

USER GUIDE

# Essential Studio for ReactJS

---

Version - v18.1.0.36 | Release Date - March 19, 2020

React JS .....	37
Getting started with React.....	37
Create React Application layout .....	37
Add React Components .....	37
Content template.....	39
React application with NPM packages.....	40
Setup .....	40
Accordion .....	45
Overview .....	45
Getting Started.....	45
Configure Accordion .....	46
Create a Simple Accordion in React JS.....	46
Configure Multiple Open .....	48
Customize Icon.....	53
AutoComplete.....	56
Overview .....	56
Getting Started.....	56
Create an AutoComplete .....	56
Data Binding.....	57
Enable Popup Button .....	58
Barcode .....	59
Overview .....	59
Create an Barcode.....	60
Supported Barcode types.....	60
Custom Design for Barcode control .....	61
Members .....	62
Events.....	71
BulletGraph .....	72
Getting Started.....	72
Adding Script Reference.....	72
Control Initialization.....	73
Using jsx Template .....	73
Without using jsx Template .....	81
Bullet Graph Dimensions .....	84
Size .....	84

Value for performance bar .....	85
Comparative measure value .....	85
Theme .....	85
Orientation.....	86
Flow direction .....	86
Qualitative range size.....	86
Quantitative scale length .....	87
Bullet Graph Caption.....	87
Title .....	87
Subtitle.....	88
Indicator .....	89
Trim .....	90
Text Placement .....	90
Localization .....	93
Data Binding.....	93
Local Data.....	93
Remote Data .....	94
Quantitative Scale .....	95
Range for Quantitative Scale .....	95
Quantitative scale location .....	95
Major ticks.....	96
Minor ticks .....	96
Tick position .....	97
Tick Placement .....	97
Quantitative scale labels .....	98
Label Placement.....	98
Performance measure bar .....	99
Comparative measure symbol .....	99
Multiple performance measures comparison .....	100
Qualitative Range.....	100
User Interaction .....	101
Animation.....	101
Responsiveness during browser resize .....	101
Applying same color to all ticks and labels in a range .....	102
Tooltip .....	102

Methods.....	103
Events.....	104
Button .....	107
Overview .....	107
Getting Started.....	108
Control Initialization with react js .....	108
Create a Button without using JSX template .....	109
Configure Properties .....	109
Chart.....	110
Getting Started.....	110
Adding Script Reference.....	110
Control Initialization.....	111
Using jsx Template .....	111
Populate chart with data .....	113
Add Data Labels .....	116
Enable Legend.....	118
Enable Tooltip .....	119
Add Chart Title .....	120
Without using jsx Template .....	121
Working with Data .....	123
Local Data.....	123
Remote Data .....	124
Chart Dimensions.....	125
Set size for the container .....	125
Set size in pixels .....	125
Setting size relative to the container size .....	125
Responsive chart.....	126
Axis.....	126
Category Axis.....	126
Numeric Axis .....	128
DateTime Axis .....	131
DateTime Category Axis.....	134
Logarithmic Axis.....	136
Label Format .....	138
Common axis features .....	140



Multiple Axis .....	148
Smart Axis Labels .....	149
Multi-level Labels .....	150
Multiple panes .....	152
Column Definitions .....	154
ChartTypes .....	155
Line Chart .....	155
Step Line Chart .....	161
Area Chart .....	166
Range Area Chart .....	167
Step Area Chart .....	168
Spline Area Chart .....	169
Stacked Area Chart .....	170
100% Stacked Area Chart .....	171
Stacked Spline Area Chart .....	172
100% Stacked Spline Area Chart .....	173
Column Chart .....	174
RangeColumn Chart .....	180
Stacked Column Chart .....	183
100% Stacked Column Chart .....	186
Bar Chart .....	189
Stacked Bar Chart .....	191
100% Stacked Bar Chart .....	194
Spline Chart .....	197
Pie Chart .....	201
Doughnut Chart .....	207
Multiple Pie Chart .....	214
Pyramid Chart .....	218
Funnel Chart .....	222
Bubble Chart .....	225
Scatter .....	226
HiLoOpenClose Chart .....	228
Candle .....	231
HiLo .....	233
Polar .....	235

Radar Chart .....	237
Waterfall Chart.....	240
Error bar Chart .....	243
Box and Whisker Chart.....	249
Pie Of Pie Chart .....	253
Chart Series .....	257
Multiple Series .....	257
Combination Series .....	259
Data Markers .....	260
Add Shapes.....	260
Add image as marker .....	261
Add labels.....	261
Contrast Color for the data label .....	266
Binding label from the datasource.....	266
Binding fill color to the points from the datasource.....	267
Customize specific points.....	267
Connect Line .....	269
Smart labels.....	270
Legend.....	270
Legend Visibility .....	270
Legend title .....	271
Position and Align the Legend .....	271
Arrange legend items in the rows and columns .....	272
Customization .....	273
Handle the legend item clicked.....	277
Series selection on legend item click .....	277
Collapsing legend item.....	278
Empty Points .....	278
EmptyPointSettings.....	279
Customizing Styles .....	279
Chart Title & Subtitle.....	280
Title .....	280
Add Subtitle to the chart.....	281
Striplines .....	283
Horizontal Stripline .....	283

Vertical Stripline.....	284
Customize the Text .....	285
Customize the Stripline .....	286
Change the Z-order of the stripline .....	286
User Interactions.....	287
Tooltip .....	287
Zooming and Panning .....	290
Crosshair .....	292
Trackball.....	293
Highlight .....	297
Selection.....	301
Data Editing.....	307
Performance .....	308
Lazy Loading.....	309
Trendlines .....	309
Customize the trendline styles.....	310
Types of Trendline.....	310
Forecasting.....	313
Trendlines Legend.....	314
Technical Indicators .....	315
Bind data to render the indicator .....	315
Indicator Types.....	316
Enable Tooltip .....	321
Annotations.....	322
Rotate the annotation template.....	322
Positioning Annotation .....	323
Annotation alignments .....	324
Appearance .....	324
Custom Color Palette .....	324
Built-in Themes .....	325
Point level customization.....	325
Series border customization .....	326
Chart area customization.....	326
Rendering Modes.....	330
VML .....	330

SVG .....	330
Canvas .....	330
Real-Time Chart .....	331
Exporting Chart .....	331
Client-side Exporting .....	332
Server-side Exporting .....	332
Naming the exported file .....	336
Rotating the chart .....	337
Setting orientation for the document .....	337
Printing Chart .....	337
Printing Multiple chart .....	338
Page Setup .....	338
3D Chart .....	338
3D Column Chart .....	338
3D Bar Chart .....	339
3D Stacked Column Chart .....	339
3D 100% Stacked Column Chart .....	340
3D Stacked Bar Chart .....	341
3D 100% Stacked Bar Chart .....	341
3D Pie Chart .....	342
3D Doughnut Chart .....	342
Configure 3D Chart .....	343
Localization .....	345
Exporting Service .....	345
Checkbox .....	346
Overview .....	346
Key Features .....	346
GettingStarted .....	346
Create a CheckBox .....	347
CircularGauge .....	350
Getting Started .....	350
Adding Script Reference .....	350
Control Initialization .....	351
Using jsx Template .....	351
Set Height and Width .....	352

Set Background Color.....	353
Provide Scale Values .....	354
Add Label Customization .....	355
Add Pointers.....	356
Add Tick Details.....	358
Add Range Values .....	359
Add Indicator Details.....	361
Add Custom Label Details .....	363
Without using jsx Template .....	365
Interaction and Animation.....	366
Gradient .....	367
Distance From Corner .....	367
Resize .....	368
Localization .....	368
Themes.....	368
Circular Gauge Values .....	369
Scales.....	369
Adding Scale Collection.....	369
Scale Customization .....	370
Multiple Scales .....	372
Pointers .....	373
Adding Pointer Collection .....	373
Adding Pointer Value .....	373
Pointer Styles .....	374
Pointer Images .....	377
Multiple Pointers .....	379
Pointer Value Text.....	380
Appearance .....	381
Font Options .....	382
Labels .....	383
Adding Label Collection.....	383
Label Customization.....	383
Multiple Labels.....	385
Ticks.....	386
Adding Tick Collection.....	386

Tick Customization .....	386
Indicators .....	387
Adding Indicator Collection.....	387
Basic Customization .....	388
State Ranges.....	388
Multiple Indicators.....	389
Ranges and Frames .....	391
Adding Range Collection .....	391
Range Customization .....	391
Colors and Border .....	392
Positioning the ranges .....	392
Multiple Ranges .....	393
Frames.....	394
Legend.....	395
Legend Visibility .....	395
Position and Align the Legend .....	396
Customization .....	396
Events.....	399
Custom labels.....	400
Adding Custom Label Collection .....	400
Multiple Custom Labels .....	401
Outer Custom Label .....	402
Tooltip .....	403
Default Tooltip .....	403
Tooltip Template.....	404
Sub Gauges.....	405
Adding SubGauges .....	405
Multiple SubGauges .....	406
Gauge Position .....	407
Exporting .....	408
Methods.....	409
Events.....	437
ColorPicker .....	441
Overview .....	441
Getting Started.....	441

Create an ColorPicker .....	442
Configure Properties .....	442
CurrencyTextbox .....	443
Getting Started.....	443
Create a CurrencyTextbox.....	443
Configure Properties .....	443
DatePicker .....	444
OverView.....	444
Key features .....	444
Getting Started.....	444
Create a DatePicker.....	444
DateTimePicker .....	446
Overview .....	446
Key Features.....	446
GettingStarted.....	446
Create a DateTimePicker .....	447
Diagram.....	450
Overview .....	450
Getting started.....	451
Adding Script Reference.....	452
Control Initialization.....	453
Using jsx Template .....	453
Without using jsx Template .....	456
Dialog .....	457
Overview .....	457
Key Features:.....	457
Getting Started.....	458
Create a simple Dialog .....	458
Add content to Dialog .....	458
Set header text.....	459
Open Dialog dynamically .....	459
DigitalGauge.....	460
Getting Started.....	460
Adding Script Reference.....	460
Control Initialization.....	461

Using jsx Template .....	461
Set Height and Width values.....	462
Set Items Property .....	463
Add Background Image .....	463
Add Location .....	464
Add Items Collection .....	465
Without using jsx Template .....	466
Basic Settings .....	467
Height and Width Customization.....	467
Responsive Layout .....	467
Themes.....	467
Frames.....	468
Inner and Outer Width Customization.....	468
Setting Background Image .....	468
Digital Elements .....	469
Segment Settings .....	470
Appearance .....	470
Dimension Modification.....	470
Character Settings.....	471
Appearance .....	471
Count and Type .....	472
Text Positioning.....	473
Shadow Effects.....	473
Font Customization .....	474
Multiple Items.....	475
Exporting the Digital Gauge .....	476
Methods.....	476
Events.....	479
DropDownList .....	480
DropDownList .....	480
Key Features.....	480
Getting Started.....	481
Creating DropDownList in React JS .....	481
Populating data .....	482
Setting Dimensions .....	483



FileExplorer .....	484
Getting Started.....	484
Create an FileExplorer.....	484
Grid.....	485
Overview .....	485
Getting started .....	486
Preparing HTML document.....	486
Create a Grid .....	488
Using jsx Template .....	488
Data binding.....	488
Enable Paging.....	489
Enable Filtering .....	490
Enable Grouping.....	491
Add Summaries .....	493
Data binding.....	495
Local Data.....	495
Remote Data .....	497
Columns .....	499
Column Template.....	500
Row .....	501
Details Template .....	501
Row Template .....	503
Editing .....	506
Toolbar with edit option .....	506
Cell edit type and its params.....	507
Cell Edit Template .....	509
Edit Modes .....	511
Confirmation messages.....	529
Column Validation.....	532
Persisting data in Server .....	534
Adding New Row Position.....	542
Render with blank row for easy add new .....	543
Default column values on add new.....	545
Filtering .....	546
Menu filter .....	548

Excel-like filter.....	551
Filter bar.....	555
Filter Operators.....	559
FilterBar Template .....	559
Selection.....	561
Types of Selection .....	561
Row Selection.....	562
Cell Selection.....	563
Column Selection .....	564
Touch options .....	565
Toggle Selection .....	567
GroupButton .....	567
Overview .....	567
Key Features.....	567
Getting Started.....	568
Create a GroupButton Using JSX template .....	568
Create a GroupButton without using JSX template .....	568
Configure Properties .....	569
Gantt .....	569
Overview .....	569
Getting Started.....	570
Create your first Gantt in ReactJS .....	570
Adding script references .....	570
Initialize the Gantt with data source.....	571
Create a Gantt.....	571
Using jsx Template .....	571
Enable Toolbar .....	573
Enable Sorting.....	574
Enable Editing .....	575
Enable Context Menu .....	576
Enable Column Menu.....	577
Provide tasks relationship.....	577
Provide Resources.....	578
Highlight Weekend.....	579
HeatMap .....	580

Overview .....	580
Key features: .....	580
Getting Started.....	580
Adding Script Reference.....	581
Control Initialization.....	582
Using jsx Template .....	582
Without using jsx Template .....	584
Initialize Legend .....	585
Kanban Board.....	586
Overview .....	586
Getting Started.....	586
Preparing HTML document.....	586
Create a Kanban .....	588
Using jsx Template .....	588
Without using jsx Template .....	592
Columns .....	593
Key Mapping .....	593
Multiple Key Mapping.....	594
Headers .....	595
Width .....	597
Visibility .....	597
Toggle.....	598
Allow Dragging .....	599
Allow Dropping .....	600
Items Count.....	601
Customize Items Count Text .....	602
Workflows .....	603
Swim lanes .....	604
Drag And Drop between swim lanes.....	605
Unassigned swim lane group .....	606
Drag and Drop.....	607
Prioritization of cards.....	608
Cards .....	608
Customization .....	608
Template .....	610

Tooltip .....	612
Editing .....	615
Configuring Edit Items.....	615
Edit modes .....	616
Cell edit type and its params.....	625
Column Validation.....	627
Filtering .....	629
Scrolling.....	630
Set width and height in pixel .....	630
Set height and width in percentage.....	631
Set width as auto .....	632
Enabling freeze swim lane .....	633
Selection and Hovering .....	634
Types of Selection .....	634
Responsive .....	635
Mobile Layout .....	636
Width .....	643
Min Width .....	644
Stacked Headers .....	645
Adding Stacked header columns.....	645
Context Menu .....	646
Default Context Menu items.....	646
Custom Context Menu .....	648
Sub Context Menu .....	649
Print.....	651
Localization .....	652
Localization .....	652
Styling.....	654
List of classes and its purposes .....	654
Web Accessibility .....	655
Keyboard Navigation.....	655
LinearGauge .....	656
Getting Started.....	656
Adding Script Reference.....	656
Control Initialization.....	657

Using jsx Template .....	658
Set Height and Width values.....	659
Set Animation option and Label Color .....	659
Provide Scale Values .....	660
Add Pointers.....	661
Add Label Customization .....	663
Add Tick Details.....	664
Add Custom Label Details .....	665
Change Scale from Degree to Fahrenheit .....	667
Add Custom Label for Current Value .....	668
Without using jsx Template .....	669
Basic Settings .....	671
Adding Dimension .....	671
Adding frame.....	671
Appearance .....	672
Responsive .....	673
Localization .....	675
Interaction and Animation.....	675
Scales.....	676
Adding scale collection.....	676
Scale Customization .....	677
Appearance .....	679
Scale Types.....	680
Adding multiple scales .....	683
Marker Pointers .....	684
Adding marker pointer collection .....	684
Add marker pointer value .....	685
Pointer Styles .....	686
Positioning the pointer .....	687
Types .....	688
Bar Pointers.....	690
Adding bar pointer collection .....	690
Adding bar pointer value .....	691
Pointer Styles .....	692
Positioning the pointer .....	693

Multiple Bar Pointers .....	694
Labels .....	695
Label Customization .....	695
Unit text and Positioning .....	696
Ticks.....	697
Adding tick collection.....	698
Tick Customization .....	699
Types .....	700
Positioning the ticks .....	701
Ranges.....	702
Adding range collection .....	702
Range Customization .....	703
Colors and Border .....	704
Positioning the ranges .....	705
Multiple Ranges .....	706
Custom labels.....	708
Adding Custom label collection .....	708
Basic Customization .....	709
Locating the CustomLabels .....	710
Multiple Custom Labels .....	711
Indicators .....	712
Setting Dimension .....	712
State Ranges.....	713
Color and Appearance .....	715
Font options .....	716
Multiple Indicator .....	718
Exporting.....	719
Methods.....	721
Events.....	748
ListBox.....	753
Getting Started.....	753
Key Features.....	753
Create an ListBox .....	753
Selection.....	754
ListView .....	755

Getting Started.....	755
Key Features.....	755
Create an ListView .....	755
Add Header .....	756
Maps .....	757
Getting Started.....	757
Create a Map.....	758
Adding Script Reference.....	758
Control Initialization.....	759
Using jsx Template .....	759
Initialize Map.....	761
Data Binding in Map.....	762
Enable Tooltip .....	766
Legend.....	767
Without using jsx Template .....	769
Populate Data .....	770
Shape Data .....	770
Data Binding.....	771
Customization .....	773
Shape Settings.....	773
Color Mapping.....	774
Color Palette .....	777
Tooltip .....	778
Map Elements .....	781
Markers .....	781
Bubbles.....	782
Legend.....	784
User Interaction .....	787
Map Selection .....	787
MultiSelection .....	787
Dragging On Selection.....	788
Zooming .....	788
Panning .....	791
Navigation Control .....	792
Layers .....	795

Multilayer .....	795
SubLayer .....	796
Map Providers .....	796
Open Street Map .....	796
Bing Map .....	797
Methods .....	797
Events .....	799
MaskEdit .....	802
Getting Started .....	802
Create a MaskEdit .....	802
Configure Properties .....	802
Menu .....	803
Overview .....	803
Getting Started .....	803
Create a Menu in React JS .....	804
Configure parent Menu items .....	804
Initialize sub-level Menu items .....	805
Define multiple level Menu items .....	806
Navigation Drawer .....	807
Overview .....	807
Getting Started .....	808
Create a Navigation Drawer .....	808
Configuring properties .....	808
Customize Direction .....	813
NumericTextbox .....	813
Getting Started .....	813
Create a NumericTextbox .....	813
Configure Properties .....	814
PDF viewer .....	814
Overview .....	814
Getting Started .....	815
Script and CSS Reference .....	815
Initialize and configure the control .....	816
Using jsx Template .....	816
Without using jsx Template .....	817



PercentageTextbox .....	818
Getting Started.....	818
Create a PercentageTextbox.....	818
Configure Properties .....	818
PivotChart .....	819
Overview .....	819
Getting Started.....	819
Script and CSS Reference .....	819
Relational .....	820
OLAP.....	823
PivotGrid .....	824
Overview .....	824
Getting Started.....	825
Script and CSS Reference .....	825
Relational .....	826
OLAP.....	831
Relational .....	832
Grouping Bar .....	832
PivotTable Field List .....	838
Value Sorting.....	845
Calculated Field.....	846
Cell Editing .....	848
Sub Total Hiding .....	849
Collapse by default.....	849
OLAP .....	850
KPI .....	850
Named Sets .....	851
Grouping Bar .....	852
PivotTable Field List .....	856
Common.....	862
PivotGrid: Elements .....	862
Number Format.....	866
Save and Load Report .....	868
Grid Layout.....	869
Grand Total Hiding .....	871

Advanced Filtering & Sorting .....	873
Exporting .....	875
Frozen Header.....	880
Drill Through .....	883
Column Resizing.....	886
Styling.....	887
Localization and Globalization .....	890
Responsive .....	902
ToolTip .....	904
PivotGauge.....	905
Overview .....	905
Getting Started.....	905
Script and CSS Reference .....	906
Relational .....	906
OLAP.....	911
PivotTreeMap.....	914
Overview .....	914
Getting Started.....	915
Script and CSS Reference .....	915
ProgressBar .....	917
Overview .....	917
Getting Started.....	918
Create a ProgressBar.....	918
Progress Control using Length of the Password Field.....	921
RadialMenu .....	924
Overview .....	924
Key Features.....	924
Getting Started.....	924
Create a RadialMenu.....	925
Configure Items.....	925
Displaying RadialMenu.....	927
RadialMenu item functionalities.....	927
RadialSlider .....	929
Overview .....	929
Key Features.....	929

Getting Started.....	929
Create a simple Radial Slider.....	929
RadioButton .....	930
Overview .....	930
GettingStarted.....	930
Create a RadioButton.....	931
RangeNavigator.....	933
Getting Started.....	933
Create your RangeNavigator.....	934
Adding Script Reference.....	934
Control Initialization.....	935
Using jsx Template .....	935
Add series.....	936
Enable tooltip.....	937
Update Chart.....	938
Set value type.....	939
Without using jsx Template .....	940
Numeric Type.....	941
DateTime.....	942
DateTime Intervals.....	942
Padding .....	943
AllowSnapping .....	944
Responsive .....	944
Auto Resizing.....	944
Customize range Navigator border.....	944
Customize size of range navigator .....	944
Value Axis Settings .....	949
Selected Range Settings.....	950
User Interactions.....	956
Highlight .....	956
Selection.....	957
Scrollbar .....	958
Methods.....	959
Events.....	959
Rating .....	962

Overview .....	962
Key Features.....	962
Getting Started.....	962
Create a Rating Widget in React JS .....	963
Set the Min and Max Value.....	965
Set Precision.....	966
ReportDesigner .....	967
Overview .....	967
Key features .....	967
Getting Started.....	967
Script and CSS Reference .....	968
Initialize and configure the control.....	969
ReportViewer .....	971
Overview .....	971
Key Features.....	971
Getting Started.....	972
Script and CSS Reference .....	972
Initialize and configure the control.....	973
Load SSRS Server Reports .....	975
Load RDLC Reports.....	977
How To .....	980
Load unsupported fonts.....	980
Ribbon .....	981
Overview .....	981
Key Features.....	981
Getting Started.....	981
Script & CSS Reference.....	981
Control Initialization.....	982
Using jsx Template .....	982
Adding Tabs.....	983
Configuring Groups .....	984
Adding Controls to Group .....	985
Without using jsx Template .....	986
Application Tab .....	988
Application Menu.....	988

Backstage Page.....	989
Tab .....	991
Group .....	992
Adding Tab Groups.....	992
Group Expander .....	996
Controls Support.....	997
Gallery .....	999
Gallery Items .....	999
Custom Gallery Items.....	1000
Resize .....	1002
Tablet Layout .....	1002
Mobile Layout .....	1004
Screen Tips .....	1006
HTML Tooltip.....	1006
Custom Tooltip.....	1007
Quick Access Toolbar .....	1013
Globalization and Localization .....	1015
Localization .....	1015
Right to Left - RTL.....	1017
RichTextEditor .....	1018
Overview .....	1018
Getting Started.....	1019
Create RTE Control in React JS .....	1019
Toolbar–Configuration.....	1019
Setting and Getting Content .....	1020
Rotator .....	1022
Overview .....	1022
Key Features.....	1022
Getting Started.....	1022
Create a Rotator.....	1022
Configure data.....	1023
Schedule.....	1024
Overview .....	1024
External and Internal Dependencies.....	1025
Getting Started.....	1026

Control Initialization.....	1027
Working with timeZone Settings .....	1029
Data Binding.....	1030
Appointment Fields.....	1030
Appointment Field Validation .....	1032
Binding to JSON Data Array.....	1033
Binding Remote Data Service.....	1034
OData V4 .....	1034
WebAPI Binding .....	1035
Data binding using OLEDB.....	1035
ASP.NET Web Method Binding .....	1039
MVC Controller Action Binding .....	1048
Loading Data on Demand.....	1051
Views.....	1053
Day .....	1054
Week .....	1054
Work Week .....	1055
Month .....	1055
Custom .....	1056
Agenda .....	1057
Restriction on View Navigation.....	1058
Timeline View.....	1058
Working with Appointments.....	1059
Appointment Types.....	1059
CRUD operation .....	1059
Handling Appointment Actions.....	1067
Read Only .....	1068
Drag and Drop.....	1068
Resize .....	1074
Categorization .....	1075
Priority.....	1077
Search or Filter Appointments .....	1079
Recurrence Options .....	1081
Reminder.....	1085
Block Time Intervals .....	1086

Context Menu .....	1090
Default Menu Options .....	1090
Custom Menu Options .....	1092
Handling Menu Actions .....	1092
Adding Categorize Option .....	1094
Resources .....	1096
Fields of Resources .....	1096
Data Binding .....	1098
Multiple Resources (Without Grouping) .....	1100
Grouping .....	1101
Different Working days and Hours for Resources .....	1103
Customization .....	1105
Hour Customization .....	1105
TimeScale .....	1106
Hide Weekend days .....	1107
Date Customization .....	1108
Appointment Window Customization .....	1109
Scheduler Customization using queryCellInfo .....	1116
Navigation .....	1117
View Navigation .....	1117
Date Navigation .....	1118
Appointment Navigation .....	1118
Template .....	1119
Appointment Template .....	1119
Cell Templates .....	1120
Date Header Template .....	1122
Resource Header Template .....	1123
TimeScale Templates .....	1125
Priority Settings Template .....	1126
Tooltip Template .....	1127
Agenda View Templates .....	1128
Globalization and Localization .....	1130
Globalization .....	1130
Localization .....	1130
Time Zone .....	1133

Time Mode .....	1135
Date Format .....	1136
First Day of Week .....	1136
Keyboard Navigation.....	1137
Setting Dimension .....	1138
Scheduler Dimension .....	1138
Scheduler Cell Dimensions .....	1138
Responsiveness .....	1140
Auto-Resizing Scheduler .....	1140
Scheduler in Mobile/Tablets .....	1141
Persistence .....	1143
Export and Print .....	1143
Export Appointments to ICS file .....	1143
PDF Export.....	1146
Excel Export.....	1150
Print.....	1152
Import Appointments .....	1154
Miscellaneous .....	1155
Time Indicator .....	1155
Show/Hide All-Day Row .....	1156
Show/Hide Header bar.....	1156
Show/Hide TimeScale .....	1157
Show/Hide Location Field .....	1157
Recurrence Editor .....	1158
Getting Started.....	1158
Control Initialization.....	1159
Generating Recurrence Rule .....	1159
How To .....	1160
Validate the Custom Appointment Window Fields .....	1160
Highlight Different Work Hours for Each Resources.....	1161
Display Scheduler with Appointments Filtered by Subject.....	1162
Customize the Default Appointment Window.....	1164
Synchronize the Schedule with Outlook .....	1165
Scroller .....	1168
Overview .....	1168



Key Features.....	1168
Getting Started.....	1168
Create a Scroller .....	1168
Signature .....	1170
Overview .....	1170
Key Features.....	1170
Getting Started.....	1170
Create a Signature component in ReactJS .....	1171
Adjusting Signature Size.....	1172
SplitButton .....	1173
Overview .....	1173
Getting Started.....	1173
Create a SplitButton.....	1173
Configure Properties .....	1174
Splitter.....	1174
Overview .....	1174
Key Features.....	1175
Getting Started.....	1175
Create a Splitter .....	1175
Configure Splitter Panes .....	1176
Configure Tree View.....	1176
Set Actions .....	1177
Sparkline .....	1177
Getting Started.....	1177
Adding Script Reference.....	1177
Control Initialization.....	1178
Using jsx Template .....	1179
Populate Sparkline with data .....	1179
Sparkline Type.....	1180
Enable Tooltip .....	1181
Without using jsx Template .....	1181
Working with Data .....	1182
Local Data.....	1182
Sparkline Dimensions.....	1182
Set size for the container .....	1183

Set size in pixels .....	1183
Responsive Sparkline .....	1183
Sparkline Types .....	1183
Line Type .....	1183
Column Type .....	1184
Area Type .....	1184
WinLoss Type .....	1184
Pie Type .....	1185
Axis Customize .....	1185
Marker Customization .....	1185
Point Customization .....	1186
Sparkline Customization .....	1186
Sparkline background .....	1186
Stroke color and width .....	1186
Sparkline border .....	1187
Opacity .....	1187
Localization .....	1187
Padding for Sparkline .....	1187
Canvas support .....	1188
Themes .....	1188
Tooltip .....	1188
Tooltip Template .....	1188
Range Band .....	1189
Methods .....	1189
Events .....	1190
SpellCheck .....	1192
Overview .....	1192
Getting Started .....	1193
Control Initialization .....	1193
Spell Checking .....	1194
SpellCheck Operations .....	1194
Dialog Mode .....	1194
Context Menu Mode .....	1199
Functionalities .....	1204
Check Spelling .....	1204

Ignore Words .....	1204
Ignore Settings .....	1206
Change Words.....	1207
Custom Words .....	1208
Checking content on typing .....	1209
Suggestion Words .....	1210
Synchronous request .....	1210
SpellCheck Customization .....	1211
Misspell Word Appearance .....	1211
Restrict Suggestion Count .....	1212
Localization .....	1213
Supported Target Elements .....	1216
Multiple Target .....	1216
Responsive .....	1217
Spreadsheet .....	1219
Overview .....	1219
Getting started .....	1220
Adding Script Reference.....	1220
Initialize Spreadsheet.....	1221
Populate Spreadsheet with data.....	1222
Apply Conditional Formatting.....	1223
Export Spreadsheet as Excel File.....	1225
Sunburst Chart .....	1226
Getting Started.....	1226
Adding Script Reference.....	1226
Control Initialization.....	1227
Using jsx Template .....	1227
Populate Data source:.....	1227
Add Title to the Sunburst Chart .....	1229
Enable Legend.....	1229
Add Data Labels .....	1230
Sunburst Elements .....	1230
Start and End Angle.....	1230
Sunburst Radius .....	1230
Sunburst Inner Radius.....	1231

Levels.....	1231
GroupMemberPath .....	1231
Legend.....	1232
Legend Icon .....	1232
Positioning the Legend.....	1233
Legend Item Size and border .....	1233
Legend Alignment .....	1233
Legend Size.....	1234
Legend title .....	1234
Customize the legend text .....	1234
Legend Row and Column .....	1235
LegendInteractivity .....	1235
ToggleSegmentSelection.....	1235
Toggle Segment Visibility .....	1236
Data Labels.....	1236
Label Overflow mode.....	1236
Label Rotation Mode.....	1237
Customizing the data labels .....	1238
Sunburst Chart Title & Subtitle .....	1238
Tooltip .....	1239
Tooltip Template.....	1239
Customize the appearance of tooltip .....	1240
Highlight .....	1240
Highlight Display mode .....	1240
Highlight Mode .....	1241
Selection.....	1242
Selection Display mode.....	1243
Selection Mode .....	1243
Zooming .....	1244
Zooming toolbar.....	1245
Animation.....	1245
Animation Types .....	1245
Appearance .....	1246
Palette .....	1246
Built- in Themes .....	1246

Methods.....	1247
Events.....	1248
Slider .....	1252
Getting Started.....	1252
Create a Slider .....	1252
Configure Properties .....	1252
Tab .....	1252
Overview .....	1252
Key Features.....	1253
Getting Started.....	1253
Create Tab Control in React JS .....	1254
Configure Content.....	1255
Create the Rating .....	1255
Ajax Content Load (Load On Demand).....	1257
Orientation Change.....	1260
Header Image Customization.....	1261
Configuring Contents to remaining Tab items .....	1262
TagCloud .....	1265
Overview .....	1265
Getting Started.....	1265
Create TagCloud widget in React JS .....	1265
Set Min and Max Font Size.....	1266
Set event to perform an operation .....	1267
Tile.....	1268
Overview .....	1268
Key Features.....	1268
Getting Started.....	1268
Create a Tile .....	1268
Configuring properties .....	1269
Template support with Tile Component.....	1271
TimePicker.....	1272
Overview .....	1272
Key Features.....	1272
Getting Started.....	1273
Adding Scripts and Style references .....	1273

Configure Properties .....	1273
ToggleButton.....	1275
Overview .....	1275
Getting Started.....	1275
Create an ToggleButton .....	1275
Configure Properties .....	1276
TreeGrid .....	1276
Overview .....	1276
Getting Started.....	1277
Create your first TreeGrid in React JS .....	1277
Create a TreeGrid .....	1278
Using jsx Template .....	1278
TreeMap .....	1282
Getting Started.....	1282
Create a TreeMap .....	1283
Adding Script Reference.....	1283
Control Initialization.....	1284
Using jsx Template .....	1284
Initialize TreeMap .....	1284
GroupTreeMap Items using Levels .....	1286
Customize TreeMap Appearance by Range .....	1287
Enable Tooltip .....	1288
Legend.....	1290
Without using jsx Template .....	1291
DataBinding.....	1292
TreeMapLevels.....	1293
Flat Level .....	1293
Hierarchical Level .....	1294
Layout.....	1295
Squarified .....	1295
SliceAndDiceAuto.....	1296
SliceAndDiceHorizontal.....	1296
SliceAndDiceVertical .....	1296
Customization .....	1296
Color .....	1296

Tooltip .....	1298
Leaf Item Setting .....	1298
Dock Position .....	1299
TreeMap Elements .....	1301
Legend .....	1301
Header .....	1303
Customizing the header .....	1303
Label .....	1304
Customizing the Overflow labels .....	1304
Drill Down Support .....	1305
Enable Drill Down .....	1305
Methods .....	1306
Events .....	1306
TreeView .....	1308
Getting Started .....	1308
Create an TreeView .....	1309
Data Binding .....	1310
Toolbar .....	1311
Overview .....	1311
Key Features .....	1312
Getting Started .....	1312
Create Toolbar for PDF Reader .....	1312
Create Toolbar control in React JS .....	1312
Initialize Toolbar Items .....	1313
Render remaining Toolbar items .....	1314
Add Actions to Toolbar Items .....	1317
Uploadbox .....	1320
Overview .....	1320
Getting Started .....	1320
Create Uploadbox widgets in React JS .....	1321
Set Restriction for File Extension .....	1324
Upload Multiple Files .....	1326
WaitingPopup .....	1327
Overview .....	1327
Getting Started .....	1327

Create Username and Password..... 1328

Add WaitingPopup Widget ..... 1330



## React JS

React is an DOM management library that is used to create user interfaces, it computes the minimal set of changes that makes keep your DOM up-to-date.

Essential JavaScript components are supported to React JavaScript library through wrappers in ej.web.react.min.js file.

### Getting started with React

This section briefly explains about how to create a web application with Syncfusion React components.

#### Create React Application layout

Include the following dependencies for render React components with Syncfusion widgets.

#### HTML

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8" />
<title>React JS Getting started</title>
<link href="http://cdn.syncfusion.com/14.3.0.49/js/web/flat-
azure/ej.web.all.min.css" rel="stylesheet" />
<script
src="http://cdn.syncfusion.com/js/assets/external/react.min.js"></script>
<script src="http://cdn.syncfusion.com/js/assets/external/react-
dom.min.js"></script>
<script src="https://cdnjs.cloudflare.com/ajax/libs/babel-
core/5.8.34/browser.min.js"></script>
<script src="http://cdn.syncfusion.com/js/assets/external/jquery-
3.0.0.min.js"></script>
<script
src="http://js.syncfusion.com/demos/web/scripts/jsondata.min.js"></script>
<script
src="http://cdn.syncfusion.com/js/assets/external/jsrender.min.js"></script>
<script
src="http://cdn.syncfusion.com/14.3.0.49/js/web/ej.web.all.min.js"></script>
<script
src="http://cdn.syncfusion.com/14.3.0.49/js/common/ej.web.react.min.js"></sc
ript>
</head>
<body>
<!--Add React JS components-->
</body>
</html>
```

**Information:** From React v16.x.x, support for `React.createClass` has been [removed](#). So if you are using react framework later versions then, you need to refer [create-react-class](#) package separately.

#### Add React Components

React components are typically written in JSX. It is allowing quoting of HTML and using HTML tag's syntax to render sub-components. HTML syntax is processed into JavaScript calls of the React library.

To translate JSX to plain JavaScript, we must use `<script type="text/babel">` and refer the `browser.min.js` file to perform the transformation in the browser.

To render the React components, you must have defined the HTML div element with id attribute in HTML file which is target element of React components.

Create a **HTML** file and paste the following content

### HTML

```
<div id="container"></div>
<script type="text/babel" src="app.jsx">
</script>
```

Create a JSX file and paste the following content

### JS

```
ReactDOM.render(
  <EJ.Grid dataSource = {window.gridData} pageSettings-currentPage={2}
  allowGrouping = {true}
  allowPaging = {true} allowSorting= {true}>
    <columns>
      <column field="OrderID" headerText="Order ID" textAlign="right" width={90}
      />
      <column field="EmployeeID" headerText="Employee ID" textAlign="right"
      width={90} />
      <column field="Freight" headertext="Freight" width={90} />
    </columns>
  </EJ.Grid>,
  document.getElementById('container')
);
```

Output of above code

Order ID	Employee ID	Freight
10260	4	55.09
10261	4	3.05
10262	8	48.29
10263	9	146.06
10264	6	3.67
10265	2	55.28
10266	3	25.73
10267	4	208.58
10268	8	66.29
10269	5	4.56
10270	1	136.54
10271	6	4.54

2 of 17 pages (200 items)

### Content template

It provides to render the multiple components inside of single components. For example if you are using the **EJ.Grid** component inside the **EJ.Dialog**, it can be rendered as like mentioned in the below code example

#### HTML

```
<div id="container"></div>
<script type="text/babel">
ReactDOM.render(
  <EJ.Dialog title="dialog">
    <EJ.Grid dataSource = {window.gridData} pageSettings-currentPage={2}
    allowGrouping = {true}
    allowPaging = {true} allowSorting= {true}>
    <columns>
    <column field="OrderID" headerText="Order ID" textAlign="right" width={90}
    />
    <column field="EmployeeID" headerText="Employee ID" textAlign="right"
    width={90} />
    <column field="Freight" headertext="Freight" width={90} />
    </columns>
    </EJ.Grid>
  </EJ.Dialog>,
  document.getElementById('container')
);
</script>
```

Output of above code

dialog ✕

Order ID	Employee ID	Freight
10260	4	55.09
10261	4	3.05
10262	8	48.29
10263	9	146.06
10264	6	3.67
10265	2	55.28
10266	3	25.73
10267	4	208.58
10268	8	66.29
10269	5	4.56
10270	1	136.54
10271	6	4.54

⏮ ⏪ 1 2 3 4 5 6 7 8 ... ⏩ ⏭

2 of 17 pages (200 items)

## React application with NPM packages

We can create the React application with NPM packages.

### Setup

#### Pre-requisites

We will need **Node.js** if you don't have it installed on your machine, check the link from the table below.

SN	Software	Description
----	----------	-------------

1	Node.js and NPM	Node.js is the platform needed for the Cordova development. Checkout <a href="#">Node.js package-manager</a>
---	-----------------	--

**Gulp** is a command line task runner using **Node.js** platform. It runs custom defined repetitious tasks.

To use Gulp, you need to install it as a global module through NPM.

- **npm install --global gulp**

### Configuration

This section briefly explains about how to configure the `package.json` and `gulpfile.js` file.

#### package.json

Create a `package.json` file which is best way to manage locally installed packages.

**Note:** If we used the `gulp-react` plugin, we don't need to include the `browser.js` file on the page either.

#### JS

```
{
  "name": "ejreact",
  "version": "1.0.0",
  "devDependencies": {
    "browser-sync": "^2.14.3",
    "gulp": "^3.9.1",
    "gulp-clean": "^0.3.2",
    "gulp-react": "^3.1.0",
    "gulp-watch": "^4.3.9"
  }
}
```

#### gulpfile.js

This file will give you a taste of what gulp does.

#### JS

```
var gulp = require('gulp');
var browserSync = require('browser-sync').create();
var react = require('gulp-react');
var watch = require('gulp-watch');
var clean = require('gulp-clean');
var reload = browserSync.reload;
gulp.task('build', ['clean'], function () {
  //Convert jsx to js
  gulp.src('source/samples/**/*.jsx')
    .pipe(react())
    .pipe(gulp.dest('source/jsx'));
});
gulp.task('clean', function () {
  return gulp.src('app', {read: false})
    .pipe(clean());
});
// Static Server + watching html files
gulp.task('serve', function () {
```

```
browserSync.init({
  server: {
    baseDir: "./"
  }
});
// Files watching
gulp.task('watch', ['build', 'serve'], function () {
  gulp.watch(["source/samples/**/*.{js,jsx,html}"], ['build', reload]);
})
gulp.task('default', ['watch']);
```

### Converting JSX to JavaScript with React

When building with React, you can write plain JavaScript or in [JSX](#). JSX is a preprocessor that gives you a more concise syntax.

which is easier and more readable, but it's needs to be converted to plain JavaScript through gulp-react plugin.

### JS

```
ReactDOM.render(
  <EJ.Grid dataSource = {window.gridData} pageSettings-currentPage={2}
  allowGrouping = {true}
  allowPaging = {true} allowSorting= {true}>
  <columns>
  <column field="OrderID" headerText="Order ID" textAlign="right" width={90}
  />
  <column field="EmployeeID" headerText="Employee ID" textAlign="right"
  width={90} />
  <column field="Freight" headertext="Freight" width={90} />
  </columns>
  </EJ.Grid>,
  document.getElementById('container')
);
```

With the help of [gulp-react](#) and its gulp plugin, you can convert JSX to JavaScript by piping react() into the **build** task

### JS

```
gulp.task('build', ['clean'], function () {
  //Convert jsx to js
  gulp.src('source/samples/**/*.jsx')
    .pipe(react())
    .pipe(gulp.dest('source/jsx'));
});
```

### Watching JS, JSX and HTML files for changes

Let's configure gulp's watch task to watch for JS, JSX and HTML files, then execute build and serve tasks.

### JS

```
// Files watching
gulp.task('watch', ['build', 'serve'], function () {
```

```
gulp.watch(["source/samples/**/*.{js,jsx,html}"], ['build', reload]);
})
```

### Removing files and folders

Let's configure gulp's clean task to removing files and folders.

### JS

```
gulp.task('clean', function () {
  return gulp.src('app', {read: false})
    .pipe(clean());
});
```

### Run Application

Create an index.html file, then include the following dependencies to render the React application with Essential JS widgets. This index.html will be act as a start page of the application.

### HTML

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8" />
<title>React JS Getting started</title>
<link href="http://cdn.syncfusion.com/14.3.0.52/js/web/flat-
azure/ej.web.all.min.css" rel="stylesheet" />
<script
src="http://cdn.syncfusion.com/js/assets/external/react.min.js"></script>
<script src="http://cdn.syncfusion.com/js/assets/external/react-
dom.min.js"></script>
<script src="http://cdn.syncfusion.com/js/assets/external/jquery-
3.0.0.min.js"></script>
<script
src="http://js.syncfusion.com/demos/web/scripts/jsondata.min.js"></script>
<script
src="http://cdn.syncfusion.com/js/assets/external/jsrender.min.js"></script>
<script
src="http://cdn.syncfusion.com/14.3.0.52/js/web/ej.web.all.min.js"></script>
<script
src="http://cdn.syncfusion.com/14.3.0.52/js/common/ej.web.react.min.js"></sc
ript>
</head>
<body>
<div id="container"></div>
<script src="source/samples/grid/grid.js">
</script>
</body>
</html>
```

Create a JSX file and paste the following content.

### JS

```
ReactDOM.render(
```

```
<EJ.Grid dataSource = {window.gridData} pageSettings-currentPage={2}
allowGrouping = {true}
allowPaging = {true} allowSorting= {true}>
<columns>
<column field="OrderID" headerText="Order ID" textAlign="right" width={90}
/>
<column field="EmployeeID" headerText="Employee ID" textAlign="right"
width={90} />
<column field="Freight" headerText="Freight" width={90} />
</columns>
</EJ.Grid>,
document.getElementById('container')
);
```

Now type the below commands step by step for launching ReactJS application in browser.

- npm install – To install the npm packages
- gulp – To launch application in browser



```
gulp
E:\reactjs>gulp
[14:21:23] Using gulpfile E:\reactjs\gulpfile.js
[14:21:23] Starting 'clean'...
[14:21:23] Starting 'serve'...
[14:21:23] Finished 'serve' after 16 ms
[14:21:23] Finished 'clean' after 203 ms
[14:21:23] Starting 'build'...
[14:21:23] Finished 'build' after 11 ms
[14:21:23] Starting 'watch'...
[14:21:24] Finished 'watch' after 204 ms
[14:21:24] Starting 'default'...
[14:21:24] Finished 'default' after 5.8 ms
[00] Access URLs:
-----
Local: http://localhost:3000
External: http://172.16.185.134:3000
-----
UI: http://localhost:3001
UI External: http://172.16.185.134:3001
-----
[00] Serving files from: ./
```

Screenshot of launching application



Order ID	Employee ID	Freight
10260	4	55.09
10261	4	3.05
10262	8	48.29
10263	9	146.06
10264	6	3.67
10265	2	55.28
10266	3	25.73
10267	4	208.58
10268	8	66.29
10269	5	4.56
10270	1	136.54
10271	6	4.54

2 of 17 pages (200 items)

## Accordion

### Overview

The **Accordion** control is an interface where lists of items can be collapsed or expanded. It has several collapsible panels where only one can be expanded at a time that is useful for dashboards where space is limited. Each **Accordion** control has a template for its header and its content.

### Key Features

- **Collapsible Header:** All headers are collapsible.
- **AJAX Load:** Load AJAX content in the **Accordion** content panel.
- **Icon Customization:** The expand and collapse icons can be customized.
- **Custom Event for Expand Header:** By default, **Accordion** panels can be expanded and collapsed on a single click. However, this action can be customized to occur on mouse-over or mouse-up through custom events.
- **Multiple Open:** Multiple items can be expanded.
- **Auto Size:** The content panel height can be predefined with values such as content, auto, and fill.
- **Theme:** Essential JavaScript controls feature 12 built-in themes (six flat themes and six with gradient effects), and also supports custom skin options to set user-defined themes.
- **RTL:** Accordion headers and content text can display RTL languages.
- **Keyboard Navigation:** You can expand and collapse panels using the keyboard.

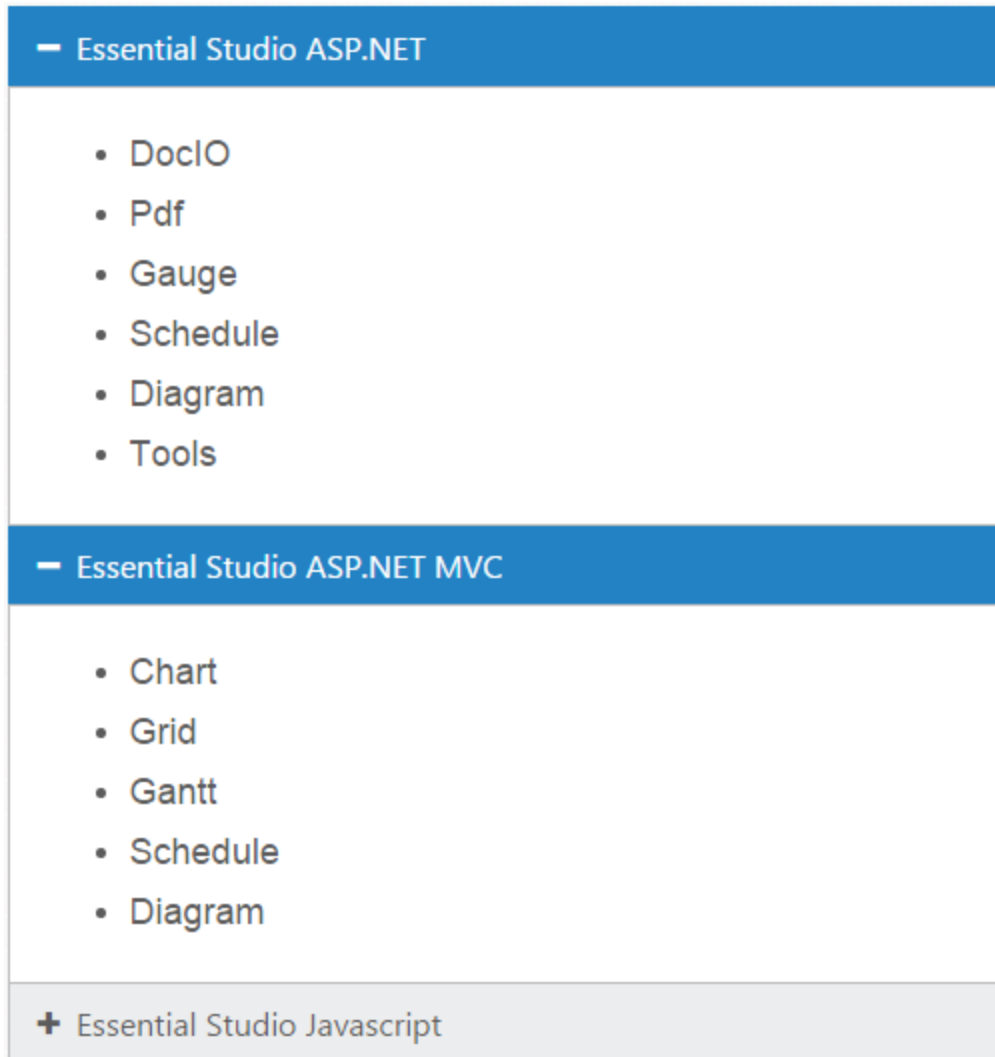
### Getting Started

This section explains briefly about how to create an **Accordion** in your application with **React JS**.

### Configure Accordion

This section encompasses the details on how you can configure the **Accordion** control in your application and customize it with various properties such as multiple open, rounded corner and icons for the **Accordion** header according to your requirement.

The following screenshot illustrates you the usage of **Accordion** control in listing the controls under the Essential Studio products.



The usage of **Accordion** control is described in the following sections.

#### Create a Simple Accordion in React JS

You can create a React application and add necessary scripts and styles with the help of the given [React Getting Started Documentation](#).

Create a JSX file for rendering Accordion component using <EJ.Accordion> syntax. Add required properties to it in <EJ.Accordion> tag element

#### JS

```
"use strict";
ReactDOM.render(
```

```

<EJ.Accordion>
<h3>
<a href="#">Essential Studio ASP.NET</a>
</h3>
<div>
<ul>
<li>
<h4>DocIO</h4>
</li>
<li>
<h4>Pdf </h4>
</li>
<li>
<h4>Gauge </h4>
</li>
<li>
<h4>Schedule </h4>
</li>
<li>
<h4>Diagram </h4>
</li>
<li>
<h4>Tools </h4>
</li>
</ul>
</div>
<h3>
<a href="#">Essential Studio ASP.NET MVC</a>
</h3>
<div>
<ul>
<li>
<h4>Chart </h4>
</li>
<li>
<h4>Grid </h4>
</li>
<li>
<h4>Gantt </h4>
</li>
<li>
<h4>Schedule </h4>
</li>
<li>
<h4>Diagram </h4>
</li>
</ul>
</div>
<h3>
<a href="#">Essential Studio Javascript</a>
</h3>
<div>
<ul>
<li>
<h4>Chart </h4>
</li>
<li>

```

```

<h4>Grid </h4>
</li>
<li>
<h4>Gantt </h4>
</li>
<li>
<h4>Schedule </h4>
</li>
<li>
<h4>Diagram </h4>
</li>
</ul>
</div>
</EJ.Accordion>,
document.getElementById('accordion-default')
);

```

Define an HTML element for adding Accordion in the application and refer the JSX file.

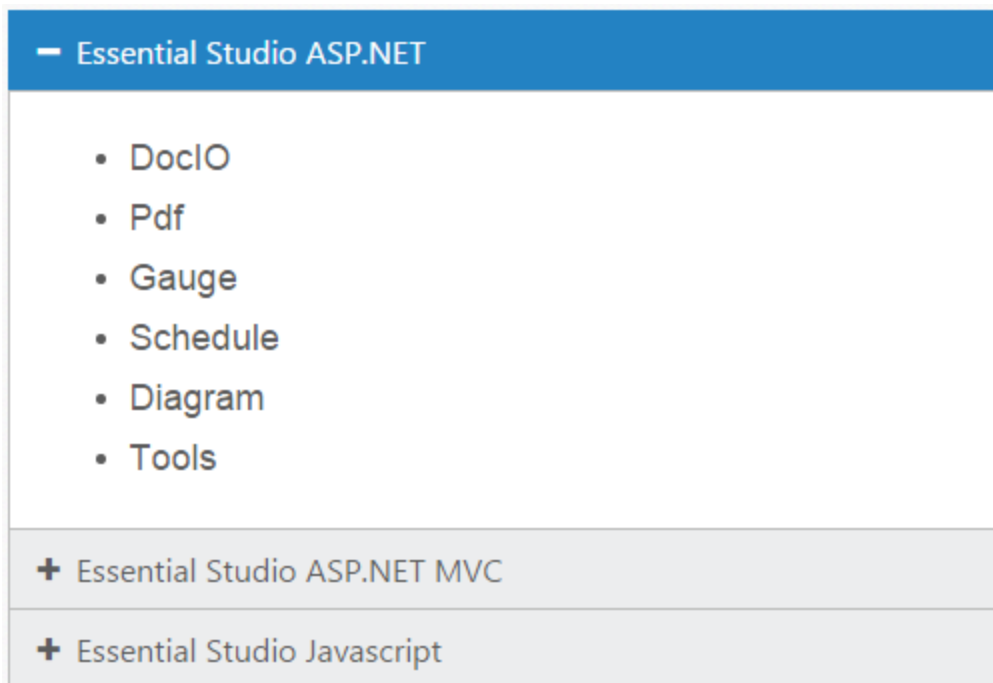
### HTML

```

<div id="accordion-default"></div>
<script src="app/accordion/default.js"></script>

```

You can execute the above code example to display the Accordion control with simple control list.



You can customize the Accordion control using various properties. The Accordion control properties and its default values are described in the following section.

### Configure Multiple Open

You can have multiple **Accordion** tabs opened to view all products at a time. To achieve this set the **enableMultipleOpen** property of the **Accordion** control to true.

**Note:** enableMultipleOpen property is false by default.

You can also open all the panels during initialization using the **selectedItems** property of the **Accordion** control. The following code sample illustrates the opening of multiple tabs by passing the tab index values of tab.

### HTML

```
<div id="accordion-multipleopen"></div>
<script src="app/accordion/multipleopen.js"></script>
```

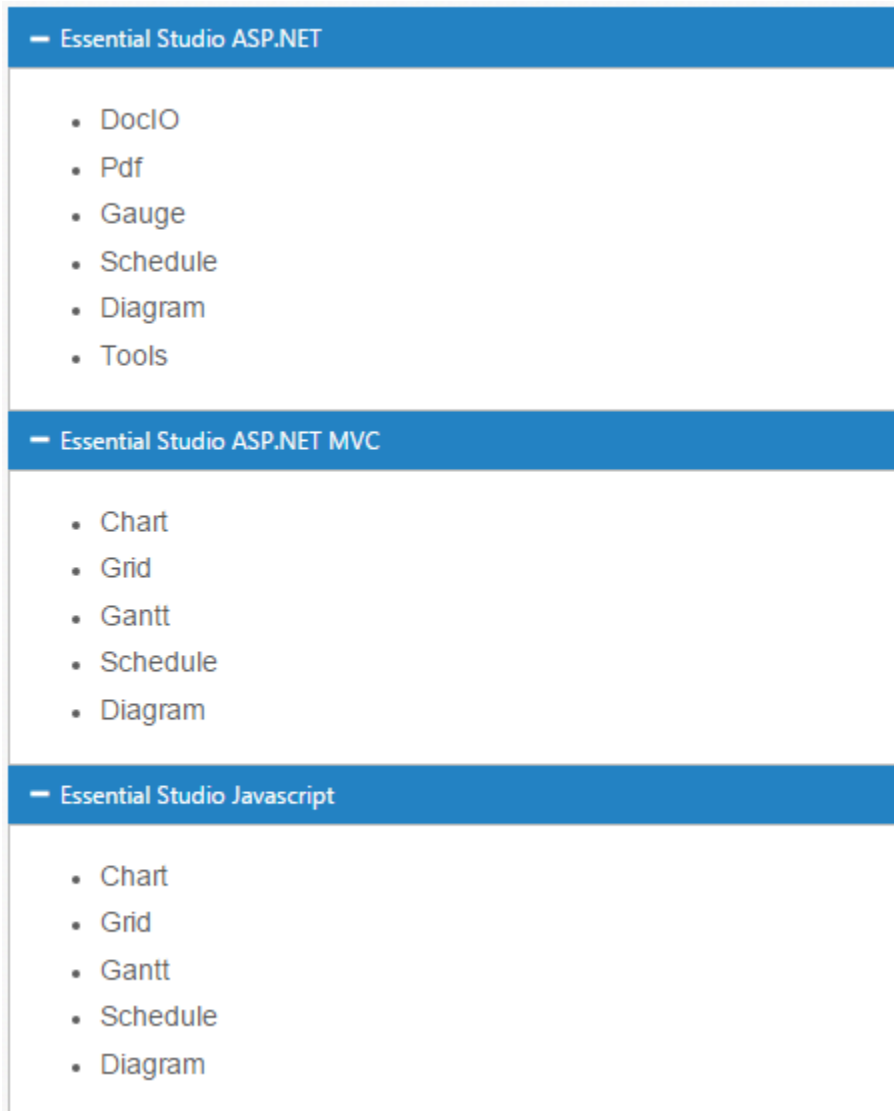
### JAVASCRIPT

```
"use strict";
ReactDOM.render(
  <EJ.Accordion enableMultipleOpen={true}>
    <h3>
      <a href="#">Essential Studio ASP.NET</a>
    </h3>
    <div>
      <ul>
        <li>
          <h4>DocIO</h4>
        </li>
        <li>
          <h4>Pdf </h4>
        </li>
        <li>
          <h4>Gauge </h4>
        </li>
        <li>
          <h4>Schedule </h4>
        </li>
        <li>
          <h4>Diagram </h4>
        </li>
        <li>
          <h4>Tools </h4>
        </li>
      </ul>
    </div>
    <h3>
      <a href="#">Essential Studio ASP.NET MVC</a>
    </h3>
    <div>
      <ul>
        <li>
          <h4>Chart </h4>
        </li>
        <li>
          <h4>Grid </h4>
        </li>
        <li>
          <h4>Gantt </h4>
        </li>
        <li>

```

```
<h4>Schedule </h4>
</li>
<li>
<h4>Diagram </h4>
</li>
</ul>
</div>
<h3>
<a href="#">Essential Studio Javascript</a>
</h3>
<div>
<ul>
<li>
<h4>Chart </h4>
</li>
<li>
<h4>Grid </h4>
</li>
<li>
<h4>Gantt </h4>
</li>
<li>
<h4>Schedule </h4>
</li>
<li>
<h4>Diagram </h4>
</li>
</ul>
</div>
</EJ.Accordion>,
document.getElementById('accordion-default')
);
```

**Accordion** control with **enableMultipleOpen** property is illustrated in the following screen shot.



#### Setting rounded corner

**Accordion** control, by default, is rendered in a regular rectangle. You can modify the regular rectangles with rounded corners by setting the **showRoundedCorner** property to **True**.

**Note:** showRoundedCorner property is False by default.

#### JAVASCRIPT

```
"use strict";
ReactDOM.render(
  <EJ.Accordion enableMultipleOpen={true} showRoundedCorner={true}>
    <h3>
      <a href="#">Essential Studio ASP.NET</a>
    </h3>
    <div>
      <ul>
        <li>
          <h4>DocIO</h4>
        </li>
      </ul>
    </div>
  </EJ.Accordion>
);
```

```

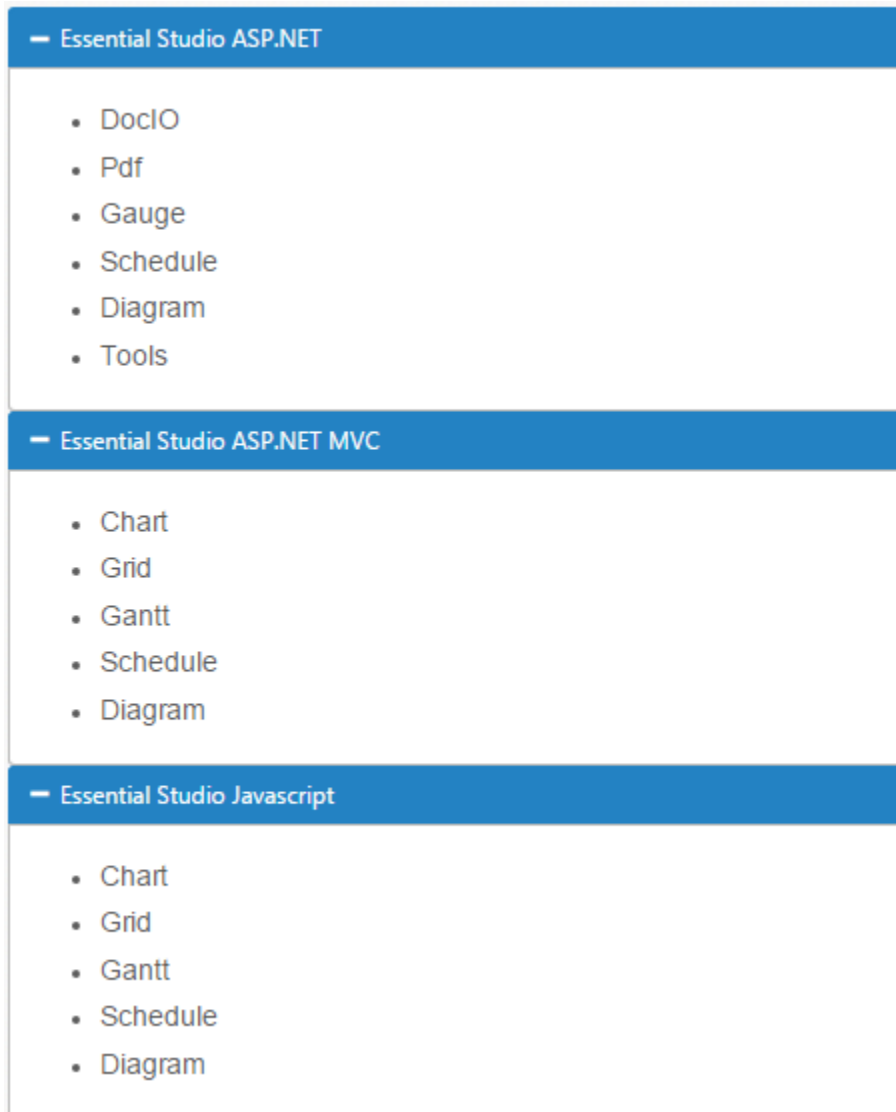
<li>
<h4>Pdf </h4>
</li>
<li>
<h4>Gauge </h4>
</li>
<li>
<h4>Schedule </h4>
</li>
<li>
<h4>Diagram </h4>
</li>
<li>
<h4>Tools </h4>
</li>
</ul>
</div>
<h3>
<a href="#">Essential Studio ASP.NET MVC</a>
</h3>
<div>
<ul>
<li>
<h4>Chart </h4>
</li>
<li>
<h4>Grid </h4>
</li>
<li>
<h4>Gantt </h4>
</li>
<li>
<h4>Schedule </h4>
</li>
<li>
<h4>Diagram </h4>
</li>
</ul>
</div>
<h3>
<a href="#">Essential Studio Javascript</a>
</h3>
<div>
<ul>
<li>
<h4>Chart </h4>
</li>
<li>
<h4>Grid </h4>
</li>
<li>
<h4>Gantt </h4>
</li>
<li>
<h4>Schedule </h4>
</li>
<li>

```



```
<h4>Diagram </h4>
</li>
</ul>
</div>
</EJ.Accordion>,
document.getElementById('accordion-default')
);
```

The following screenshot illustrates the **Accordion** control with rounded corners.



### Customize Icon

You can customize the **Header** icon using **customIcon** property. This property has two features such as **header** and **selectedHeader**. By default, the classes of **header** and **selectedHeader** are **e-collapse** and **e-expand** respectively.

You can change the + and - symbols in the **Accordion** header, that are the default icons with Up or Down arrow icons.

Up or Down arrow icons are available in **e-arrowheadup** and **e-arrowheaddown** classes respectively in the ej.widgets.core.min.css stylesheets from the sample.

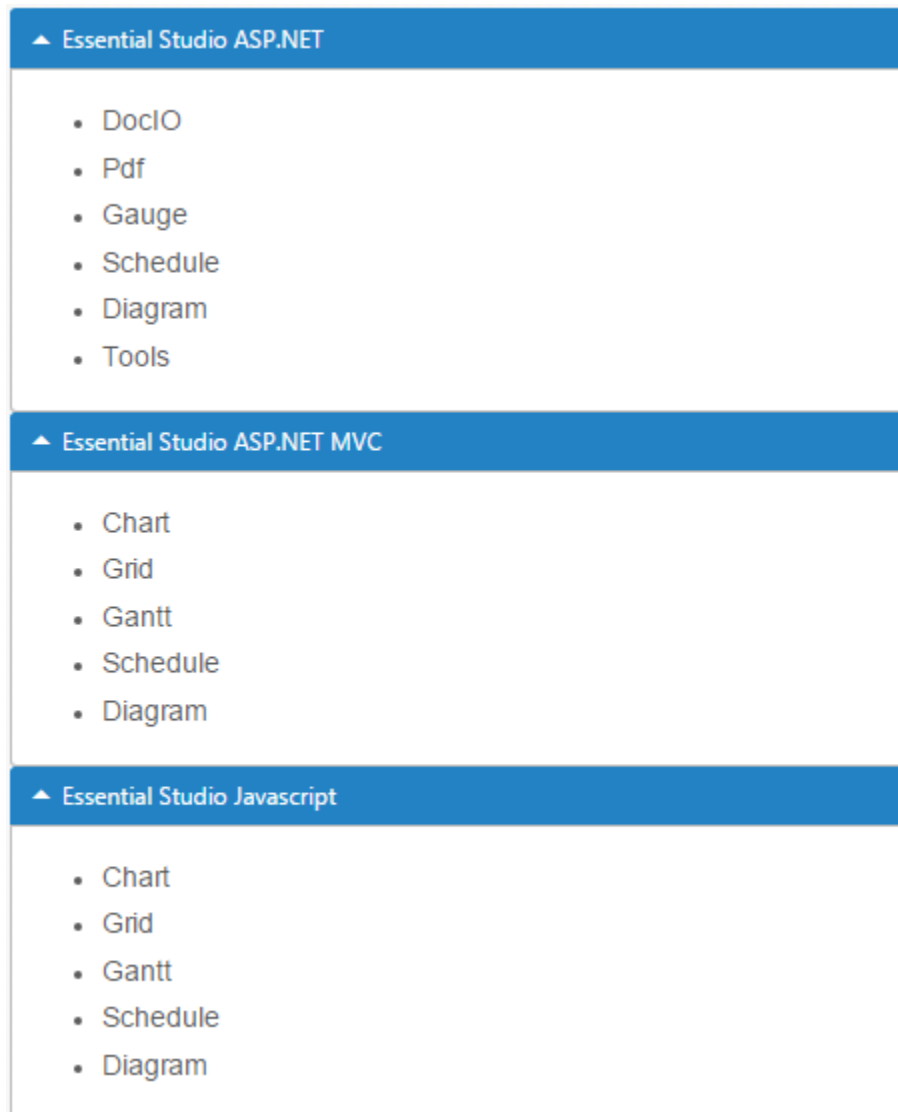
You can set the Up or Down arrow icon to **Accordion** header, by adding **e-arrowheadup** and **e-arrowheaddown** class to **selectedHeader** and **header** properties respectively.

### HTML

```
"use strict";
ReactDOM.render(
  <EJ.Accordion enableMultipleOpen={true} showRoundedCorner={true} customIcon-
  header="header-close" customIcon-selectedHeader="header-expand">
    <h3>
      <a href="#">Essential Studio ASP.NET</a>
    </h3>
    <div>
      <ul>
        <li>
          <h4>DocIO</h4>
        </li>
        <li>
          <h4>Pdf </h4>
        </li>
        <li>
          <h4>Gauge </h4>
        </li>
        <li>
          <h4>Schedule </h4>
        </li>
        <li>
          <h4>Diagram </h4>
        </li>
        <li>
          <h4>Tools </h4>
        </li>
      </ul>
    </div>
    <h3>
      <a href="#">Essential Studio ASP.NET MVC</a>
    </h3>
    <div>
      <ul>
        <li>
          <h4>Chart </h4>
        </li>
        <li>
          <h4>Grid </h4>
        </li>
        <li>
          <h4>Gantt </h4>
        </li>
        <li>
          <h4>Schedule </h4>
        </li>
        <li>
          <h4>Diagram </h4>
        </li>
      </ul>
    </div>
  </EJ.Accordion>
);
```

```
</ul>
</div>
<h3>
<a href="#">Essential Studio Javascript</a>
</h3>
<div>
<ul>
<li>
<h4>Chart </h4>
</li>
<li>
<h4>Grid </h4>
</li>
<li>
<h4>Gantt </h4>
</li>
<li>
<h4>Schedule </h4>
</li>
<li>
<h4>Diagram </h4>
</li>
</ul>
</div>
</EJ.Accordion>,
document.getElementById('accordion-default')
);
```

The following screenshot illustrates the customization of **selectedHeader** and **header** of the **Accordion** control.



## AutoComplete

### Overview

The **React** AutoComplete component is a textbox control that provides a list of suggestions based on your query. When you enter a text into the text box, the control performs a search operation and provides a list of results. There are several filter types available to perform the search.

### Getting Started

Using the following steps, you can create a React AutoComplete component. The basic rendering of React AutoComplete is achieved with default functionality.

#### Create an AutoComplete

You can create a React application and add necessary scripts and styles with the help of the given [React Getting Started Documentation](#).

Create a JSX file for rendering AutoComplete component using `<EJ.Autocomplete>` syntax. Add required properties to it in `<EJ.Autocomplete>` tag element

JS

```
"use strict";
ReactDOM.render(
  <EJ.AutoComplete id="default">
</EJ.AutoComplete>,
  document.getElementById('autocomplete-default')
);
```

Define an HTML element for adding AutoComplete in the application and refer the JSX file.

HTML

```
<div id="autocomplete-default"></div>
<script type="text/babel" src="autocomplete.jsx"></script>
```

This will render an AutoComplete with no suggestion on executing.



## Data Binding

The data for AutoComplete suggestion list which can be populated using the `dataSource` property.

JS

```
"use strict";
var autocomplete_data = [
  { text: "Algeria", sprite: "flag-dz" }, { text: "Argentina", sprite: "flag-ar" },
  { text: "Armenia", sprite: "flag-am" }, { text: "Brazil", sprite: "flag-br" },
  { text: "Bangladesh", sprite: "flag-bd" }, { text: "Canada", sprite: "flag-ca" },
  { text: "Cuba", sprite: "flag-cu" }, { text: "China", sprite: "flag-cn" },
  { text: "Denmark", sprite: "flag-dk" }, { text: "Estonia", sprite: "flag-ee" },
  { text: "Egypt", sprite: "flag-eg" }, { text: "France", sprite: "flag-fr" },
  { text: "Finland", sprite: "flag-fi" }, { text: "Greenland", sprite: "flag-gl" },
  { text: "India", sprite: "flag-in" }, { text: "Indonesia", sprite: "flag-id" },
  { text: "Malaysia", sprite: "flag-my" }, { text: "Mexico", sprite: "flag-mx" },
  { text: "New Zealand", sprite: "flag-nz" }, { text: "Netherlands", sprite: "flag-nl" },
  { text: "Norway", sprite: "flag-no" }, { text: "Portugal", sprite: "flag-pt" },
  { text: "Poland", sprite: "flag-pl" }, { text: "Qatar", sprite: "flag-qa" },
  { text: "Romania", sprite: "flag-ro" }, { text: "Spain", sprite: "flag-es" },
  { text: "Singapore", sprite: "flag-sg" }, { text: "Saudi Arabia", sprite: "flag-sa" },
];
```

```

{ text: "Thailand", sprite: "flag-th" }, { text: "Turkey", sprite: "flag-tr"
},
{ text: "Ukraine", sprite: "flag-ua" }, { text: "United States", sprite:
"flag-us" },
{ text: "Uruguay", sprite: "flag-uy" }, { text: "Viet Nam", sprite: "flag-
vn" },
{ text: "Yemen", sprite: "flag-ye" }
];
var fields={"text":"text","key":"sprite"}
ReactDOM.render(
<EJ.AutoComplete id="default" dataSource = {autocomplete_data}
fields={fields} watermarkText="Select a country">
</EJ.AutoComplete>,
document.getElementById('autocomplete-default')
);

```



### Enable Popup Button

You can enable the popup button of AutoComplete by using `showPopupButton` property which helps you to show all the available suggestions on clicking it.

### JS

```

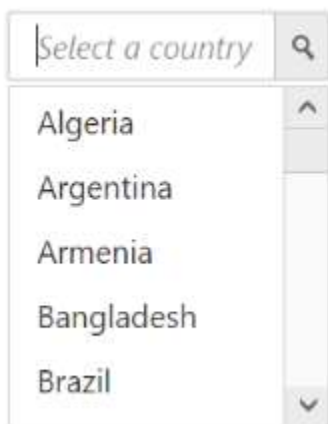
"use strict";
var autocomplete_data = [
{ text: "Algeria", sprite: "flag-dz" }, { text: "Argentina", sprite: "flag-
ar" },
{ text: "Armenia", sprite: "flag-am" }, { text: "Brazil", sprite: "flag-br"
},
{ text: "Bangladesh", sprite: "flag-bd" }, { text: "Canada", sprite: "flag-
ca" },
{ text: "Cuba", sprite: "flag-cu" }, { text: "China", sprite: "flag-cn" },
{ text: "Denmark", sprite: "flag-dk" }, { text: "Estonia", sprite: "flag-ee"
},
{ text: "Egypt", sprite: "flag-eg" }, { text: "France", sprite: "flag-fr" },
{ text: "Finland", sprite: "flag-fi" }, { text: "Greenland", sprite: "flag-
gl" },
{ text: "India", sprite: "flag-in" }, { text: "Indonesia", sprite: "flag-id"
},
{ text: "Malaysia", sprite: "flag-my" }, { text: "Mexico", sprite: "flag-mx"
},
{ text: "New Zealand", sprite: "flag-nz" }, { text: "Netherlands", sprite:
"flag-nl" },
{ text: "Norway", sprite: "flag-no" }, { text: "Portugal", sprite: "flag-pt"
},
{ text: "Poland", sprite: "flag-pl" }, { text: "Qatar", sprite: "flag-qa" },

```

```

{ text: "Romania", sprite: "flag-ro" }, { text: "Spain", sprite: "flag-es"
},
{ text: "Singapore", sprite: "flag-sg" }, { text: "Saudi Arabia", sprite:
"flag-sa" },
{ text: "Thailand", sprite: "flag-th" }, { text: "Turkey", sprite: "flag-tr"
},
{ text: "Ukraine", sprite: "flag-ua" }, { text: "United States", sprite:
"flag-us" },
{ text: "Uruguay", sprite: "flag-uy" }, { text: "Viet Nam", sprite: "flag-
vn" },
{ text: "Yemen", sprite: "flag-ye" }
];
var fields={"text":"text","key":"sprite"}
ReactDOM.render(
<EJ.Autocomplete id="default" dataSource = {autocomplete_data}
showPopupButton={true} fields={fields} watermarkText="Select a country">
</EJ.Autocomplete>,
document.getElementById('autocomplete-default')
);

```



**Note:** You can find the Rotator properties from the [API reference](#) document

## Barcode

### Overview

The Syncfusion Essential JS Barcode widget enables rendering of one dimension and two dimension barcodes in web page. Barcode provides you a simple and inexpensive method of encoding text information that can be easily read by electronic readers.

### Key Features

- Supports 10 one-dimensional barcodes including Code 39 and Code 32 barcodes.
- Supports 2 two-dimensional barcodes such as QR and Data Matrix barcodes.

This section helps to understand the getting started of the ReactJS Barcode with the step-by-step instructions.

### Create an Barcode

You can create a React application and add necessary scripts and styles with the help of the given [ReactJs Getting Started](#) Documentation.

Define an HTML element for adding Barcode in the application and refer the JSX file created.

#### HTML

```
<div id="barcode-qrbarcode" align="center"></div>
<script src="app/barcode/qrbarcode.js"></script>
```

Create a JSX file for rendering Barcode component using <EJ.Barcode> syntax.

#### JS

```
"use strict";
ReactDOM.render(
  <div className="control">
    <EJ.Barcode id="default" text="http://www.syncfusion.com"
      symbologyType="qrbarcode" xDimension="5" >
    </EJ.Barcode>
  </div>,
  document.getElementById('barcode-default')
);
```

Run the above code to render the following output:



### Supported Barcode types

The following table contains the supported types and associated valid characters.

Symbol	Enum Value	Supported characters	Length
<a href="#">QR Code</a>	QRBarcode	[0-9]; [A-Z (upper-case only)]; [space \$ % * + - . / , :]; [Shift JIS characters]	variable
<a href="#">DataMatrix</a>	DataMatrix	All ASCII characters	variable
<a href="#">Code 39</a>	Code39	[0-9]; [A-Z]; [- . \$ / + % SPACE]	variable
<a href="#">Code 39 Extended</a>	Code39Extended	All ASCII characters	variable
<a href="#">Code 11</a>	Code11	[0-9]; [-]	variable
<a href="#">Codabar</a>	Codabar	[0-9]; [- \$ : / . +]	variable
Code 32	Code32	[0-9]	8



<a href="#">Code 93</a>	Code93	[0-9]; [A-Z]; [- . \$ / + % SPACE]	variable
<a href="#">Code 93 Extended</a>	Code93Extended	All 128 ASCII characters	variable
<a href="#">Code 128A</a>	Code128A	[0-9]; [A-Z]; [NUL (0x00) SOH (0x01) STX (0x02) ETX (0x03) EOT(0x04) ENQ (0x05) ACK (0x06) BEL (0x07) BS (0x08) HT (0x09) LF (0x0A) VT(0x0B) FF (0x0C) CR (0x0D) SO (0x0E) SI (0x0F) DLE (0x10) DC1 (0x11) DC2(0x12) DC3 (0x13) DC4 (0x14) NAK (0x15) SYN (0x16) ETB (0x17) CAN(0x18) EM (0x19) SUB (0x1A) ESC (0x1B) FS (0x1C) GS (0x1D) RS (0x1E) US(0x1F) SPACE (0x20)]; [" ! # \$ % & ' ( ) * + , - . / ; < = > ? @ [ / ] ^ _ ]	variable
<a href="#">Code 128B</a>	Code128B	[0-9]; [A-Z]; [a-z]; [SPACE (0x20) ! " # \$ % & ' ( ) * + , - . / ; < = > ? @ [ / ] ^ _ ` {   } ~ DEL (â€¢) ]	variable
<a href="#">Code 128C</a>	Code128C	ASCII 00-99(encodes each two digit with one code)	variable
<a href="#">UPC Barcode</a>	UPCBarcode	[0-9]	11 or 12

### Custom Design for Barcode control.

```
<ts root="datavisualization" />
```

The Barcode can be easily configured to the DOM element, such as div. you can create a Barcode with a highly customizable look and feel.

#### Syntax

```
ReactDOM.render(EJ.Barcode{})
```

#### Example

#### HTML

```
//Create the Barcode by setting the symbologyType and providing input URL to
the text property. QR Barcode is rendered by default.
"use strict";
ReactDOM.render(
  <div className="control">
    <EJ.Barcode id="default" text="http://www.syncfusion.com"
      symbologyType="qrbarcode" xDimension="5" >
    </EJ.Barcode>
  </div>,
  document.getElementById('barcode- default')
);
```

## Members

*barcodeToTextGapHeight* `number`

Specifies the distance between the Barcode and text below it.

**Note:** This property is applicable only for one dimensional barcode.

Default Value:

- 10 pixels

## Example

### HTML

```
// Add the below code to set barcodeToTextGapHeight during initialization
inside the div container.
"use strict";
ReactDOM.render(
  <div className="control">
    <EJ.Barcode id=" code39" text=" SYNCFUSION " symbologyType=" code39'"
    barcodeToTextGapHeight=50 enabled=true>
  </EJ.Barcode>
</div>,
document.getElementById('barcode- code39')
);
```

*barHeight* `number`

Specifies the height of bars in the Barcode. By modifying the barHeight, the entire Barcode height can be customized. Please refer to [xDimension](#) for two dimensional barcode height customization.

**Note:** This property is applicable only for one dimensional barcode.

Default Value:

- 150 pixels

## Example

### HTML

```
// Add the below code to set barHeight during initialization with inside the
div container.
"use strict";
ReactDOM.render(
  <div className="control">
    <EJ.Barcode id=" code39" text=" SYNCFUSION " barHeight: 50 symbologyType="
    code39">
  </EJ.Barcode>
</div>,
document.getElementById('barcode- code39')
);
```

*darkBarColor`object`*

Specifies the dark bar color of the Barcode. One dimensional barcode contains a series of dark and light bars which are usually colored as black and white respectively.

**Note:** 1. For the Barcode should be properly detected by all scanners, choose the best possible contrast color.

2. This property is applicable only for one dimensional barcode.

Default Value:

- black

## Example

**HTML**

```
// Add the below code to set darkBarColor during initialization with inside
the div container.
"use strict";
ReactDOM.render(
  <div className="control">
    <EJ.Barcode id=" code39" text=" SYNCFUSION " darkBarColor="blue"
    symbologyType=" code39">
  </EJ.Barcode>
</div>,
  document.getElementById('barcode- code39')
);
```

*displayText`boolean`*

Specifies whether the text below the Barcode is visible or hidden.

**Note:** This property is applicable only for one dimensional barcode.

Default Value:

- true

## Example

**HTML**

```
// Add the below code to hide displayText during initialization with inside
the div container.
"use strict";
ReactDOM.render(
  <div className="control">
    <EJ.Barcode id=" code39" text=" SYNCFUSION " displayText=false
    symbologyType=" code39">
  </EJ.Barcode>
</div>,
  document.getElementById('barcode- code39')
);
```

*enabled* `boolean`

Specifies whether the control is enabled.

Default Value:

- true

Example

**HTML**

```
// Add the below code to hide displayText during initialization with inside the div container.
"use strict";
ReactDOM.render(
  <div className="control">
    <EJ.Barcode id=" code39" text=" SYNCFUSION " enabled=false symbologyType=" code39">
    </EJ.Barcode>
  </div>,
  document.getElementById('barcode- code39')
);
```

*encodeStartStopSymbol* `boolean`

Specifies the start and stop encode symbol in the Barcode. In one dimensional barcodes, an additional character is added as start and stop delimiters. These symbols are optional and the unique of the symbol allows the reader to determine the direction of the Barcode being scanned.

**Note:** This property is applicable only for one dimensional barcode.

Default Value:

- true

Example

**HTML**

```
// Add the below code to remove encodeStartStopSymbol during initialization with inside the div container.
"use strict";
ReactDOM.render(
  <div className="control">
    <EJ.Barcode id=" code39" text=" SYNCFUSION " encodeStartStopSymbol=false symbologyType=" code39">
    </EJ.Barcode>
  </div>,
  document.getElementById('barcode- code39')
);
```

*lightBarColor* `object`

Specifies the light bar color of the Barcode. One dimensional barcode contains a series of dark and light bars which are usually colored as black and white respectively.

**Note:** 1. For the Barcode should be properly detected by all scanners, choose the best possible contrast color.

2. This property is applicable only for one dimensional barcode.

Default Value:

- white

Example

#### HTML

```
// Add the below code to set lightBarColor during initialization with inside
the div container.
"use strict";
ReactDOM.render(
  <div className="control">
    <EJ.Barcode id=" code39" text=" SYNCFUSION " lightBarColor="blue"
    symbologyType=" code39">
  </EJ.Barcode>
</div>,
  document.getElementById('barcode- code39')
);
```

*narrowBarWidth`number`*

Specifies the width of the narrow bars in the Barcode. The dark bars in the one dimensional barcode contains random narrow and wide bars based on the provided input which can be specified during initialization.

**Note:** This property is applicable only for one dimensional barcode.

Default Value:

- 1 pixel

Example

#### HTML

```
// Add the below code to set narrowBarWidth during initialization with
inside the div container.
"use strict";
ReactDOM.render(
  <div className="control">
    <EJ.Barcode id=" code39" text=" SYNCFUSION " narrowBarWidth= 5
    symbologyType=" code39">
  </EJ.Barcode>
</div>,
  document.getElementById('barcode- code39')
);
```

*quietZone`object`*

Specifies the width of the quiet zone. In barcode, a quiet zone is the blank margin on either side of a barcode which informs the reader where a barcode's symbology starts and stops. The purpose of a quiet zone is to prevent the reader from picking up unrelated information.

*quietZone.all`number`*

Specifies the quiet zone around the Barcode.

Default Value:

- 1 pixel

## Example

**HTML**

```
// Add the below code to set quiet zone during initialization with inside
the div container.
"use strict";
ReactDOM.render(
  <div className="control">
    <EJ.Barcode id=" code39" text=" SYNCFUSION " quietZone= {   all: 10   }
    symbologyType=" code39">
  </EJ.Barcode>
</div>,
document.getElementById('barcode- code39')
);
```

*quietZone.bottom`number`*

Specifies the bottom quiet zone of the Barcode.

Default Value:

- 1 pixel

## Example

**HTML**

```
// Add the below code to set quiet zone during initialization with inside
the div container.
"use strict";
ReactDOM.render(
  <div className="control">
    <EJ.Barcode id=" code39" text=" SYNCFUSION " quietZone= {   bottom: 10   }
    symbologyType=" code39">
  </EJ.Barcode>
</div>,
document.getElementById('barcode- code39')
);
```

*quietZone.left`number`*

Specifies the left quiet zone of the Barcode.

Default Value:

- 1 pixel

Example

#### HTML

```
// Add the below code to set quiet zone during initialization.
container.
"use strict";
ReactDOM.render(
  <div className="control">
    <EJ.Barcode id=" code39" text=" SYNCFUSION " quietZone= {   left: 10   }
    symbologyType=" code39">
  </EJ.Barcode>
</div>,
document.getElementById('barcode- code39')
);
```

*quietZone.right`number`*

Specifies the right quiet zone of the Barcode.

Default Value:

- 1 pixel

Example

#### HTML

```
// Add the below code to set quiet zone during initialization with inside
the div container.
"use strict";
ReactDOM.render(
  <div className="control">
    <EJ.Barcode id=" code39" text=" SYNCFUSION " quietZone= {   right: 10   }
    symbologyType=" code39">
  </EJ.Barcode>
</div>,
document.getElementById('barcode- code39')
);
```

*quietZone.top`number`*

Specifies the top quiet zone of the Barcode.

Default Value:

- 1 pixel

Example

#### HTML

```
// Add the below code to set quiet zone during initialization with inside
the div container.
"use strict";
ReactDOM.render(
  <div className="control">
    <EJ.Barcode id=" code39" text=" SYNCFUSION " quietZone= {   top: 10   }
    symbologyType=" code39">
  </EJ.Barcode>
</div>,
  document.getElementById('barcode- code39')
);
```

*symbologyType`enum`*

<ts name="ej.datavisualization.Barcode.SymbologyType"/>

Specifies the type of the Barcode. See <a href="global.html#SymbologyType">SymbologyType</a>

Name	Description
QRBarcode	Represents the QR code
DataMatrix	Represents the Data Matrix barcode
Code39	Represents the Code 39 barcode
Code39Extended	Represents the Code 39 Extended barcode
Code11	Represents the Code 11 barcode
Codabar	Represents the Codabar barcode
Code32	Represents the Code 32 barcode
Code93	Represents the Code 93 barcode
Code93Extended	Represents the Code 93 Extended barcode
Code128A	Represents the Code 128 A barcode
Code128B	Represents the Code 128 B barcode
Code128C	Represents the Code 128 C barcode
UPCBarcode	Represents the UPC barcode

Default Value:

- symbologyType="qrbarcode"

Example

#### HTML

```
// Add the below code to set the SymbologyType during initialization with
inside the div container.
```



```
"use strict";
ReactDOM.render(
  <div className="control">
    <EJ.Barcode id="default" text="http://www.syncfusion.com"
      symbologyType="qrbarcode" xDimension="5" >
    </EJ.Barcode>
  </div>,
  document.getElementById('barcode- default')
);
```

*text`string`*

Specifies the text to be encoded in the Barcode.

Default Value:

- empty

Example

#### HTML

```
// Add the below code to set the text to be encoded while initialization
with inside the div container.
"use strict";
ReactDOM.render(
  <div className="control">
    <EJ.Barcode id="default" text="http://www.syncfusion.com"
      symbologyType="qrbarcode" xDimension="5" >
    </EJ.Barcode>
  </div>,
  document.getElementById('barcode- default')
);
```

*textColor`object`*

Specifies the color of the text/data at the bottom of the Barcode.

**Note:** This property is applicable only for one dimensional barcode.

Default Value:

- black

Example

#### HTML

```
// Add the below code to set the textColor while initialization with inside
the div container.
"use strict";
ReactDOM.render(
  <div className="control">
    <EJ.Barcode id=" code39" text=" SYNCFUSION " symbologyType="code39"
      textColor="blue" >
    </EJ.Barcode>
  </div>,
```

```
document.getElementById('barcode- code39')
);
```

#### *wideBarWidth `number`*

Specifies the width of the wide bars in the Barcode. One dimensional barcode usually contains random narrow and wide bars based on the provided which can be customized during initialization.

**Note:** This property is applicable only for one dimensional barcode.

Default Value:

- 3 pixels

#### Example

##### HTML

```
// Add the below code to set wideBarWidth during initialization with inside
the div container.
"use strict";
ReactDOM.render(
  <div className="control">
    <EJ.Barcode id=" code39" text=" SYNCFUSION " symbologyType="code39"
    wideBarWidth= 5>
  </EJ.Barcode>
</div>,
document.getElementById('barcode- code39')
);
```

#### *xDimension `number`*

Specifies the width of the narrowest element(bar or space) in a Barcode. The greater the x dimension, the more easily a barcode reader will scan.

**Note:** This property is applicable only for two dimensional barcode.

Default Value:

- 4 pixels

#### Example

##### HTML

```
// Add the below code to set xDimension during initialization with inside
the div container.
"use strict";
ReactDOM.render(
  <div className="control">
    <EJ.Barcode id="default" text="http://www.syncfusion.com"
    symbologyType="qrbarcode" xDimension="5" >
  </EJ.Barcode>
</div>,
document.getElementById('barcode- default')
);
```

## Events

*load*

Fires after Barcode control is loaded.

Name	Type	Description		
{% highlight html %}argument{% endhighlight %}	Object	Event parameters from barcode		
		Name	Type	Description
		{% highlight html %}cancel{% endhighlight %}	boolean	if the event should be canceled; otherwise, false.
		{% highlight html %}model{% endhighlight %}	object	returns the barcode model
		{% highlight html %}type{% endhighlight %}	string	returns the name of the event
		{% highlight html %}status{% endhighlight %}	boolean	return the barcode state

## Example

**HTML**

```
"use strict";
function load(args) {
  debugger;
}
ReactDOM.render(
  <div className="control">
    <EJ.Barcode id="code39" text="SYNCFUSION" symbologyType="code39"
    load="load">
  </EJ.Barcode>
  </div>,
  document.getElementById('barcode-code39')
);
```

## BulletGraph

### Getting Started

This section explains briefly about how to create a BulletGraph control in your application with **ReactJS**

#### Adding Script Reference

Create an **HTML** page and add the scripts references in the order mentioned in the following code example.

- [jQuery](#) 1.10.2 and later versions
- [jsRender](#) - to render the templates

The required ReactJS script dependencies as follows. And you can also refer [React](#) to know more about ReactJS.

- react.min.js - <http://cdn.syncfusion.com/js/assets/external/react.min.js>
- react-dom.min.js - <http://cdn.syncfusion.com/js/assets/external/react-dom.min.js>
- ej.web.react.min.js - <http://cdn.syncfusion.com/{{ site.releaseversion }}/js/common/ej.web.react.min.js>

To get started, you can use the `ej.web.all.min.js` file that encapsulates all the `ej` controls and frameworks in one single file.

#### HTML

```
<!DOCTYPE html>
<html>
<head>
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<meta name="description" content="Essential Studio for React JS">
<meta name="author" content="Syncfusion">
<title>Getting Started for Ribbon React JS</title>
<!-- Essential Studio for JavaScript theme reference -->
<link href="http://cdn.syncfusion.com/{{ site.releaseversion }}/js/web/flat-
azure/ej.web.all.min.css" rel="stylesheet" />
<!-- Essential Studio for JavaScript script references -->
<script src="http://cdn.syncfusion.com/js/assets/external/jquery-
3.0.0.min.js"></script>
<script
src="http://cdn.syncfusion.com/js/assets/external/jsrender.min.js"></script>
<script
src="http://cdn.syncfusion.com/js/assets/external/react.min.js"></script>
<script src="http://cdn.syncfusion.com/js/assets/external/react-
dom.min.js"></script>
<script src="http://cdn.syncfusion.com/{{ site.releaseversion
}}/js/web/ej.web.all.min.js"></script>
<script src="http://cdn.syncfusion.com/{{ site.releaseversion
}}/js/common/ej.web.react.min.js"></script>
<!-- Add your custom scripts here -->
</head>
<body>
</body>
</html>
```

**Note:** 1. In production, we highly recommend you to use our [custom script generator](#) to create custom script file with required controls and its dependencies only. Also to reduce the file size further please use [GZip compression](#) in your server.

2. For themes, you can use the `ej.web.all.min.css` CDN link from the code snippet given. To add the themes in your application, please refer to [this link](#).

### Control Initialization

Control can be initialized in two ways.

- Using jsx Template
- Without using jsx Template

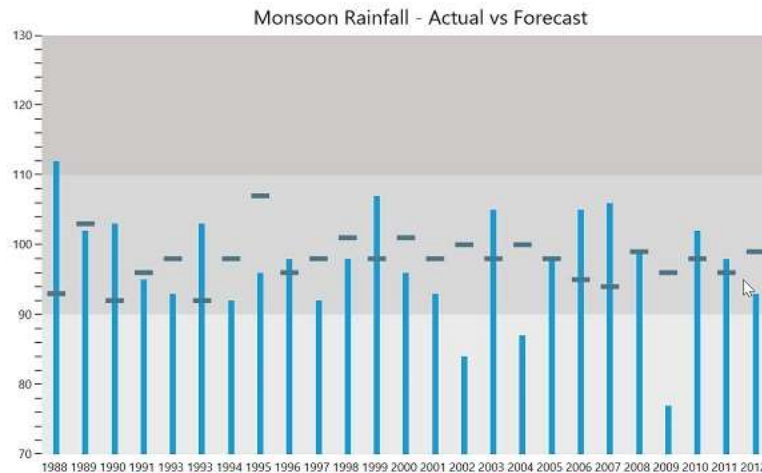
### Using jsx Template

By using the jsx template, we can create the HTML file and jsx file. The `.jsx` file can be convert to `.js` file and it can be referred in HTML page.

### Creating the first Bullet Graph

This section encompasses the details on how to configure the BulletGraph control in your application. It also allows you to learn how to pass the required data to it and customize its various options according to your requirements.

In the following screenshot, a BulletGraph is used to compare the actual monsoon rainfall received in a state versus its forecasted values for the years ranging from 1988 to 2013.



1. Create a

tag.

### HTML

```
<!DOCTYPE html>
<html>
<body>
<div id="bulletGraph-default" style="height:99%; "></div>
<script src="app/bulletgraph/default.js"></script>
</body>
```

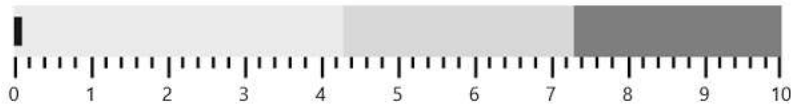
```
</html>
```

2. Initialize the BulletGraph by using the `EJ.BulletGraph` tag.

### JAVASCRIPT

```
"use strict";
ReactDOM.render(
  <div className="default">
    <EJ.BulletGraph id="bulletGraph1"></EJ.BulletGraph>,
  </div>,
  document.getElementById('bulletGraph-default')
);
```

Execute the above code to display the BulletGraph. To customize the measure bars in the BulletGraph, you can pass the data either locally or remotely.



### Provide Required Data

You can customize the values of feature and comparative measure bars in a BulletGraph, either locally or remotely. The category data is optional, and it is used to display label values parallel to the measure bars.

Assign the data in LocalData variable to the DataSource property of BulletGraph as illustrated in the following code example.

### JAVASCRIPT

```
var localData = [
  {
    value: 90, comparativeMeasureValue: 100,
    category: 2013
  },
  {
    value: 93, comparativeMeasureValue: 99,
    category: 2012
  },
  {
    value: 98, comparativeMeasureValue: 96,
    category: 2011
  },
  {
    value: 102, comparativeMeasureValue: 98,
    category: 2010
  },
  {
    value: 77, comparativeMeasureValue: 96,
    category: 2009
  },
  {
    value: 99, comparativeMeasureValue: 99,
    category: 2008
  }
];
```

```
},
{
  value: 106, comparativeMeasureValue: 94,
  category: 2007
},
{
  value: 105, comparativeMeasureValue: 95,
  category: 2006
},
{
  value: 98, comparativeMeasureValue: 98,
  category: 2005
},
{
  value: 87, comparativeMeasureValue: 100,
  category: 2004
},
{
  value: 105, comparativeMeasureValue: 98,
  category: 2003
},
{
  value: 84, comparativeMeasureValue: 101,
  category: 2002
},
{
  value: 93, comparativeMeasureValue: 98,
  category: 2001
},
{
  value: 90, comparativeMeasureValue: 96,
  category: 2000
},
{
  value: 95, comparativeMeasureValue: 107,
  category: 1999
},
{
  value: 104, comparativeMeasureValue: 98,
  category: 1998
},
{
  value: 102, comparativeMeasureValue: 92,
  category: 1997
},
{
  value: 103, comparativeMeasureValue: 98,
  category: 1996
},
{
  value: 100, comparativeMeasureValue: 96,
  category: 1995
},
{
  value: 110, comparativeMeasureValue: 92,
  category: 1994
},
},
```

```
{
  value: 100, comparativeMeasureValue: 103,
  category: 1993
},
{
  value: 94, comparativeMeasureValue: 93,
  category: 1992
},
{
  value: 91, comparativeMeasureValue: 95,
  category: 1991
},
{
  value: 107, comparativeMeasureValue: 103,
  category: 1990
},
{
  value: 101, comparativeMeasureValue: 102,
  category: 1989
},
{
  value: 119, comparativeMeasureValue: 112,
  category: 1988
}
];
```

Once the DataSource property is assigned with the required values, you can bind the variable names used in the JSON data to the corresponding fields of the BulletGraph as shown in the following code example.

### HTML

```
<script type="text/babel">
var fields= {
dataSource: localData, category: "category",
featureMeasures: "value",
comparativeMeasure: "comparativeMeasureValue"
};
<!DOCTYPE html>
<html>
<body>
<script type="text/babel">
ReactDOM.render(
<div className="default">
<EJ.BulletGraph id="bulletGraph1" fields: {fields} </EJ.BulletGraph>,
</div>,
document.getElementById('bulletGraph-default')
);
</script>
</body>
</html>
```

### Set Default and Scale Values

You can plot any number of measure bars within the BulletGraph by increasing the height and width of the control to locate all the measure bars within the graph. Set the QualitativeRangeSize and QuantitativeScaleLength properties according to the following code example.

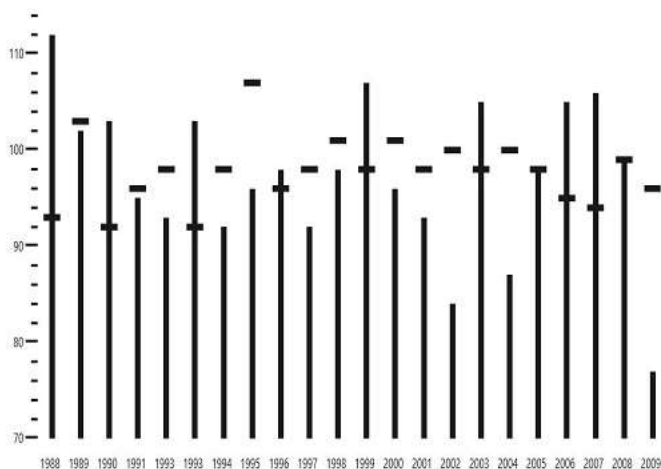


By default, the BulletGraph is rendered in the horizontal orientation with its flow direction set to Forward. In this example, to achieve the desired output, change the horizontal orientation to vertical orientation with the flow direction set to Backward.

Minimum, Maximum and Interval values for the QuantitativeScale of the BulletGraph are set, as illustrated in the following code example. The ticks and labels within the scale are also positioned.

### HTML

```
<script type="text/babel">
var quantitativeScaleSettings = {
  minimum: 70,
  maximum: 130,
  interval: 10,
  tickPosition: ej.datavisualization.BulletGraph.TickPosition.Far,
  labelSettings: {
    position: ej.datavisualization.BulletGraph.LabelPosition.Below, offset: 14,
    size: 10
  },
};
<!DOCTYPE html>
<html>
<body>
<script type="text/babel">
ReactDOM.render(
  <div className="default">
    <EJ.BulletGraph id="bulletGraph1" quantitativeScaleSettings=
    {quantitativeScaleSettings} </EJ.BulletGraph>,
  </div>,
  document.getElementById('bulletGraph-default')
);
</script>
</body>
</html>
```



The above image illustrates the BulletGraph without any ranges displayed in the background.

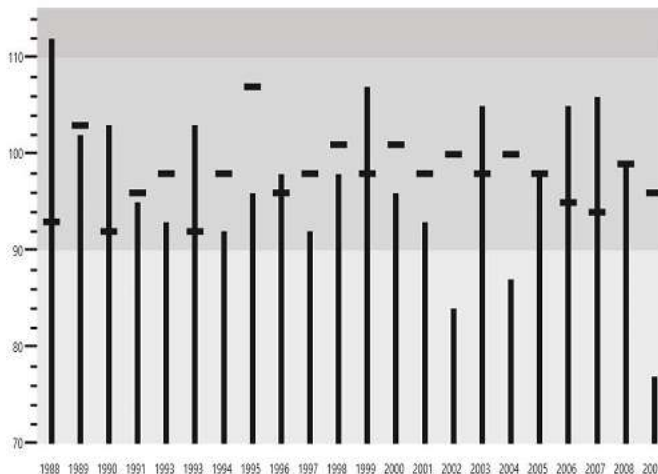
### Add Qualitative Ranges

By default, 3 ranges are displayed in the BulletGraph control during the initial rendering of the control with its default values. To customize it, you can set appropriate values for the RangeEnd and RangeStroke properties. Any number of QualitativeRanges can be added to the control.

### HTML

```
<script type="text/babel">
var qualitativeRanges= [{
rangeEnd: 90
},
{
rangeEnd: 110
},
{
rangeEnd: 130, rangeStroke: "#CDC9C9"
}];
<!DOCTYPE html>
<html>
<body>
<script type="text/babel">
ReactDOM.render(
<div className="default">
<EJ.BulletGraph id="bulletGraph1"    qualitativeRanges={qualitativeRanges}
</EJ.BulletGraph>,
</div>,
document.getElementById('bulletGraph-default')
);
</script>
</body>
</html>
```

After adding QualitativeRanges to the BulletGraph, the control appears as follows.



### Ticks and Measure Bars Customization

You can do the following code changes in the quantitative scale to customize the tick size, the color of the feature bar and the comparative measure symbols.

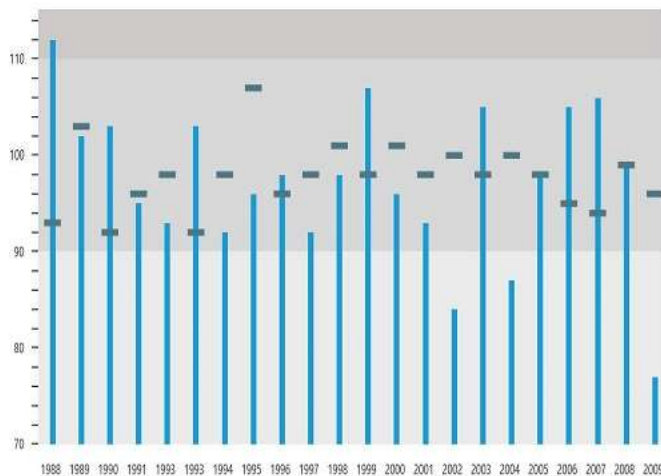
### HTML

```

<script type="text/babel">
var quantitativeScaleSettings = {
  minimum: 70,
  maximum: 130,
  interval: 10,
  tickPosition: ej.datavisualization.BulletGraph.TickPosition.Far,
  labelSettings: {
    position: ej.datavisualization.BulletGraph.LabelPosition.Below, offset: 14,
    size: 10
  },
  majorTickSettings:{width:1, size:7},
  minorTickSettings:{width:1},
  comparativeMeasureSettings:{stroke:"#507786"},
  featuredMeasureSettings:{stroke: "#169DD8"}
};
<!DOCTYPE html>
<html>
<body>
<script type="text/babel">
ReactDOM.render(
  <div className="default">
    <EJ.BulletGraph id="bulletGraph1" quantitativeScaleSettings=
    {quantitativeScaleSettings} ></EJ.BulletGraph>,
  </div>,
  document.getElementById('bulletGraph-default')
);
</script>
</body>
</html>

```

When you customize the ticks and measure bar, the BulletGraph appears as follows.



#### Add Caption and Subtitle

You can add the following code example to display an appropriate Caption and Subtitle to the BulletGraph.

#### HTML

```

<script type="text/babel">

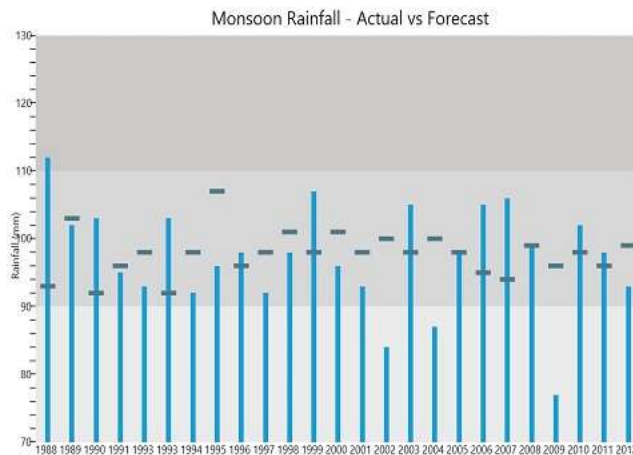
```

```

var captionSettings= {
  textAngle: 0,
  location: { x: 470, y: 270 },
  text: "Monsoon Rainfall - Actual vs Forecast",
  font: { fontFamily: 'Segoe UI', size: '20px', fontWeight:
ej.datavisualization.BulletGraph.FontWeight.Normal, opacity: 1 },
  subTitle: {
    textAngle: -90,
    text: "Rainfall (mm)", location: { x: 180, y: 4 },
    font: { fontFamily: 'Segoe UI', size: '14px',
fontWeight: 'ej.datavisualization.BulletGraph.FontWeight.Normal', opacity:
1}
  }
};
<!DOCTYPE html>
<html>
<body>
<script type="text/babel">
ReactDOM.render(
<div className="default">
<EJ.BulletGraph id="bulletGraph1" captionSettings= {captionSettings}
</EJ.BulletGraph>,
</div>,
document.getElementById('bulletGraph-default')
);
</script>
</body>
</html>

```

The following screenshot displays a BulletGraph with a Caption and Subtitle.



#### Show Tooltip

You can use a Tooltip in your application to display the values of forecasted rainfall, actual rainfall received in millimeter and also the appropriate year. The Tooltip Visible property is set to true to enable the Tooltip option. To set the template Tooltip, you can pass the template id to it as illustrated in the following code example.

#### HTML

```

<script type="text/babel">
var tooltipSettings= {template: "Tooltip", visible: true};
<!DOCTYPE html>
<html>
<body>
<script type="text/babel">
ReactDOM.render(
<div className="default">
<EJ.BulletGraph id="bulletGraph1"    tooltipSettings = {tooltipSettings}
</EJ.BulletGraph>,
</div>,
document.getElementById('bulletGraph-default')
);
</script>
</body>
</html>

```

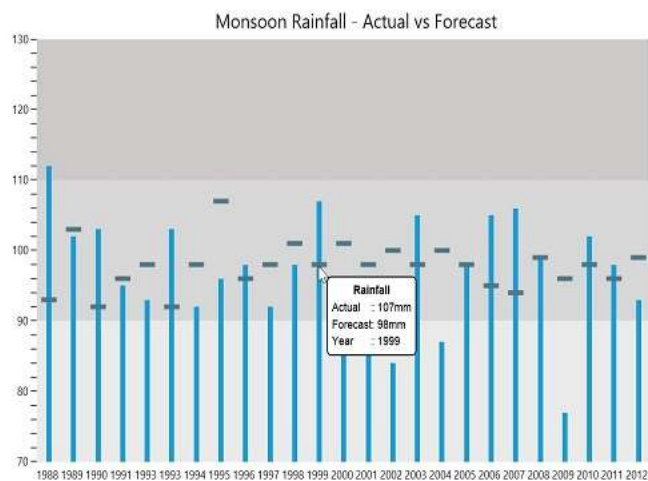
## HTML

```

<div id="Tooltip"style="display:none; width:125px;padding-top: 10px;padding-
bottom:10px">
<div align="center"style="font-weight:bold">
Rainfall </div>
<table>
<tr><td>Actual</td>
<td>{{:currentValue}}mm</td></tr>
<tr><td>Forecast</td>
<td>{{:targetValue}}mm</td></tr>
<tr><td>Year</td>
<td>{{:category}}</td></tr>
</table>
</div>

```

The following screenshot displays a customized BulletGraph.



## Without using jsx Template

The BulletGraph can be created from a HTML **DIV** element with the HTML **id** attribute set to it. Refer to the following code example.

## HTML

```
<div id="bulletGraph-default" style="height:99%;"></div>
```

## JAVASCRIPT

```
var localData = [
{
value: 90, comparativeMeasureValue: 100,
category: 2013
},
{
value: 93, comparativeMeasureValue: 99,
category: 2012
},
{
value: 98, comparativeMeasureValue: 96,
category: 2011
},
{
value: 102, comparativeMeasureValue: 98,
category: 2010
},
{
value: 77, comparativeMeasureValue: 96,
category: 2009
},
{
value: 99, comparativeMeasureValue: 99,
category: 2008
},
{
value: 106, comparativeMeasureValue: 94,
category: 2007
},
{
value: 105, comparativeMeasureValue: 95,
category: 2006
},
{
value: 98, comparativeMeasureValue: 98,
category: 2005
},
{
value: 87, comparativeMeasureValue: 100,
category: 2004
},
{
value: 105, comparativeMeasureValue: 98,
category: 2003
},
{
value: 84, comparativeMeasureValue: 101,
category: 2002
},
{

```

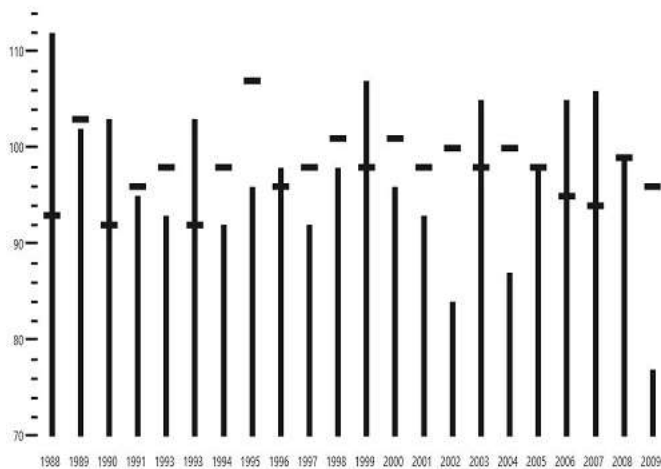
```
value: 93, comparativeMeasureValue: 98,
category: 2001
},
{
value: 90, comparativeMeasureValue: 96,
category: 2000
},
{
value: 95, comparativeMeasureValue: 107,
category: 1999
},
{
value: 104, comparativeMeasureValue: 98,
category: 1998
},
{
value: 102, comparativeMeasureValue: 92,
category: 1997
},
{
value: 103, comparativeMeasureValue: 98,
category: 1996
},
{
value: 100, comparativeMeasureValue: 96,
category: 1995
},
{
value: 110, comparativeMeasureValue: 92,
category: 1994
},
{
value: 100, comparativeMeasureValue: 103,
category: 1993
},
{
value: 94, comparativeMeasureValue: 93,
category: 1992
},
{
value: 91, comparativeMeasureValue: 95,
category: 1991
},
{
value: 107, comparativeMeasureValue: 103,
category: 1990
},
{
value: 101, comparativeMeasureValue: 102,
category: 1989
},
{
value: 119, comparativeMeasureValue: 112,
category: 1988
}];
var quantitativeScaleSettings = {
minimum: 70,
```

```

maximum: 130,
interval: 10,
tickPosition: ej.datavisualization.BulletGraph.TickPosition.Far,
labelSettings: {
  position: ej.datavisualization.BulletGraph.LabelPosition.Below, offset: 14,
  size: 10
},
};
ReactDOM.render(
  React.createElement(EJ.BulletGraph, {id: "bulletCore0",
    height: 400,
    enableAnimation: true,
    qualitativeRangeSize: 320,
    quantitativeScaleLength: 475,
    quantitativeScaleSettings: quantitativeScaleSettings,
    fields: fields,
  }
),
document.getElementById('bulletGraph-default')
);
</script>

```

The above image illustrates the BulletGraph without any ranges displayed in the background.



## Bullet Graph Dimensions

This section explains you on how to change the dimensions of the **Bullet Graph**. You can change various dimensions and properties of **Bullet Graph** like width, height, quantitative scale length, qualitative range size etc. By default, **Bullet Graph** uses 595 pixel width and 90 pixel height. You can customize width and height of a **Bullet Graph** using **width** and **height** properties of **Bullet Graph** respectively.

### Size

#### JAVASCRIPT

```

"use strict";
ReactDOM.render(
  <EJ.BulletGraph id="bulletCore0"
    width={850}
    height={540}
  >

```



```
</EJ.BulletGraph>,
document.getElementById('bulletgraph')
);
```

In the above code example, width is set as 500 pixel and height is set as 100 pixel. The output of the above code example with dimension 500 \* 100 is as follows.



#### Value for performance bar

The feature measure bar value is customized using the **value** property. Default value of this property is 0.

#### JAVASCRIPT

```
"use strict";
ReactDOM.render(
<EJ.BulletGraph id="bulletCore0"
value={5}
>
</EJ.BulletGraph>,
document.getElementById('bulletgraph')
);
```

The following screenshot displays **Bullet Graph** with a performance measure value of 5.



#### Comparative measure value

The **Comparative measure value** is set using **comparativeMeasureValue** property. The default value of this property is 0.

#### JAVASCRIPT

```
"use strict";
ReactDOM.render(
<EJ.BulletGraph id="bulletCore0"
comparativeMeasureValue= {5}
>
</EJ.BulletGraph>,
document.getElementById('bulletgraph')
);
```

The following screenshot displays **Bullet Graph** with comparative measure value of 5.



#### Theme

**Bullet Graph Theme** is customized using **theme** property. Default value is **flatlight**. **Bullet Graph** supports **flatlight** and **flatdark** themes. **Flatdark** theme improves **Bullet Graph** appearance when background of **Bullet Graph** container uses dark color like black.

#### JAVASCRIPT

```
"use strict";
ReactDOM.render(
```

```
<EJ.BulletGraph id="bulletCore0"
  theme = {"flatdark"}
>
</EJ.BulletGraph>,
document.getElementById('bulletgraph')
);
```

The following screenshot displays **Bullet Graph** with **flatdark** theme



### Orientation

Bullet Graph is oriented either horizontally or vertically using orientation property. Default value of this property is horizontal.

### JAVASCRIPT

```
"use strict";
ReactDOM.render(
<EJ.BulletGraph id="bulletCore0"
  orientation={'vertical'}
  width={100}
  height={550}
  flowDirection={'backward'}
>
</EJ.BulletGraph>,
document.getElementById('bulletgraph')
);
```

### Flow direction

The Flow direction of Bullet Graph is customized using flowDirection property. Default value of this property is forward. Setting forward renders Bullet Graph left to right and backward renders from right to left.

### JAVASCRIPT

```
"use strict";
ReactDOM.render(
<EJ.BulletGraph id="bulletCore0"
  comparativeMeasureValue={2}
  flowDirection={'backward'}
>
</EJ.BulletGraph>,
document.getElementById('bulletgraph')
);
```

The following screenshot displays **Bullet Graph** in a backward direction.



### Qualitative range size

Size of the Qualitative range is customized using qualitativeRangeSize property. Default value of this property is 32.

### JAVASCRIPT

```
"use strict";
ReactDOM.render(
  <EJ.BulletGraph id="bulletCore0"
    qualitativeRangeSize={50}
  >
</EJ.BulletGraph>,
  document.getElementById('bulletgraph')
);
```

The following screenshot displays **Bullet Graph** with Qualitative range of size 50



### Quantitative scale length

Length of the **Quantitativescale** is customized using **quantitativeScaleLength** property. Default value of this property is 475.

### JAVASCRIPT

```
"use strict";
ReactDOM.render(
  <EJ.BulletGraph id="bulletCore0"
    quantitativeScaleLength={500}
  >
</EJ.BulletGraph>,
  document.getElementById('bulletgraph')
);
```

The following screenshot displays **Bullet Graph** with Quantitative scale length of 500.



### Bullet Graph Caption

**Bullet Graph** supports title and **subtitle** to convey what is represented in **Bullet Graph**. They are customized using **captionSettings** property.

### Title

**Title** is set to **Bullet Graph** using **text** property in **captionSettings**. Caption settings also include properties like location, font, and textAngle for customizing the caption of **Bullet Graph**.

### JAVASCRIPT

```
"use strict";
var quantitativeScaleSettings= {
  location: { x: 100, y: 200 },
  minimum: 0,
  maximum: 5,
  interval: 0.5,
  featureMeasures: [{ value: 4, comparativeMeasureValue: 3.5, category: "" }]
};
var captionSettings= {
  text: "Revenue YTD",
  textAngle: 0,
  location: {
    x: 10, y: 220
  },
```

```

font: {
  color: 'gray',
  fontFamily: 'Segoe UI',
  fontStyle: 'bold',
  size: '14px',
  fontWeight: 'regular',
  opacity: 1
}
};
ReactDOM.render(
  <EJ.BulletGraph id="bulletCore0"
  height={700}
  width={600}
  quantitativeScaleSettings={quantitativeScaleSettings}
  captionSettings={captionSettings}
  >
  </EJ.BulletGraph>,
  document.getElementById('bulletGraph')
);

```

The following screenshot displays a **Bullet Graph** with customized caption using the above code.

![[/js/BulletGraph/Bullet-Graph-Captionimages/Bullet-Graph-Captionimg1.png]

### Subtitle

**Subtitle** is added to **Bullet Graph** using **text** property of **subtitle** in **captionSettings**. **Subtitle** also provides properties like location, textAngle and font to customize subtitle similar to caption.

### JAVASCRIPT

```

"use strict";
var captionSettings= {
  subTitle: {
    text: "Subtitle",
    location: { x: 20, y: 225 },
    font: {
      color: 'black',
      fontFamily: 'segoe ui',
      fontStyle: 'italic',
      size: '16px',
      fontWeight: 'regular',
      opacity: 1
    }
  },
};
ReactDOM.render(
  <EJ.BulletGraph id="bulletCore0"
  height={700}
  width={600}
  captionSettings={captionSettings}
  >
  </EJ.BulletGraph>,
  document.getElementById('bulletGraph')
);

```

The following screenshot displays **Bullet Graph** with a subtitle



### Indicator

You can add **Indicator** to bullet graph by enabling **visible** and setting **text** properties of **indicator** in **captionSettings**. Indicator is used to represent whether target is achieved or not with text and symbol by comparing current and target values in bullet graph.

Indicator displays a symbol along with text which is different from caption and subtitle. Images like logos can be used in indicator instead of symbols. Indicator has properties such as **symbol**, **text**, **textSpacing**, **textAngle**, **location** and **font**.

### JAVASCRIPT

```
"use strict";
var captionSettings = {
  indicator: {
    visible: true,
    textAngle: 0,
    location: { x: 15, y: 240 },
    text: "+ $0.5 K",
    textSpacing: 5,
    symbol: {
      color: 'green',
      shape: 'triangle',
      imageURL: "Column.png",
      size: {
        width: 10,
        height: 10
      },
      border: {
        width: 1,
        color: 'green'
      }
    },
    font: {
      color: 'green',
      fontFamily: 'Segoe UI',
      fontStyle: 'Normal ',
      size: '12px',
      fontWeight: 'regular',
      opacity: 1
    },
  },
};
ReactDOM.render(
  <ej.bulletgraph id="bulletGraph1" captionsettings=
    {captionSettings}></ej.bulletgraph>,
  document.getElementById('bulletGraph')
);
```

The following screenshot displays a bullet graph with indicator.



### Trim

The title, subtitle and indicator text can be overlapped to the scale group. You can avoid the overlapped text by using the enableTrim property of the captionSettings. The default value of the enableTrim is true.

### JAVASCRIPT

```
"use strict";
var captionSettings = {
  text: 'Bullet Graph Title',
  enableTrim : true,
}
ReactDOM.render(
  <ej.bulletgraph id="bulletGraph1" captionsettings=
  {captionSettings}></ej.bulletgraph>,
  document.getElementById('bulletGraph')
);
```

The following screenshot displays the BulletGraph with Trim.



### Text Placement

All the caption group elements (caption, subtitle, and indicator) in the **Bullet Graph** support text positioning by using the property **textPosition** available in all caption group elements. The properties, **textAlignment** and **textAnchor** are used to customize text placement further.

### Text Position

The property, textPosition, is used to position the text at the top, bottom, left, and right side of the quantitative scale. The default value of this property is float. By default, text can be placed at any desired location by using the location property.

### JAVASCRIPT

```
"use strict";
var quantitativeScaleSettings={ location: { x: 120, y:40 }};
var captionSettings={
  text: 'Bullet Graph Title',
  textPosition: 'top',
  font:{
    size: '20px',
    fontWeight: 'bold',
  }
}
ReactDOM.render(
  <EJ.BulletGraph id="bulletCore0"
  value={8}
  comparativeMeasureValue={5}
  quantitativeScaleSettings={quantitativeScaleSettings}
  height={150}
  width={650}
  captionSettings={captionSettings}
  >
  </EJ.BulletGraph>,
  document.getElementById('bulletGraph')
```

```
);
```

The following screenshot displays the Bullet Graph with the title positioned above.



#### Text Alignment

Alignment of text at different positions with respect to scale can be customized by using the **textAlignment** property. Text can be aligned in the **near**, **center**, and **far** locations of the scale. Text alignment depends upon **textPosition** property and is not applicable when the value of the **textPosition** property is **float**. The default value of the **textAlignment** property is **near**.

#### JAVASCRIPT

```
"use strict";
var quantitativeScaleSettings={ location: { x: 120, y:40 }};
var captionSettings={
  text: 'Revenue',
  textPosition: 'left',
  textAnchor: 'middle',
  font:{
    size: '16px',
    fontWeight: 'bold',
  },
  subTitle: {
    text: '$ in thousands',
    textPosition: 'left',
    textAlignment: 'center',
    font: {
      size: '12px',
      fontWeight: 'bold',
    }
  }
}
ReactDOM.render(
  <EJ.BulletGraph id="bulletCore0"
  value={8}
  comparativeMeasureValue={5}
  quantitativeScaleSettings={quantitativeScaleSettings}
  height={150}
  width={650}
  captionSettings={captionSettings}
  >
  </EJ.BulletGraph>,
  document.getElementById('bulletGraph')
);
```

The following screenshot displays the Bullet Graph with the title and subtitle at different alignments.



#### Text Anchor

Text elements aligned at the same position are anchored by using the **textAnchor** property. These can be anchored at the start, middle, and end. The default value of this property is start and applicable only when two or more text elements are aligned at the same position.

**JAVASCRIPT**

```

"use strict";
var quantitativeScaleSettings={ location: { x: 120, y:40 }};
var captionSettings={
  text: 'Revenue',
  textPosition: 'left',
  textAnchor: 'middle',
  font:{
    size: '16px',
    fontWeight: 'bold',
  },
  subTitle: {
    text: '$ in thousands',
    textPosition: 'left',
    textAlignment: 'center',
    font: {
      size: '12px',
      fontWeight: 'bold',
    }
  }
}
ReactDOM.render(
  <EJ.BulletGraph id="bulletCore0"
    value={8}
    comparativeMeasureValue={5}
    quantitativeScaleSettings={quantitativeScaleSettings}
    height={150}
    width={650}
    captionSettings={captionSettings}
  >
  </EJ.BulletGraph>,
  document.getElementById('bulletGraph')
);

```



*Padding*

The space required between text and quantitative scale is customized by using the padding property. The default value of this property is 5 and not applicable when the value of the textPosition property is float.

**JAVASCRIPT**

```

"use strict";
var quantitativeScaleSettings={ location: { x: 120, y:40 }};
var captionSettings={
  text: 'Profit in %',
  textPosition: 'left',
  textAlignment: 'center',
  padding: 10,
  font:{
    size: '16px',
    fontWeight: 'bold',
  }
}

```



```
ReactDOM.render(
  <EJ.BulletGraph id="bulletCore0"
    value={8}
    comparativeMeasureValue={5}
    quantitativeScaleSettings={quantitativeScaleSettings}
    height={150}
    width={650}
    captionSettings={captionSettings}
  >
</EJ.BulletGraph>,
  document.getElementById('bulletGraph')
);
```

![[/js/BulletGraph/Bullet-Graph-Captionimages/Bullet-Graph-Captionimg8.png]

### Localization

Bullet graph supports localization for its axis labels and tooltip. To render the bullet graph with specific culture you have to refer the corresponding globalize culture script and need to specify the culture name in locale property of bullet graph.

Enable Group Separator is used to Convert the date object to string while using the locale settings, you can set enableGroupSeparator property as true.

### JAVASCRIPT

```
"use strict";
ReactDOM.render(
  <EJ.BulletGraph id="bulletCore0"
    locale = "en-fr"
    enableGroupSeparator = {true}
  >
</EJ.BulletGraph>,
  document.getElementById('bulletGraph')
);
```

### Data Binding

**Bullet Graph** supports binding JSON data from a remote server or data created in client-side. You can use the **fields** property to customize the data bound with **Bullet Graph**.

#### Local Data

Data available in client-side (local data) can be bound with **Bullet Graph** using **fields** property. This property provides option to specify data source, fields representing progress measure bar value, comparative measure value and category value.

### JAVASCRIPT

```
"use strict";
var localData = [
  {
    value: 9.5, comparativeMeasureValue: 7.5,
    category: 2001
  },
  {
    value: 9.5, comparativeMeasureValue: 5,
    category: 2002
  }
];
```

```

    }];
    var quantitativeScaleSettings= { location: { x:50, y:20 } };
    var fields= {
    dataSource: localData, category: "category",
    featureMeasures: "value",
    comparativeMeasure: "comparativeMeasureValue"
    };
    ReactDOM.render(
    <EJ.BulletGraph id="bulletCore0"
    qualitativeRangeSize={60}
    quantitativeScaleSettings={quantitativeScaleSettings}
    >
    </EJ.BulletGraph>,
    document.getElementById('bulletgraph')
    );

```

The following screenshot displays **Bullet Graph** with local data generated using JavaScript

![[/js/BulletGraph/Data-Bindingimages/Data-Bindingimg1.png]

### Remote Data

**Bullet Graph** provides option to bind data from a remote server using **ejDataManager** as data source in **fields** property. A query object should also be passed to **query** property when using data manager as data source.

### JAVASCRIPT

```

"use strict";
var dataManger = new ej.DataManager({
url: "http://mvc.syncfusion.com/Services/Northwnd.svc/"
});
// Query creation
var query = ej.Query().from("Order_Details").take(3).where("UnitPrice",
ej.FilterOperators.greaterThan, 18, false)
.where("UnitPrice", ej.FilterOperators.lessThanOrEqual, 40, false)
.where("Quantity", ej.FilterOperators.greaterThan, 5, false)
.where("Quantity", ej.FilterOperators.lessThanOrEqual, 45, false);
var fields = {
dataSource: dataManger,
query: query,
category: "ProductID",
featureMeasures: "UnitPrice",
comparativeMeasure: "Quantity"
};
var qualitativeRanges = [{ rangeEnd: 25 }, { rangeEnd: 37 }, { rangeEnd: 45
}];
var quantitativeScaleSettings = {
location: { x: 50, y: 20 },
minimum: 5,
maximum: 45,
interval: 10,
};
ReactDOM.render(
<EJ.BulletGraph id="bulletCore0"
qualitativeRanges = {qualitativeRanges}

```

```
quantitativeScaleSettings ={quantitativeScaleSettings} qualitativeRangeSize
={60}
fields ={fields}
>
</EJ.BulletGraph>,
document.getElementById('bulletgraph')
);
```

The following screenshot displays a Bullet Graph bounded with data from a remote server



### Quantitative Scale

The **Quantitative Scale** appearance is customized using **quantitativeScaleSettings** property. It has properties to customize labels, major ticks, minor ticks, comparative measure and performance measure of the bullet graph

#### Range for Quantitative Scale

**Quantitative Scale** range is set using the properties **minimum**, **maximum** and **interval** of **quantitativeScaleSettings** property. **Minimum** specifies the start range of the scale, **maximum** specifies the end range of scale and **interval** specifies the number of intervals between start and end range. Default values of **minimum**, **maximum** and **interval** are 0, 10 and 1 respectively. The number of minor ticks (ticks between intervals) are specified using **minorTicksPerInterval** property.

#### JAVASCRIPT

```
"use strict";
var quantitativeScaleSettings = {
  minimum: 0,
  maximum: 10,
  interval: 1,
  minorTicksPerInterval: 4
};
ReactDOM.render(
  <ej.bulletgraph id="bulletgraph1"
  quantitativeScaleSettings ={quantitativeScaleSettings} >
  </ej.bulletgraph>,
  document.getElementById('bulletgraph')
);
```

The following screenshot displays a **Bullet Graph** with start range 0, end range 10 and interval 1 with 4 minor ticks per interval



### Quantitative scale location

Bullet Graph does not position Quantitative scale automatically based on its size or space required for caption text, etc. By default Quantitative scale is positioned at 10 pixels from left and 10 pixels from top. Quantitative scale location is customized as per the requirement using the location property available in **quantitativeScaleSettings**.

#### JAVASCRIPT

```
"use strict";
var quantitativeScaleSettings = { location: { x: 20, y:20 }};
```

```
ReactDOM.render(
  <ej.bulletgraph id="bulletgraph1"
    quantitativeScaleSettings={quantitativeScaleSettings} ></ej.bulletgraph>,
  document.getElementById('bulletgraph')
);
```

The following screenshot displays **Bullet Graph** with Quantitative scale at 20 pixels from left and 20 pixels from top

![[/js/BulletGraph/Quantitative-Scaleimages/Quantitative-Scaleimg2.png)]

### Major ticks

Color, size and width of **Major tick** lines are customized using **majorTickSettings** property in **quantitativeScaleSettings**. Default value of **size** and **width** properties are 13 and 2 respectively. **Ticks** are drawn in black color by default. The property **size** represents the height of **tick** lines and **width** represents the width of **tick** lines and **ticks** color are customized using **stroke** property.

### JAVASCRIPT

```
"use strict";
var quantitativeScaleSettings = {
  majorTickSettings: {
    size: 15,
    width: 3,
    stroke: 'gray'
  }
};
ReactDOM.render(
  <ej.bulletgraph id="bulletgraph1"
    quantitativeScaleSettings={quantitativeScaleSettings} >
  </ej.bulletgraph>,
  document.getElementById('bulletgraph')
);
```

The following screenshot displays **Major ticks** in **gray** color with a width of 3 pixels and height 15 pixels.

![[/js/BulletGraph/Quantitative-Scaleimages/Quantitative-Scaleimg3.png)]

### Minor ticks

Minor ticks can also be customized similar to major ticks. The properties **stroke**, **width** and **size** of **minorTickSettings** are used to customize Minor ticks in quantitative scale. **Stroke** specifies the color of ticks, **width** specifies the width of ticks and **size** specifies the height of the ticks.

### JAVASCRIPT

```
"use strict";
var quantitativeScaleSettings = {
  minorTickSettings: {
    size: 7,
    width: 3,
    stroke: 'gray'
  }
};
ReactDOM.render(
  <ej.bulletgraph id="bulletgraph1"
    quantitativeScaleSettings={quantitativeScaleSettings} >
```

```
</ej.bulletgraph>,
document.getElementById('bulletgraph')
);
```

The following screenshot displays **Bullet Graph** with customized **Minor ticks** in quantitative scale



#### Tick position

**Ticks** are positioned below, above or inside the quantitative scale. By default **ticks** are positioned below the quantitative scale. The **tickPosition** property is used to customize the position of ticks in quantitative scale. **Ticks** can be placed inside the quantitative scale by setting **tickPosition** to **cross**.

#### JAVASCRIPT

```
"use strict";
var quantitativeScaleSettings = {
  tickPosition: 'above'
};
ReactDOM.render(
  <ej.bulletgraph id="bulletgraph1"
    quantitativeScaleSettings={quantitativeScaleSettings} >
  </ej.bulletgraph>,
  document.getElementById('bulletgraph')
);
```

The following screenshot displays **Bullet Graph** with ticks positioned above quantitative scale



#### Tick Placement

**Quantitativescaleticks** can be placed either inside or outside the scale using “**tickPlacement**” property. By default ticks are placed outside the scale.

#### JAVASCRIPT

```
"use strict";
var quantitativeScaleSettings = {
  location: { x: 108, y: 10 },
  tickPlacement: 'inside',
  labelSettings: { offset: 5, size: 10, labelPrefix: '$', labelSuffix: ' K' },
};
var captionSettings = {
  textAngle: 0, location: { x: 17, y: 28 }, text: "Revenue YTD",
  subTitle: {
    textAngle: 0,
    text: "$ in Thousands", location: { x: 10, y: 42 }
  }
};
ReactDOM.render(
  <ej.bulletgraph id="bulletgraph1" value={8} comparativeMeasureValue={5}
    qualitativeRangeSize={50} quantitativeScaleSettings
    ={quantitativeScaleSettings} >
  </ej.bulletgraph>,
  document.getElementById('bulletgraph')
);
```

The following screenshot displays **Bullet Graph** ticks inside **Quantitative Scale**

![[/js/BulletGraph/Quantitative-Scaleimages/Quantitative-Scaleimg6.png)]

### Quantitative scale labels

**Quantitivescalelabels** are customized with prefix, suffix, font, color and size using **labelSettings** property. By default, label text is displayed in black color with 12 pixel 'Segoe UI' font and there is a padding of 20 pixels space between quantitative scale and labels.

### JAVASCRIPT

```
"use strict";
var quantitativeScaleSettings = {
  labelSettings: {
    stroke: 'blue',
    labelPrefix: '$',
    labelSuffix: 'K',
    font: {
      fontFamily: 'Segoe UI',
      fontStyle: 'bold',
      fontWeight: 'regular',
      opacity: 0.8
    },
    size: 12,
    offset: 15
  }
};
ReactDOM.render(
  <ej.bulletgraph id="bulletgraph1"
    quantitativeScaleSettings={quantitativeScaleSettings} >
  </ej.bulletgraph>,
  document.getElementById('bulletgraph')
);
```

The following screenshot displays **Bullet Graph** labels in blue color

![[/js/BulletGraph/Quantitative-Scaleimages/Quantitative-Scaleimg7.png)]

### Label Placement

**Quantitivescalelabels** can be placed either inside or outside the scale using **"labelPlacement"** property. By default labels are placed 15 pixels outside the scale.

### JAVASCRIPT

```
"use strict";
var quantitativeScaleSettings = {
  location: { x: 108, y: 10 },
  labelSettings: {
    offset: 5, size: 10, labelPrefix: '$', labelSuffix: ' K', font: {
      fontWeight:
        'bold' },
    labelPlacement: 'inside'
  },
};
var captionSettings = {
```

```

textAngle: 0, location: { x: 17, y: 28 }, text: "Revenue YTD",
subTitle: {
textAngle: 0,
text: "$ in Thousands", location: { x: 10, y: 42 }
}
};
ReactDOM.render(
<ej.bulletgraph id="bulletgraph1" value ={8} comparativeMeasureValue ={5}
captionSettings ={captionSettings} quantitativeScaleSettings =
{quantitativeScaleSettings} >
</ej.bulletgraph>,
document.getElementById('bulletgraph')
);

```

The following screenshot displays **Bullet Graph** labels inside **Quantitative Scale**.



#### Performance measure bar

Performance measure bar is customized using featuredMeasureSettings in quantitativeScaleSettings property. Color of the bar is customized using stroke property and width using width property. By default bar is drawn in black color with 6 pixels of width.

#### JAVASCRIPT

```

"use strict";
var quantitativeScaleSettings = {
  featuredMeasureSettings: {
    stroke: 'blue',
    width: 4
  },
};
ReactDOM.render(
<ej.bulletgraph id="bulletgraph1" value ={5}
quantitativeScaleSettings ={quantitativeScaleSettings} >
</ej.bulletgraph>,
document.getElementById('bulletgraph')
);

```

The following screenshot displays **Bullet Graph** with customized **Performance measure bar**.



#### Comparative measure symbol

Comparative symbol color and width are customized using comparativeMeasureSettings through quantitativeScaleSettings property. Color of the symbol is customized using stroke property and width using width property. By default Comparative measure symbol is displayed in black color with a width of 5 pixels.

#### JAVASCRIPT

```

"use strict";
var quantitativeScaleSettings = {
  comparativeMeasureSettings: {
    stroke: 'blue',
    width: 5
  }
};

```

```

},
};
ReactDOM.render(
<ej.bulletgraph id="bulletgraph1" comparativeMeasureValue={5}
quantitativeScaleSettings = {quantitativeScaleSettings} >
</ej.bulletgraph>,
document.getElementById('bulletgraph')
);

```

The following screenshot displays **Bullet Graph** with customized **Comparative measure value**.



### Multiple performance measures comparison

**Bullet Graph** supports comparing more than one performance at a time, given that all the comparisons are related using **featureMeasure** in **quantitativeScaleSettings** property.

### JAVASCRIPT

```

"use strict";
var quantitativeScaleSettings = {
  featureMeasures: [
    { value: 6, comparativeMeasureValue: 3, category: 2010 },
    { value: 9, comparativeMeasureValue: 6, category: 2011 },
    { value: 5, comparativeMeasureValue: 5, category: 2012 },
  ]
};
ReactDOM.render(
<ej.bulletgraph id="bulletgraph1" height={120} qualitativeRangeSize = {60}
quantitativeScaleSettings = {quantitativeScaleSettings} >
</ej.bulletgraph>,
document.getElementById('bulletgraph')
);

```

The following screenshot displays **Bullet Graph** that compares 3 related performance measures.



### Qualitative Range

**Qualitative Range** represents the quality of a specific range in quantitative scale like good, bad and satisfactory. Color for each qualitative range is customized using **rangeStroke** property. The **rangeEnd** property specifies the ending point of the qualitative range. Minimum value of quantitative scale is considered as the starting point of first qualitative range and previous end points are considered as starting point for other qualitative ranges.

### JAVASCRIPT

```

"use strict";
var quantitativeScaleSettings = {
  location: { x: 50, y: 20 },
  minimum: 0,
  maximum: 100,
  interval: 10,
  featureMeasures: [
    { value: 55, comparativeMeasureValue: 75, category: "Year 1" },
    { value: 65, comparativeMeasureValue: 70, category: "Year 2" },
  ]
};

```



```

{ value: 80, comparativeMeasureValue: 65, category: "Year 3" }
];
};
var qualitativeRanges = [
{ rangeEnd: 35, rangeStroke: 'darkred', rangeOpacity: 0.5 },
{ rangeEnd: 50, rangeStroke: 'red', rangeOpacity: 1 },
{ rangeEnd: 75, rangeStroke: 'blue', rangeOpacity: 0.7 },
{ rangeEnd: 90, rangeStroke: 'lightgreen', rangeOpacity: 1 },
{ rangeEnd: 100, rangeStroke: 'green', rangeOpacity: 1 }
];
ReactDOM.render(
<ej.bulletgraph id="bulletgraph1"
qualitativeRanges={qualitativeRanges}
qualitativeRangeSize={80}
height={200}
quantitativeScaleSettings={quantitativeScaleSettings}>
</ej.bulletgraph>,
document.getElementById('bulletgraph')
);

```

The following screenshot displays **Bullet Graph** with different qualitative ranges in different colors. In this image, range 0 to 35 represents bad performance, 35 to 50 represents average performance, 50 to 75 represents that the performance is above average, 75 to 90 represents good performance and above 90 represents excellent performance.



## User Interaction

### Animation

**Bullet Graph** supports animation that makes the performance measure bar to animate when rendering the **Bullet Graph**. **Animation** is enabled or disabled using **enableAnimation** property. By default, **Animation** is enabled in **Bullet Graph**.

### JAVASCRIPT

```

"use strict";
ReactDOM.render(
<ej.bulletgraph id="bulletGraph1"
value = {8}
comparativeMeasureValue = {5}
enableAnimation = {true}>
</ej.bulletgraph>,
document.getElementById('bulletGraph')
);

```

### Responsiveness during browser resize

**Bullet Graph** is made responsive when resizing the browser by using **isResponsive** property. By default the value of this property is **true** in **Bullet Graph**.

### JAVASCRIPT

```

"use strict";
ReactDOM.render(
<ej.bulletgraph id="bulletGraph1" isResponsive = {true} value ={8}

```

```
comparativeMeasureValue = {5} >
</ej.bulletgraph>,
document.getElementById('bulletGraph')
);
```

Responsiveness of the linear gauge is controlled by using enableResize property.

#### JAVASCRIPT

```
"use strict";
ReactDOM.render(
<ej.bulletgraph id="bulletGraph1" enableResize = {true}>
</ej.bulletgraph>,
document.getElementById('bulletGraph')
);
```

Applying same color to all ticks and labels in a range

Background color for qualitative range is applied to major ticks and minor ticks of the **Bullet Graph** using **applyRangeStrokeToTicks** property. The range colors are applied to labels using **applyRangeStrokeToLabels** property. By default same colors are not applied to a qualitative range and its corresponding ticks or labels.

#### JAVASCRIPT

```
"use strict";
var qualitativeRanges = [
{ rangeEnd: 3.5, rangeStroke: 'yellow', rangeOpacity: 0.5 },
{ rangeEnd: 5.0, rangeStroke: 'red', rangeOpacity: 1 },
{ rangeEnd: 7.5, rangeStroke: 'blue', rangeOpacity: 0.7 },
{ rangeEnd: 9.0, rangeStroke: 'pink', rangeOpacity: 1 },
{ rangeEnd: 10.0, rangeStroke: 'green', rangeOpacity: 1 }
];
ReactDOM.render(
<ej.bulletgraph id="bulletGraph1"
qualitativeRanges={qualitativeRanges}
value = {8} applyRangeStrokeToTicks = {true}
applyRangeStrokeToLabels = {true}
comparativeMeasureValue = {5} >
</ej.bulletgraph>,
document.getElementById('bulletGraph')
);
```



#### Tooltip

By default **Bullet Graph** displays **Tooltip** when mouse is hovered over feature measure bar. **Tooltip** is enabled or disabled using visible property in **tooltipSettings**.



Bullet Graph supports Tooltip template instead of default Tooltip to customize the appearance and contents of Tooltip. The Tooltip template should be a <div> element with display set to 'none', so it is displayed only when mouse is placed on feature measure bar. The id value of the <div> element should be provided as value to the template property in tooltipSettings of Bullet Graph to display the

customized <div> element as Tooltip instead of default Tooltip. The values displayed in default Tooltip such as current value, target value and category are accessed in template <div> element by using {{currentValue}}, {{targetValue}} and {{category}} respectively.

### HTML

```
<div id="BulletGraphTooltip" style="display:none; width:125px; padding-top:10px; padding-bottom:10px; color: blue">
<div align="center" style="color:blue; font-weight:bold"> Sales </div>
<table style="color:green"> <tr> <td> Current </td> <td> : </td> </tr> <tr>
<td> Target </td> <td> : </td> </tr> </table>
</div>
```

### JAVASCRIPT

```
"use strict";
var tooltipSettings = { template: 'BulletGraphTooltip' };
ReactDOM.render(
  <ej.bulletgraph id="bulletGraph1" tooltipSettings =
  {tooltipSettings} value ={8}
    comparativeMeasureValue ={6} height = {150} >
  </ej.bulletgraph>,
  document.getElementById('bulletGraph')
);
```

The following screenshot displays **Bullet Graph** with a customized **Tooltip** including a header and contents such as current value and target value in different colors.



### Methods

#### *destroy()*

To destroy the bullet graph

### HTML

```
<div id="bullet"></div>
```

### JAVASCRIPT

```
ReactDOM.render(
  <EJ.BulletGraph id="bulletCore0"></EJ.BulletGraph>,
  document.getElementById('bullet')
);
function BulletGraphMethod(){
  var bulletObj = $("#bulletCore0").data("ejBulletGraph");
  bulletObj.destroy();
};
```

#### *redraw()*

To redraw the bullet graph

### HTML

```
<div id="bullet"></div>
```

**JAVASCRIPT**

```
ReactDOM.render(
  <EJ.BulletGraph id="bulletCore0"></EJ.BulletGraph>,
  document.getElementById('bullet')
);
function BulletGraphMethod() {
  var bulletObj = $("#bulletCore0").data("ejBulletGraph");
  bulletObj.redraw();
};
```

*setComparativeMeasureSymbol()*

To set the value for comparative measure in bullet graph.

**HTML**

```
<div id="bullet"></div>
```

**JAVASCRIPT**

```
ReactDOM.render(
  <EJ.BulletGraph id="bulletCore0"></EJ.BulletGraph>,
  document.getElementById('bullet')
);
function BulletGraphMethod() {
  var bulletObj = $("#bulletCore0").data("ejBulletGraph");
  bulletObj.setComparativeMeasureSymbol();
};
```

*setFeatureMeasureBarValue()*

To set the value for feature measure bar.

**HTML**

```
<div id="bullet"></div>
```

**JAVASCRIPT**

```
ReactDOM.render(
  <EJ.BulletGraph id="bulletCore0"></EJ.BulletGraph>,
  document.getElementById('bullet')
);
function BulletGraphMethod() {
  var bulletObj = $("#bulletCore0").data("ejBulletGraph");
  bulletObj.setFeatureMeasureBarValue();
};
```

*Events**drawCaption*

Fires on rendering the caption of bullet graph.

**HTML**

```
<div id="bullet"></div>
```

### JAVASCRIPT

```
ReactDOM.render(
  <EJ.BulletGraph id="bulletCore0"
    drawCaption = {DrawCaption}
  >
</EJ.BulletGraph>,
  document.getElementById('bullet')
);
function DrawCaption() {
  // Do Something
};
```

#### *drawCategory*

Fires on rendering the category.

### HTML

```
<div id="bullet"></div>
```

### JAVASCRIPT

```
ReactDOM.render(
  <EJ.BulletGraph id="bulletCore0"
    drawCategory = {DrawCategory}
  >
</EJ.BulletGraph>,
  document.getElementById('bullet')
);
function DrawCategory() {
  // Do Something
};
```

#### *drawComparativeMeasureSymbol*

Fires on rendering the comparative measure symbol.

### HTML

```
<div id="bullet"></div>
```

### JAVASCRIPT

```
ReactDOM.render(
  <EJ.BulletGraph id="bulletCore0"
    drawComparativeMeasureSymbol = {DrawComparativeMeasureSymbol}
  >
</EJ.BulletGraph>,
  document.getElementById('bullet')
);
function DrawComparativeMeasureSymbol() {
  // Do Something
};
```

*drawFeatureMeasureBar*

Fires on rendering the feature measure bar.

**HTML**

```
<div id="bullet"></div>
```

**JAVASCRIPT**

```
ReactDOM.render(  
  <EJ.BulletGraph id="bulletCore0"  
    drawFeatureMeasureBar = {DrawFeatureMeasureBar}  
  >  
    </EJ.BulletGraph>,  
  document.getElementById('bullet')  
);  
function DrawFeatureMeasureBar() {  
  // Do Something  
};
```

*drawIndicator*

Fires on rendering the indicator of bullet graph.

**HTML**

```
<div id="bullet"></div>
```

**JAVASCRIPT**

```
ReactDOM.render(  
  <EJ.BulletGraph id="bulletCore0"  
    drawIndicator = {DrawIndicator}  
  >  
    </EJ.BulletGraph>,  
  document.getElementById('bullet')  
);  
function DrawIndicator() {  
  // Do Something  
};
```

*drawLabels*

Fires on rendering the labels.

**HTML**

```
<div id="bullet"></div>
```

**JAVASCRIPT**

```
ReactDOM.render(  
  <EJ.BulletGraph id="bulletCore0"  
    drawLabels = {DrawLabels}  
  >
```

```

</EJ.BulletGraph>,
document.getElementById('bullet')
);
function drawLabels() {
  // Do Something
};

```

### *drawQualitativeRanges*

Fires on rendering the qualitative ranges.

### HTML

```

<div id="bullet"></div>

```

### JAVASCRIPT

```

ReactDOM.render(
  <EJ.BulletGraph id="bulletCore0"
  drawQualitativeRanges = {DrawQualitativeRanges}
  >
  </EJ.BulletGraph>,
  document.getElementById('bullet')
);
function DrawQualitativeRanges() {
  // Do Something
};

```

### *load*

Fires on loading bullet graph.

### HTML

```

<div id="bullet"></div>

```

### JAVASCRIPT

```

ReactDOM.render(
  <EJ.BulletGraph id="bulletCore0"
  load = {load}
  >
  </EJ.BulletGraph>,
  document.getElementById('bullet')
);
function load() {
  // Do Something
};

```

## Button

### Overview

The **Button** control allows you to perform an action by clicking on it. The **Button** control has the feature of displaying both text and images. When the **Button** is clicked, it looks as if it is being pushed in and released.

## Key Features

- **Trendy Look:** Rich Appearance with Theme Support
- **RTL:** Supports for Right to Left alignment
- **Repeat-Button:** Supports the rising of click event Repeatedly
- **Text and Image:** Supports both text and image as **Button** content
- **Built-in Icon:** Supports the built-in icon libraries
- **Easy Customization:** The customization of **Button** control to any form is made simple

## Getting Started

This section explains briefly about how to create a Button in your application with ReactJS.

You can create a React JS application and add necessary scripts and styles with the help of the given [link](#)

Define an HTML element to rendering as the Button in the application and refer the JSX file. Please check with the below code example,

### HTML

```
<body>
  <!--Add React JS components-->
  <div id="btn"></div>
  <!-- Button.jsx created in previous step-->
  <script type="text/babel" src="Button.jsx">
  </script>
</body>
```

## Control Initialization with react js

Control can be initialized in two ways.

- Using JSX Template.
- Without using JSX Template

### Using JSX Template

With the JSX template concept, we need to create the html file and JSX file. The .jsx file can be convert to .js file and it can be referred in html page.

Create a JSX file for rendering Button component with below code.

### HTML

```
ReactDOM.render (
  <EJ.Button id="btn" text="Button">
</EJ.Button>,
  document.getElementById('btn')
);
```

Run the above code to render the following output.



My First Button

BUTTON

### Configuring Button

This section encompasses the details on how you can configure the Button control in your application and customize it with various properties such as various size, resize the Button and rounded corner according to your requirement.

To render the text and size properties add the following code example in your JS file.

#### HTML

```
ReactDOM.render(  
  <EJ.Button id="btn" text="Button" showRoundedCorner={true} size="large"  
    text="Click here">  
  </EJ.Button>,  
  document.getElementById('btn')  
) ;
```

The following screenshot illustrates the Button control with text, size and rounded corner properties.

### Create a Button without using JSX template

The Button can be created from a normal script section without using JSX template file also. Please check with the below code example,

#### HTML

```
<script>  
ReactDOM.render(  
  React.createElement(EJ.Button, { id: "container",text:"Button" }  
),  
  document.getElementById('container')  
);  
</script>
```

### Configure Properties

This section encompasses the details on how you can configure the Button control in your application and customize it with various properties such as various size, rename and rounded corner for Button according to your requirement.

To render the text, size and rounded corner properties add the following snippet in your HTML file.

#### JAVASCRIPT

```
<script>  
ReactDOM.render(  
  React.createElement(EJ.Button, { id:  
    "container",text:"Button", showRoundedCorner:true, size:"large", text:"Click  
    here" }  
),  
  document.getElementById('container')
```

```
);
</script>
```

## Chart

### Getting Started

This section explains you the steps required to populate the Chart with data, add data labels, tooltips and title to the Chart. This section covers only the minimal features that you need to know to get started with the Chart.

### Adding Script Reference

Create an **HTML** page and add the scripts references in the order mentioned in the following code example.

- [jQuery](#) 1.10.2 and later versions

The required ReactJS script dependencies as follows. And you can also refer [React](#) to know more about react js.

- react.min.js - <http://cdn.syncfusion.com/js/assets/external/react.min.js>
- react-dom.min.js - <http://cdn.syncfusion.com/js/assets/external/react-dom.min.js>
- ej.web.react.min.js - <http://cdn.syncfusion.com/{{ site.releaseversion }}/js/common/ej.web.react.min.js>

To get started, you can use the `ej.web.all.min.js` file that encapsulates all the `ej` controls and frameworks in one single file.

### HTML

```
<!DOCTYPE html>
<html>
<head>
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<meta name="description" content="Essential Studio for React JS">
<meta name="author" content="Syncfusion">
<title>Getting Started for Ribbon React JS</title>
<!-- Essential Studio for JavaScript theme reference -->
<link href="http://cdn.syncfusion.com/{{ site.releaseversion }}/js/web/flat-azure/ej.web.all.min.css" rel="stylesheet" />
<!-- Essential Studio for JavaScript script references -->
<script src="http://cdn.syncfusion.com/js/assets/external/jquery-3.0.0.min.js"></script>
<script src="http://cdn.syncfusion.com/js/assets/external/react.min.js"></script>
<script src="http://cdn.syncfusion.com/js/assets/external/react-dom.min.js"></script>
<script src="http://cdn.syncfusion.com/{{ site.releaseversion }}/js/web/ej.web.all.min.js"></script>
<script src="http://cdn.syncfusion.com/{{ site.releaseversion }}/js/common/ej.web.react.min.js"></script>
<!-- Add your custom scripts here -->
</head>
```

```
<body>  
</body>  
</html>
```

**Note:** 1. In production, we highly recommend you to use our [custom script generator](#) to create custom script file with required controls and its dependencies only. Also to reduce the file size further please use [GZip compression](#) in your server.

2. For themes, you can use the `ej.web.all.min.css` CDN link from the code snippet given. To add the themes in your application, please refer to [this link](#).

### Control Initialization

Control can be initialized in two ways.

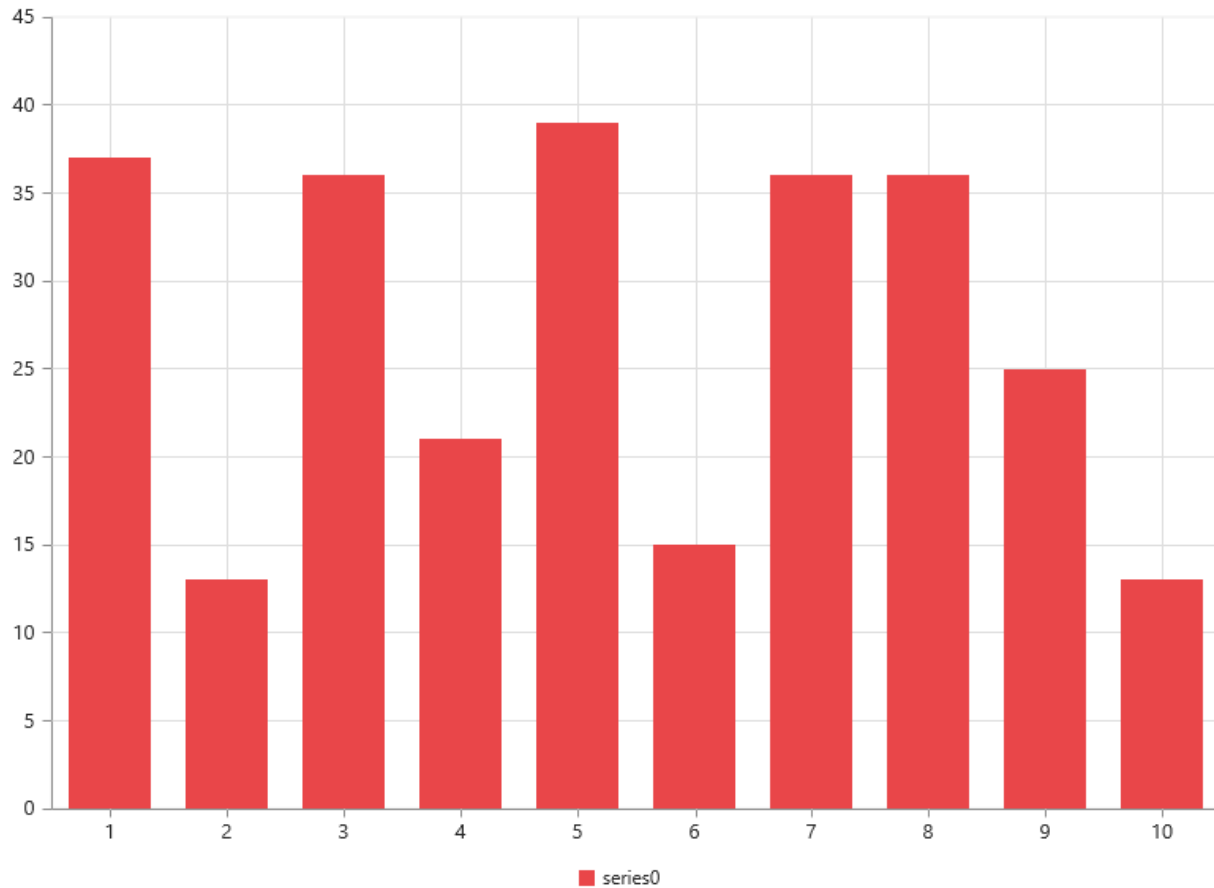
- Using jsx Template.
- Without using jsx Template.

### Using jsx Template

By using the jsx template, we can create the html file and jsx file. The `.jsx` file can be convert to `.js` file and it can be referred in html page.

### Create your chart

In this tutorial, you will learn how to create a simple chart. The following screen shot displays the output after completing this tutorial.



1.Create a  
tag.

### HTML

```
<!DOCTYPE html>
<html>
<body>
<div id="chart-default" style="height:99%;"></div>
<script src="app/chart/default.js"></script>
</body>
</html>
```

2.Initialize the Chart by using the `EJ.Chart` tag.

### JAVASCRIPT

```
"use strict";
ReactDOM.render(
  <div className="default">
    <EJ.Chart id="chart1"></EJ.Chart>,
  </div>,
  document.getElementById('chart-default')
);
```

Now, the Chart is rendered with some auto-generated random values and with default Column chart type.

The chart is rendered to the size of its container, by default. You can also customize the chart dimension either by setting the width and height of the container element as in the above code example or by using the **Size** option of the Chart.

### Populate chart with data

Now, this section explains how to plot JSON data to the Chart. First, let us prepare a sample JSON data with each object containing following fields – month and sales.

#### JAVASCRIPT

```
var chartData = [
  { month: 'Jan', sales: 35 },
  { month: 'Feb', sales: 28 },
  { month: 'Mar', sales: 34 },
  { month: 'Apr', sales: 32 },
  { month: 'May', sales: 40 },
  { month: 'Jun', sales: 32 },
  { month: 'Jul', sales: 35 },
  { month: 'Aug', sales: 55 },
  { month: 'Sep', sales: 38 },
  { month: 'Oct', sales: 30 },
  { month: 'Nov', sales: 25 },
  { month: 'Dec', sales: 32 }];
```

Add a Series to the Chart using **Series** option and set the chart type as **Line** using **type** option.

#### JAVASCRIPT

```
<script type="text/babel">
var series= {
  type: 'line'
};
<!DOCTYPE html>
<html>
<body>
<script type="text/babel">
ReactDOM.render(
<div className="default">
<EJ.Chart id="chart1" series = {series} </EJ.Chart>,
</div>,
document.getElementById('chart-default')
);
</script>
</body>
</html>
```

You can also add multiple series tags based on your requirement.

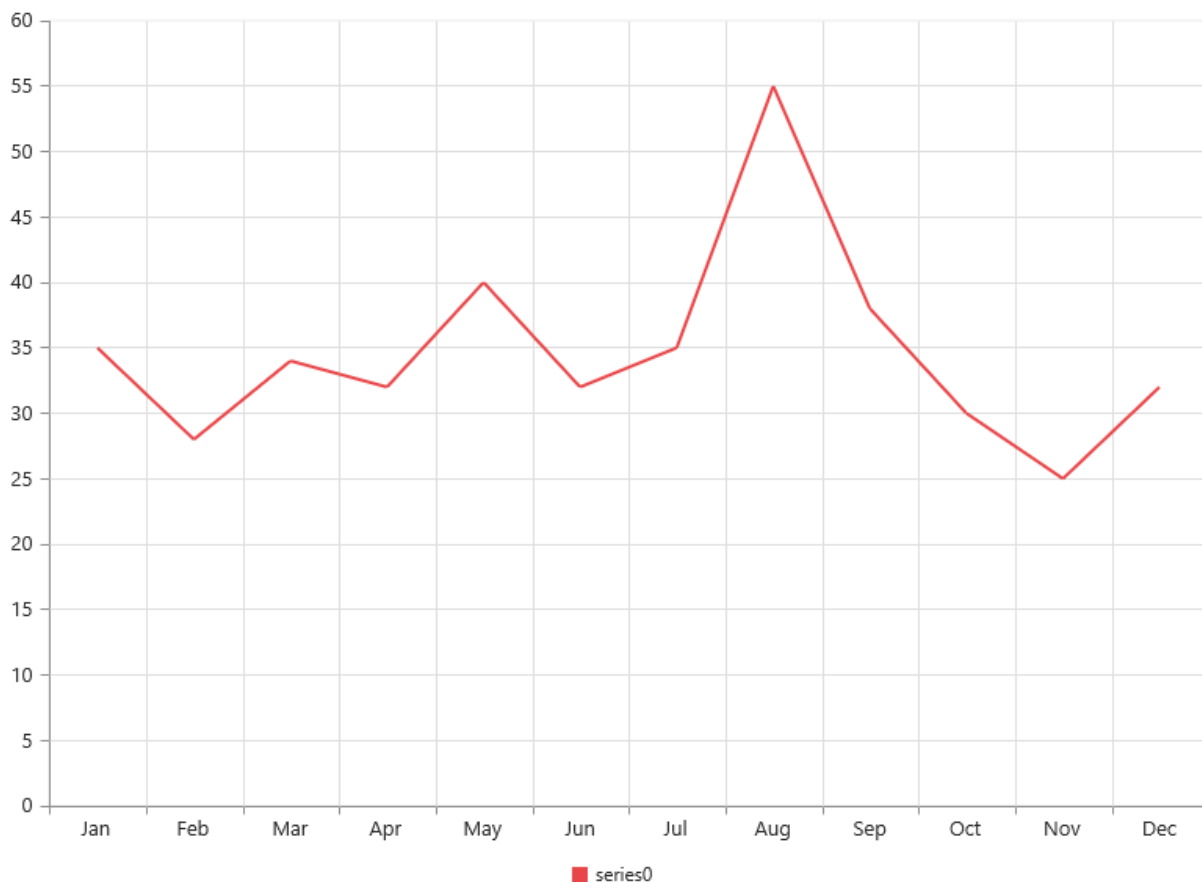
Next, map the Month and Sales values in the data source to the Line series by setting XName and YName with the field names respectively, and then set the actual data using DataSource option.

#### JAVASCRIPT

```

<script type="text/babel">
var series= {
dataSource: chartData,
xName: "month",
yName: "sales",
};
var size = {height:"400",width:"600"};
<!DOCTYPE html>
<html>
<body>
<script type="text/babel">
ReactDOM.render(
<div className="default">
<EJ.Chart id="chart1" series={series} ></EJ.Chart>,
</div>,
document.getElementById('chart-default')
);
</script>
</body>
</html>

```

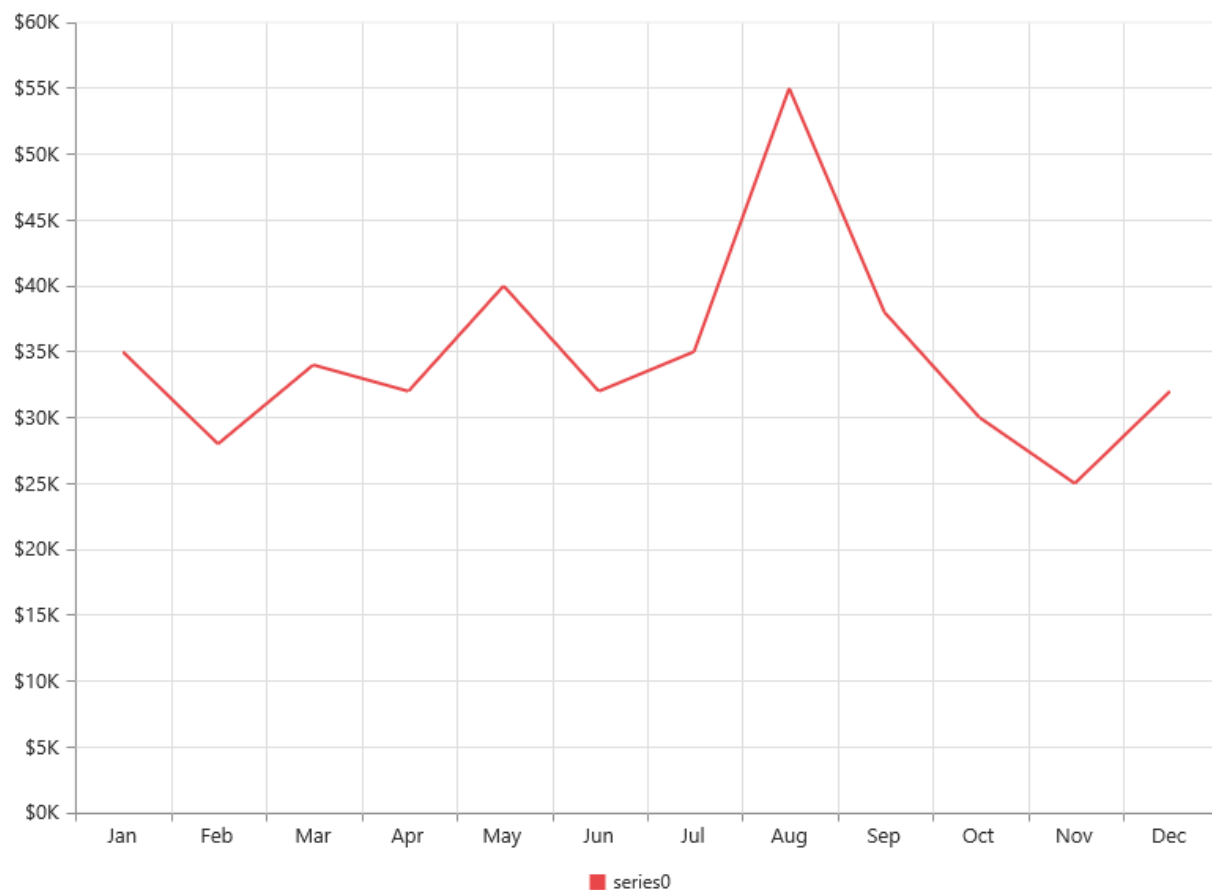


Since the data is related to Sales, format the vertical axis labels by adding '\$' as a prefix and 'K' as a suffix to each label. This can be achieved by setting the "\${value}K" to the **labelFormat** option of the axis. Here, {value} acts as a placeholder for each axis label, "\$" and "K" are the actual prefix and suffix added to each axis label.

The following code example illustrates this,

### JAVASCRIPT

```
<script type="text/babel">
var primaryYAxis= {
  labelFormat: "{value}k"
};
<!DOCTYPE html>
<html>
<body>
<script type="text/babel">
ReactDOM.render(
<div className="default">
<EJ.Chart id="chart1" primaryYAxis={primaryYAxis} ></EJ.Chart>,
</div>,
document.getElementById('chart-default')
);
</script>
</body>
</html>
```



### Add Data Labels

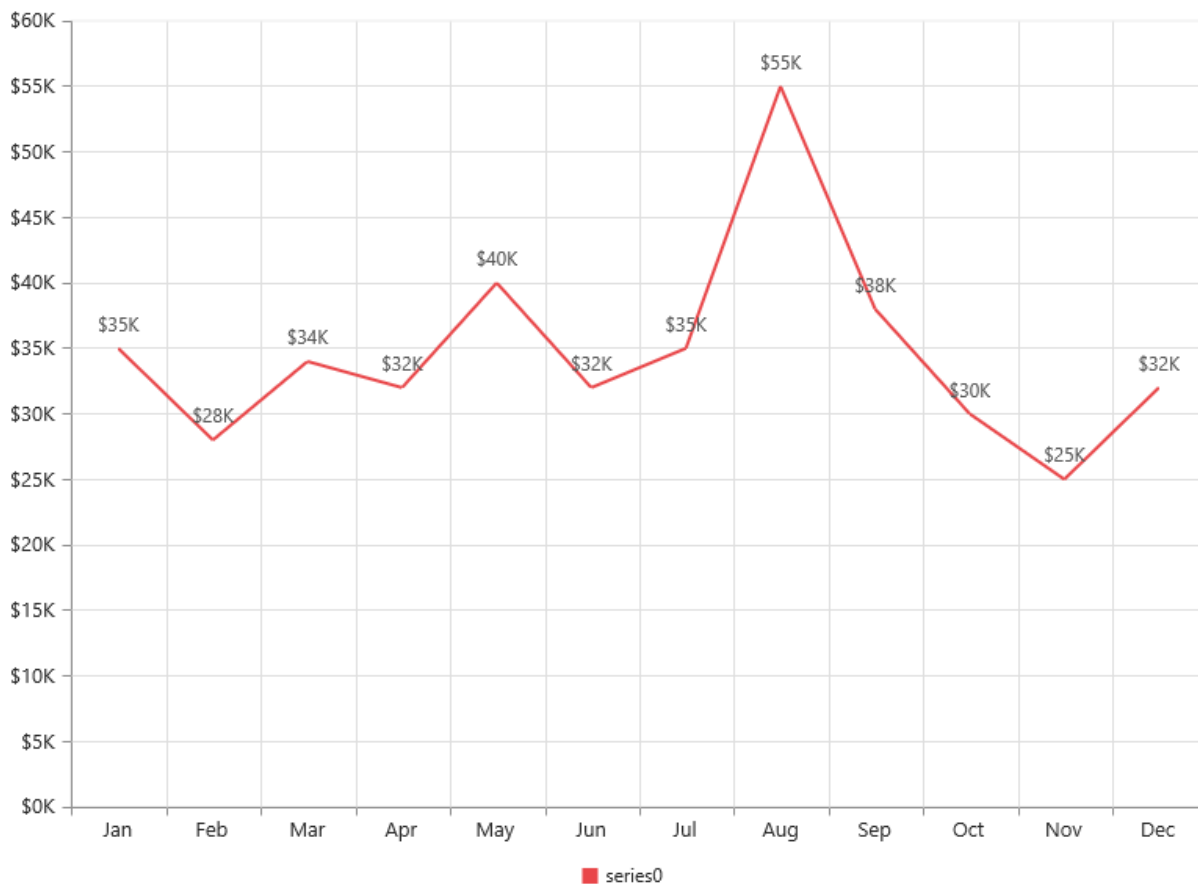
You can add data labels to improve the readability of the chart. This can be achieved by enabling the Visible option in the **dataLabel** option. Now, the data labels are rendered at the top of all the data points.

The following code example illustrates this,

#### JAVASCRIPT

```
<script type="text/babel">
var series= {
marker: {
dataLabel: {
//Enable data label in the chart
visible: true
} }
};
<!DOCTYPE html>
<html>
<body>
<script type="text/babel">
ReactDOM.render(
<div className="default">
<EJ.Chart id="chart1" series={series} </EJ.Chart>,
</div>,
document.getElementById('chart-default')
);
</script>
</body>
</html>
```





There are situations where the default label content is not sufficient to the user. In this case, you can use the **template** option to format the label content with some additional information.

### HTML

```
<!DOCTYPE html>
<html>
<body>
<div id="dataLabelTemplate" style="display:none; padding:3px;background-
color:#B9C5C9; opacity:0.8;">
<div id="point">#point.x#:$#point.y#K</div>
</div>
</body>
</html>
```

The above HTML template is used as a template for each data label. Here, “point.x” and “point.y” are the placeholder text used to display the corresponding data point’s x & y value.

The following code example shows how to set the id of the above template to dataLabel template option,

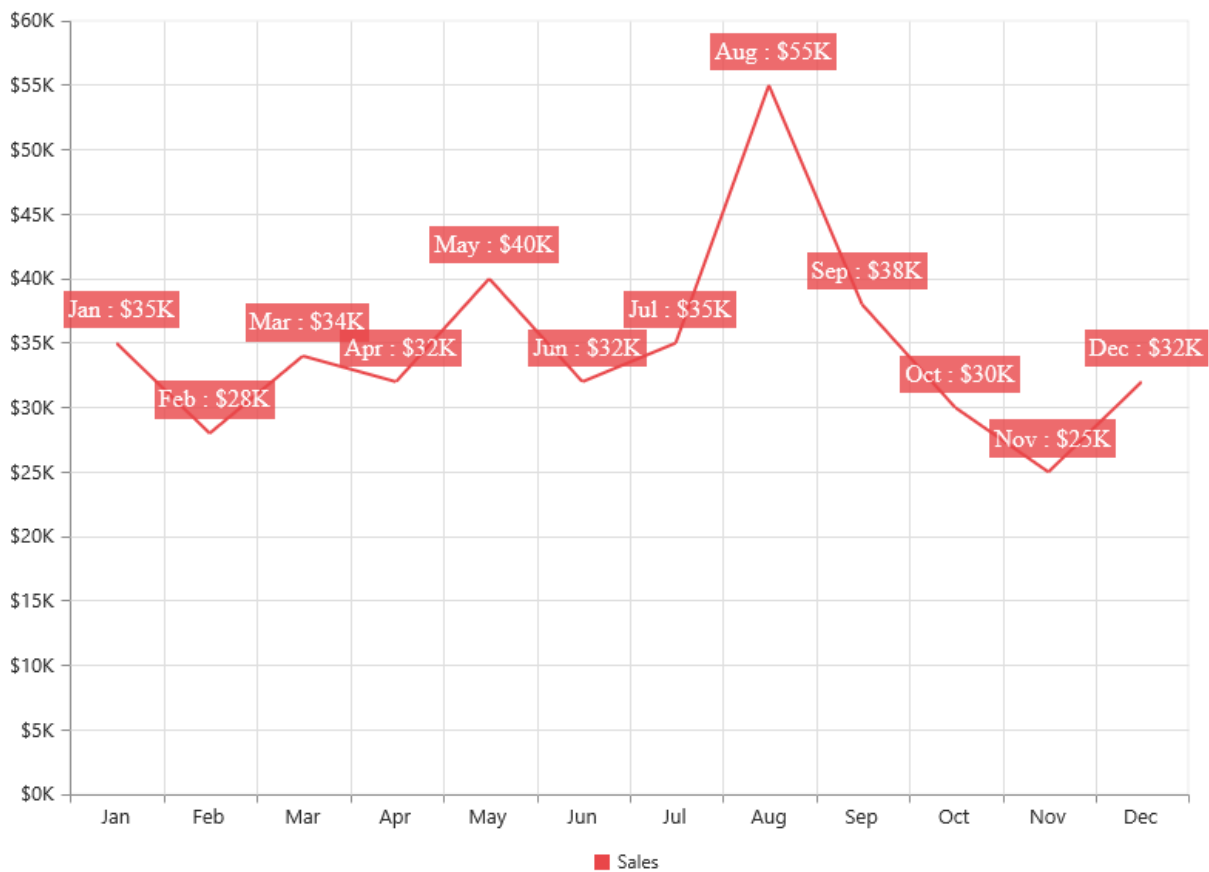
### JAVASCRIPT

```
var series= {
  marker: {
    dataLabel: {
```

```

visible: true,
//Set the id of HTML template to the chart series
template: "dataLabelTemplate"
} }
};
<!DOCTYPE html>
<html>
<body>
<script type="text/babel">
ReactDOM.render(
<div className="default">
<EJ.Chart id="chart1" series={series} </EJ.Chart>,
</div>,
document.getElementById('chart-default')
);
</script>
</body>
</html>

```



### Enable Legend

You can enable or disable the legend by using the Visible option in the **legend** option. It is enabled in the chart, by default.

### JAVASCRIPT

```
var series= {
```

```
//Add series name to display on the legend item
name: "Sales"
};
var legend = {
  //Enable chart legend
  visible: true
};
<!DOCTYPE html>
<html>
<body>
<script type="text/babel">
  ReactDOM.render(
    <div className="default">
      <EJ.Chart id="chart1" series={series} legend={legend} ></EJ.Chart>,
    </div>,
    document.getElementById('chart-default')
  );
</script>
</body>
</html>
```



### Enable Tooltip

The Tooltip is useful when you cannot display information by using the **dataLabel** due to the space constraints. You can enable tooltip by using the Visible option of the **tooltip** option in the specific series.

The following code example illustrates this,

### JAVASCRIPT

```
var series= {  
  //Enable tooltip in chart area  
  tooltip: {visible: true}  
};  
<!DOCTYPE html>  
<html>  
<body>  
<script type="text/babel">  
  ReactDOM.render(  
    <div className="default">  
      <EJ.Chart id="chart1" series={series} </EJ.Chart>,  
    </div>,  
    document.getElementById('chart-default')  
  );  
</script>  
</body>  
</html>
```

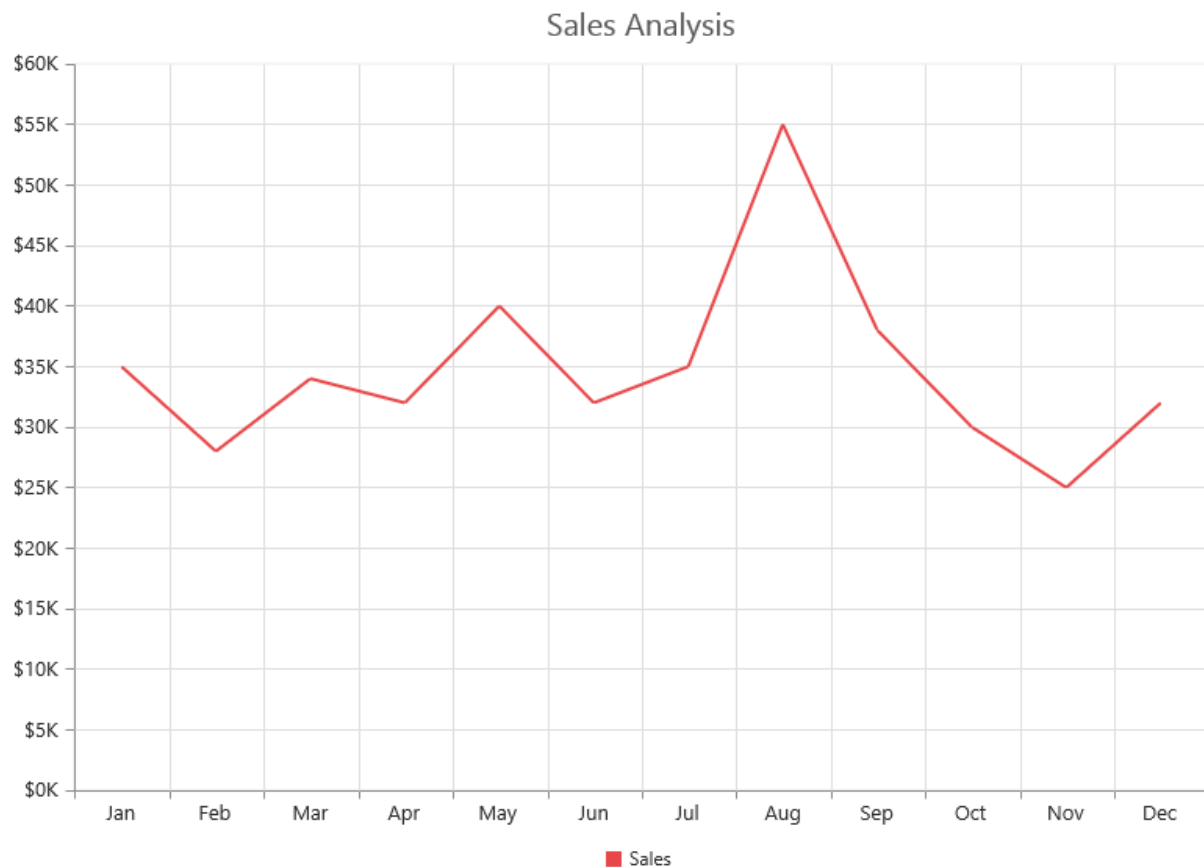


### Add Chart Title

You need to add a title to the chart to provide quick information to the user about the data being plotted in the chart. You can add it by using the text option of the **title** option.

## JAVASCRIPT

```
var title= {  
  //Add chart title  
  text: 'Sales Analysis'  
};  
<!DOCTYPE html>  
<html>  
<body>  
<script type="text/babel">  
  ReactDOM.render(  
    <div className="default">  
      <EJ.Chart id="chart1" title={title} </EJ.Chart>,  
    </div>,  
    document.getElementById('chart-default')  
  );  
</script>  
</body>  
</html>
```



### Without using jsx Template

The Chart can be created from a HTML **DIV** element with the HTML **id** attribute set to it. Refer to the following code example.

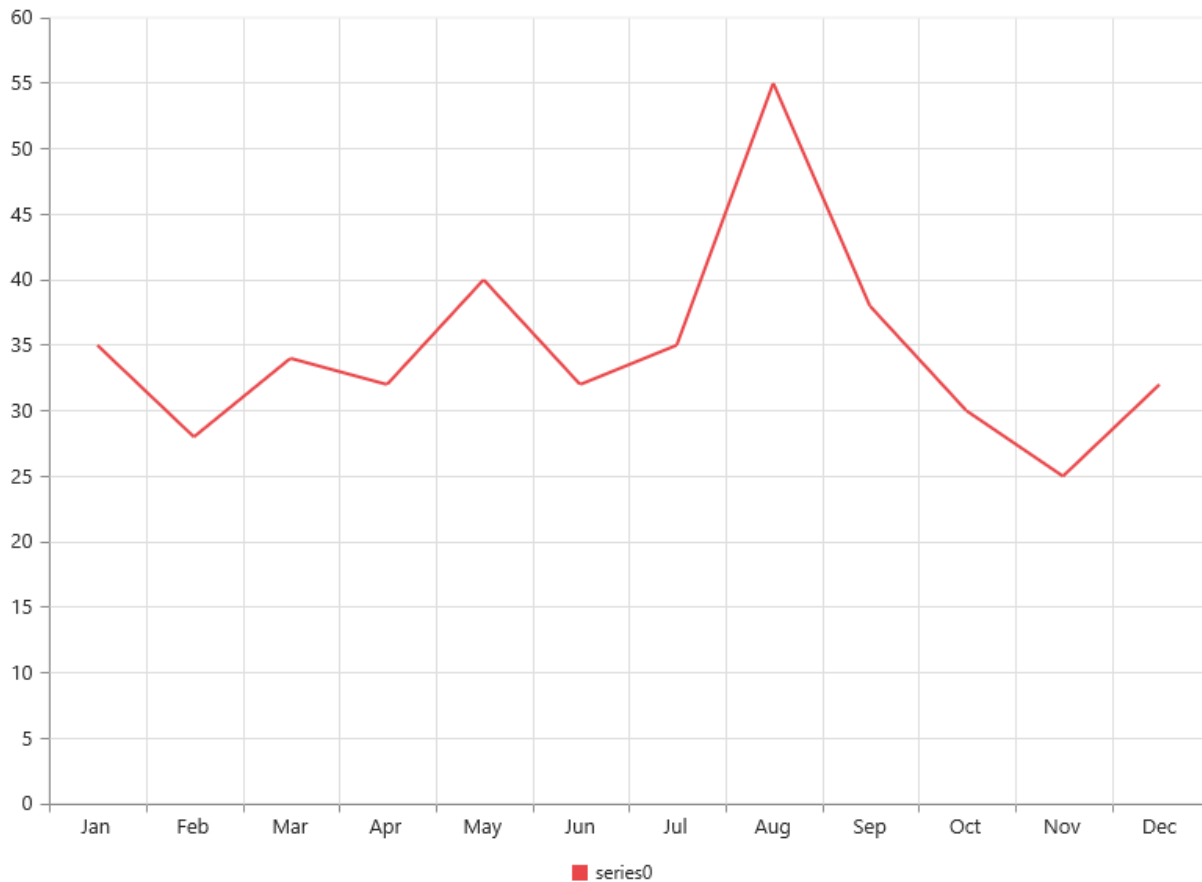
## HTML

```
<div id="chart-default" style="height:99%;"></div>
```

### JAVASCRIPT

```
var chartData = [
  { month: 'Jan', sales: 35 },
  { month: 'Feb', sales: 28 },
  { month: 'Mar', sales: 34 },
  { month: 'Apr', sales: 32 },
  { month: 'May', sales: 40 },
  { month: 'Jun', sales: 32 },
  { month: 'Jul', sales: 35 },
  { month: 'Aug', sales: 55 },
  { month: 'Sep', sales: 38 },
  { month: 'Oct', sales: 30 },
  { month: 'Nov', sales: 25 },
  { month: 'Dec', sales: 32 }];
var series= [{
  dataSource: chartData,
  xName: "month",
  yName: "sales",
  type:"line"
}];
var size = {height:"400",width:"600"};
ReactDOM.render(
  React.createElement(EJ.Chart, {id: "default_chart0",
  series:series,
  size:size
  }
),
document.getElementById('chart-default')
);
</script>
```

Now the line chart is rendered



## Working with Data

### Local Data

There are two ways to provide local data to chart.

1. You can bind the data to the chart by using the [dataSource](#) property of the series and then you need to map the X and Y value with the [xName](#) and [yName](#) properties respectively.

**Note:** For the **OHLC** type series, you have to map four [dataSource](#) fields ([high](#), [low](#), [open](#) and [close](#)) to bind the data source and for the **bubble** series you have to map the [size](#) field along with the [xName](#) and [yName](#).

### JAVASCRIPT

```
var chartData = [
  { month: 'Jan', sales: 35 }, { month: 'Feb', sales: 28 }, { month: 'Mar',
sales: 34 },
  { month: 'Apr', sales: 32 }, { month: 'May', sales: 40 }, { month: 'Jun',
sales: 32 },
  { month: 'Jul', sales: 35 }, { month: 'Aug', sales: 55 }, { month: 'Sep',
sales: 38 },
  { month: 'Oct', sales: 30 }, { month: 'Nov', sales: 25 }, { month: 'Dec',
sales: 32 }];
var series=[
{
```

```

dataSource: chartData,
xName: "month",
yName: "sales"
}
];
ReactDOM.render(
<EJ.Chart id="default_chart_sample_0"
series={series}
>
</EJ.Chart>,
document.getElementById('chart-default')
);

```



[Click](#) here to view the local data binding online demo sample.

2.You can also plot data to chart using [points](#) option in the series. Using this property you can customize each and every point in the data.

### JAVASCRIPT

```

"use strict";
var series = [{
  //Adding data points using x and y field of points
  points: [{ x: "John", y: 10000 }, { x: "Jake", y: 12000 }, { x: "Peter", y:
18000 },
  { x: "James", y: 11000 }, { x: "Mary", y: 9700 }],
  // ...
}];
ReactDOM.render(
<EJ.Chart id="chart1" series = {series} ></EJ.Chart>,
document.getElementById('chart-default')
);

```



### Remote Data

You can bind the remote data to the chart by using the DataManager and you can use the [query](#) property of the series to filter the data from the dataSource.

### JAVASCRIPT

```

"use strict";
//Remote URL
var dataManger = new ej.DataManager({
url: "http://mvc.syncfusion.com/Services/Northwnd.svc/"
});
// Query creation
var query = ej.Query().from("Orders").take(6);
var series = [{
type: 'column',
dataSource: dataManger,
xName: "ShipCity",
yName: "Freight",
query: query,

```



```

    });
    ReactDOM.render(
      <EJ.Chart id="chart1" series = {series} ></EJ.Chart>,
      document.getElementById('chart-default')
    );

```



[Click](#) here to view the remote data binding online demo sample.

## Chart Dimensions

You can set the size of the chart directly on the chart or to the container of the chart. When you do not specify the size, it takes 450px as the height and window size as its width, by default.

### Set size for the container

You can customize the chart dimension by setting the width and height for the container element.

#### JAVASCRIPT

```

<body>
<div id="container" style="width:820px; height:500px;"></div>
</body>
"use strict";
<EJ.Chart id="default_chart_sample_0">
</EJ.Chart>,

```

### Set size in pixels

You can also set the chart dimension by using the [size](#) property of the chart.

#### JAVASCRIPT

```

"use strict";
var size = { width: '600', height: '450' };
ReactDOM.render(
  <EJ.Chart id="chart1" size = {size}></EJ.Chart>,
  document.getElementById('chart')
);

```



### Setting size relative to the container size

You can specify the chart size in percentage by using the [size](#) property. The chart gets its dimension with respect to its container.

#### HTML

```

<body>
<div id="container" style="width:700px; height:500px"></div>
</body>
"use strict";
var size = { width: '80%', height: '90%' };
ReactDOM.render(
  <EJ.Chart id="chart1" size = {size}></EJ.Chart>,
  document.getElementById('chart')
);

```



### Responsive chart

To resize the Chart when the browser or the chart container is resized, set the [isResponsive](#) property to **true**, where the chart adapts to the changes in size of the container.

### JAVASCRIPT

```
"use strict";
ReactDOM.render(
  <EJ.Chart id="chart1" isResponsive = 'true' >/EJ.Chart>,
  document.getElementById('chart')
);
```

### Axis

**Charts** typically have two axes that are used to measure and categorize data: a vertical (y) axis, and a horizontal (x) axis.

Vertical axis always uses numerical or logarithmic scale. Horizontal(x) axis supports the following types of scale:

- Category
- Numeric
- DateTime
- DateTime Category
- Logarithmic

### Category Axis

Category axis displays the text labels instead of numbers. To use the categorical axis, you can set the [valueType](#) property of the axis to the **category**. Default value of [valueType](#) is **double**.

### JS

```
"use strict";
var primaryXAxis={
  //Use categorical scale in primary X axis
  valueType: 'category',
};
ReactDOM.render(
  <EJ.Chart id="default_chart_sample_0"
  primaryXAxis= {primaryXAxis}>
  </EJ.Chart>,
  document.getElementById('chart')
);
```



[Click](#) here to view our online demo sample that uses Category axis.

### Place labels on ticks

Labels in the category axis can be placed on the ticks by setting the [labelPlacement](#) property of axis to the **onticks**. The default value of the [labelPlacement](#) property is **betweenticks** i.e. labels are placed between the ticks, by default.

#### JAVASCRIPT

```
"use strict";
var primaryXAxis={
  //Placing X-axis labels on the ticks
  labelPlacement: 'onticks',
};
ReactDOM.render(
  <EJ.Chart id="default_chart_sample_0"
  primaryXAxis= {primaryXAxis}
  >
  </EJ.Chart>,
  document.getElementById('chart')
);
```



### Display labels after a fixed interval

To display the labels after a fixed interval *n*, you can set the [interval](#) property of the axis range as *n*. The default value of the interval is 1 i.e. all the labels are displayed.

#### JAVASCRIPT

```
"use strict";
var primaryXAxis={
  //Displaying labels after 2 intervals
  range: { interval: 2 },
};
ReactDOM.render(
  <EJ.Chart id="default_chart_sample_0"
  primaryXAxis= {primaryXAxis}
  >
  </EJ.Chart>,
  document.getElementById('chart')
);
```



### Indexed Category Axis

Category axis can also plot points based on index value of data points. Index based plotting can be enabled by setting [isIndexed](#) property to true in the axis.

#### JAVASCRIPT

```
"use strict";
var primaryXAxis={
  isIndexed: true
};
var series=[
  {
```

```

points:[{ x: "Monday", y: 50 }, { x: "Tuesday", y: 40 }, { x: "Wednesday",
y: 70 },
{ x: "Thursday", y: 60 }, { x: "Friday", y: 50 },
{ x: "Monday", y: 40 }, { x: "Monday", y: 30 }
]
}
];
ReactDOM.render(
<EJ.Chart id="default_chart_sample_0"
primaryXAxis= {primaryXAxis}
series = {series}
>
</EJ.Chart>,
document.getElementById('chart')
);

```



**While Category axis is Indexed value false**



### Numeric Axis

Numeric axis uses numerical scale and displays numbers as labels. To use numeric axis, you can set the [valueType](#) property of the axis to **double**.

### JAVASCRIPT

```

"use strict";
var primaryYAxis={
//Use numerical scale in primary Y axis
valueType: 'double',
};
ReactDOM.render(
<EJ.Chart id="default_chart_sample_0"
primaryYAxis= {primaryYAxis}>
</EJ.Chart>,
document.getElementById('chart')
);

```



### Customize numeric range

To customize the range of an axis, you can use the [range](#) property of the axis to set the [minimum](#), [maximum](#) and [interval](#) values. Nice range is calculated automatically based on the provided data, by default.

### JAVASCRIPT

```

"use strict";
var primaryYAxis={
//Customizing Y-axis range
range: { min: 0, max: 50 },
};
ReactDOM.render(
<EJ.Chart id="default_chart_sample_0"

```

```
primaryYAxis= {primaryYAxis}
>
</EJ.Chart>,
document.getElementById('chart')
);
```



#### Customizing numeric interval

Axis interval can be customized by using the [interval](#) property of the axis range. Nice interval is calculated based on the minimum and maximum value of the provided data, by default.

#### JAVASCRIPT

```
"use strict";
var primaryYAxis={
  //Customizing Y-axis interval
  range: { interval: 5 },
};
ReactDOM.render(
<EJ.Chart id="default_chart_sample_0"
primaryYAxis= {primaryYAxis}
>
</EJ.Chart>,
document.getElementById('chart')
);
```



#### Apply padding to the range

Padding can be applied to the minimum and maximum extremes of the axis range by using the [rangePadding](#) property. Numeric axis supports the following types of padding

- None
- Round
- Additional
- Normal

#### None

When the value of the [rangePadding](#) property is **none**, padding can not be applied to the axis. This is also the default value of the rangePadding.

#### JAVASCRIPT

```
"use strict";
var primaryYAxis={
  //Applying none as range padding
  rangePadding: 'none',
};
ReactDOM.render(
<EJ.Chart id="default_chart_sample_0"
primaryYAxis= {primaryYAxis}
>
</EJ.Chart>,
```

```
document.getElementById('chart')
);
```



### Round

When the value of [rangePadding](#) property is **round**, the axis range is rounded to the nearest possible value divided by the interval.

### JAVASCRIPT

```
"use strict";
var primaryYAxis={
  //Applying round as range padding
  rangePadding: 'round',
};
ReactDOM.render(
  <EJ.Chart id="default_chart_sample_0"
  primaryYAxis= {primaryYAxis}
  >
  </EJ.Chart>,
  document.getElementById('chart')
);
```

### Chart before rounding axis range



### Chart after rounding axis range



### Additional

When the value of the [rangePadding](#) property is **additional**, the axis range is rounded and an interval of the axis is added as padding to the minimum and maximum values of the range.

### JAVASCRIPT

```
"use strict";
var primaryYAxis={
  //Applying additional as range padding
  rangePadding: 'additional',
};
ReactDOM.render(
  <EJ.Chart id="default_chart_sample_0"
  primaryYAxis= {primaryYAxis}
  >
  </EJ.Chart>,
  document.getElementById('chart')
);
```



### Normal

When the value of the [rangePadding](#) property is **normal**, the padding is applied to the axis based on the range calculation.

### JAVASCRIPT

```
"use strict";
var primaryYAxis={
  //Applying normal as range padding
  rangePadding: 'normal',
};
ReactDOM.render(
  <EJ.Chart id="default_chart_sample_0"
  primaryYAxis= {primaryYAxis}
  >
  </EJ.Chart>,
  document.getElementById('chart')
);
```



### Customizing the starting range of the axis

By default the Y axis will be always calculated from the value 0 for column, bar, stacking column and stacking bar series types. You can modify this behavior by setting false to the property [startFromZero](#) in the axis. On setting this the axis minimum value will be calculated based on the value for the data points.

### JAVASCRIPT

```
"use strict";
//Initializing Primary Y Axis
var primaryYAxis={
  rangePadding: "none",
  startFromZero: false
};
ReactDOM.render(
  <EJ.Chart id="default_chart_sample_0"
  primaryYAxis= {primaryYAxis}
  >
  </EJ.Chart>,
  document.getElementById('chart')
);
```



### DateTime Axis

Date time axis uses date time scale and displays the date time values as axis labels in the specified format. To use date time axis, set the [valueType](#) property of the axis to **datetime**.

### JAVASCRIPT

```
"use strict";
var primaryXAxis={
  //Use date time scale in primary X axis
  valueType: 'datetime',
};
ReactDOM.render(
```

```
<EJ.Chart id="default_chart_sample_0"
primaryXAxis= {primaryXAxis}
>
</EJ.Chart>,
document.getElementById('chart')
);
```



[Click](#) here to view our online demo sample for date time axis.

#### Customizing date time range

Axis range can be customized by using the [range](#) property to set the [minimum](#), [maximum](#) and [interval](#) values. Nice range is calculated automatically based on the provided data, by default.

#### JAVASCRIPT

```
"use strict";
var primaryXAxis={
//Customizing X-axis date time range
range: {
min: new Date(2000, 6, 1),
max: new Date(2010, 6, 1)
},
};
ReactDOM.render(
<EJ.Chart id="default_chart_sample_0"
primaryXAxis= {primaryXAxis}
>
</EJ.Chart>,
document.getElementById('chart')
);
```



#### Date time intervals

Date time intervals can be customized by using the [interval](#) and [intervalType](#) properties of the axis. For example, when you set [interval](#) as **2** and [intervalType](#) as **years**, it considers the 2 years as interval.

Essential Chart supports the following types of interval for date time axis.

- Days
- Hours
- Milliseconds
- Minutes
- Months
- Seconds
- Years

#### JAVASCRIPT

```
"use strict";
var primaryXAxis={
//Customizing X-axis date time range
```



```

range: {
  interval: 2,
},
intervalType: 'years',
};
ReactDOM.render(
  <EJ.Chart id="default_chart_sample_0"
    primaryXAxis= {primaryXAxis}
  >
  </EJ.Chart>,
  document.getElementById('chart')
);

```



#### *Apply padding to the range*

Padding can be applied to the minimum and maximum extremes of the range by using the [rangePadding](#) property. Date time axis supports the following types of padding

- None
- Round
- Additional

#### **None**

When the value of the [rangePadding](#) property is **none**, padding is applied to the axis. This is also the default value of the rangePadding.

#### **JAVASCRIPT**

```

"use strict";
var primaryXAxis={
  //Applying none as range padding
  rangePadding: 'none',
  // ...
};
ReactDOM.render(
  <EJ.Chart id="default_chart_sample_0"
    primaryXAxis= {primaryXAxis}
  >
  </EJ.Chart>,
  document.getElementById('chart')
);

```



#### **Round**

When the value of the [rangePadding](#) property is **round**, the axis range is rounded to the nearest possible date time value.

#### **JAVASCRIPT**

```

"use strict";
var primaryXAxis={

```

```
//Applying round as range padding
rangePadding: 'round',
// ...
};
ReactDOM.render(
<EJ.Chart id="default_chart_sample_0"
primaryXAxis= {primaryXAxis}
>
</EJ.Chart>,
document.getElementById('chart')
);
```

### Chart before rounding axis range



### Chart after rounding axis range



### Additional

When the value of the [rangePadding](#) property is **additional**, the range is rounded and date time interval of the axis are added as padding to the minimum and maximum extremes of the range.

### JAVASCRIPT

```
"use strict";
var primaryXAxis={
//Applying additional as range padding
rangePadding: 'additional',
// ...
};
ReactDOM.render(
<EJ.Chart id="default_chart_sample_0"
primaryXAxis= {primaryXAxis}
>
</EJ.Chart>,
document.getElementById('chart')
);
```



### DateTime Category Axis

DateTime category axis takes date time value as input but behaves like category axis. This is used to display the date time values with nonlinear intervals (used to depict the business days by skipping holidays). To use date time axis, set the `[valueType]` (../api/ejchart#members:primaryxaxis-valuetype) property of the axis to **datetimeCategory**.

### JAVASCRIPT

```
"use strict";
var primaryXAxis={
valueType: 'datetimeCategory',
// ...
};
ReactDOM.render(
```

```
<EJ.Chart id="default_chart_sample_0"
primaryXAxis= {primaryXAxis}
>
</EJ.Chart>,
document.getElementById('chart')
);
```



[Click](#) here to view our online demo sample for date time axis.

#### *Customizing DateTime Category range*

Axis range can be customized by using the [range](#) property to set the [minimum](#), [maximum](#) and [interval](#) values. Datetime category axis takes numeric input for minimum and maximum property.

#### **JAVASCRIPT**

```
"use strict";
var primaryXAxis={
range: { //Customizing X-axis date time category range
min: 0,
max: 4
},
// ...
};
ReactDOM.render(
<EJ.Chart id="default_chart_sample_0"
primaryXAxis= {primaryXAxis}
>
</EJ.Chart>,
document.getElementById('chart')
);
```



#### *DateTime Category intervals*

Date time category intervals can be customized by using the [interval](#) and [intervalType](#) properties of the axis. For example, when you set the intervalType as months, it displays only the first label of all the months from the data.

Essential Chart supports the following types of interval for date time category axis.

- Days
- Hours
- Milliseconds
- Minutes
- Months
- Seconds
- Years
- Auto

#### **JAVASCRIPT**

```
"use strict";
```

```

var primaryXAxis={
  intervalType: "months"
  // ...
};
ReactDOM.render(
  <EJ.Chart id="default_chart_sample_0"
  primaryXAxis= {primaryXAxis}
  >
</EJ.Chart>,
  document.getElementById('chart')
);

```



### Logarithmic Axis

Logarithmic axis uses logarithmic scale and it is very useful in visualizing when the data has values with both lower order of magnitude (eg:  $10^{-6}$ ) and higher order of magnitude (eg:  $10^6$ ). To use logarithmic axis, set the [valueType](#) property of the axis to **logarithmic**.

#### JAVASCRIPT

```

"use strict";
var primaryXAxis={
  //Use logarithmic scale in primary X axis
  valueType: 'logarithmic',
  // ...
};
ReactDOM.render(
  <EJ.Chart id="default_chart_sample_0"
  primaryXAxis= {primaryXAxis}
  >
</EJ.Chart>,
  document.getElementById('chart')
);

```



[Click](#) here to view our online demo sample link for logarithmic axis.

### Customize Logarithmic range

Logarithmic range can be customized by using the [range](#) property of the axis to change the [minimum](#), [maximum](#) and [interval](#) values. Nice range is calculated automatically based on the provided data, by default.

#### JAVASCRIPT

```

"use strict";
var primaryXAxis={
  //Customizing logarithmic range
  range: { min: 1, max: 5 },
  // ...
};
ReactDOM.render(
  <EJ.Chart id="default_chart_sample_0"
  primaryXAxis= {primaryXAxis}
  >

```

```
</EJ.Chart>,
document.getElementById('chart')
);
```



#### Logarithmic base

Logarithmic base can be customized by using the [logBase](#) property of the axis. The default value of the [logBase](#) is 10.

#### JAVASCRIPT

```
"use strict";
var primaryXAxis={
  //Customizing logarithmic base
  logBase: 2,
  // ...
};
ReactDOM.render(
  <EJ.Chart id="default_chart_sample_0"
  primaryXAxis= {primaryXAxis}
  >
  </EJ.Chart>,
  document.getElementById('chart')
);
```



#### Logarithmic interval

Logarithmic axis interval can be customized by using the [interval](#) property of the axis. When the logarithmic base is 10 and logarithmic interval is 2, then the axis labels are placed at an interval of  $10^{\sup>2</sup>}$ . The default value of the interval is 1.

#### JAVASCRIPT

```
"use strict";
var primaryXAxis={
  //Customizing logarithmic interval
  range: { interval: 2 },
  // ...
};
ReactDOM.render(
  <EJ.Chart id="default_chart_sample_0"
  primaryXAxis= {primaryXAxis}
  >
  </EJ.Chart>,
  document.getElementById('chart')
);
```



## Label Format

### Format numeric labels

Numeric labels can be formatted by using the [labelFormat](#) property. Numeric values can be formatted with n (number with decimal points), c (currency) and p (percentage) commands.

### JAVASCRIPT

```
"use strict";
var primaryXAxis={
  //Applying currency format to axis labels
  labelFormat: 'c',
  // ...
};
ReactDOM.render(
  <EJ.Chart id="default_chart_sample_0"
    primaryXAxis= {primaryXAxis}
  >
  </EJ.Chart>,
  document.getElementById('chart')
);
```



The following table describes the result of applying some commonly used label formats on numeric values.

Label Value	Label Format property value	Result	Description
1000	n1	1000.0	The Number is rounded to 1 decimal place
1000	n2	1000.00	The Number is rounded to 2 decimal place
1000	n3	1000.000	The Number is rounded to 3 decimal place
0.01	p1	1.0%	The Number is converted to percentage with 1 decimal place
0.01	p2	1.00%	The Number is converted to percentage with 2 decimal place
0.01	p3	1.000%	The Number is converted to percentage with 3 decimal place
1000	c1	\$1,000.0	The Currency symbol is appended to number and number is rounded to 1 decimal place
1000	c2	\$1,000.00	The Currency symbol is appended to number and number is rounded to 2 decimal place

### Format date time values

Date time labels can be formatted by using the [labelFormat](#) property of the axis.

### JAVASCRIPT

```

"use strict";
var primaryXAxis={
  //Formatting date time labels in date/Month name/Year format
  labelFormat: 'dd/MMMM/yyyy',
  // ...
};
ReactDOM.render(
  <EJ.Chart id="default_chart_sample_0"
  primaryXAxis= {primaryXAxis}
  >
  </EJ.Chart>,
  document.getElementById('chart')
);

```



The following table describes the result of applying some common date time formats to the labelFormat property

Label Value	Label Format Property Value	Result	Description
new Date(2000, 03, 10)	dddd	Monday	The Date is displayed in day format
new Date(2000, 03, 10)	MM/dd/yyyy	04/10/2000	The Date is displayed in month/date/year format
new Date(2000, 03, 10)	n3	1000.000	The Number is rounded to 3 decimal place
new Date(2000, 03, 10)	MMM	Apr	The Shorthand month for the date is displayed
new Date(2000, 03, 10)	t	12:00 AM	Time of the date value is displayed as label
new Date(2000, 03, 10)	hh:mm:ss	12:00:00	The Label is displayed in hours:minutes:seconds format

#### Custom label format

Prefix and suffix can be added to the category labels by using the [labelFormat](#) property. You can use the `{value}` as placeholder text in your custom text, it is replaced with the corresponding axis label at the runtime.

#### JAVASCRIPT

```

"use strict";
var primaryXAxis={
  //...
  //Adding prefix and suffix to axis labels
  labelFormat: '${value} K',
};
ReactDOM.render(
  <EJ.Chart id="default_chart_sample_0"
  primaryXAxis= {primaryXAxis}

```

```
>
</EJ.Chart>,
document.getElementById('chart')
);
```



### Common axis features

Customization of features such as axis crossing, title, labels, grid lines and tick lines are common to all the axis. Each of these features are explained in this section.

#### Axis Crossing

Axis can be positioned anywhere in chart area using the [crossesAt](#) property of axis. This property specifies where the horizontal axis should intersect or cross the vertical axis and vice versa. Default value of [crossesAt](#) property is null.

#### JAVASCRIPT

```
"use strict";
var primaryXAxis={
  //Crosses primary Y axis at 0
  crossesAt: 0,
  //...
};
ReactDOM.render(
  <EJ.Chart id="default_chart_sample_0"
  primaryXAxis= {primaryXAxis}
  >
  </EJ.Chart>,
  document.getElementById('chart')
);
```



#### Crossing a specific Axis

The [crossesInAxis](#) property takes axis name as input and determines the axis used for crossing. By default all the horizontal axes crosses in primary Y axis and all the vertical axes crosses in primary X axis.

#### JAVASCRIPT

```
"use strict";
var primaryXAxis={
  //Crosses vertical axis at -0.2
  crossesAt: -0.2,
  //Crosses in secondary Y axis
  crossesInAxis: 'secondaryYAxis',
  //...
};
var axes=[{
  orientation: 'vertical',
  name: 'secondaryYAxis',
  //...
}];
ReactDOM.render(
  <EJ.Chart id="default_chart_sample_0"
  primaryXAxis= {primaryXAxis}
  axes= {axes}
  >
  </EJ.Chart>,
  document.getElementById('chart')
);
```



```
axes={axes}
>
</EJ.Chart>,
document.getElementById('chart')
);
```



Axis will be placed in the opposite side if value of [crossesAt](#) property is greater than the maximum value of crossing axis (axis name provided through [crossesInAxis](#) property or primary Y axis for horizontal axis).

#### JAVASCRIPT

```
"use strict";
var primaryXAxis={
  //Crosses primary Y axis at 200
  crossesAt: 200,
  //...
};
ReactDOM.render(
  <EJ.Chart id="default_chart_sample_0"
  primaryXAxis= {primaryXAxis}
  >
  </EJ.Chart>,
  document.getElementById('chart')
);
```



#### Crossing in DateTime Axis

For crossing in a date time horizontal axis, date object should be provided as value for [crossesAt](#) property of vertical axes.

#### JAVASCRIPT

```
"use strict";
var primaryYAxis={
  //Crosses horizontal axis at 5/29/2010
  crossesAt: new Date(2010, 4, 29),
  //...
};
ReactDOM.render(
  <EJ.Chart id="default_chart_sample_0"
  primaryYAxis= {primaryYAxis}
  >
  </EJ.Chart>,
  document.getElementById('chart')
);
```



#### Crossing in Category Axis

For crossing in a category type horizontal axis, either a string object or a number corresponding to the index of category value can be used for [crossesAt](#) property of vertical axes.

**Warning:** String value provided for [crossesAt](#) property is case-sensitive.

#### JAVASCRIPT

```
"use strict";
var primaryYAxis={
  //Crosses horizontal axis at category value 'Third'
  crossesAt: 'Tuesday',
  //...
};
ReactDOM.render(
  <EJ.Chart id="default_chart_sample_0"
  primaryYAxis= {primaryYAxis}
  >
</EJ.Chart>,
  document.getElementById('chart')
);
```



#### Positioning the axis elements while crossing

The [showNextToAxisLine](#) property is used for controlling the axis elements movement along with the axis line while axis crossing is performed. When the showNextToAxisLine is set as false only the axis line and the tick lines are placed at the crossing Value and the axis elements will be placed outside the chart area. The default value of [showNextToAxisLine](#) is **true**.

#### JAVASCRIPT

```
var primaryXAxis = {
  //Crosses primary Y axis at 0
  crossesAt: 0,
  showNextToAxisLine: false,
  //...
}
ReactDOM.render(
  <EJ.Chart id="default_chart_sample_0"
  width="100%"
  primaryXAxis={primaryXAxis}>
</EJ.Chart>,
  document.getElementById('chart')
);
```

The axis is placed at the crossing value without the axis elements



#### Axis Visibility

Axis visibility can be controlled by using the [visible](#) property of the axis. The default value of the [visible](#) property is **true**.

#### JAVASCRIPT

```
"use strict";
var primaryYAxis={
  //Disabling visibility of Y-axis
  visible: false
};
```

```
// ...
};
ReactDOM.render(
  <EJ.Chart id="default_chart_sample_0"
    primaryYAxis= {primaryYAxis}
  >
</EJ.Chart>,
  document.getElementById('chart')
);
```



#### Axis title

The [title](#) property in the axis provides options to customize the text and font of the axis title. Axis does not display the title, by default. Title text can also be trimmed based on the title text length or specified length.

#### JAVASCRIPT

```
"use strict";
var primaryXAxis = {
  //Customizing axis title
  title : {
    text : 'Month',
    font : {
      fontFamily : 'Segoe UI',
      size : '16px',
      fontWeight : 'bold' ,
      color : 'grey',
    },
    enableTrim : true,
    maximumTitleWidth : 80
  }
  // ...
}
ReactDOM.render(
  <EJ.Chart id="default_chart_sample_0"
    primaryXAxis= {primaryXAxis}
  >
</EJ.Chart>,
  document.getElementById('chart')
);
```



You can modify the position of the axis title either inside or outside the chart area using the property `[position]`. By default, it will be placed outside the chart area. In addition, you can also change the alignment of the title to near, far and center by `[alignment]` property, using `[offset]` property you can change the position with respect to pixels.

#### JAVASCRIPT

```
"use strict";
var primaryXAxis = {
  //Customizing axis title
```

```

title : {
  text : 'Month',
  position: 'inside',
  alignment: 'near',
  offset: 10
}
// ...
}
ReactDOM.render(
  <EJ.Chart id="default_chart_sample_0"
    primaryXAxis= {primaryXAxis}
  >
  </EJ.Chart>,
  document.getElementById('chart')
);

```



#### Label customization

The [font](#) property of the axis provides options to customize the [font-family](#), [color](#), [opacity](#), [size](#) and [font-weight](#) of the axis labels.

#### JAVASCRIPT

```

"use strict";
var primaryXAxis = {
  //Customizing label appearance
  font : {
    fontFamily : 'Segoe UI',
    size : '14px',
    fontWeight : 'bold' ,
    color : 'blue',
  },
  // ...
}
ReactDOM.render(
  <EJ.Chart id="default_chart_sample_0"
    primaryXAxis= {primaryXAxis}
  >
  </EJ.Chart>,
  document.getElementById('chart')
);

```



#### Label and tick positioning

Axis labels and ticks can be positioned inside or outside the chart area by using the [labelPosition](#) and [tickPosition](#) properties. The labels and ticks are positioned outside the chart area, by default.

#### JAVASCRIPT

```

"use strict";
var primaryXAxis = {
  //Customizing label and tick positions
  labelPosition : 'inside',
  tickLinesPosition : 'inside',

```

```
// ...
}
ReactDOM.render(
  <EJ.Chart id="default_chart_sample_0"
    primaryXAxis= {primaryXAxis}
  >
</EJ.Chart>,
  document.getElementById('chart')
);
```



#### Edge labels placement

Labels with long text at the edges of an axis may appear partially outside the chart. The [edgeLabelPlacement](#) property can be used to avoid the partial appearance of the labels at the corners.

#### JAVASCRIPT

```
"use strict";
var primaryXAxis = {
  //Customizing edge label placement
  edgeLabelPlacement : 'shift',
  // ...
}
ReactDOM.render(
  <EJ.Chart id="default_chart_sample_0"
    primaryXAxis= {primaryXAxis}
  >
</EJ.Chart>,
  document.getElementById('chart')
);
```

#### Chart before setting edge label placement to X-axis



#### Chart after setting edge label placement to X-axis



#### Grid lines customization

The [majorGridLines](#) and [minorGridLines](#) properties in the axis are used to customize the major grid lines and minor grid lines of an axis. They provide options to change the width, color, visibility and opacity of the grid lines. The minor grid lines are not visible, by default.

#### JAVASCRIPT

```
"use strict";
var primaryXAxis = {
  //Customizing grid lines
  majorGridLines : {
    color : 'blue',
    visible : true,
    width : 1
  },
  minorTicksPerInterval: 0,
  minorGridLines : {
```

```

color : 'red',
visible : false,
width : 1
}
}
// ...
ReactDOM.render(
<EJ.Chart id="default_chart_sample_0"
primaryXAxis= {primaryXAxis}
>
</EJ.Chart>,
document.getElementById('chart')
);

```



#### *Tick lines customization*

The [majorTickLines](#) and [minorTickLines](#) properties in the axis are used to customize the major tick lines of an axis and minor tick lines of an axis. They provide options to change the width, size, color and visibility of the grid lines. The minor tick lines are not visible, by default.

#### **JAVASCRIPT**

```

"use strict";
var primaryXAxis = {
//Customizing tick lines
majorTickLines : {
color : 'blue',
visible : true,
width : 1 ,
size : 5,
},
minorTicksPerInterval: 0,
minorTickLines : {
color : 'red',
visible : false,
width : 1 ,
size : 5
}
}
// ...
}
ReactDOM.render(
<EJ.Chart id="default_chart_sample_0"
primaryXAxis= {primaryXAxis}
>
</EJ.Chart>,
document.getElementById('chart')
);

```



#### *Inversing axis*

Axis can be inversed by using the [isInversed](#) property of the axis. The default value of the [isInversed](#) property is **false**.

**JAVASCRIPT**

```

"use strict";
var primaryXAxis = {
  //Inversing the X-axis
  isInversed: false
  // ...
}
ReactDOM.render(
  <EJ.Chart id="default_chart_sample_0"
  primaryXAxis= {primaryXAxis}
  >
  </EJ.Chart>,
  document.getElementById('chart')
);

```

**Chart before inverting the axes**



**Chart after inverting the axes**



*Place axes at the opposite side*

The [opposedPosition](#) property of axis can be used to place the axis at the opposite side of its default position. The default value of the [opposedPosition](#) property is **false**.

**JAVASCRIPT**

```

"use strict";
var primaryXAxis = {
  //Placing axis at the opposite side of its normal position
  opposedPosition: true
  // ...
}
ReactDOM.render(
  <EJ.Chart id="default_chart_sample_0"
  primaryXAxis= {primaryXAxis}
  >
  </EJ.Chart>,
  document.getElementById('chart')
);

```

**Chart with X and Y axes at normal position**



**Chart with Y-axis at opposed position**



*Maximum number of labels per 100 pixels*

A maximum of 3 labels are displayed for each 100 pixels in the axis, by default. The maximum number of labels that is present within the 100 pixels length can be customized by using the [maximumLabels](#) property of the axis. This property is applicable only for an automatic range calculation and it does not work when you set the value for [interval](#) property in the axis range.

**JAVASCRIPT**

```

"use strict";
var primaryXAxis = {
  //Maximum number of labels per 100 pixels
  maximumLabels : 1
  // ...
}
ReactDOM.render(
  <EJ.Chart id="default_chart_sample_0"
  primaryXAxis= {primaryXAxis}
  >
  </EJ.Chart>,
  document.getElementById('chart')
);

```

**Chart before setting maximum labels per 100 pixels**



**Chart after setting maximum labels one per 100 pixels**



**Multiple Axis**

Multiple axes can be used in the Chart and chart area can be split into multiple panes to draw multiple series with multiple axes.



An additional horizontal or vertical axis can be added to the chart by adding an axis instance to the **axes** collection and then you can associate it to a series by specifying the name of the axis to the [xAxisName](#) or [yAxisName](#) property of the series.

**JAVASCRIPT**

```

"use strict";
var axes = [{
  name : 'SecondaryX',
  // ...
},{
  name : 'SecondaryY',
  // ...
}]
var series = [{
  // Binding secondary X-axis with a series
  xAxisName : 'SecondaryX',
  // Binding secondary Y-axis with a series
  yAxisName : 'SecondaryY',
  // ...
},
{
  // ...
}]
ReactDOM.render(
  <EJ.Chart id="default_chart_sample_0"
  axes={axes}
  series={series}
  >
  </EJ.Chart>
);

```



```
>
</EJ.Chart>,
document.getElementById('chart')
);
```



[Click](#) here to view the multiple axis online demo sample.

### Smart Axis Labels

When the Axis labels overlap with each other based on the chart dimensions and label size, you can use the [labelIntersectAction](#) property of the axis to avoid overlapping. The default value of the [labelIntersectAction](#) is **none**. The other available values of the Label Intersect Actions are **rotate45**, **rotate90**, **trim**, **multipleRows**, **wrap**, **wrapByWord** and **hide**.

### JAVASCRIPT

```
"use strict";
var primaryXAxis = {
  // Avoid overlapping of x-axis labels
  labelIntersectAction : 'multipleRows',
  // ...
}
ReactDOM.render(
  <EJ.Chart id="default_chart_sample_0"
  primaryXAxis={primaryXAxis}
  >
  </EJ.Chart>,
  document.getElementById('chart')
);
```



[Click](#) here to view our online demo sample for smart axis labels.

The following screenshot displays the result, when the [labelIntersectAction](#) property is set as **rotate45**.



The following screenshot displays the result, when the [labelIntersectAction](#) property is set as **rotate90**.



The following screenshot displays the result, when the [labelIntersectAction](#) property is set as **wrap**.



The following screenshot displays the result, when of setting the **trim** as value to the [labelIntersectAction](#) property.



The following screenshot displays the result, when the [labelIntersectAction](#) property is set as **hide**.



The following screenshot displays the result, when the [labelIntersectAction](#) property is set as **multipleRows**.



The following screenshot displays the result, when the [labelIntersectAction](#) property is set as **wrapByWord**.



### Multi-level Labels

Axis can be customized with multiple levels of labels using the `multiLevelLabels` property. These labels are placed based on the start and end range values and we can add any number of labels to an axis.

#### JAVASCRIPT

```
"use strict";
var primaryXAxis=
{
  multiLevelLabels: [
    {
      visible: true,
      start: -0.5,
      end: 2.5,
      text: "Quater1"
    }
  ]
};
ReactDOM.render(
  <EJ.Chart id="default_chart_sample_0"
  primaryXAxis={primaryXAxis}
  >
  </EJ.Chart>,
  document.getElementById('chart')
);
```



### Customizing the multi-Level labels

The color, width and type of the border can be customized. The default border type is `Rectangle`. And the other supported border types are namely brace, curly brace, without top/bottom border and none.

#### JAVASCRIPT

```
"use strict";
var primaryXAxis = {
  multiLevelLabels: [
    {
      // customizing the border properties
      border:{
        type: "brace",
        width: 2,
        color: "black"
      }
    }
  ]
};
ReactDOM.render(
  <EJ.Chart id="default_chart_sample_0"
  primaryXAxis={primaryXAxis}
  >
```

```

</EJ.Chart>,
document.getElementById('chart')
);

```



The text of the labels can be customized using the [text] and [font] properties

### JAVASCRIPT

```

"use strict";
var primaryXAxis = {
  multiLevelLabels: [
    {
      // customizing the text and font properties
      text: "Year - 2015",
      font:{
        fontFamily: "Algerian",
        size: "12px",
        color: "black"
      }
    }
  ]
}
ReactDOM.render(
  <EJ.Chart id="default_chart_sample_0"
  primaryXAxis={primaryXAxis}
  >
  </EJ.Chart>,
  document.getElementById('chart')
);

```



You can change the alignment of the text to far, near and center position using the [textAlignment] property. By default, the text will be center aligned.

### JAVASCRIPT

```

"use strict";
var primaryXAxis = {
  multiLevelLabels: [
    {
      // customizing the text alignment
      textAlignment: "far",
    }
  ]
}
ReactDOM.render(
  <EJ.Chart id="default_chart_sample_0"
  primaryXAxis={primaryXAxis}
  >
  </EJ.Chart>,
  document.getElementById('chart')
);

```



You can trim, wrap or wrapAndTrim the text if it exceeds the maximum text width value using the property `textOverflow`

#### JAVASCRIPT

```
"use strict";
var primaryXAxis = {
  multiLevelLabels: [
    {
      // customizing the text overflow
      textOverflow: "trim",
      maximumTextWidth: 40
    }
  ]
}
ReactDOM.render(
  <EJ.Chart id="default_chart_sample_0"
  primaryXAxis={primaryXAxis}
  >
  </EJ.Chart>,
  document.getElementById('chart')
);
```

The below screenshot shows the trimmed multi-level labels



And these labels can be placed in various rows using the `level` property.

[Click](#) here to view the multi-level labels online demo sample.

#### Multiple panes

Chart area can be divided into multiple panes using the [rowDefinitions](#) and [columnDefinitions](#) properties.

##### Row Definitions

To split the chart area vertically into a number of rows, use [rowDefinitions](#) of the chart.

- You can allocate space for each row by using the [unit](#) option that determines whether the chart area should be split by *percentage* or *pixels* for the given [rowHeight](#) value of the rowDefinitions.
- To associate a vertical axis to a row, specify the rowDefinitions **index** value to the [rowIndex](#) property of the chart axis.
- To customize each row's horizontal line, use [lineColor](#) and [lineWidth](#) property.

#### JAVASCRIPT

```
"use strict";
var rowDefinitions=[{
  // Split first row of the chart area
  unit : 'percentage',
  lineColor : 'Gray',
  rowHeight : 50,
  linewidth : 0
}, {
  // Split second row of the chart area
  unit : 'percentage',
```

```

lineColor : 'green',
rowHeight : 50,
linewidth : 0
}]
var axes=[{
  //Create secondary axis and bind it to second row of chart area
  name: "yAxis1",
  rowIndex: 1
}]
var series=[{
  //Binding vertical axis name
  yAxisName: "yAxis1",
  // ...
}]
ReactDOM.render(
  <EJ.Chart id="default_chart_sample_0"
  rowDefinitions={rowDefinitions}
  axes={axes}
  series={series}
  >
  </EJ.Chart>,
  document.getElementById('chart')
);

```



[Click](#) here to view the online demo sample for multiple panes.

### Row Span

For spanning the vertical axis along multiple panes vertically, you can use [rowSpan](#) property of axis.

### JAVASCRIPT

```

"use strict";
var rowDefinitions=[{
  // ...
},{
  // ...
}]
var axes=[{
  // ...
}]
var primaryYAxis={
  // Span the PrimaryYAxis
  rowspan : 2,
}
var series=[{
  // ...
}]
ReactDOM.render(
  <EJ.Chart id="default_chart_sample_0"
  rowDefinitions={rowDefinitions}
  axes={axes}
  series={series}
  primaryYAxis={primaryYAxis}
  >

```

```
</EJ.Chart>,
document.getElementById('chart')
);
```



### Column Definitions

To split the chart area horizontally into a number of columns, use [columnDefinitions](#) of the chart.

- You can allocate space for each column by using the [unit](#) option that determines whether the chart area should be split by *percentage* or *pixels* for the given [columnWidth](#) value of the columnDefinitions.
- To associate a horizontal axis to a column, specify the columnDefinitions **index** value to the [columnIndex](#) property of the chart axis.

### JAVASCRIPT

```
"use strict";
var columnDefinitions= [{
// Split first column of the chart area
unit : 'percentage',
columnWidth : 50,
}, {
// Split second column of the chart area
unit : 'percentage',
columnWidth : 50,
}]
var axes=[{
//Create secondary axis and bind it to second column of chart area
name: "xAxis1",
columnIndex: 1
// ...
}]
var series=[{
//Binding horizontal axis name
xAxisName: "xAxis1",
// ...
}]
ReactDOM.render(
<EJ.Chart id="default_chart_sample_0"
columnDefinitions={columnDefinitions}
axes={axes}
series={series}
>
</EJ.Chart>,
document.getElementById('chart')
);
```



### Column Span

For spanning the horizontal axis along multiple panes horizontally, you can use [columnSpan](#) property of axis.

**JAVASCRIPT**

```

"use strict";
var columnDefinitions= [{
  // ...
}, {
  // ...
}]
var axes=[{
  // ...
}]
var series=[{
  // ...
}]
var primaryXAxis= {
  // Span the PrimaryXAxis
  columnSpan : 2,
}
ReactDOM.render(
<EJ.Chart id="default_chart_sample_0"
  primaryXAxis={primaryXAxis}
  columnDefinitions={columnDefinitions}
  axes={axes}
  series={series}
>
</EJ.Chart>,
document.getElementById('chart')
);

```



**ChartTypes****Line Chart**

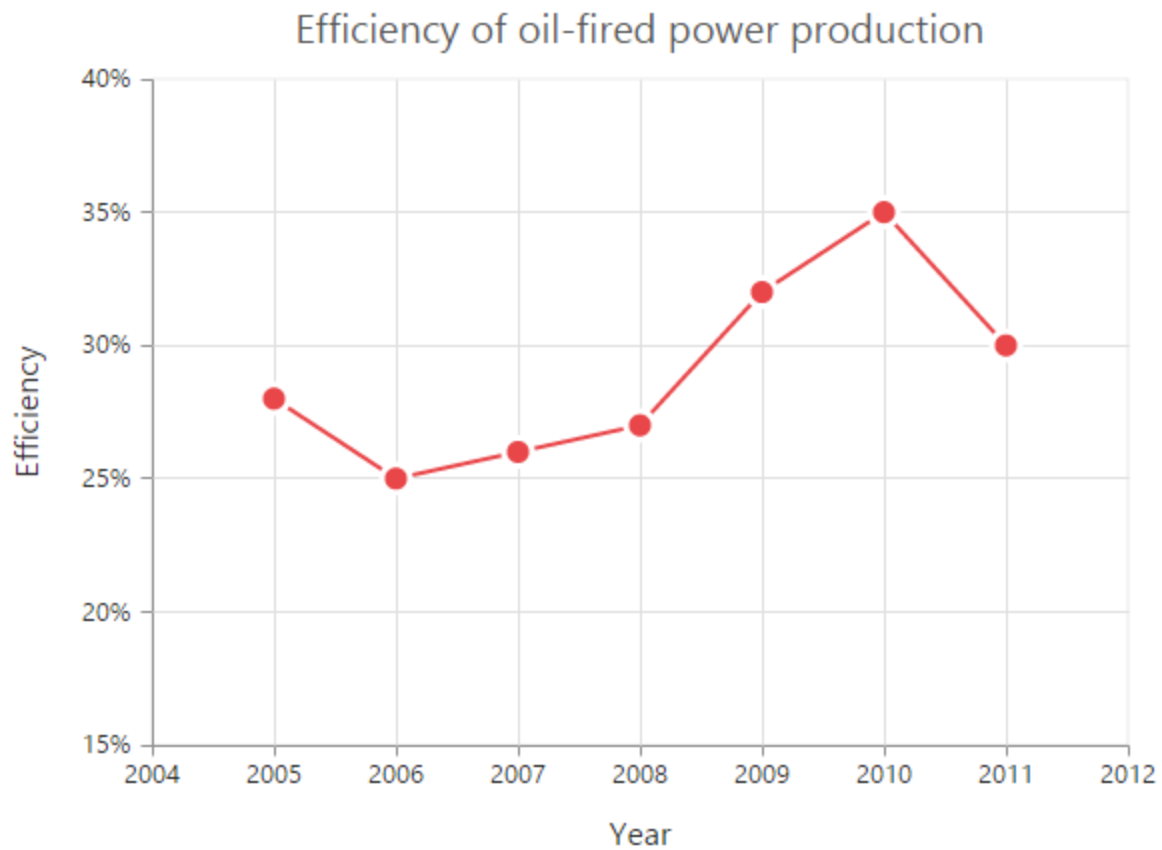
To render a Line Chart, set the series [type](#) as “line” in the chart series. To change the line segment color, you can use the [fill](#) property of the series.

**JAVASCRIPT**

```

"use strict";
// ...
var series= [{
  //Change type and color of the series.
  type: 'line',
  fill: "#E94649",
  // ...
}];
// ...
ReactDOM.render(
<EJ.Chart id="default_chart_sample_0"
  series={series}
>
</EJ.Chart>,
document.getElementById('chart')
);

```



[Click](#) here to view the Line Chart online demo sample.

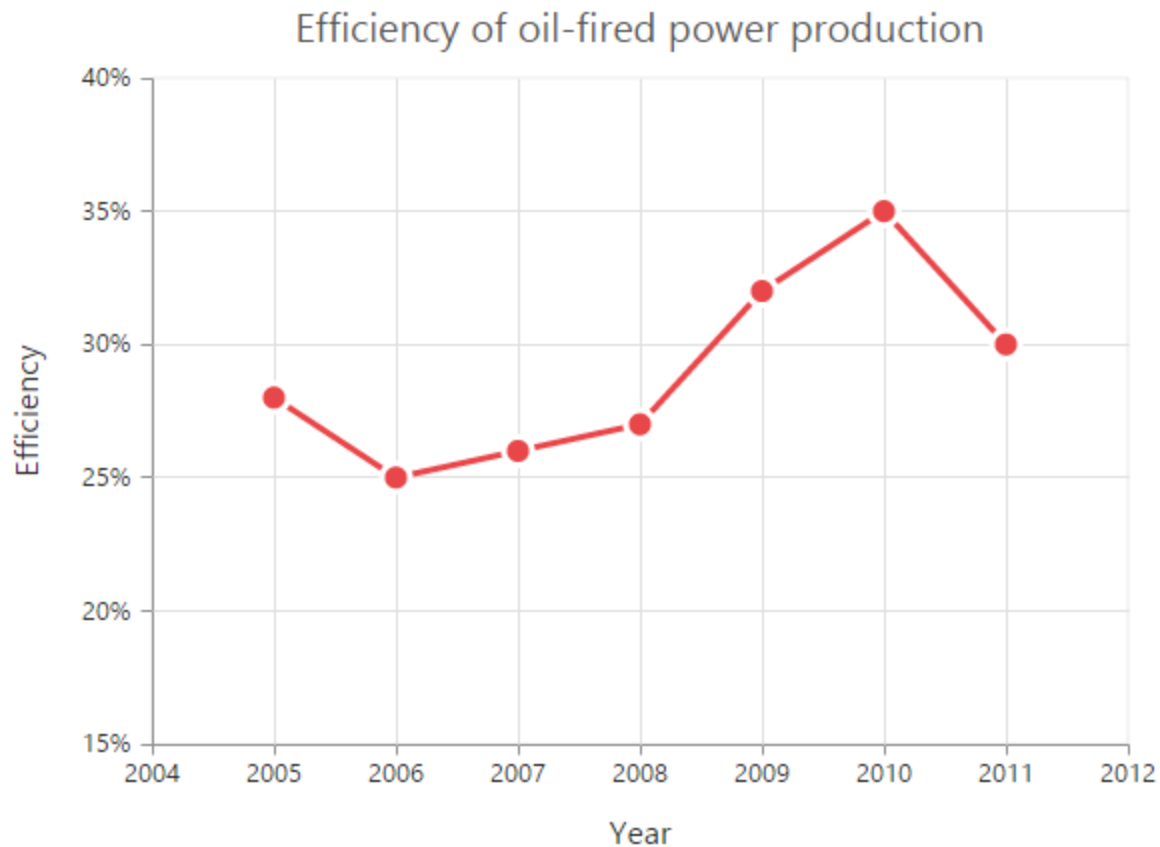
[Change the line width](#)

To change the width of the line segment, you can use the [width](#) property in the series.

#### JAVASCRIPT

```
"use strict";  
// ...  
var series= [{  
  //Change the width of line series  
  width: 3,  
  // ...  
}];  
// ...  
ReactDOM.render(  
  <EJ.Chart id="default_chart_sample_0"  
    series={series}  
  >  
    </EJ.Chart>,  
  document.getElementById('chart')  
)
```



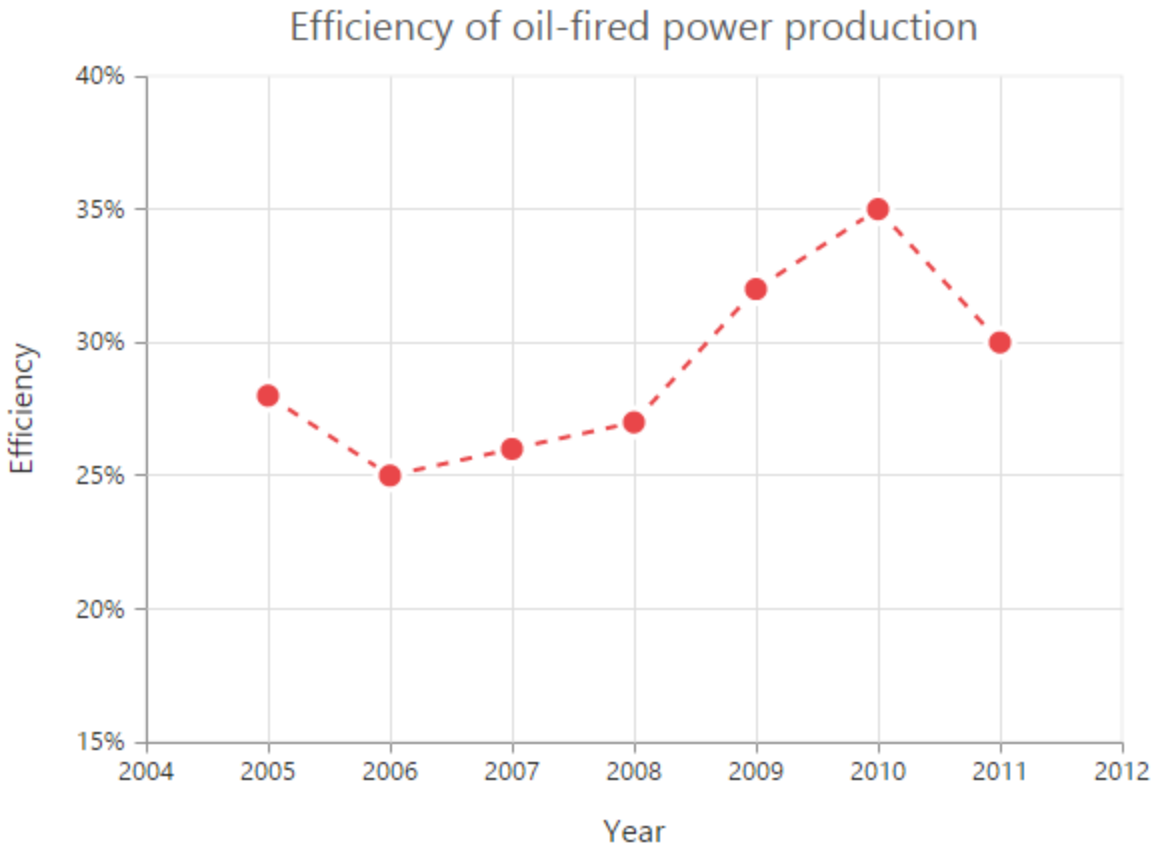


#### Dashed lines

To render the line series with dotted lines, you can use the [dashArray](#) option of the series.

#### JAVASCRIPT

```
"use strict";
// ...
var series= [{
  //Change dash array to display dotted or dashed lines
  dashArray: '5,5',
  // ...
}];
// ...
ReactDOM.render(
  <EJ.Chart id="default_chart_sample_0"
    series={series}
  >
  </EJ.Chart>,
  document.getElementById('chart')
);
```

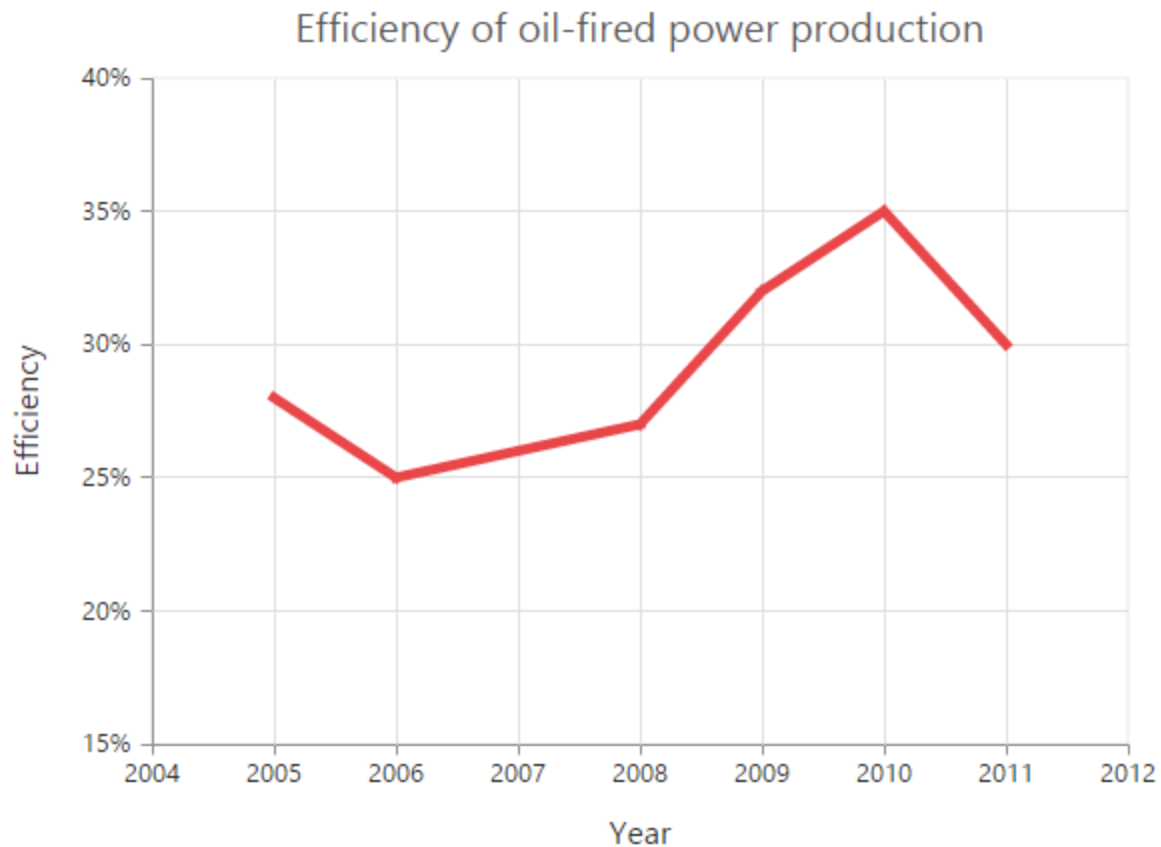


#### Changing the line cap

For customizing the start and end caps of the line segment, you can use the [lineCap](#) property.

#### JAVASCRIPT

```
"use strict";
// ...
var series= [{
  //Change line cap
  lineCap: 'square',
  // ...
}];
// ...
ReactDOM.render(
  <EJ.Chart id="default_chart_sample_0"
  series={series}
  >
  </EJ.Chart>,
  document.getElementById('chart')
);
```

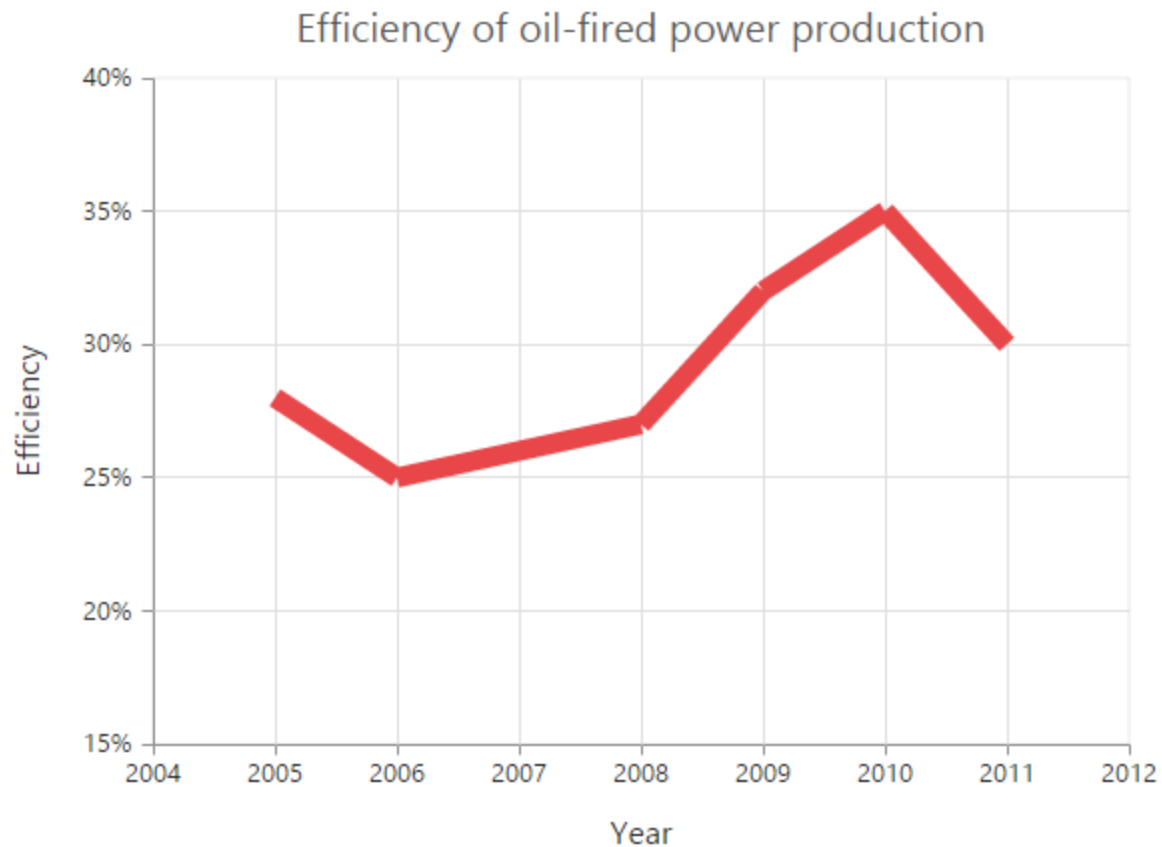


#### Changing the line join

You can use the [lineJoin](#) property to specify how two intersecting line segments should be joined.

#### JAVASCRIPT

```
"use strict";
// ...
var series= [{
  //Change line join
  lineJoin: 'round',
  //...
}];
// ...
ReactDOM.render(
  <EJ.Chart id="default_chart_sample_0"
    series={series}
  >
  </EJ.Chart>,
  document.getElementById('chart')
);
```

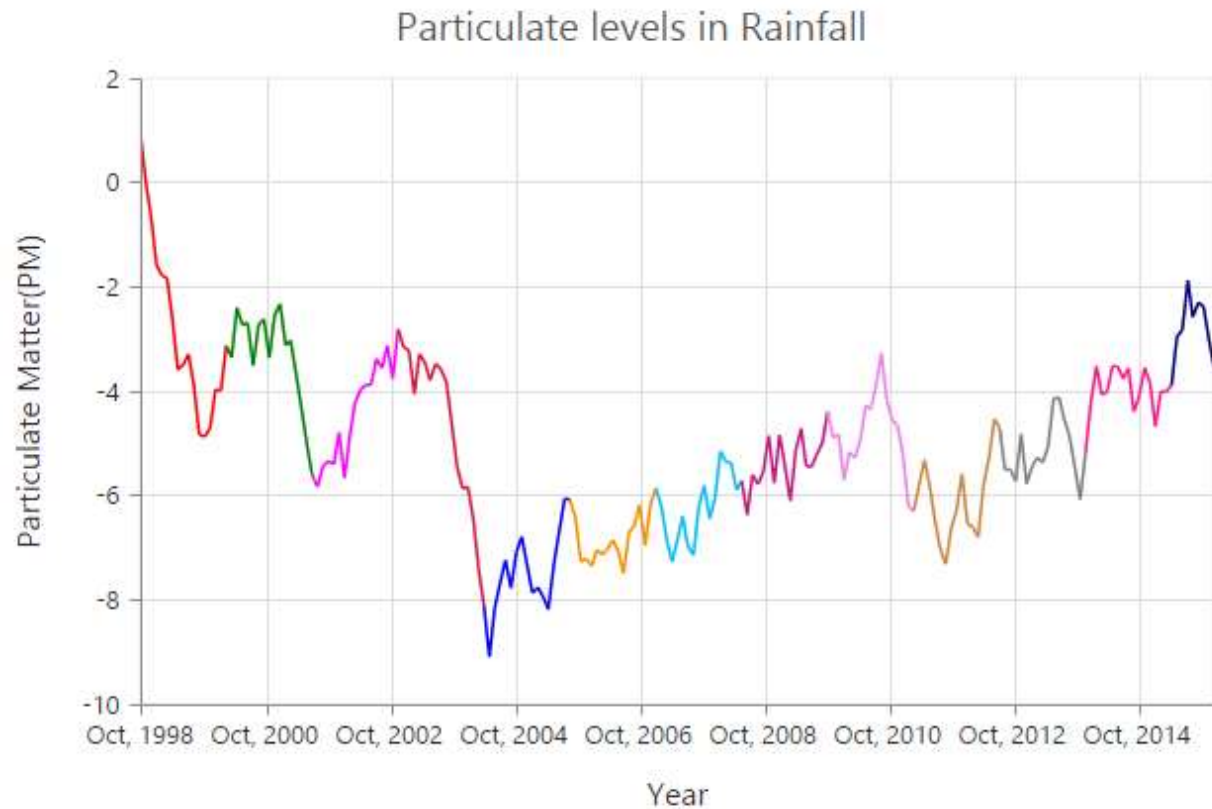


#### MultiColor Line

You can change the color of the line segments by using the [fill](#) property of the each [points](#) in the series.

#### JAVASCRIPT

```
"use strict";
// ...
var series= [{
  // Change the color of a line
  points:[{ fill: 'red' },
  // ...
],
  // ...
}];
// ...
ReactDOM.render(
  <EJ.Chart id="default_chart_sample_0"
  series={series}
  >
  </EJ.Chart>,
  document.getElementById('chart')
);
```



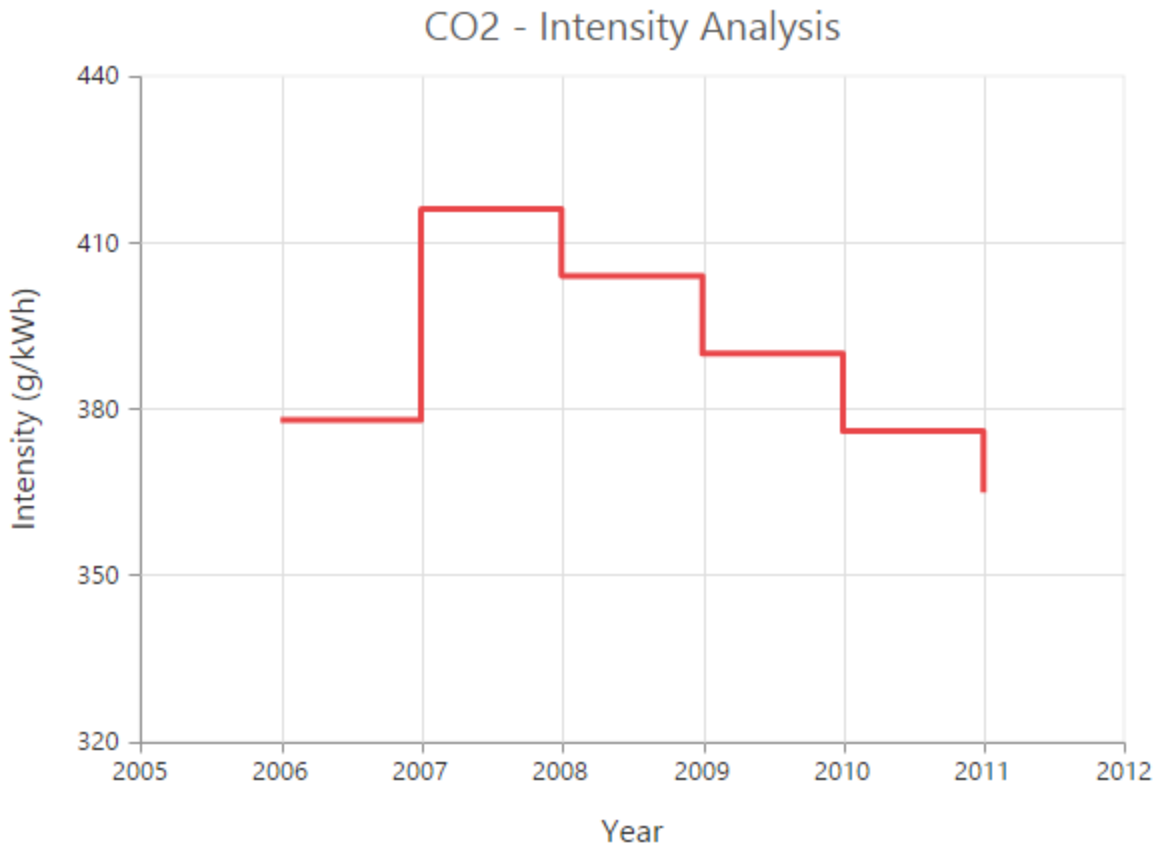
[Click](#) here to view the MultiColor Line Chart online demo sample.

### Step Line Chart

To render a Step Line Chart, set the series [type](#) as “**stepline**” in the chart series. To change the StepLine segment color, you can use the [fill](#) property of the series.

### JAVASCRIPT

```
"use strict";
// ...
var series= [{
  //Change type and color of the series.
  type: 'stepline',
  fill: "#E94649",
  // ...
}];
// ...
ReactDOM.render(
  <EJ.Chart id="default_chart_sample_0"
  series={series}
  >
  </EJ.Chart>,
  document.getElementById('chart')
);
```



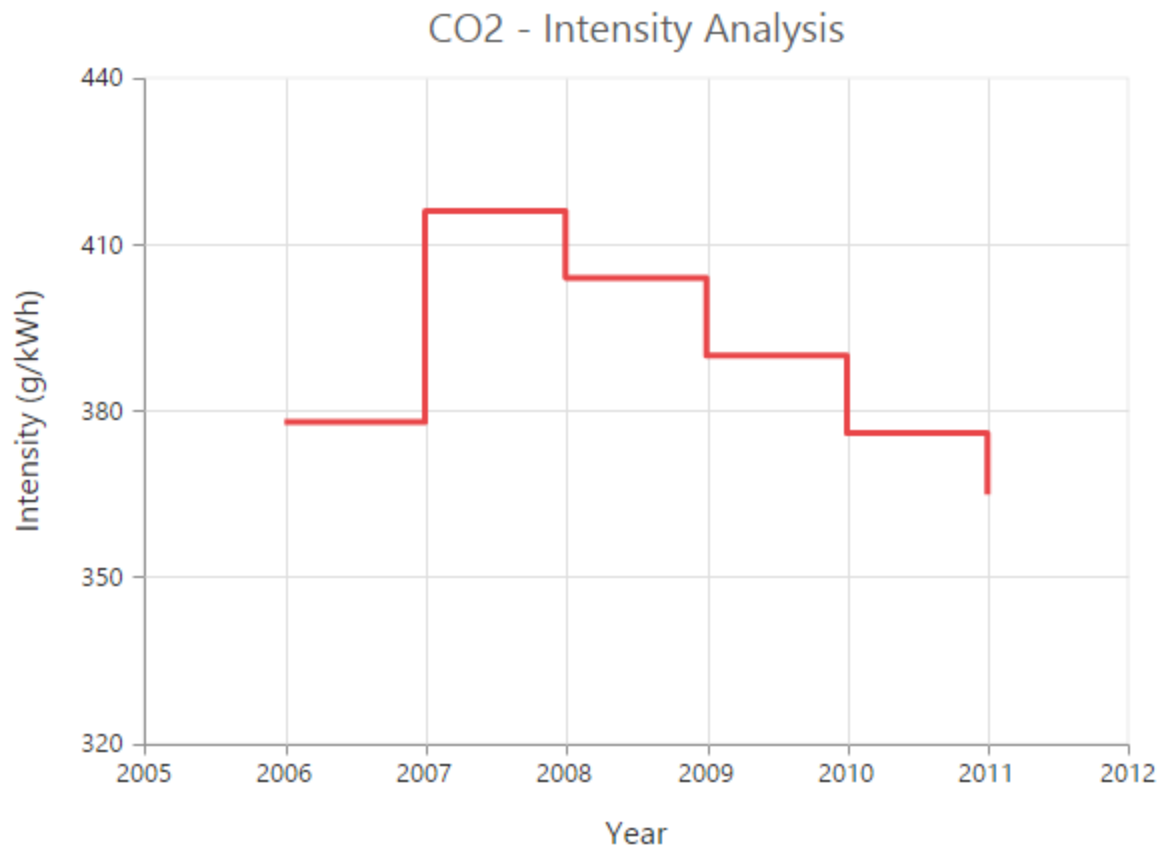
[Click](#) here to view the Step Line Chart online demo sample.

*Changing the line width*

To change the line width, you can use the **width** property.

#### JAVASCRIPT

```
"use strict";
// ...
var series= [{
  //Change the width of step line series
  width: 3,
  // ...
}];
// ...
ReactDOM.render(
  <EJ.Chart id="default_chart_sample_0"
    series={series}
  >
  </EJ.Chart>,
  document.getElementById('chart')
);
```

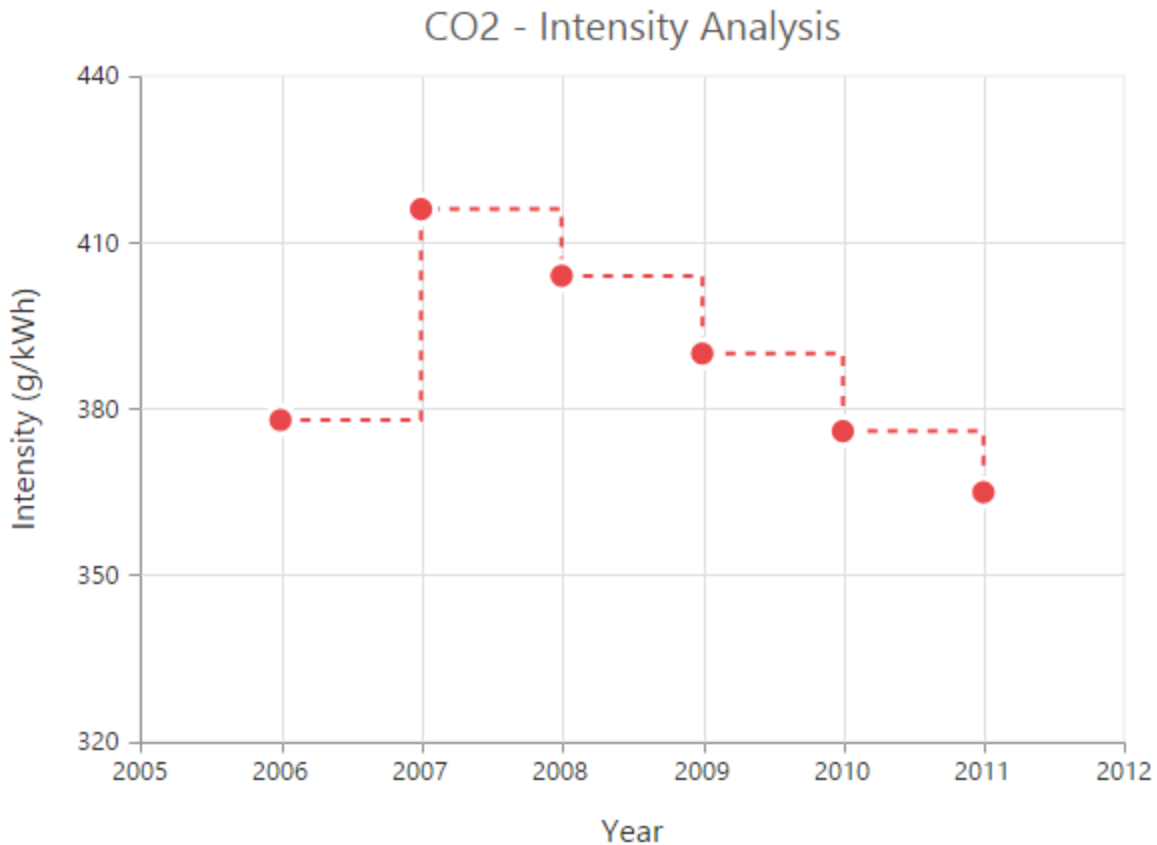


#### Dashed lines

To render the step line series with dotted lines, you can use the [dashArray](#) option of the series.

#### JAVASCRIPT

```
"use strict";
// ...
var series= [{
  //Change dash array to display dotted or dashed lines
  dashArray: '5,5',
  // ...
}];
// ...
ReactDOM.render(
  <EJ.Chart id="default_chart_sample_0"
    series={series}
  >
  </EJ.Chart>,
  document.getElementById('chart')
);
```



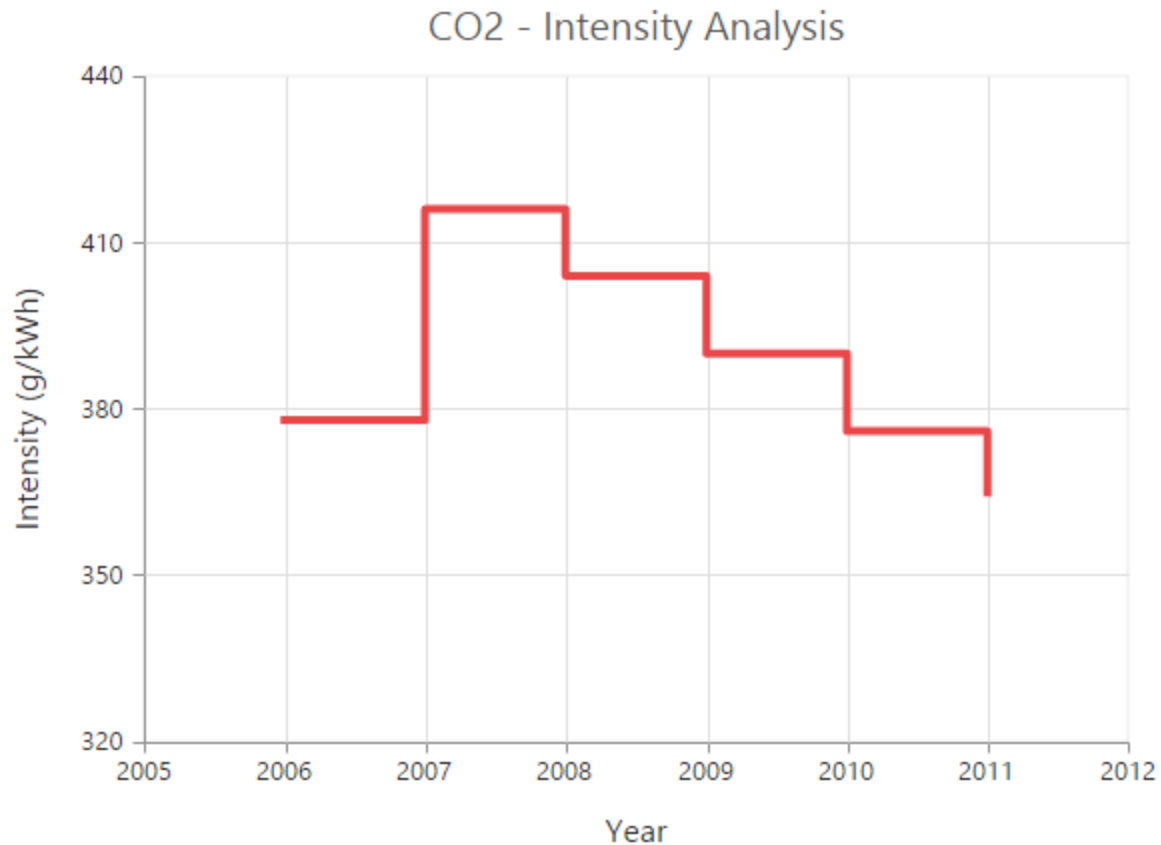
#### Changing the line cap

For customizing the start and end caps of the line segment, you can use the [lineCap](#) property.

#### JAVASCRIPT

```
"use strict";
// ...
var series= [{
  //Change line cap
  lineCap: 'square'
  // ...
}];
// ...
ReactDOM.render(
  <EJ.Chart id="default_chart_sample_0"
    series={series}
  >
  </EJ.Chart>,
  document.getElementById('chart')
);
```



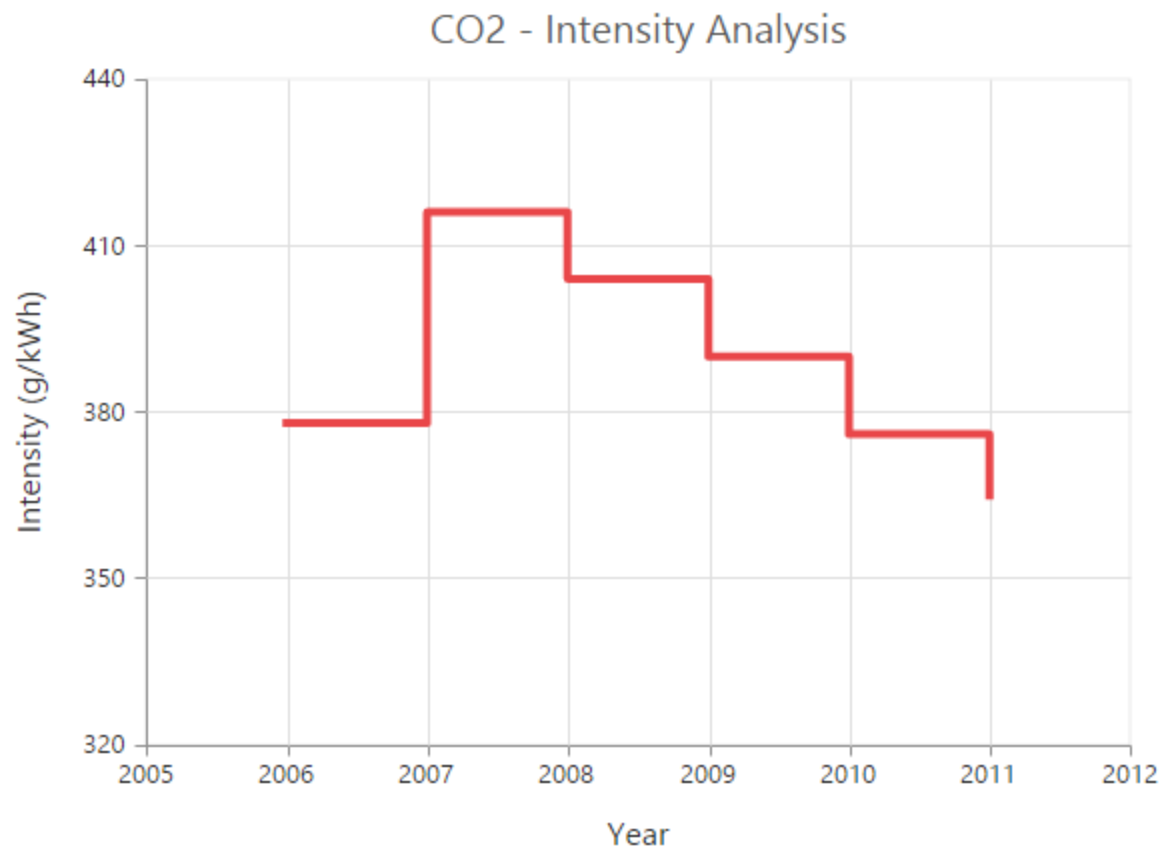


#### Changing the line join

You can use the [lineJoin](#) property to specify how two intersecting line segments should be joined.

#### JAVASCRIPT

```
"use strict";
// ...
var series= [{
  //Change line join
  lineJoin: 'round'
  // ...
}];
// ...
ReactDOM.render(
  <EJ.Chart id="default_chart_sample_0"
  series={series}
  >
  </EJ.Chart>,
  document.getElementById('chart')
);
```

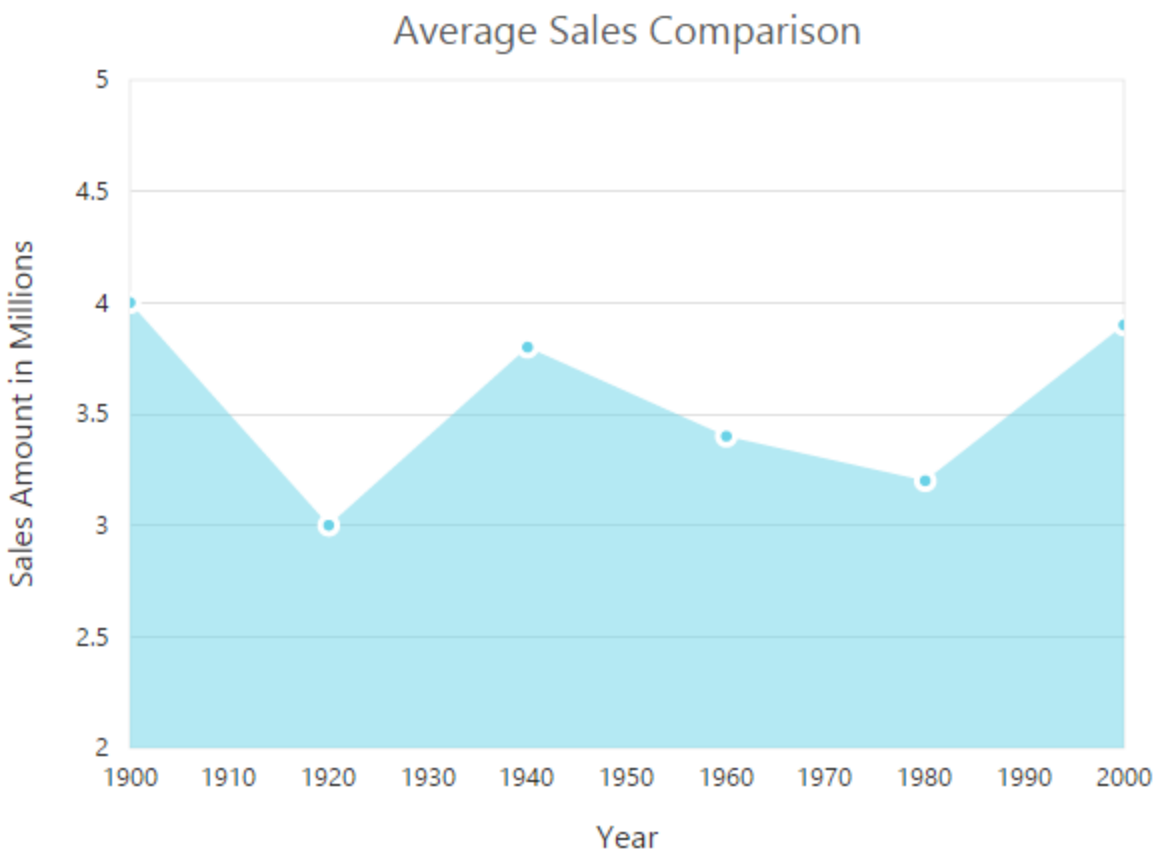


#### Area Chart

To render an Area chart, you can specify the series [type](#) as “area” in the chart series. To change the Area color, you can use the [fill](#) property of the series.

#### JAVASCRIPT

```
"use strict";
// ...
var series= [{
  // Change the series type and fill color
  type: 'area',
  fill: '#69D2E7'
  // ...
}];
// ...
ReactDOM.render(
  <EJ.Chart id="default_chart_sample_0"
  series={series}
  >
  </EJ.Chart>,
  document.getElementById('chart')
);
```



[Click](#) here to view the Area Chart online demo.

#### Range Area Chart

To render a Range Area Chart, set the [type](#) as “**rangeArea**” in the chart series. To change the RangeArea color, you can use the [fill](#) property of the series.

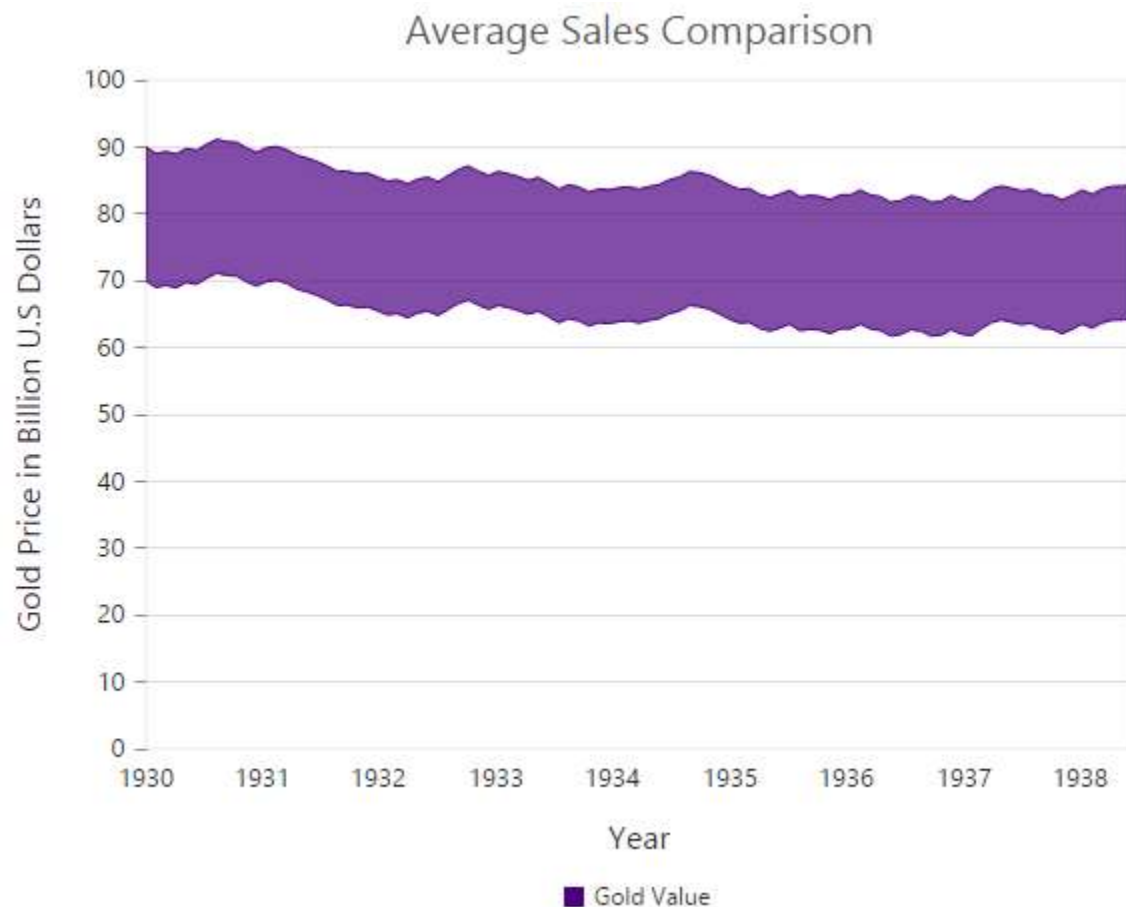
Since the RangeArea series requires two y values for a point, you have to add the [high](#) and [low](#) value. High and Low value specifies the maximum and minimum range of the points.

- When you are using the [points](#) option, specify the high and low values by using the [high](#) and [low](#) option of the point.
- When you are using the [dataSource](#) option to assign the data, map the fields from the dataSource that contain high and low values by using the [series.high](#) and [series.low](#) options.

#### JAVASCRIPT

```
"use strict";
// ...
var series= [{
  //Change the series type and fill color
  type: 'rangeArea',
  fill: "Indigo",
  //Use high and low values instead of y
  points:[{ x: 1935, high:80, low:70 },
  // ...
  ],
```

```
// ...
}];
// ...
ReactDOM.render(
  <EJ.Chart id="default_chart_sample_0"
  series={series}
  >
</EJ.Chart>,
  document.getElementById('chart')
);
```



[Click](#) here to view our Range Area Chart online demo.

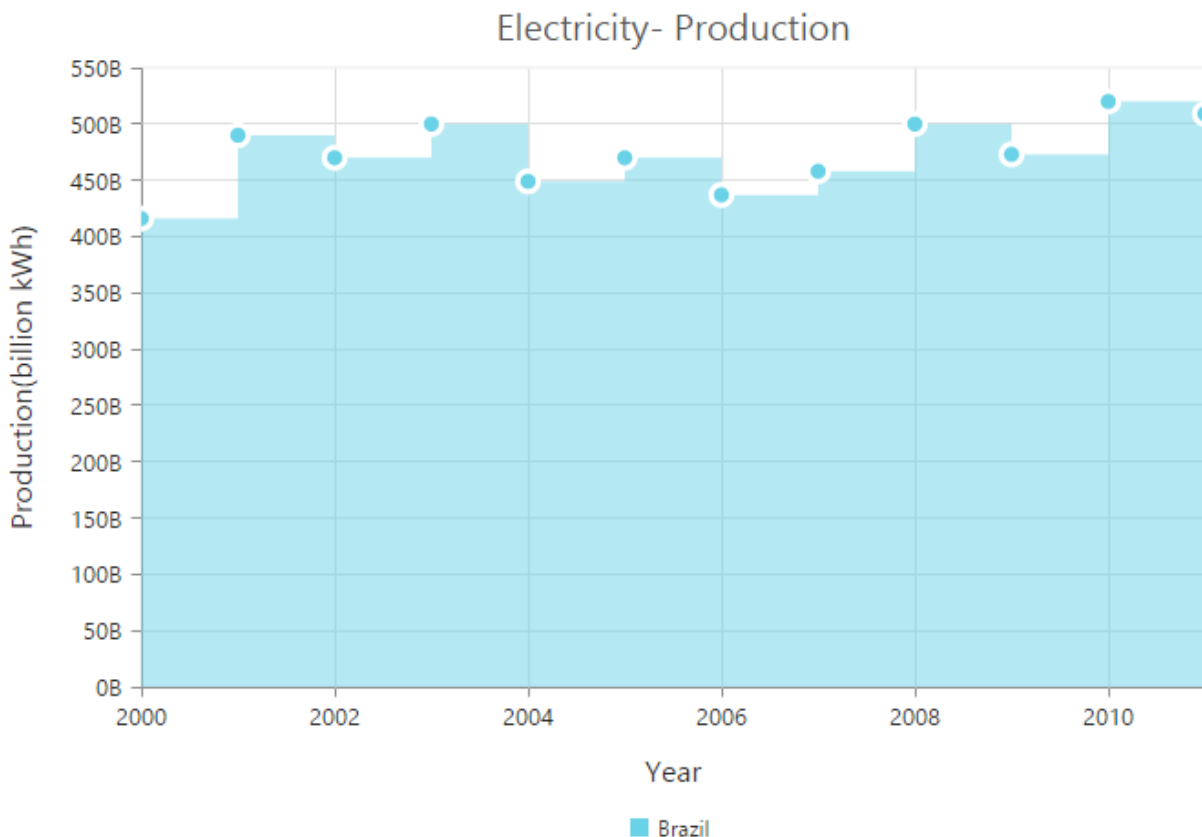
### Step Area Chart

To render a Step Area Chart, set the [type](#) as “**stepArea**” in the chart series. To change the StepArea color, you can use the [fill](#) property of the series.

### JAVASCRIPT

```
"use strict";
// ...
var series= [{
  // Change the series type and fill color
  type: 'stepArea',
  fill: " #69D2E7",
```

```
// ...
}];
// ...
ReactDOM.render(
  <EJ.Chart id="default_chart_sample_0"
    series={series}
  >
  </EJ.Chart>,
  document.getElementById('chart')
);
```



[Click](#) here to view our Step Area Chart online demo.

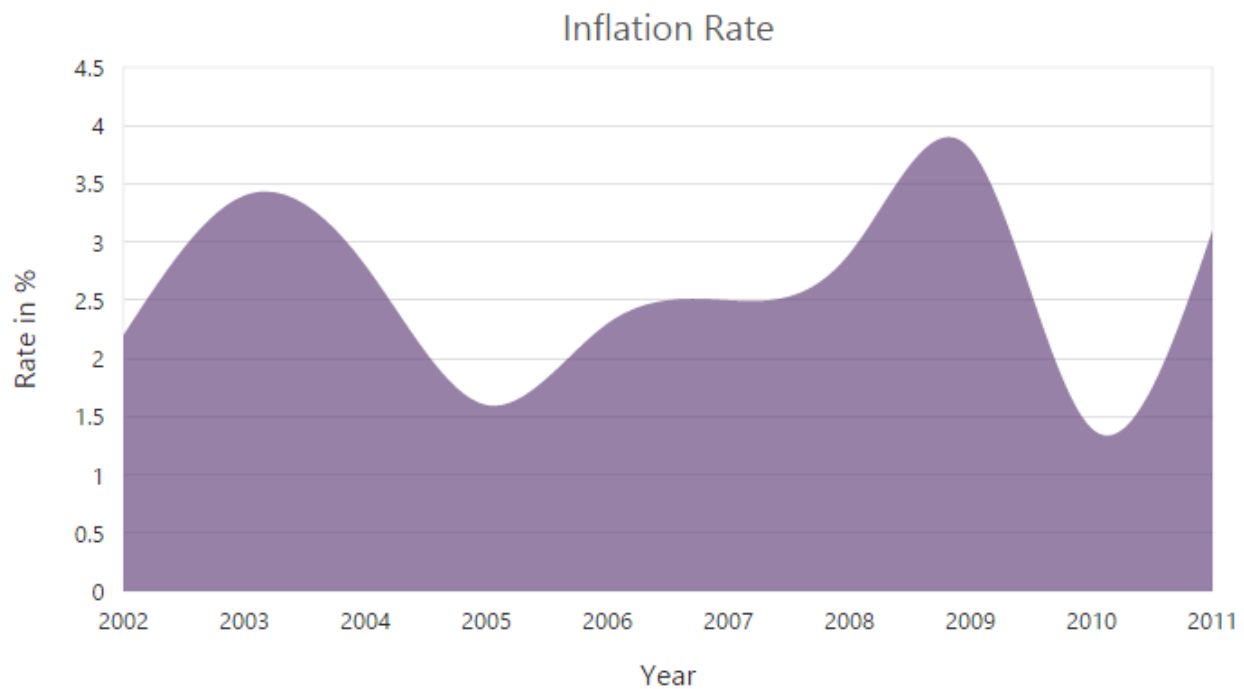
### Spline Area Chart

To render a Spline Area Chart, set the [type](#) as “**splineArea**” in the chart series. To change the SplineArea color, you can use the [fill](#) property of the series.

### JAVASCRIPT

```
"use strict";
// ...
var series= [{
  // Change the series type and fill color
  type: 'splineArea',
  fill: "#C4C24A",
  // ...
}];
```

```
// ...
ReactDOM.render(
  <EJ.Chart id="default_chart_sample_0"
    series={series}
  >
  </EJ.Chart>,
  document.getElementById('chart')
);
```



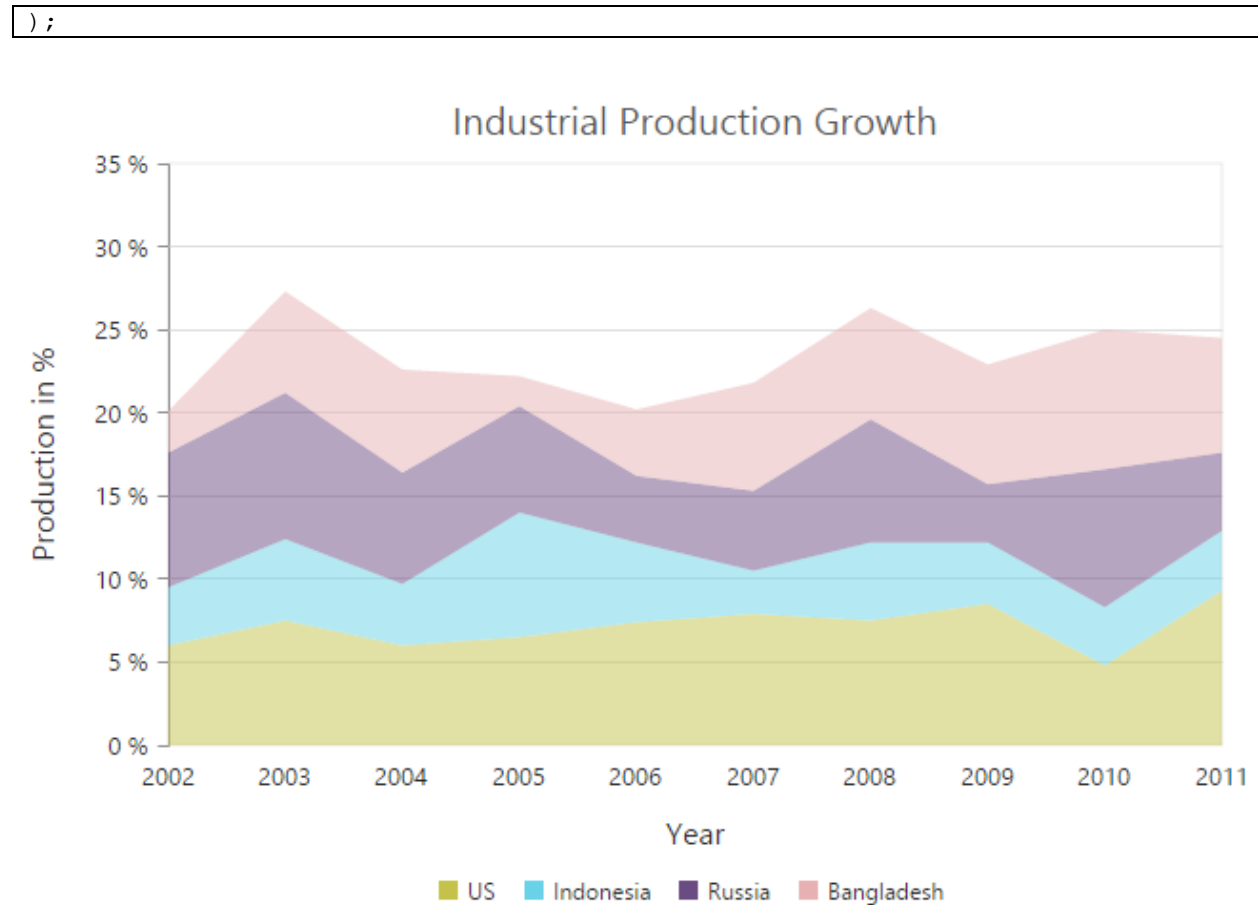
[Click](#) here to view our Spline Area Chart online demo.

### Stacked Area Chart

To render a Stacked Area Chart, set the [type](#) as “**stackingArea**” in the chart series. To change the StackingArea color, you can use the [fill](#) property of the series.

### JAVASCRIPT

```
"use strict";
// ...
var series= [{
  // Change the series type and fill color
  type: 'stackingArea',
  fill: "#69D2E7",
  // ...
}];
// ...
ReactDOM.render(
  <EJ.Chart id="default_chart_sample_0"
    series={series}
  >
  </EJ.Chart>,
  document.getElementById('chart')
```



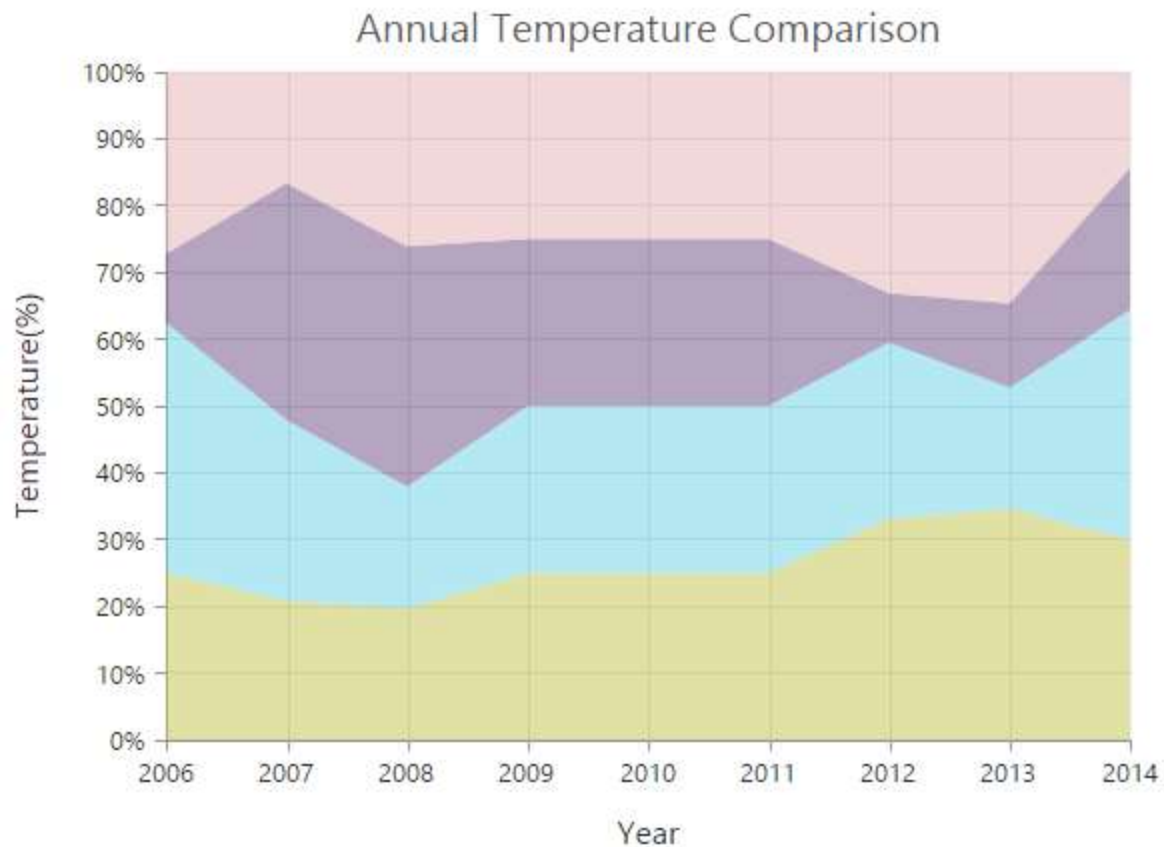
[Click](#) here to view our Stacked Area Chart online demo.

### 100% Stacked Area Chart

To render a 100% Stacked Area Chart, set the [type](#) as “**stackingArea100**” in the chart series. To change the StackingArea100 color, you can use the [fill](#) property of the series.

### JAVASCRIPT

```
"use strict";
// ...
var series= [{
// Change the series type and fill color
type: 'stackingArea100',
fill: "#C4C24A",
// ...
}];
// ...
ReactDOM.render(
<EJ.Chart id="default_chart_sample_0"
series={series}
>
</EJ.Chart>,
document.getElementById('chart')
);
```



[Click](#) here to view our 100% Stacked Area Chart online demo.

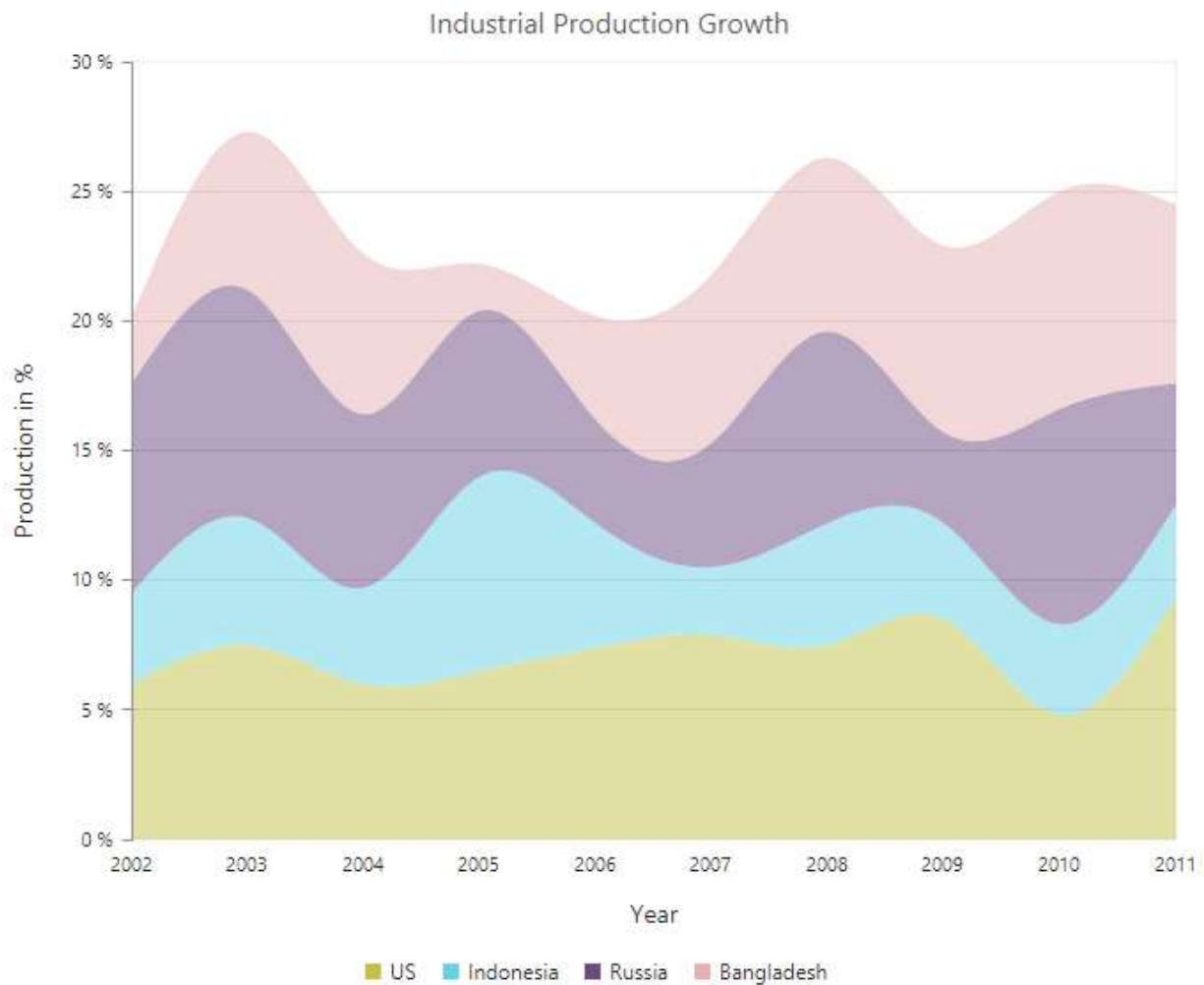
#### Stacked Spline Area Chart

To render a Stacked Spline Area Chart, set the [type](#) as “**stackingSplineArea**” in the chart series.

#### JAVASCRIPT

```
"use strict";
// ...
var series= [{
  // Change the series type
  type: 'stackingSplineArea',
  // ...
}];
// ...
ReactDOM.render(
  <EJ.Chart id="default_chart_sample_0"
  series={series}
  >
  </EJ.Chart>,
  document.getElementById('chart')
);
```



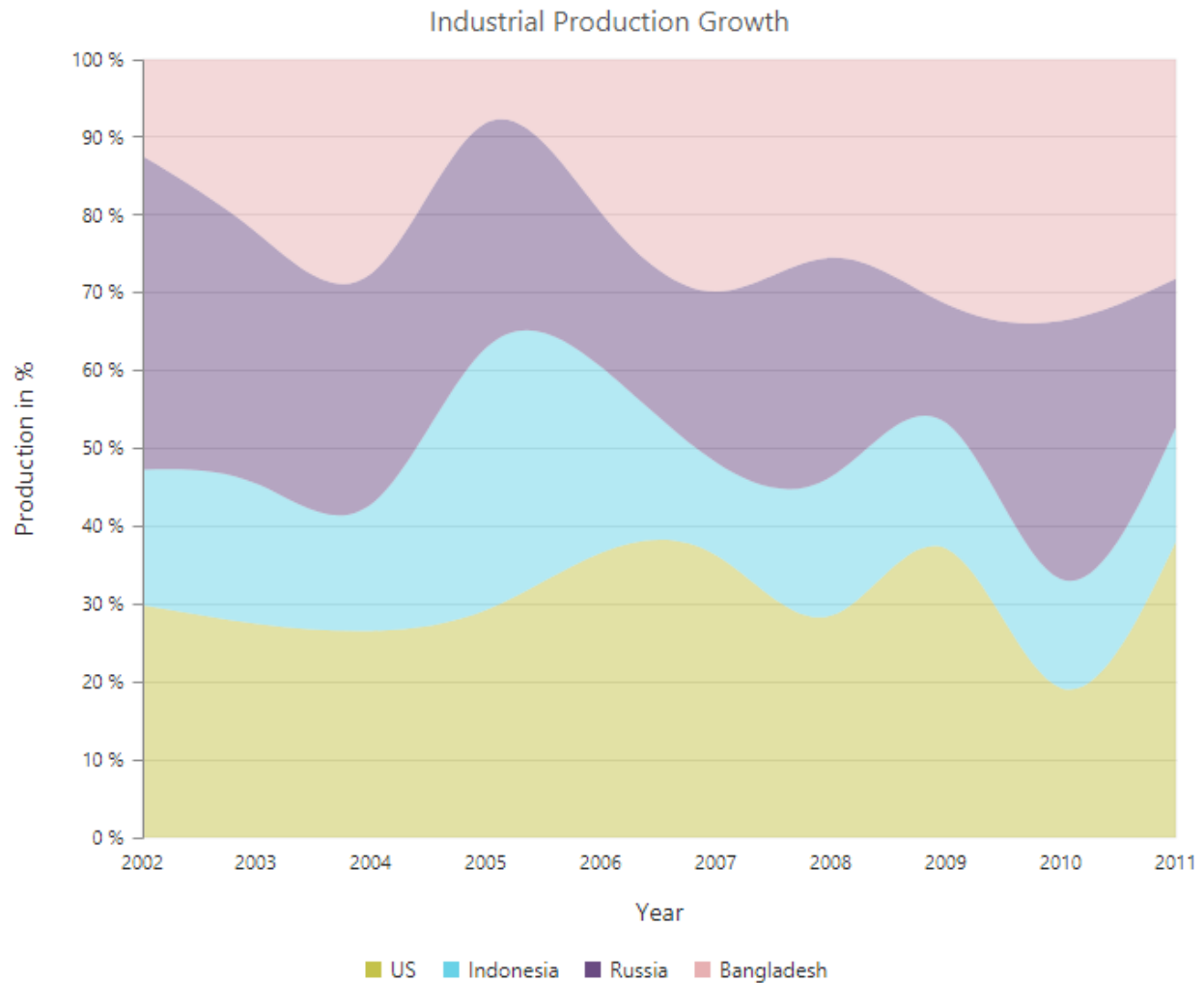


### 100% Stacked Spline Area Chart

To render a 100% Stacked Spline Area Chart, set the [type](#) as “**stackingSplineArea100**” in the chart series.

#### JAVASCRIPT

```
"use strict";
// ...
var series= [{
// Change the series type
type: 'stackingSplineArea100',
// ...
}];
// ...
ReactDOM.render(
<EJ.Chart id="default_chart_sample_0"
series={series}
>
</EJ.Chart>,
document.getElementById('chart')
);
```

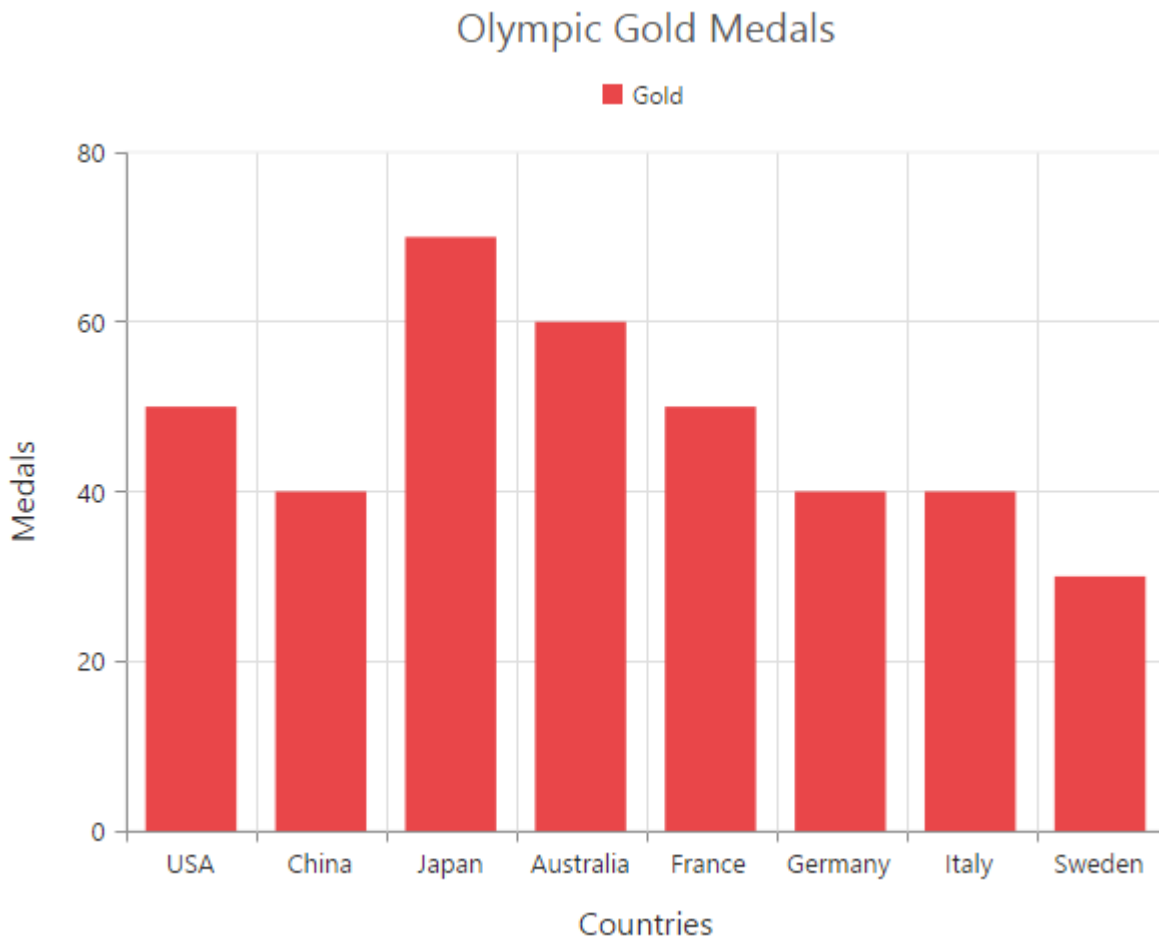


### Column Chart

To render a Column Chart, set the [type](#) as “**column**” in the chart series. To change the color of the column series, you can use the [fill](#) property.

### JAVASCRIPT

```
"use strict";
// ...
var series= [{
  // Change the series type and fill color
  type: 'column',
  fill: "#E94649",
  // ...
}];
// ...
ReactDOM.render(
  <EJ.Chart id="default_chart_sample_0"
    series={series}
  >
    </EJ.Chart>,
  document.getElementById('chart')
);
```



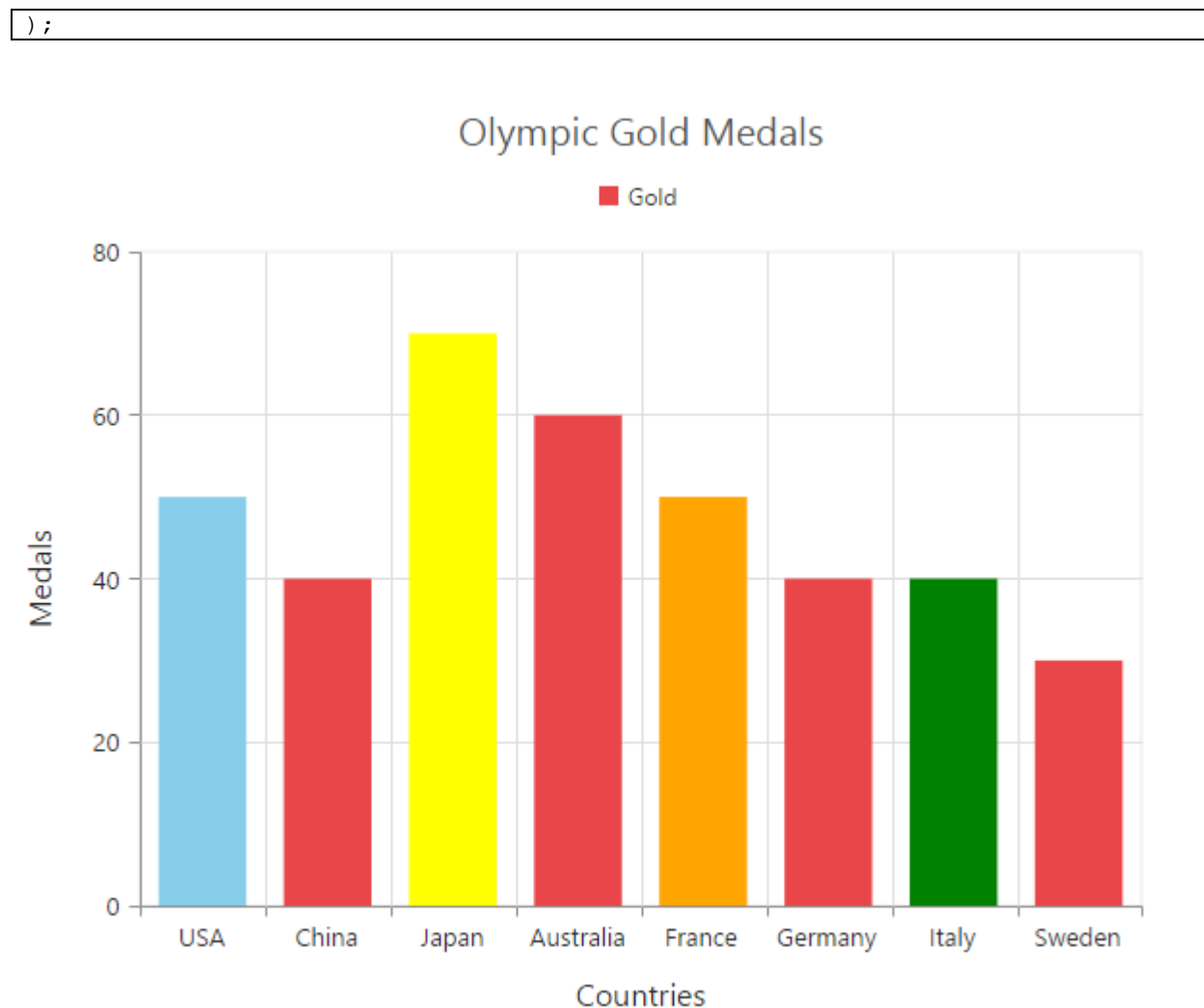
[Click](#) here to view our Column Chart demo.

*Change a point color*

You can change the color of a column by using the [fill](#) property of the point.

#### JAVASCRIPT

```
"use strict";
// ...
var series= [{
// Change the color of a column
points:[{ fill: 'skyblue' },
// ...
],
// ...
}];
// ...
ReactDOM.render(
<EJ.Chart id="default_chart_sample_0"
series={series}
>
</EJ.Chart>,
document.getElementById('chart')
```



#### Column width customization

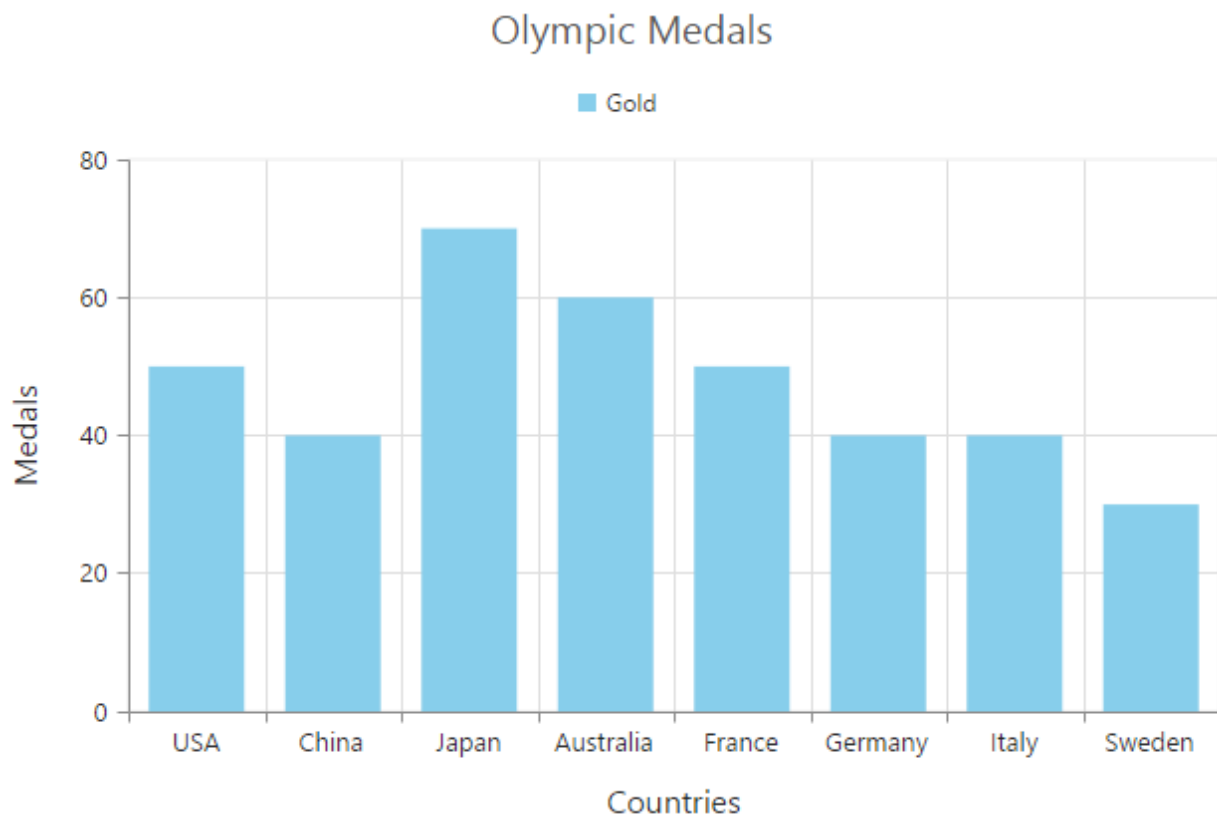
Width of the column type series can be customized by using the [columnWidth](#) property. Default value of [columnWidth](#) is 0.7. Value ranges from 0 to 1. Here 1 corresponds to 100% of available width and 0 corresponds to 0% of available width.

**Note:** Width of a column also depends upon the [columnSpacing](#) property, because [columnSpacing](#) will reduce the space available for drawing a column. This is also applicable for StackingColumn, StackingColumn100, Bar, StackingBar, StackingBar100, RangeColumn, HiLo, HiLoOpenClose, Candle and Waterfall charts.

#### JAVASCRIPT

```
"use strict";
//Common settings for all series
var commonSeriesOptions= {
//Width of columns in column type series
columnWidth: 0.7
//...
};
//Settings specific to individual series
var series= [{
```

```
//Width of columns in column type series
columnWidth: 0.8
//...
}];
//...
ReactDOM.render(
  <EJ.Chart id="default_chart_sample_0"
  commonSeriesOptions={commonSeriesOptions}
  series={series}
  >
  </EJ.Chart>,
  document.getElementById('chart')
);
```



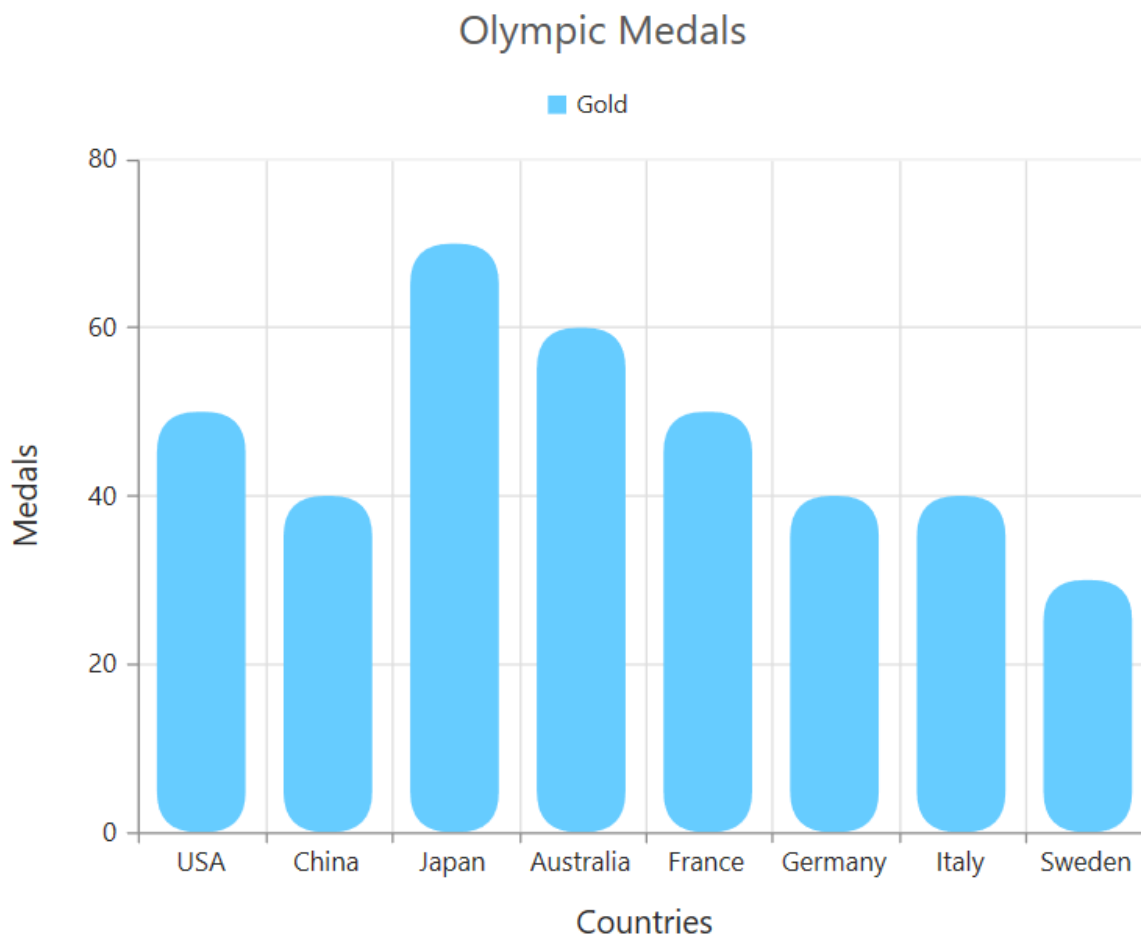
#### Column with rounded corners

Corners of the column chart can be customized by setting value to the `cornerRadius` property.

#### JAVASCRIPT

```
"use strict";
//Common settings for all series
var commonSeriesOptions= {
  cornerRadius:20
  //...
};
ReactDOM.render(
  <EJ.Chart id="default_chart_sample_0"
```

```
commonSeriesOptions={commonSeriesOptions}
>
</EJ.Chart>,
document.getElementById('chart')
);
```



#### Spacing between column series

Spacing between column type series can be customized using the [columnSpacing](#) property. Default value of [columnSpacing](#) is 0. Value ranges from 0 to 1. Here 1 corresponds to 100% available space and 0 corresponds to 0% available space.

**Note:** Column spacing will also affect the width of the column. For example, setting 20% spacing and 100% width will render columns with 80% of total width. This is also applicable for StackingColumn, StackingColumn100, Bar, StackingBar, StackingBar100, RangeColumn, HiLo, HiLoOpenClose, Candle and Waterfall charts.

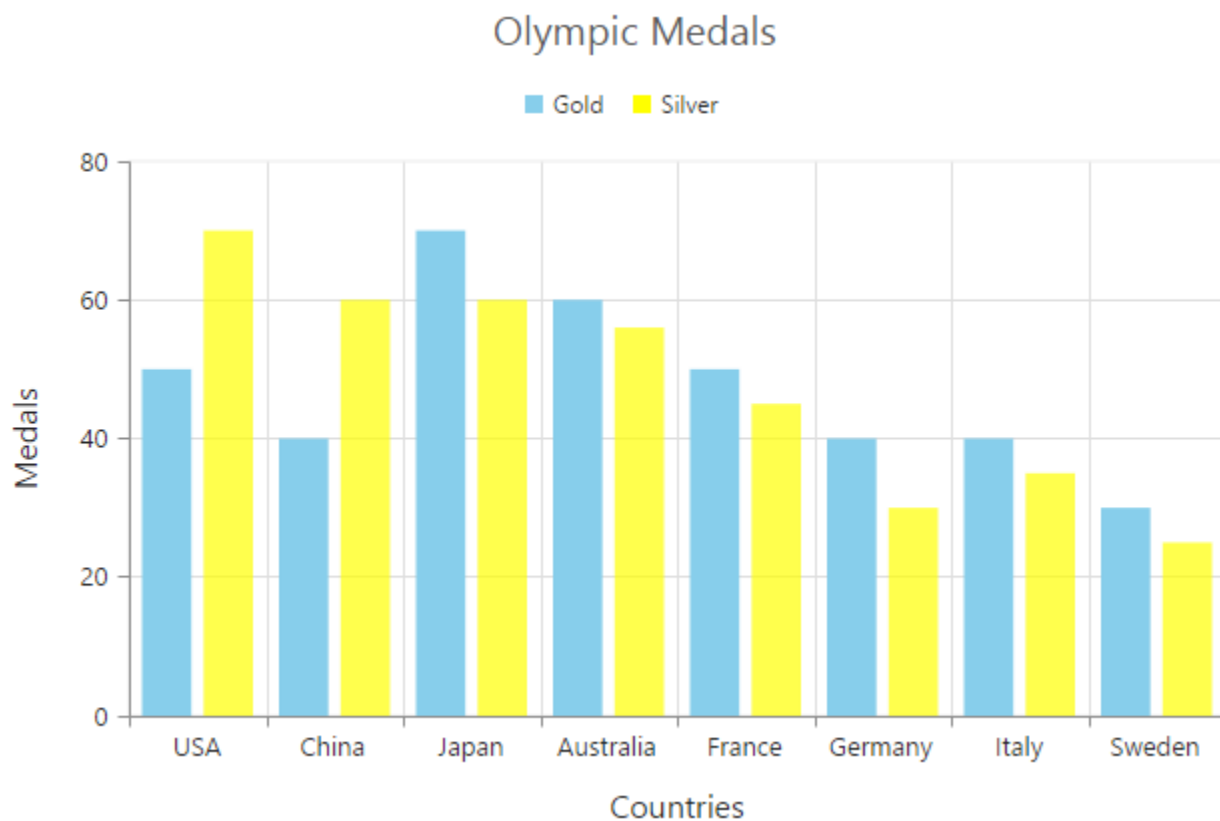
#### JAVASCRIPT

```
"use strict";
//Common settings for all series
var commonSeriesOptions= {
//Spacing between column series
```

```

columnSpacing: 0,
//...
};
//Settings specific to individual series
var series= [{
//Spacing between column series
columnSpacing: 0.2,
//...
}];
//...
ReactDOM.render(
<EJ.Chart id="default_chart_sample_0"
commonSeriesOptions={commonSeriesOptions}
series={series}
>
</EJ.Chart>,
document.getElementById('chart')
);

```



#### Cylindrical Chart

To render a cylindrical chart, set the [columnFacet](#) property as "cylinder" in the chart series along with the series type.

The following chart types can be rendered as cylinder in both 2D and in 3D view.

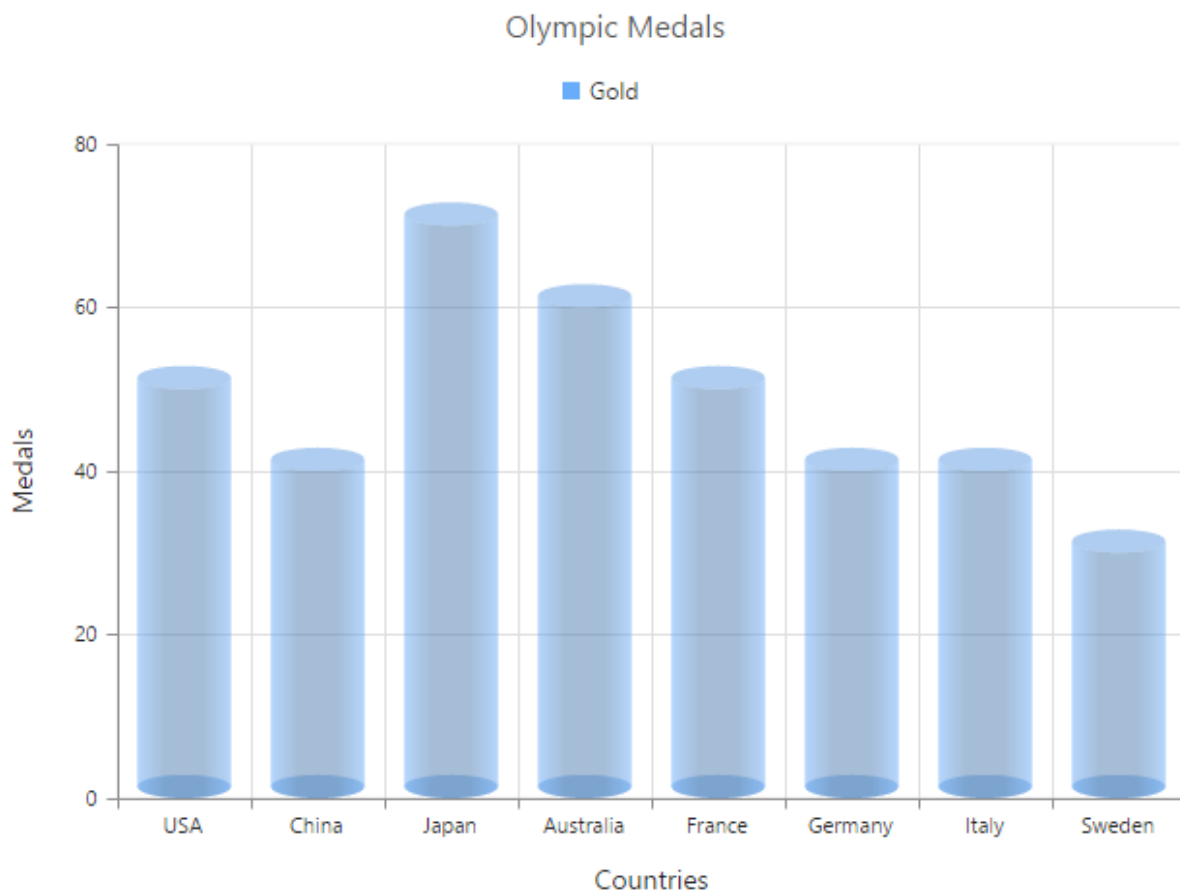
- Column Chart, Bar Chart, Stacked Column Chart, Stacked Bar Chart, 100% Stacked Column Chart, 100% Stacked Bar Chart.

**JAVASCRIPT**

```

"use strict";
var series= [{
  //To change the shape of the series
  columnFacet: 'cylinder',
  type: 'column',
  // ...
}];
//...
ReactDOM.render(
  <EJ.Chart id="default_chart_sample_0"
  series={series}
  >
  </EJ.Chart>,
  document.getElementById('chart')
);

```

**RangeColumn Chart**

To render a Range Column Chart, set the [type](#) as “**rangeColumn**” in the chart series. To change the RangeColumn color, use the [fill](#) property of the series.

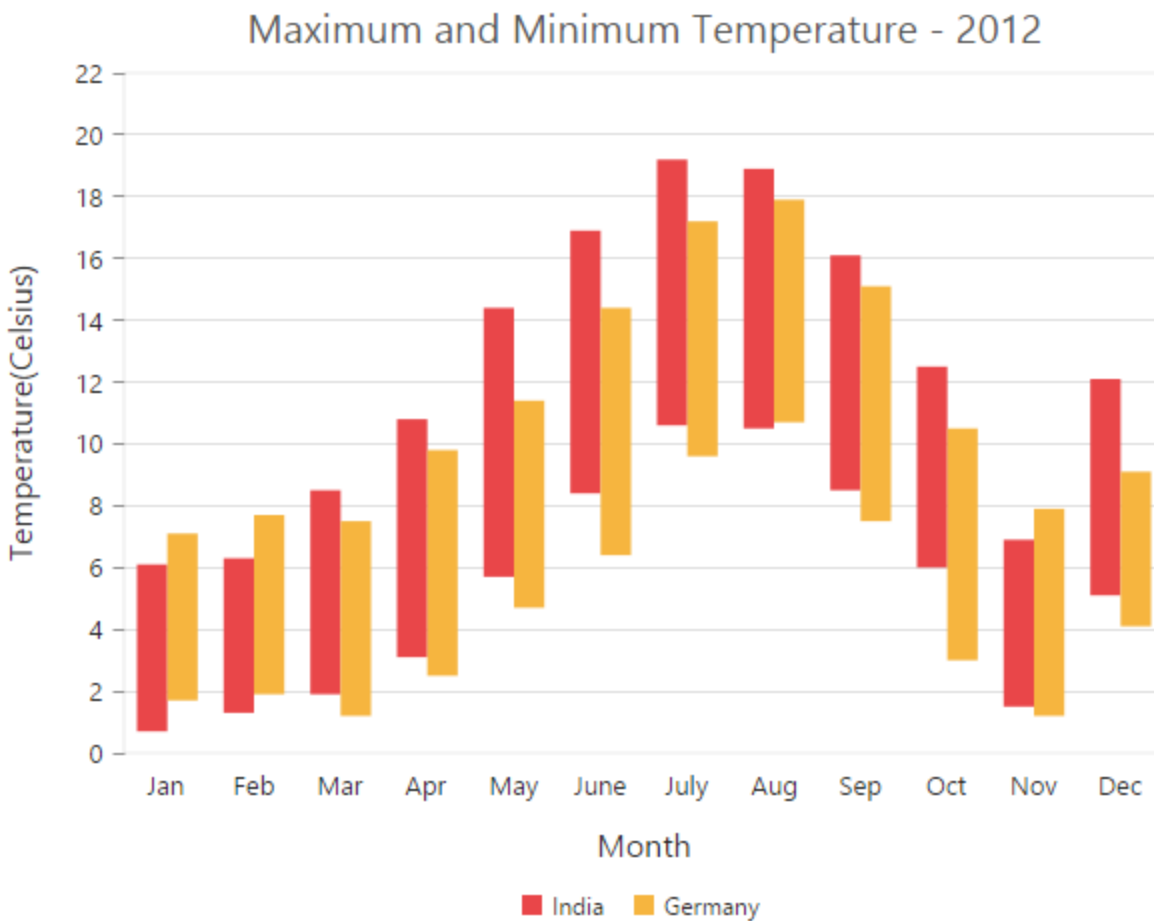
Since, the RangeColumn series requires two y values for a point, add the [high](#) and [low](#) value. High and Low value specifies the maximum and minimum range of the points.



- When you are using the [points](#) option, specify the high and low values by using the [high](#) and [low](#) option of the point.
- When you are using the [dataSource](#) option to assign the data, you have to map the fields from the dataSource that contains high and low values by using the [series.high](#) and [series.low](#) options.

## JAVASCRIPT

```
"use strict";
var series= [{
  //Set chart type to series
  type: 'rangeColumn',
  fill: "#E94649",
  //Use high and low values instead of y
  points:[{ high: 6.1, low:0.7 },
  // ...
  ],
  / ...
}];
//...
ReactDOM.render(
  <EJ.Chart id="default_chart_sample_0"
  series={series}
  >
  </EJ.Chart>,
  document.getElementById('chart')
);
```



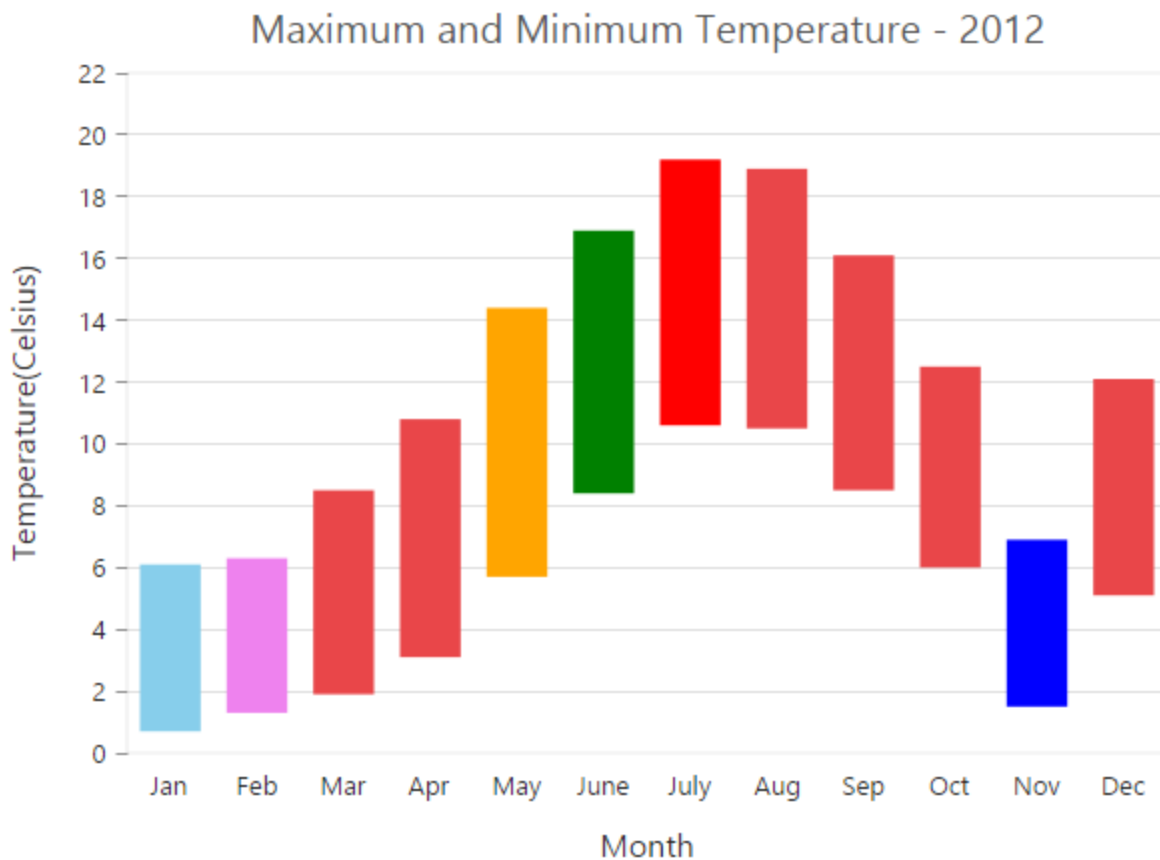
[Click](#) here to view our Range Column Chart online demo.

[Change a point color](#)

To change the color of a range column, you can use the [fill](#) property of point.

### JAVASCRIPT

```
"use strict";
// ...
var series= [{
// Change the color of a range column
points:[{ fill: 'skyblue' },
// ...
],
// ...
}];
// ...
ReactDOM.render(
<EJ.Chart id="default_chart_sample_0"
series={series}
>
</EJ.Chart>,
document.getElementById('chart')
);
```

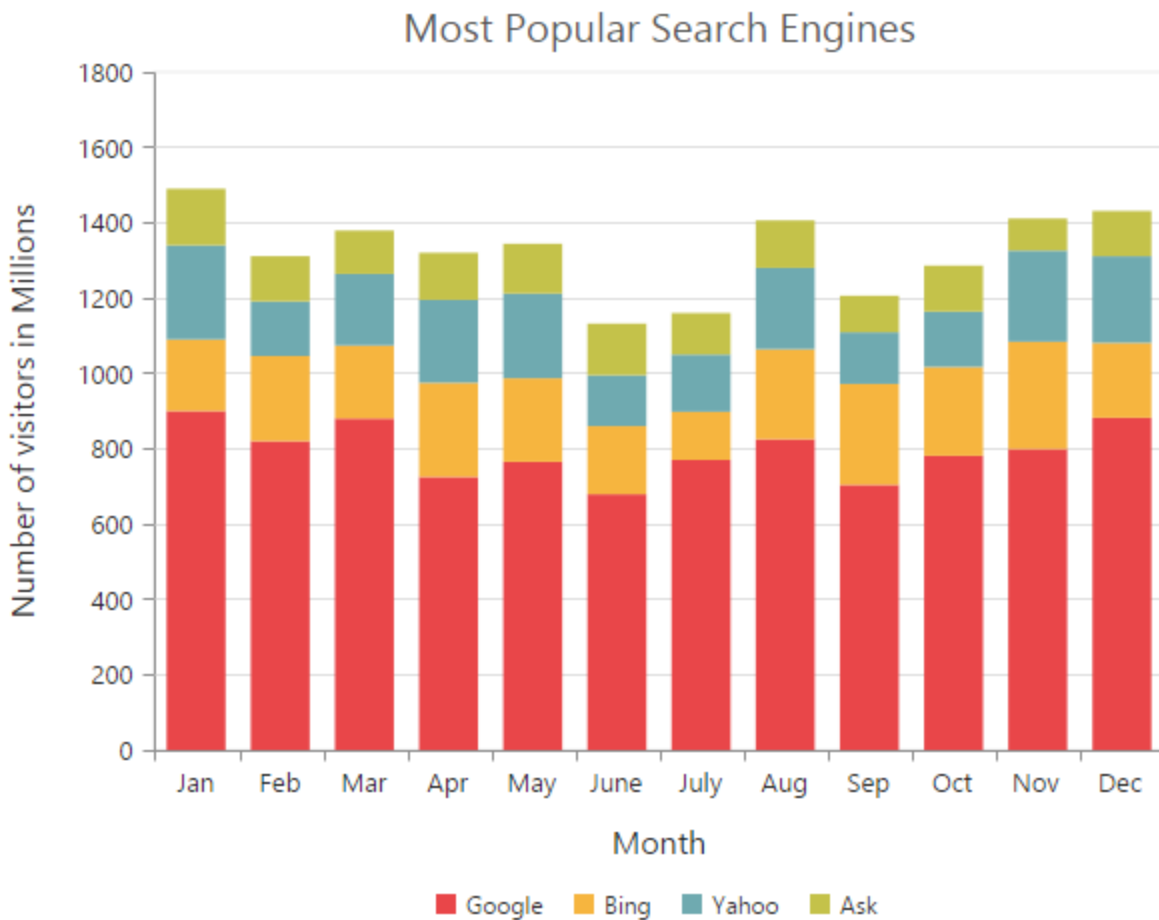


### Stacked Column Chart

To render a Stacked Column Chart, set the [type](#) as “**stackingColumn**” in the chart series. To change the StackingColumn color, you can use the [fill](#) property of the series.

### JAVASCRIPT

```
"use strict";
// ...
var series= [{
//Change type and color of the series
type: 'stackingColumn',
fill: "#E94649",
// ...
}];
// ...
ReactDOM.render(
<EJ.Chart id="default_chart_sample_0"
series={series}
>
</EJ.Chart>,
document.getElementById('chart')
);
```



[Click](#) here to view our Stacked Column Chart online demo.

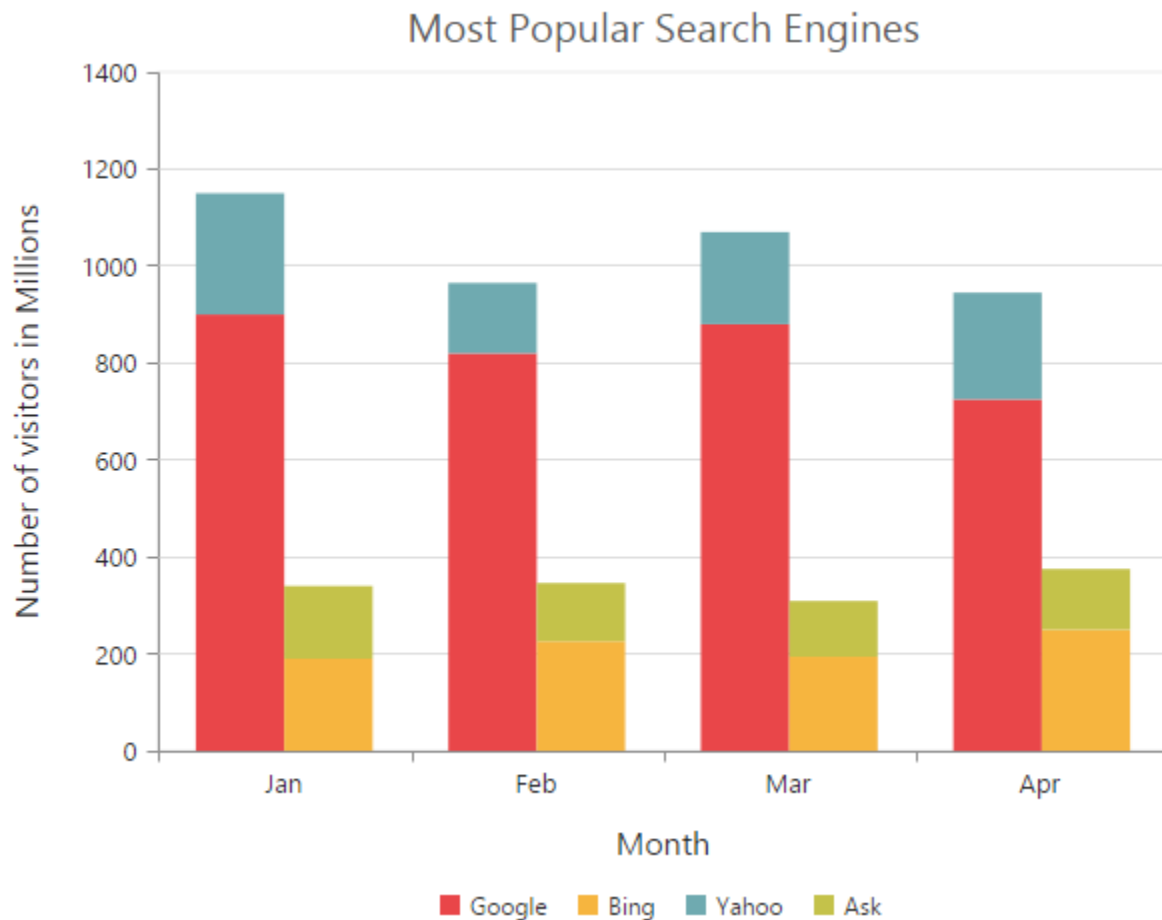
*Cluster / Group stacked columns*

You can use the [stackingGroup](#) property to group the stacked columns. Columns with same group name are stacked on top of each other.

#### JAVASCRIPT

```
"use strict";
var series= [{
  // For grouping stacked columns
  stackingGroup: 'GroupOne'
  // ...
},
{
  // For grouping stacked columns
  stackingGroup: 'GroupOne'
  // ...
}];
// ...
ReactDOM.render(
  <EJ.Chart id="default_chart_sample_0"
    series={series}
  >
  </EJ.Chart>,
```

```
document.getElementById('chart')
);
```



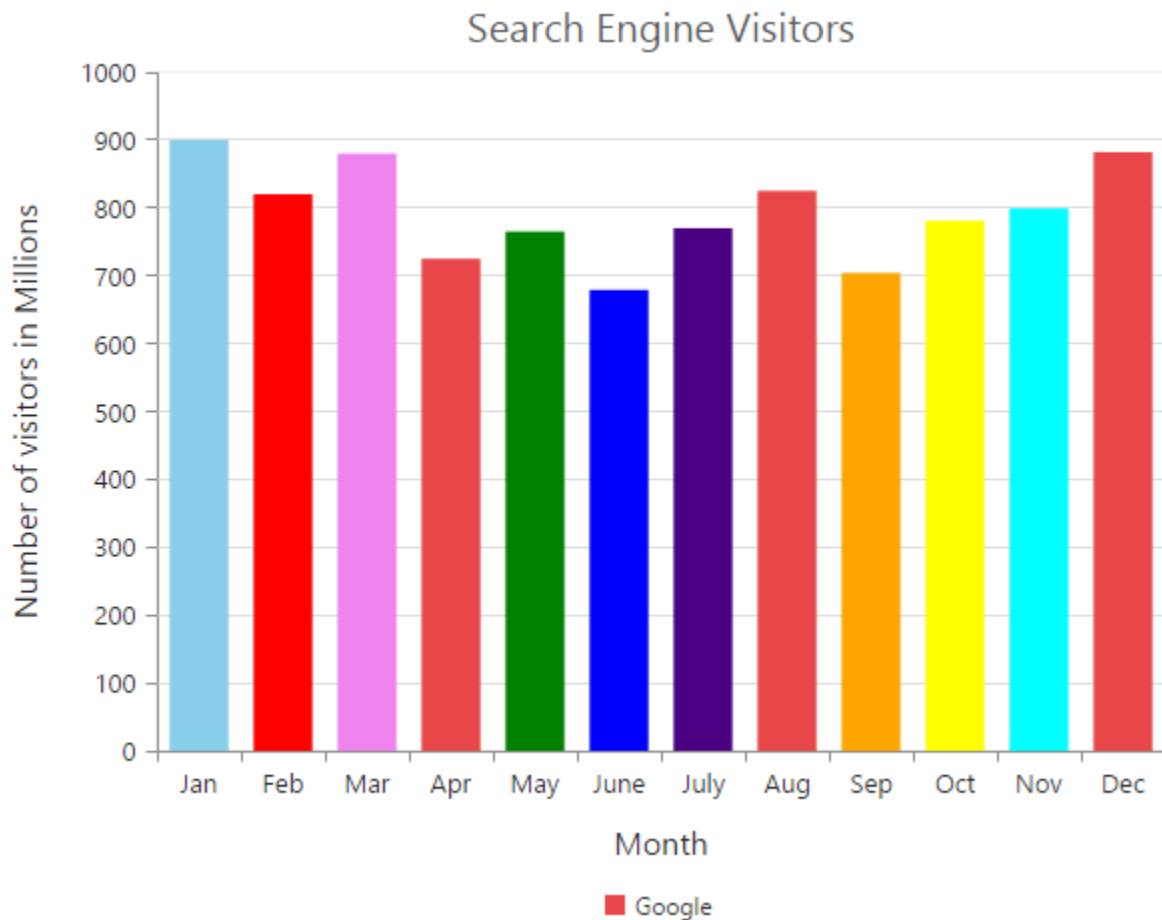
#### Change a point color

To change the color of a stacking column, you can use the [fill](#) property of the point.

#### JAVASCRIPT

```
"use strict";
// ...
var series= [{
// Change the color of a stacking column
points:[{ fill: 'skyblue' },
// ...
],
// ...
}];
// ...
ReactDOM.render(
<EJ.Chart id="default_chart_sample_0"
series={series}
>
</EJ.Chart>,
document.getElementById('chart')
```

```
);
```

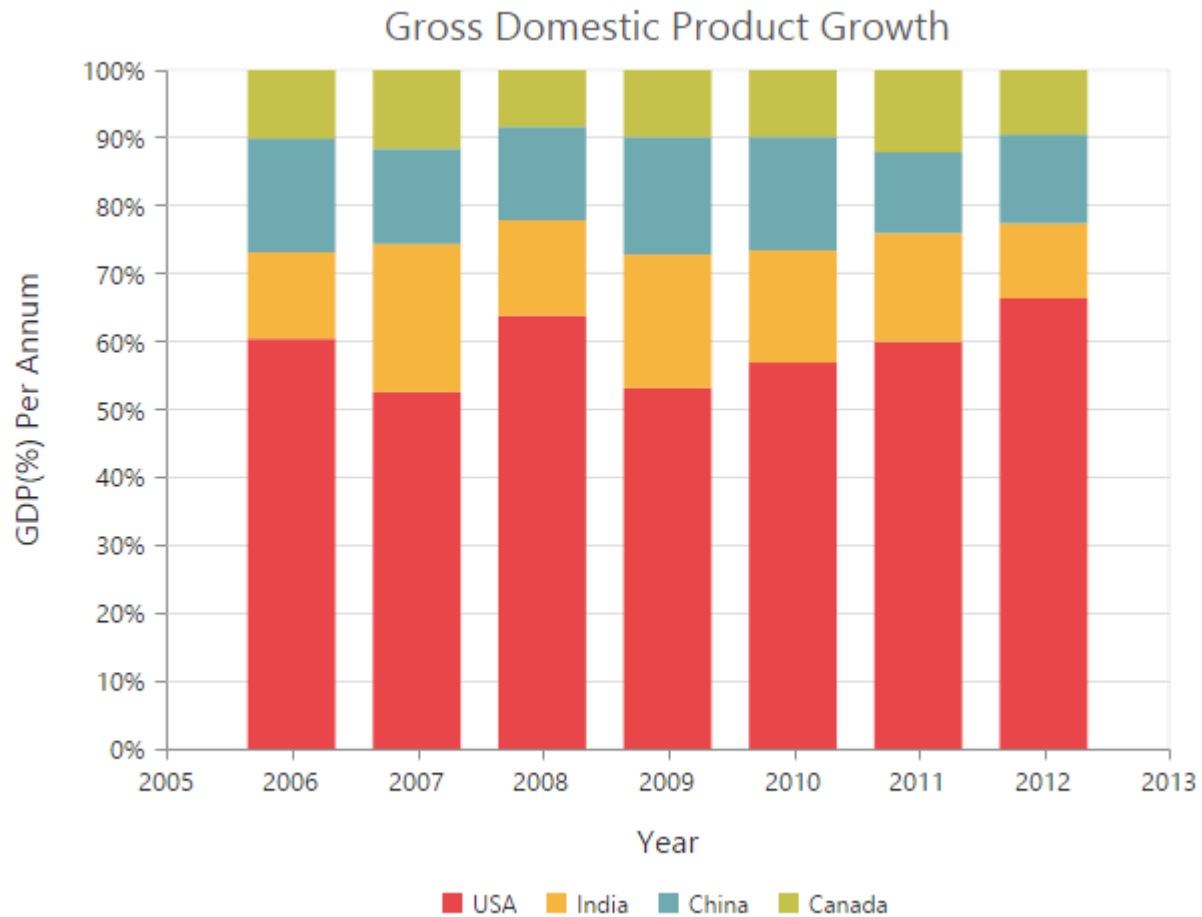


### 100% Stacked Column Chart

To render a 100% Stacked Column Chart, set the [type](#) as “**stackingColumn100**” in the chart series. To change the StackingColumn100 color, you can use the [fill](#) property of the series.

### JAVASCRIPT

```
"use strict";
// ...
var series= [{
//Change type and color of the series
type: 'stackingColumn100',
fill: "#E94649",
// .....
}];
// ...
ReactDOM.render(
<EJ.Chart id="default_chart_sample_0"
series={series}
>
</EJ.Chart>,
document.getElementById('chart')
);
```



[Click](#) here to view our 100% Stacked Column Chart online demo.

*Cluster / Group 100% stacked columns*

By using the [stackingGroup](#) property, you can group the 100% stacking columns. Columns with same group name are stacked on top of each other.

#### JAVASCRIPT

```
"use strict";
// ...
var series= [{
  // For grouping 100% stacked columns
  stackingGroup: 'GroupOne'
  // ...
},
{
  // For grouping 100% stacked columns
  stackingGroup: 'GroupOne'
  // ...
}];
// ...
ReactDOM.render(
  <EJ.Chart id="default_chart_sample_0"
```

```

series={series}
>
</EJ.Chart>,
document.getElementById('chart')
);

```



#### Change a point color

To change the color of a 100% stacking column, you can use the [fill](#) property of the point.

#### JAVASCRIPT

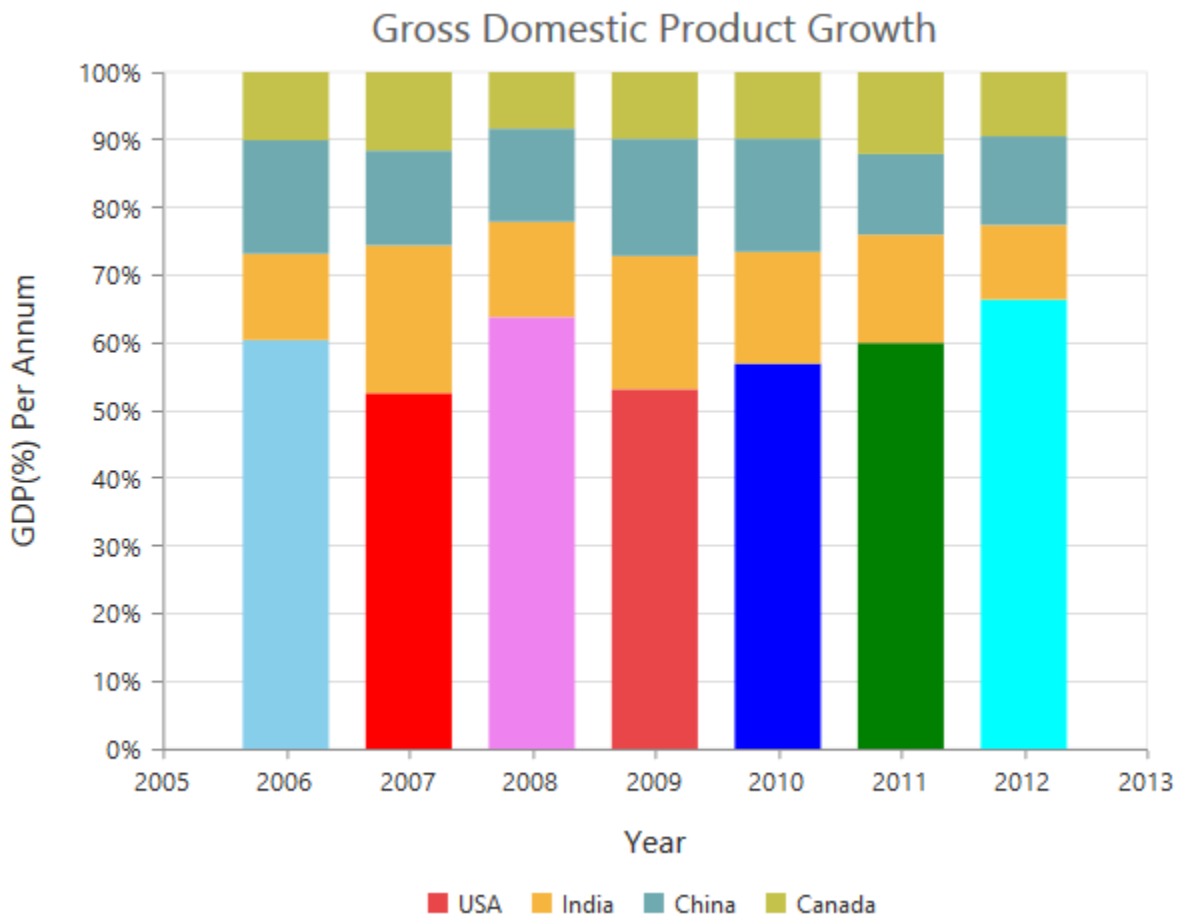
```

"use strict";
// ...
var series= [{
// Change the color of a 100% stacking column
points:[{ fill: 'skyblue' },
// ...
],
// ...
}];
// ...
ReactDOM.render(
<EJ.Chart id="default_chart_sample_0"
series={series}

```



```
>
</EJ.Chart>,
document.getElementById('chart')
);
```



### Bar Chart

To render a bar Chart, set the [type](#) as “bar” in the chart series. To change the bar color, you can use the [fill](#) property of the series.

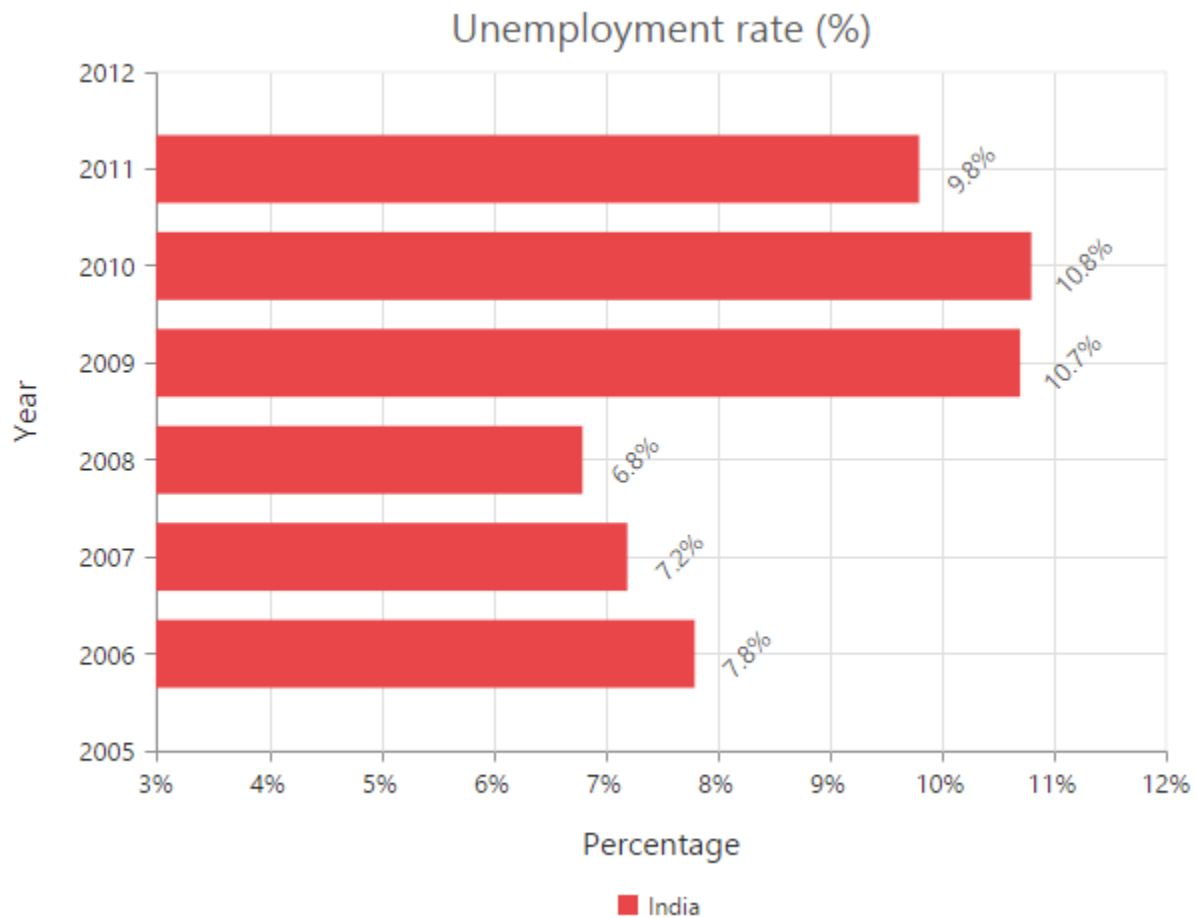
### JAVASCRIPT

```
"use strict";
// ...
var series= [{
//Change type and color of the series
type: 'bar',
fill: "#E94649",
// ...
}];
// ...
ReactDOM.render(
<EJ.Chart id="default_chart_sample_0"
series={series}
>
```

```

</EJ.Chart>,
document.getElementById('chart')
);

```



[Click](#) here to view our Bar Chart demo.

*Change the color of a bar*

By using the [fill](#) property of the point, you can change the specific point of the series.

#### JAVASCRIPT

```

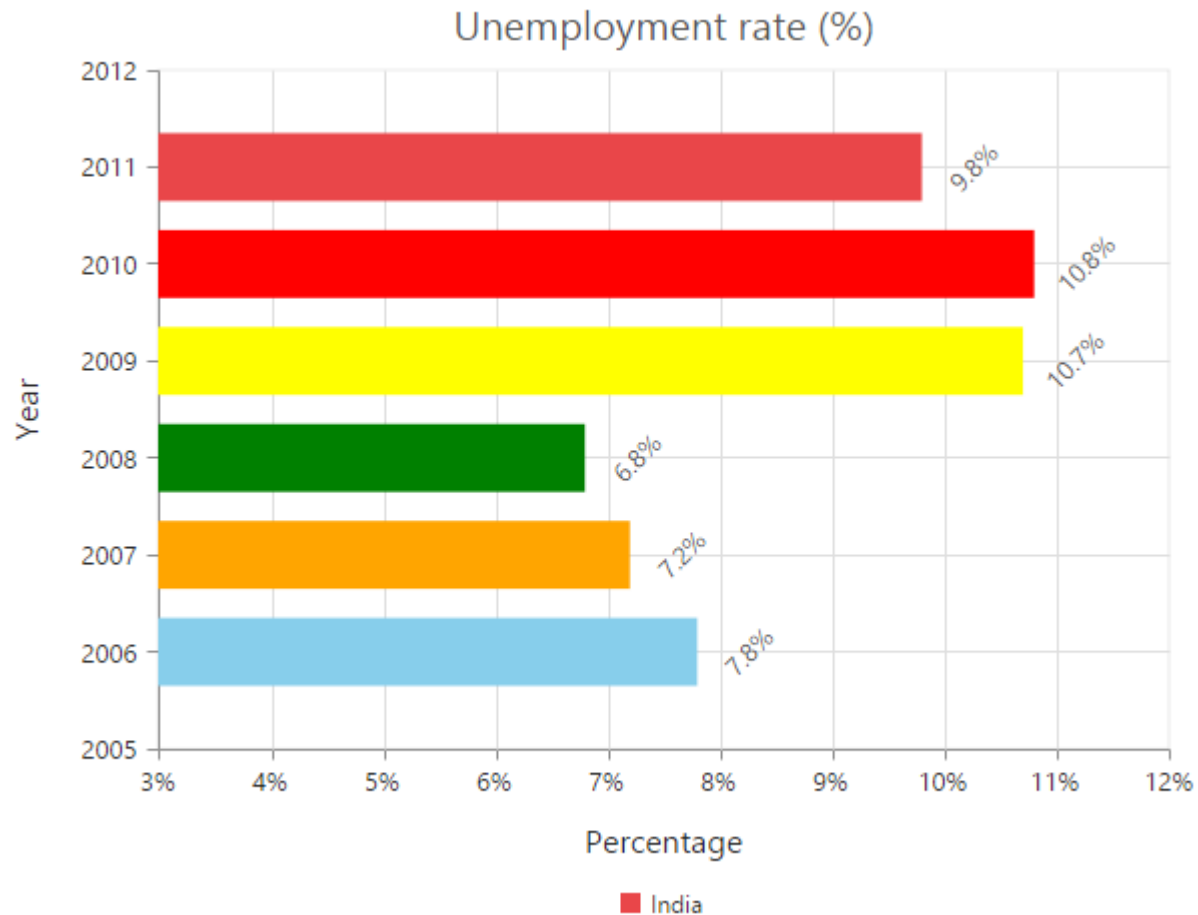
"use strict";
// ...
var series= [{
// Change the color of a bar
points:[{ fill: 'skyblue' },
// ...
],
// ...
}];
// ...
ReactDOM.render(
<EJ.Chart id="default_chart_sample_0"
series={series}
>

```

```

</EJ.Chart>,
document.getElementById('chart')
);

```



### Stacked Bar Chart

To render a Stacked Bar Chart, set the [type](#) as “**stackingBar**” in the chart series. To change the StackingBar color, you can use the [fill](#) property of the series.

### JAVASCRIPT

```

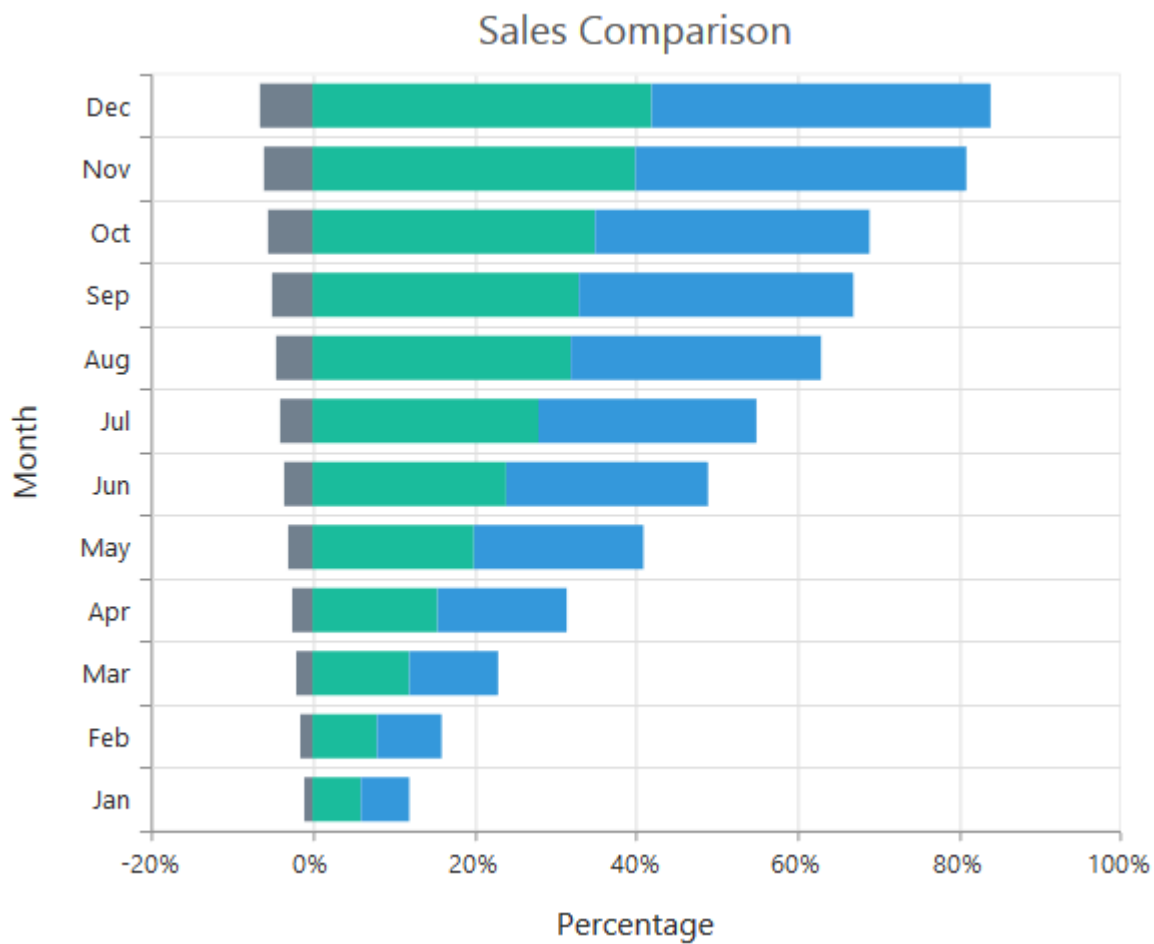
"use strict";
// ...
var series= [{
//Change type and color of the series.
type: 'stackingBar',
fill: "#E94649",
// ...
}];
// ...
ReactDOM.render(
<EJ.Chart id="default_chart_sample_0"
series={series}
>

```

```

</EJ.Chart>,
document.getElementById('chart')
);

```



[Click](#) here to view our Stacked Bar Chart online demo.

*Cluster / Group stacked bars*

You can use the [stackingGroup](#) property to group the stacking bars with the same group name.

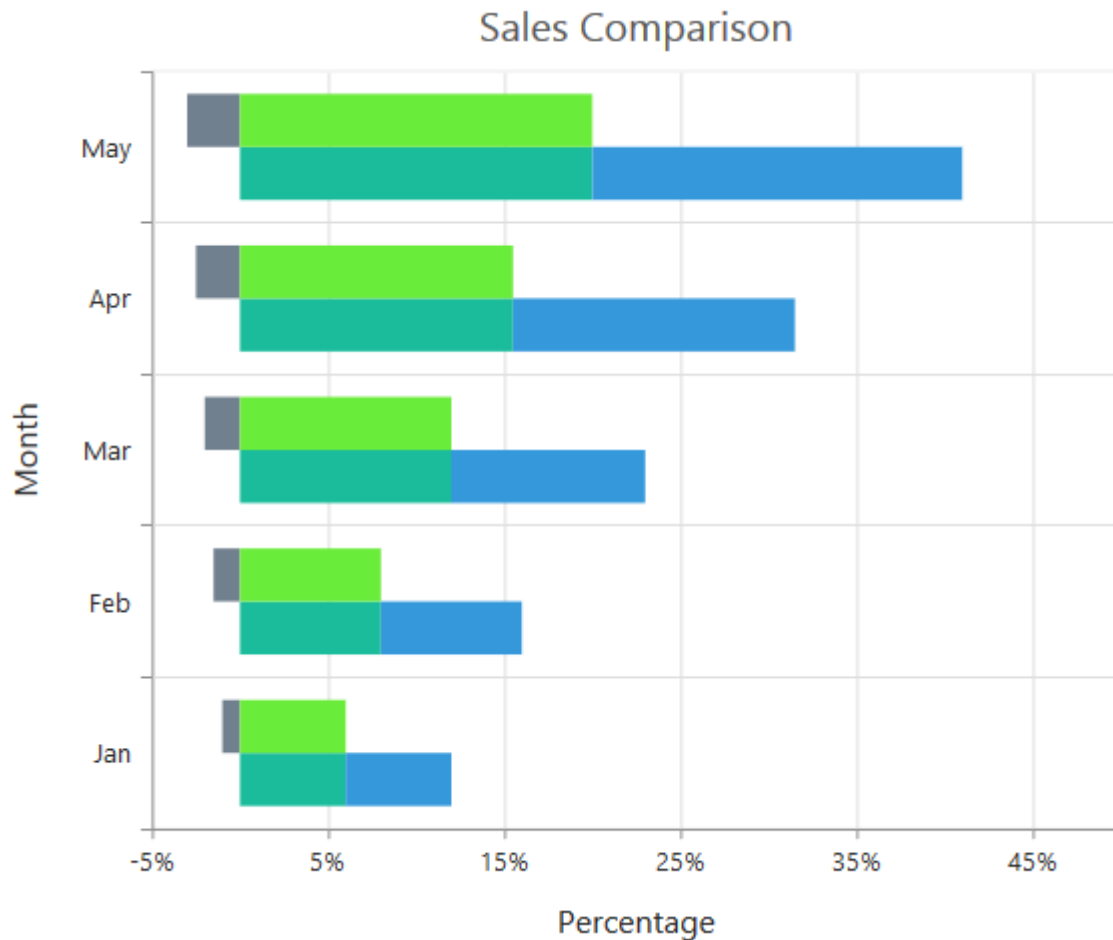
### **JAVASCRIPT**

```

"use strict";
// ...
var series= [{
// For grouping stacked bar
stackingGroup: 'GroupOne'
// ...
},
{
// For grouping stacked bar
stackingGroup: 'GroupOne'
// ...
}
];
// ...

```

```
ReactDOM.render(
  <EJ.Chart id="default_chart_sample_0"
    series={series}
  >
  </EJ.Chart>,
  document.getElementById('chart')
);
```



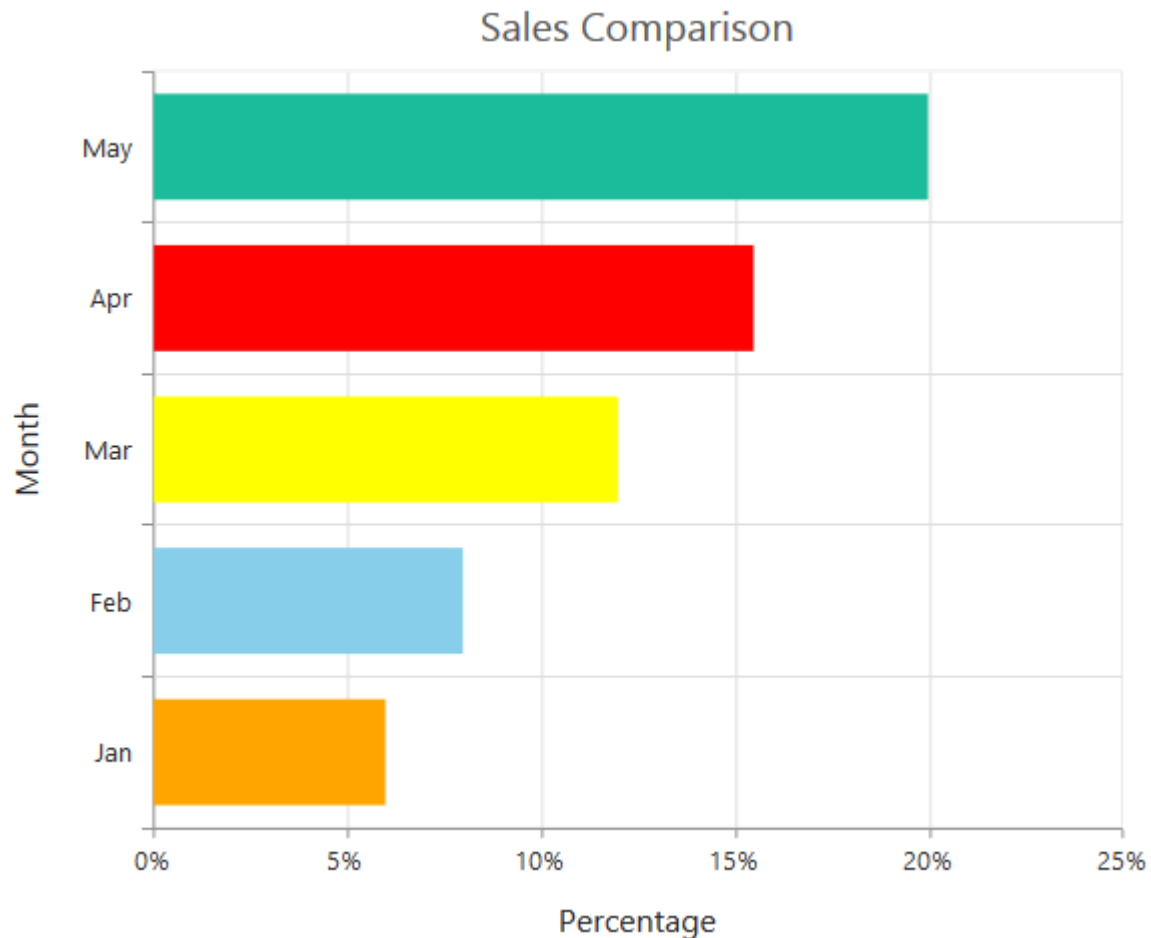
#### Change a point color

You can change the color of a stacking bar by using the [fill](#) property of the point.

#### JAVASCRIPT

```
"use strict";
// ...
var series= [{
  // Change the color of a stacking bar
  points:[{ fill: 'skyblue' },
  // ...
],
  // ...
}];
// ...
ReactDOM.render(
```

```
<EJ.Chart id="default_chart_sample_0"
series={series}
>
</EJ.Chart>,
document.getElementById('chart')
);
```



### 100% Stacked Bar Chart

To render a 100% Stacked Bar Chart, set the [type](#) as “**stackingBar100**” in the chart series. To change the StackingBar100 color, you can use the [fill](#) property of the series.

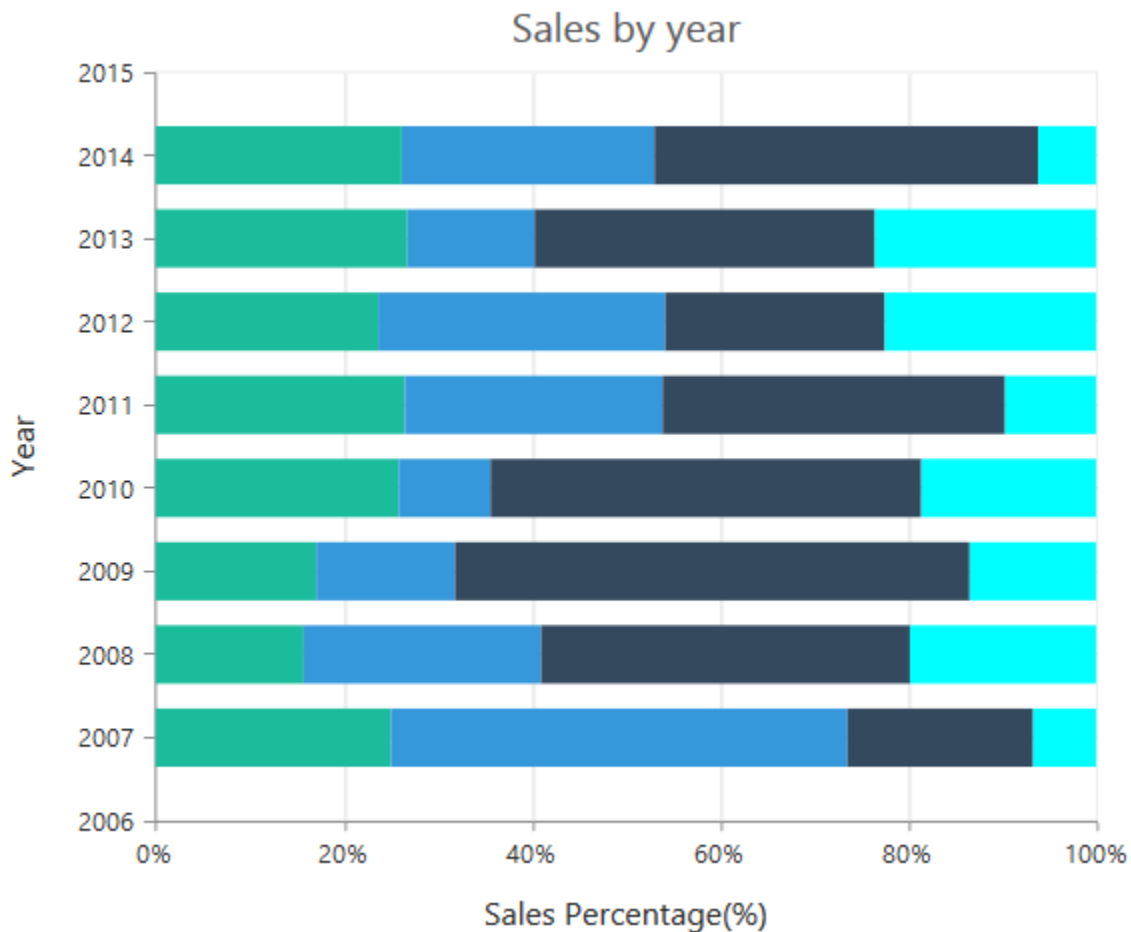
### JAVASCRIPT

```
"use strict";
// ...
var series= [{
//Change type and color of the series.
type: 'stackingBar100',
fill: "#E94649",
// ...
}];
// ...
ReactDOM.render(
<EJ.Chart id="default_chart_sample_0"
```

```

series={series}
>
</EJ.Chart>,
document.getElementById('chart')
);

```



[Click](#) here to view our 100% Stacked Bar Chart online demo.

By using the [stackingGroup](#) property, you can group the 100% stacking bars with the same group name.

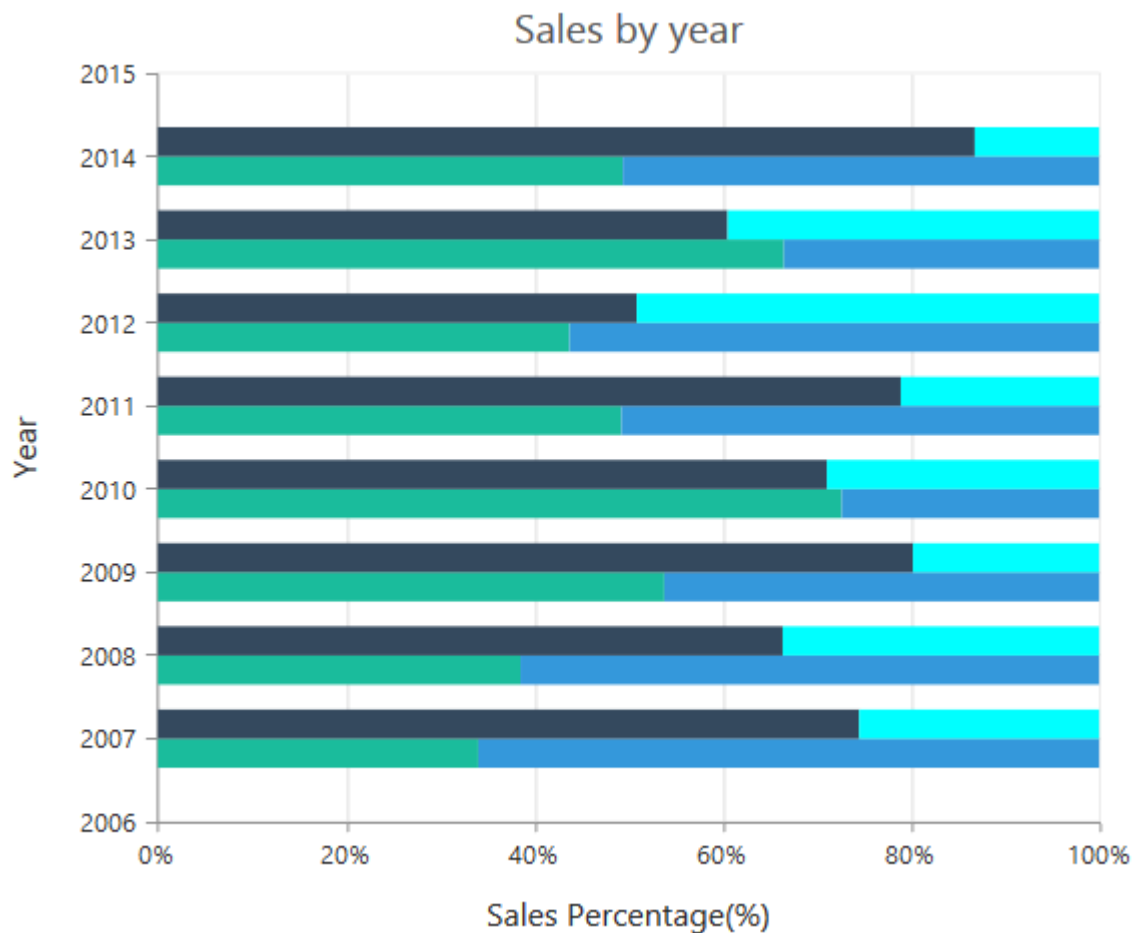
#### JAVASCRIPT

```

"use strict";
// ...
var series= [{
// For grouping 100% stacked bar
stackingGroup: 'GroupOne'
// ...
},
{
// For grouping 100% stacked bar
stackingGroup: 'GroupOne'
// ...
}
];

```

```
// ...
ReactDOM.render(
  <EJ.Chart id="default_chart_sample_0"
  series={series}
  >
</EJ.Chart>,
  document.getElementById('chart')
);
```



#### Change a point color

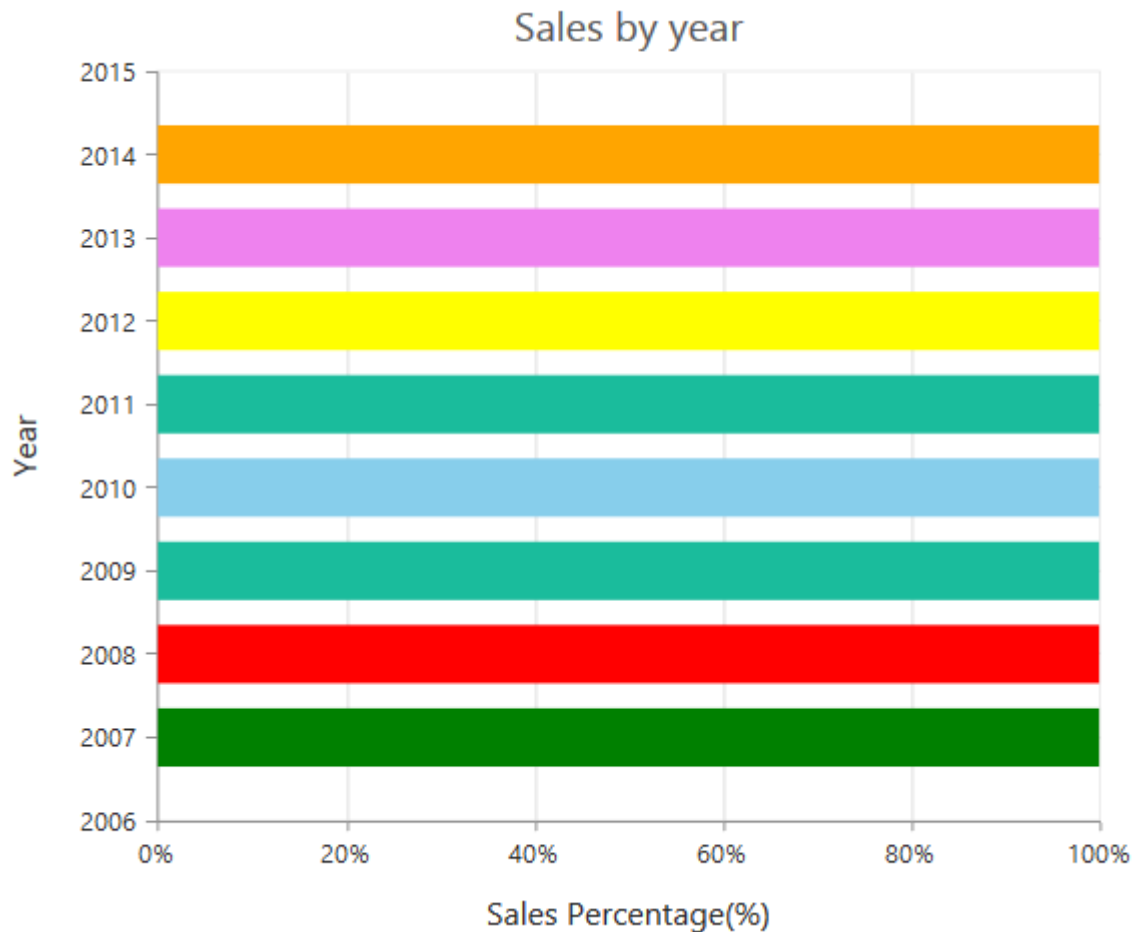
To change the color of a 100% stacking bar, you can use the [fill](#) property of the point.

#### JAVASCRIPT

```
"use strict";
// ...
var series= [{
  // Change the color of a 100% stacking bar
  points:[{ fill: 'skyblue' },
  // ...
],
  // ...
}];
// ...
```



```
ReactDOM.render(
  <EJ.Chart id="default_chart_sample_0"
    series={series}
  >
  </EJ.Chart>,
  document.getElementById('chart')
);
```



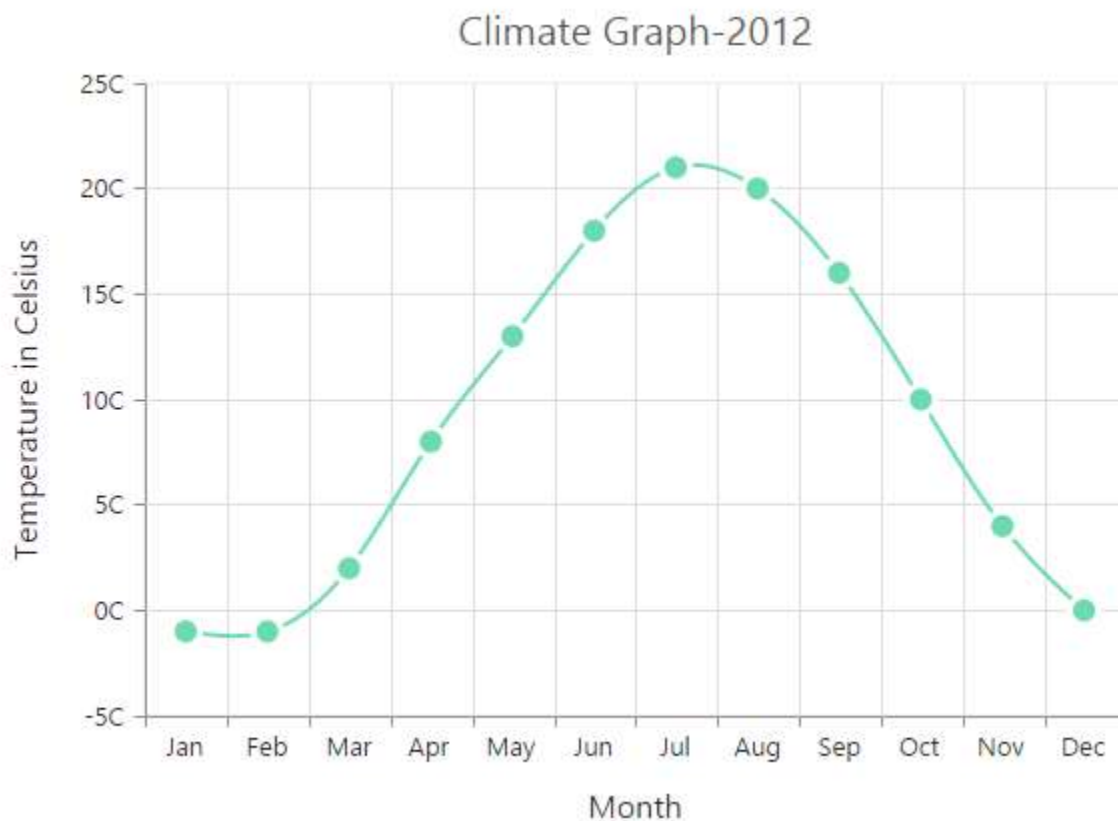
### Spline Chart

To render a Spline Chart, set the [type](#) as “**spline**” in the chart series. To change the Spline segment color, you can use the [fill](#) property of the series.

#### JAVASCRIPT

```
"use strict";
// ...
var series= [{
  //Change type and color of the series.
  type: 'spline',
  fill: "#6ADCB0",
  // ...
}];
// ...
```

```
ReactDOM.render(
  <EJ.Chart id="default_chart_sample_0"
    series={series}
  >
  </EJ.Chart>,
  document.getElementById('chart')
);
```



[Click](#) here to view the Spline Chart online demo sample.

#### *Spline Types*

Spline series supports four types of curves, namely natural, monotonic, cardinal and clamped. To change the spline type, you can use the [splineType](#) property in the series.

#### **JAVASCRIPT**

```
"use strict";
// ...
var series= [{
  //Change the spline series type.
  type: 'spline',
  splineType: 'Natural'
  // ...
}];
// ...
ReactDOM.render(
  <EJ.Chart id="default_chart_sample_0"
```

```
series={series}
>
</EJ.Chart>,
document.getElementById('chart')
);
```

### *Change the cardinal spline tension*

To change cardinal spline tension, you can use the [cardinalSplineTension](#) property in the series. The default value of cardinalSplineTension is **0.5**. Its value ranges from 0 to 1.

#### **JAVASCRIPT**

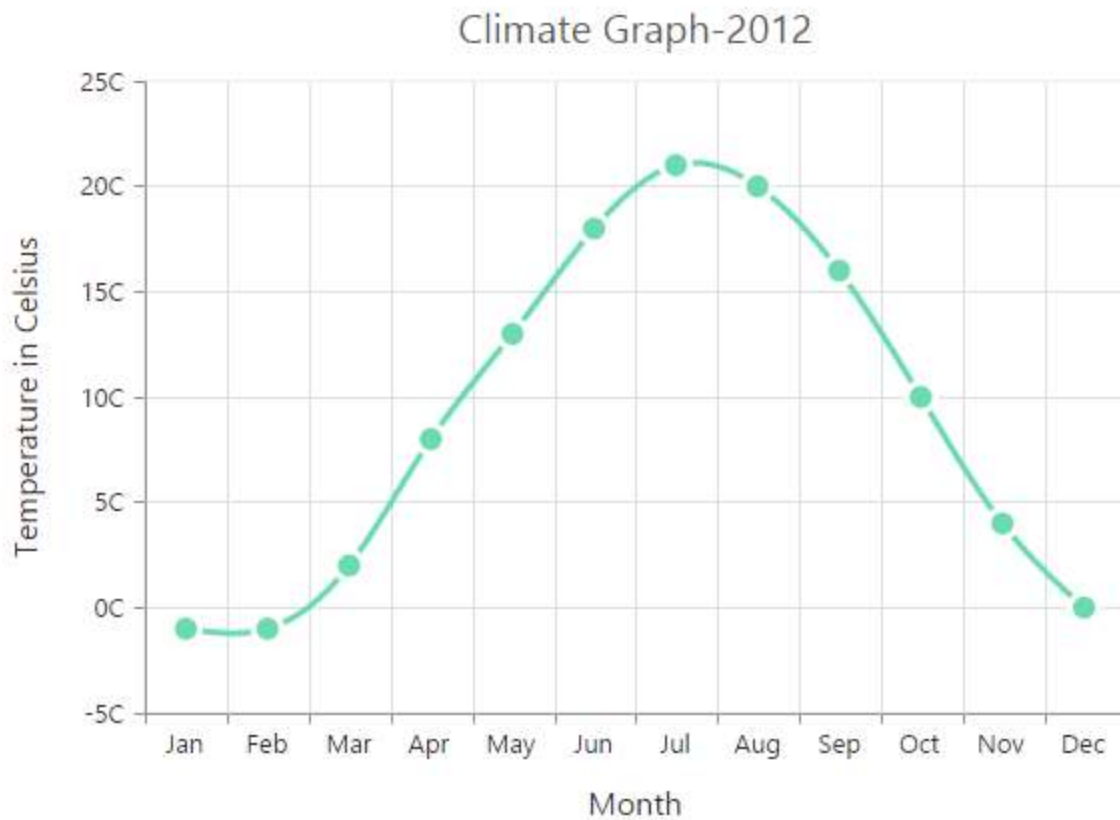
```
"use strict";
// ...
var series= [{
type: 'spline',
//Change the shape of cardinal spline
splineType: 'Natural',
cardinalSplineTension: 0.7,
// ...
}];
// ...
ReactDOM.render(
<EJ.Chart id="default_chart_sample_0"
series={series}
>
</EJ.Chart>,
document.getElementById('chart')
);
```

### *Change the spline width*

To change the spline segment width, you can use the [width](#) property of the series.

#### **JAVASCRIPT**

```
"use strict";
// ...
var series= [{
//Change the width of spline series
width: 3,
// ...
}];
// ...
ReactDOM.render(
<EJ.Chart id="default_chart_sample_0"
series={series}
>
</EJ.Chart>,
document.getElementById('chart')
);
```

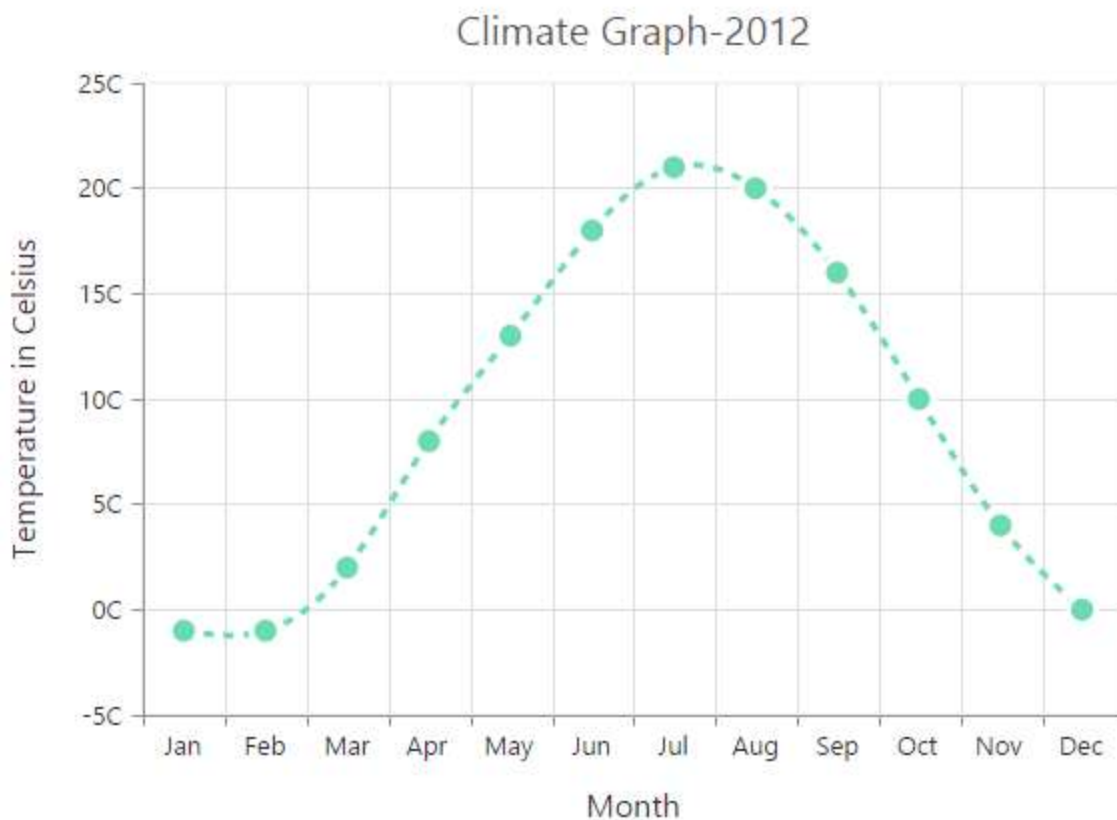


#### Dashed lines

To render the spline series with dotted lines, you can use the [dashArray](#) option of the series.

#### JAVASCRIPT

```
"use strict";
// ...
var series= [{
  //Change dash array to display dotted or dashed splines
  dashArray: '5,5',
  // ...
}];
// ...
ReactDOM.render(
  <EJ.Chart id="default_chart_sample_0"
  series={series}
  >
  </EJ.Chart>,
  document.getElementById('chart')
);
```

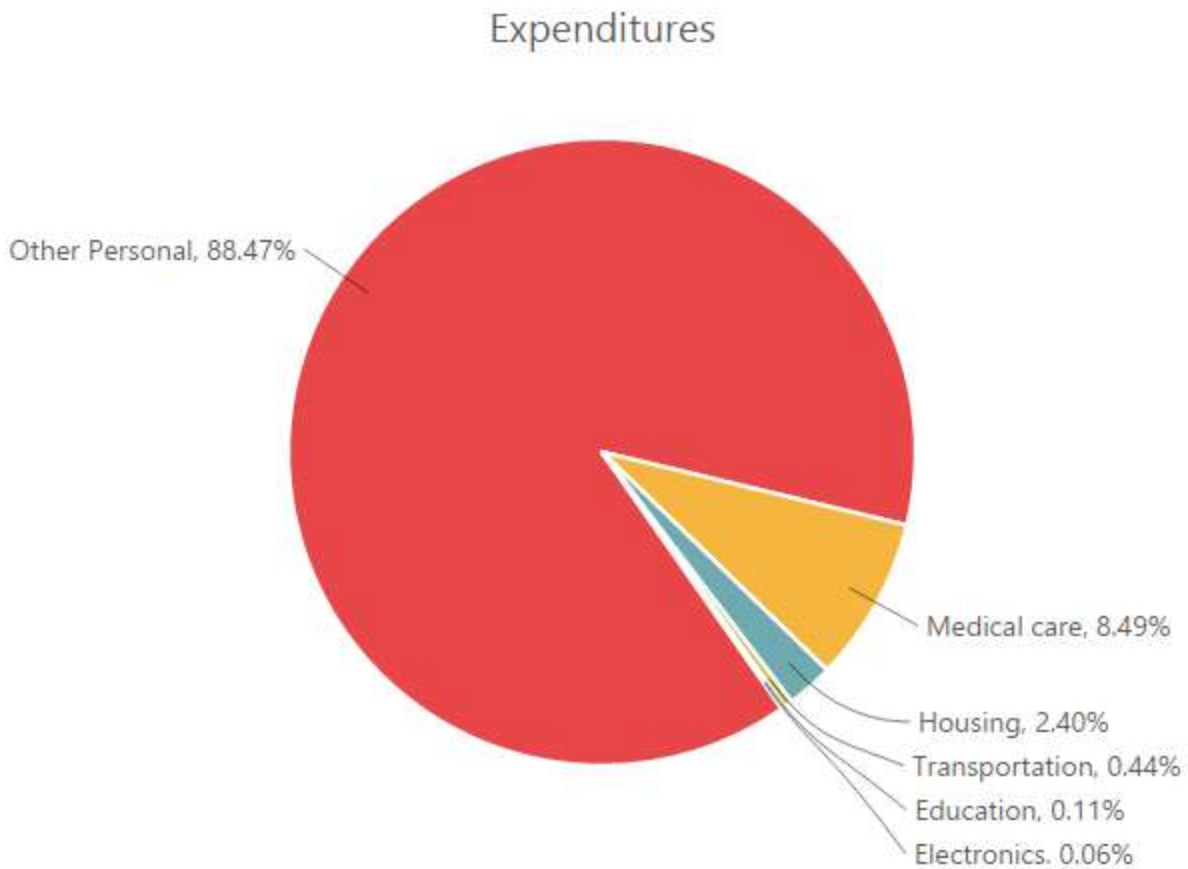


### Pie Chart

You can create a pie chart by setting the series [type](#) as “**pie**” in the chart series.

### JAVASCRIPT

```
"use strict";
// ...
var series= [{
  //Set chart type to series
  type: 'pie',
  // ...
}];
// ...
ReactDOM.render(
  <EJ.Chart id="default_chart_sample_0"
  series={series}
  >
  </EJ.Chart>,
  document.getElementById('chart')
);
```



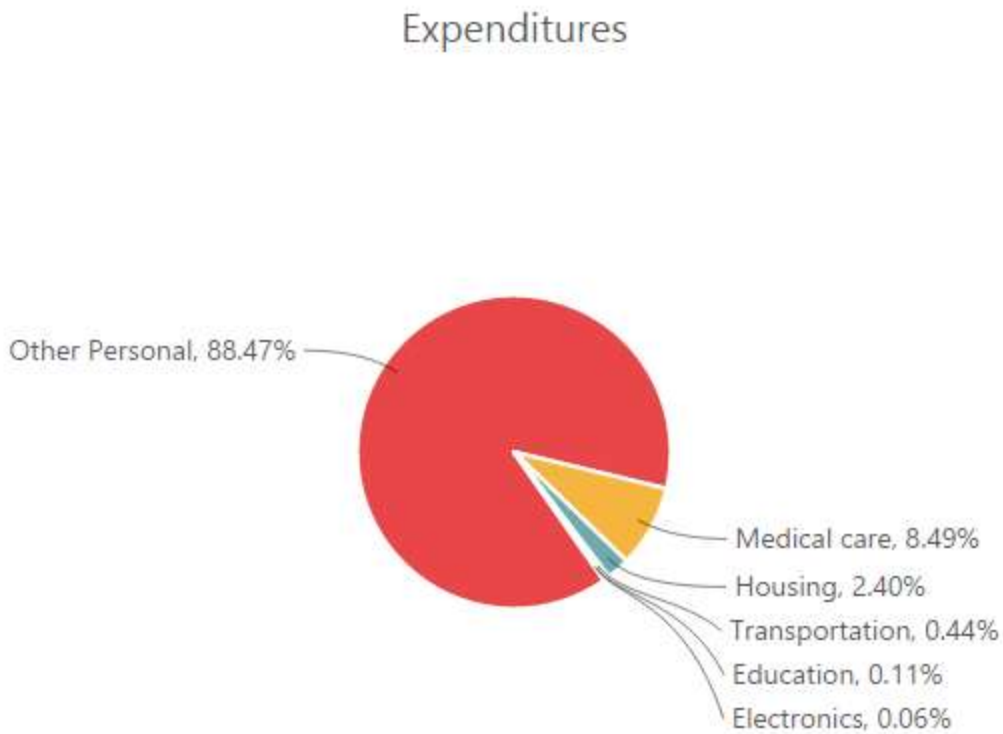
[Click](#) here to view the Pie chart online demo sample.

#### *Change the pie size*

You can use the [pieCoefficient](#) property to change the diameter of the Pie chart with respect to the plot area. It ranges from 0 to 1 and the default value is **0.8**.

#### **JAVASCRIPT**

```
"use strict";
// ...
var series= [{
  //Change pie chart coefficient value
  pieCoefficient: 0.4,
  // ...
}];
// ...
ReactDOM.render(
  <EJ.Chart id="default_chart_sample_0"
    series={series}
  >
  </EJ.Chart>,
  document.getElementById('chart')
);
```

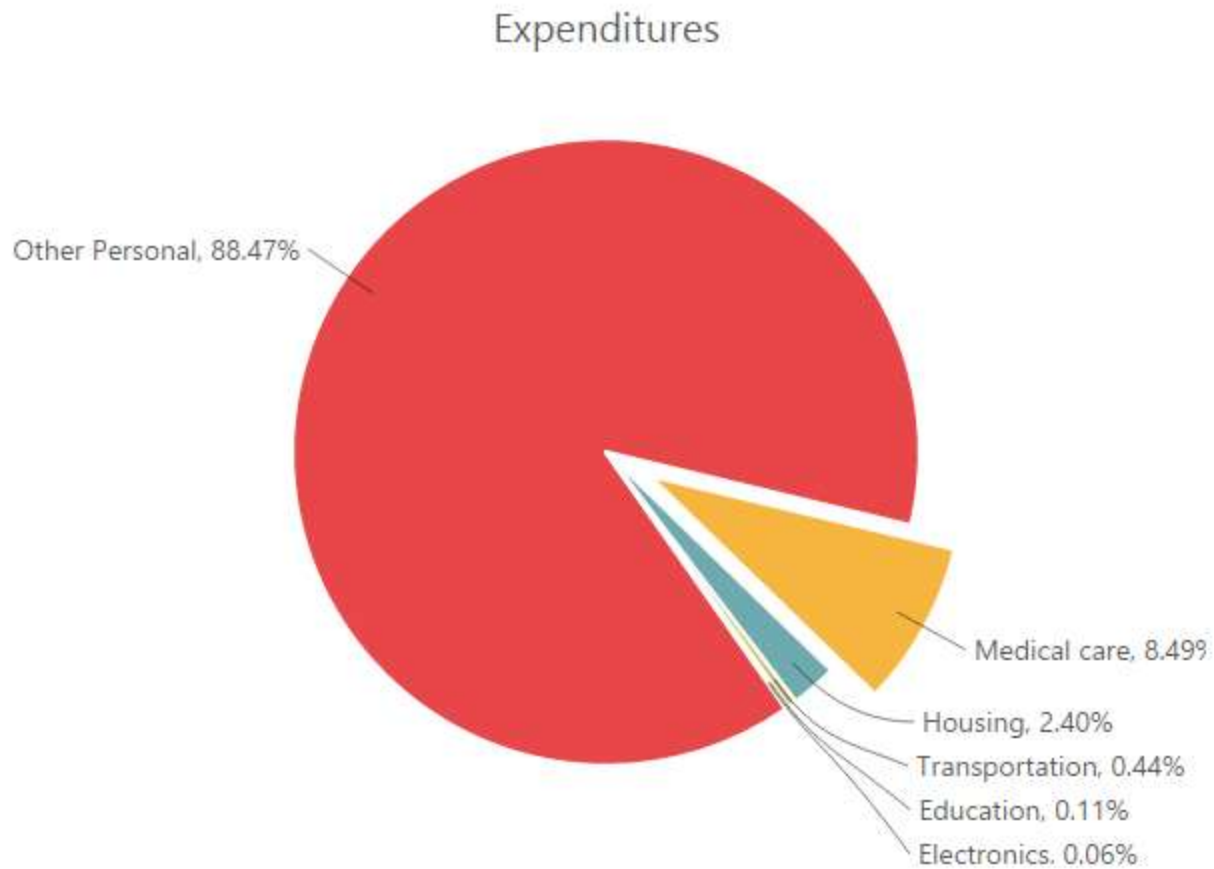


#### *Explode a pie segment*

You can explode a pie segment on the chart load by using the [explodeIndex](#) of the series.

#### **JAVASCRIPT**

```
"use strict";
// ...
var series= [{
  //Set point index value to explode the pie segment.
  explodeIndex: 1,
  // ...
}];
// ...
ReactDOM.render(
  <EJ.Chart id="default_chart_sample_0"
  series={series}
  >
</EJ.Chart>,
  document.getElementById('chart')
);
```



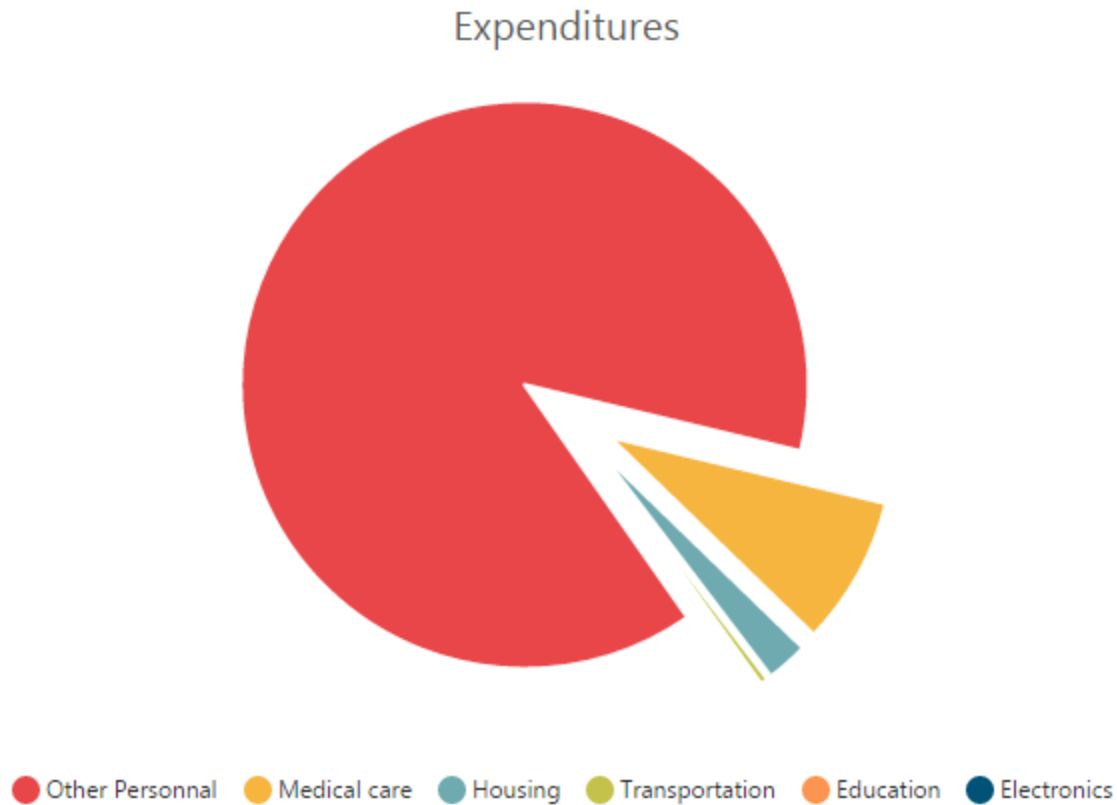
#### *Explode all the segments*

To explode all the segments of the Pie chart, you can enable the [explodeAll](#) property.

#### **JAVASCRIPT**

```
"use strict";
// ...
var series= [{
  //Enable explodeAll property for pie chart.
  explodeAll: true,
  // ...
}];
// ...
ReactDOM.render(
  <EJ.Chart id="default_chart_sample_0"
    series={series}
  >
  </EJ.Chart>,
  document.getElementById('chart')
);
```



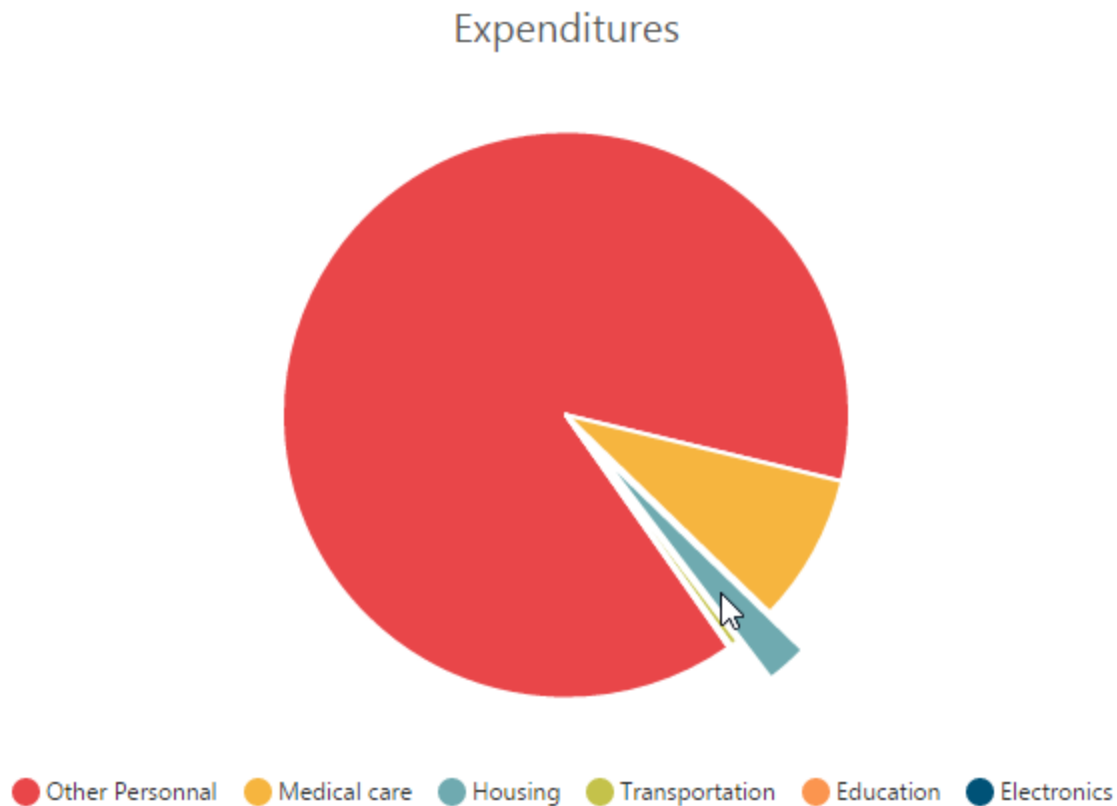


*Explode a pie segment on mouse over*

To explode a pie segment on a mouse over, you can enable the [explode](#) property of the series.

#### JAVASCRIPT

```
"use strict";
// ...
var series= [{
  //Enable pie explode option on mouse over the chart
  explode: true,
  // ...
}];
// ...
ReactDOM.render(
  <EJ.Chart id="default_chart_sample_0"
    series={series}
  >
  </EJ.Chart>,
  document.getElementById('chart')
);
```



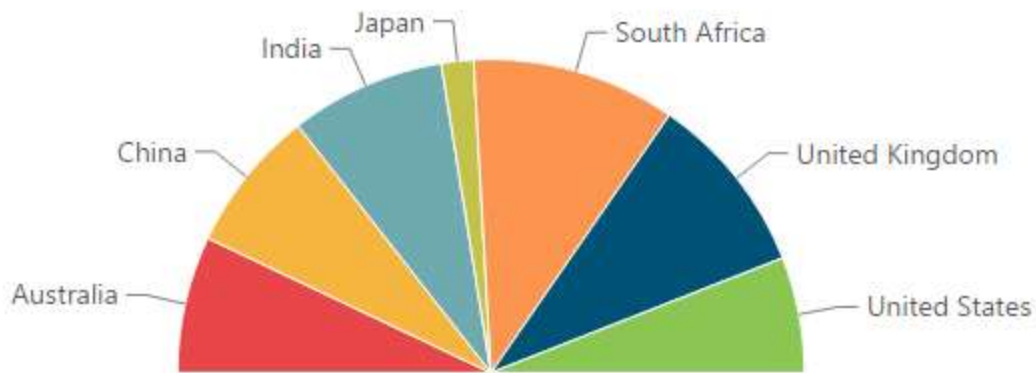
#### Sector of Pie

EjChart allows you to render all the data points/segments in the semi-pie, quarter-pie or in any sector by using the [startAngle](#) and [endAngle](#) options.

#### JAVASCRIPT

```
"use strict";
// ...
var series= [{
  type: 'pie',
  //Set startAngle and endAngle to draw the semi pie chart
  startAngle: -90, endAngle: 90
  // ...
}];
// ...
ReactDOM.render(
  <EJ.Chart id="default_chart_sample_0"
  series={series}
  >
</EJ.Chart>,
  document.getElementById('chart')
);
```

Agricultural land in 2011 (% of land area)



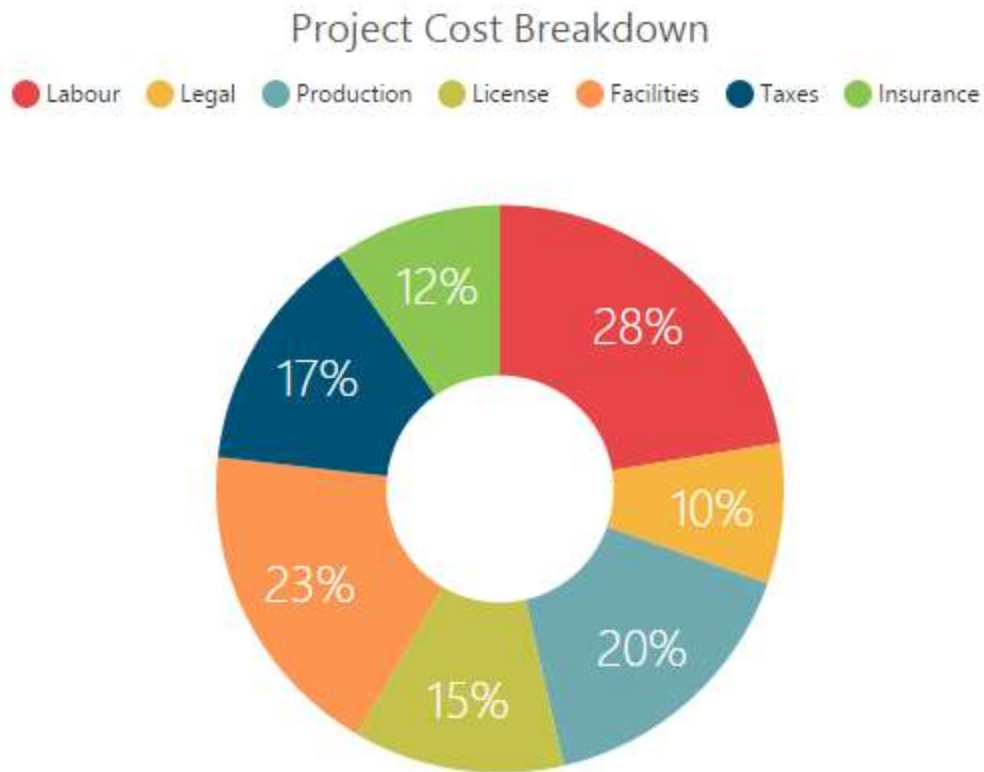
[Click](#) here to view the Semi Pie Chart online demo sample.

### Doughnut Chart

To create a Doughnut chart, you can specify the series [type](#) as “doughnut” in the chart series.

### JAVASCRIPT

```
"use strict";
// ...
var series= [{
  //Set chart type to series
  type: 'doughnut',
  // ...
}];
// ...
ReactDOM.render(
  <EJ.Chart id="default_chart_sample_0"
  series={series}
  >
  </EJ.Chart>,
  document.getElementById('chart')
);
```



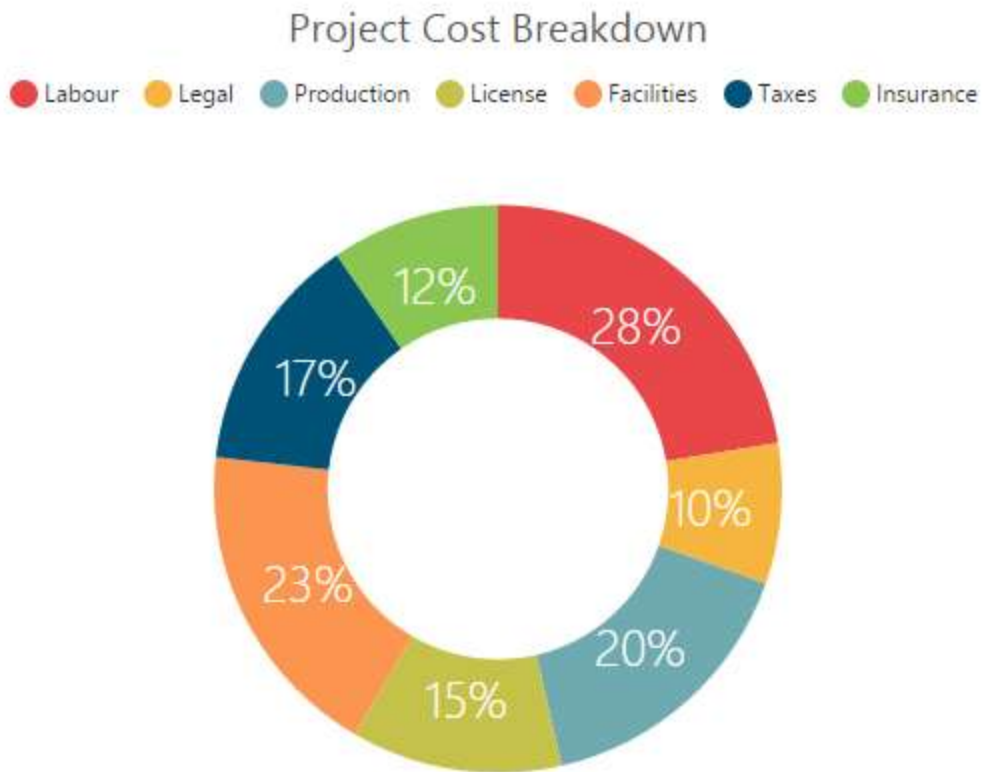
[Click](#) here to view the Doughnut Chart online demo sample.

#### *Change Doughnut inner radius*

You can change the doughnut chart inner radius by using the [doughnutCoefficient](#) with respect to the plot area. It ranges from 0 to 1 and the default value is **0.4**.

#### **JAVASCRIPT**

```
"use strict";
// ...
var series= [{
  //Change doughnut chart coefficient value
  doughnutCoefficient: 0.6
  // ...
}];
// ...
ReactDOM.render(
  <EJ.Chart id="default_chart_sample_0"
    series={series}
  >
    </EJ.Chart>,
  document.getElementById('chart')
);
```



#### *Change the doughnut size*

You can use the [doughnutSize](#) property to change the diameter of the Doughnut chart with respect to the plot area. It ranges from 0 to 1 and the default value is **0.8**.

#### **JAVASCRIPT**

```
"use strict";
// ...
var series= [{
  //Change doughnut chart coefficient value
  doughnutSize: 0.4,
  // ...
}];
// ...
ReactDOM.render(
  <EJ.Chart id="default_chart_sample_0"
  series={series}
  >
</EJ.Chart>,
  document.getElementById('chart')
);
```

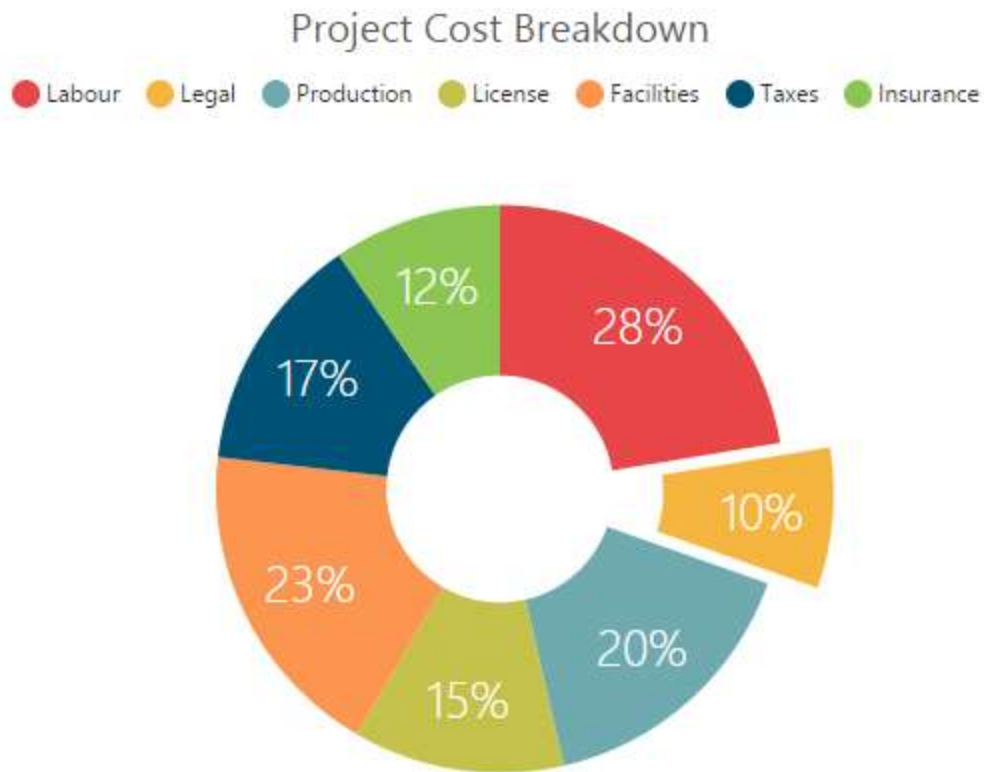


#### *Explode a doughnut segment*

To explode a specific doughnut segment, set the index to be exploded by using the [explodeIndex](#) option of the series.

#### **JAVASCRIPT**

```
"use strict";
// ...
var series= [{
  //Set point index value to explode the doughnut segment.
  explodeIndex: 1,
  // ...
}];
// ...
ReactDOM.render(
  <EJ.Chart id="default_chart_sample_0"
    series={series}
  >
  </EJ.Chart>,
  document.getElementById('chart')
);
```

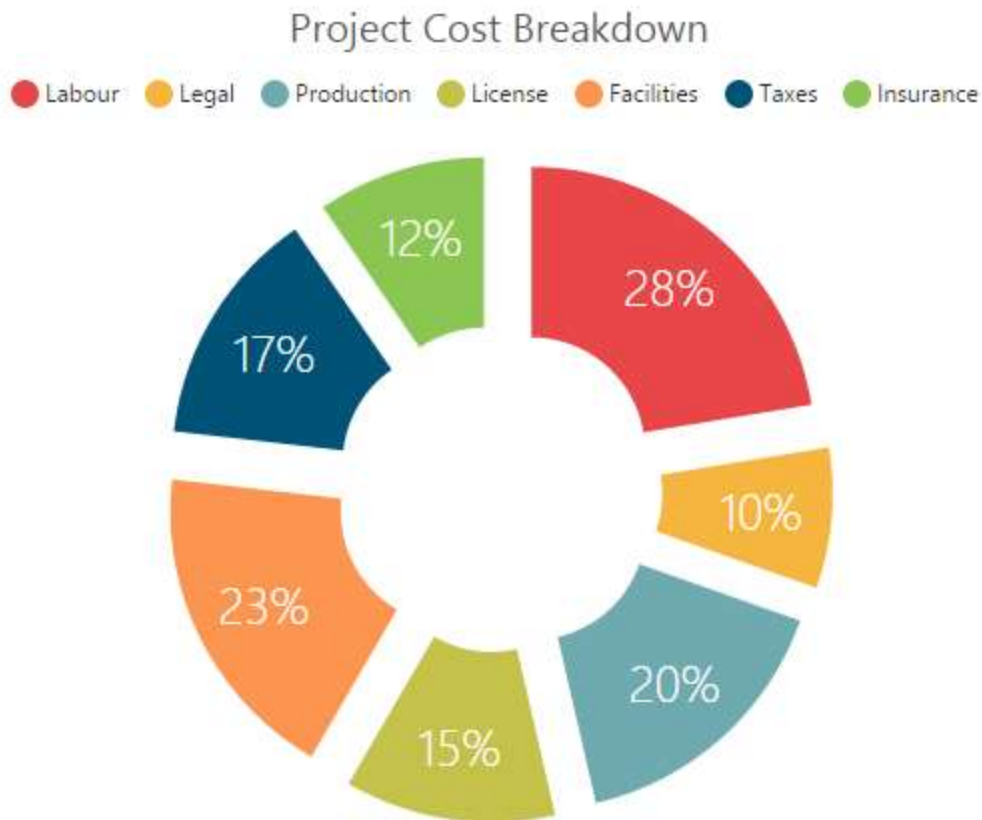


*Explode all the segments*

To explode all the segments, you can enable the [explodeAll](#) property of the series.

#### **JAVASCRIPT**

```
"use strict";
// ...
var series= [{
  //Enable explodeAll property of doughnut chart
  explodeAll: true,
  // ...
}];
// ...
ReactDOM.render(
  <EJ.Chart id="default_chart_sample_0"
  series={series}
  >
</EJ.Chart>,
  document.getElementById('chart')
);
```



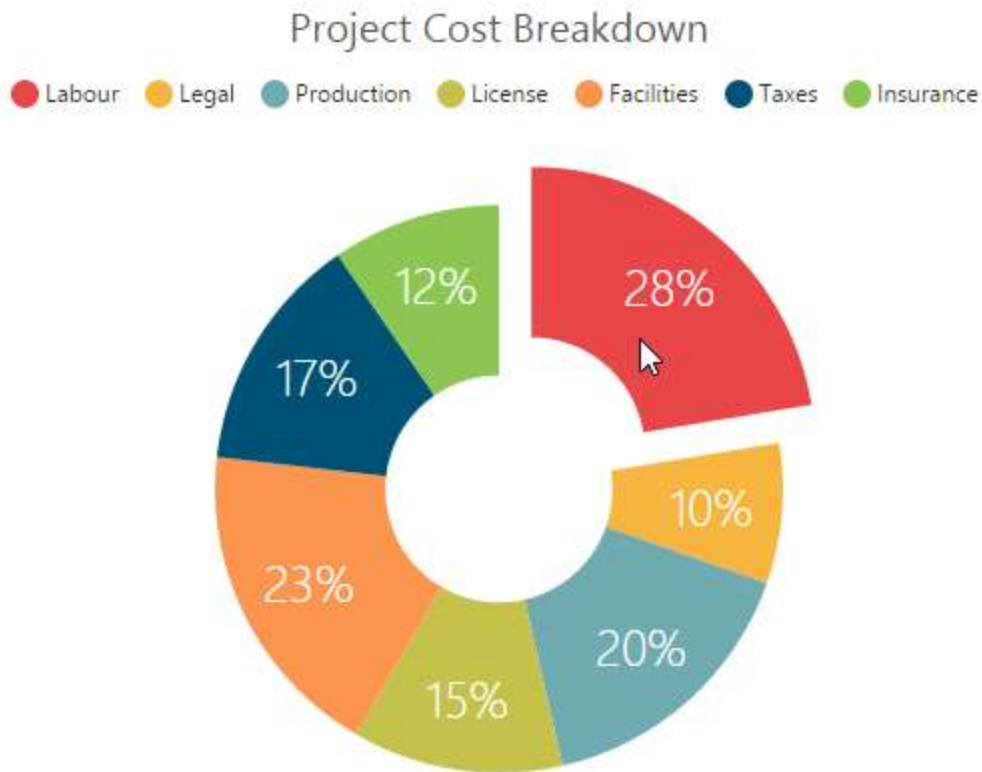
*Explode a doughnut segment on mouse over*

To explode a doughnut segment on a mouse over, you can enable the [explode](#) property of the series.

#### **JAVASCRIPT**

```
"use strict";
// ...
var series= [{
  //Enable doughnut explode option on mouse over the chart
  explode: true,
  // ...
}];
// ...
ReactDOM.render(
  <EJ.Chart id="default_chart_sample_0"
    series={series}
  >
  </EJ.Chart>,
  document.getElementById('chart')
);
```





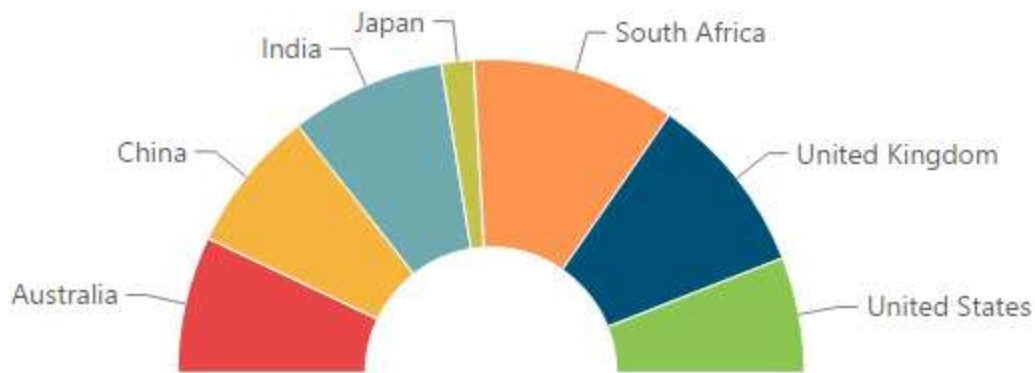
#### Sector of Doughnut

EjChart allows you to render all the data points/segments in the semi-doughnut, quarter- doughnut or in any sector by using the [startAngle](#) and [endAngle](#) options.

#### JAVASCRIPT

```
"use strict";
// ...
var series= [{
  type: 'doughnut',
  //Set startAngle and endAngle to draw the semi pie chart
  startAngle: -90, endAngle: 90
  // ...
}];
// ...
ReactDOM.render(
  <EJ.Chart id="default_chart_sample_0"
  series={series}
  >
  </EJ.Chart>,
  document.getElementById('chart')
);
```

## Agricultural land in 2011 (% of land area)



[Click](#) here to view the Semi Doughnut Chart online demo sample.

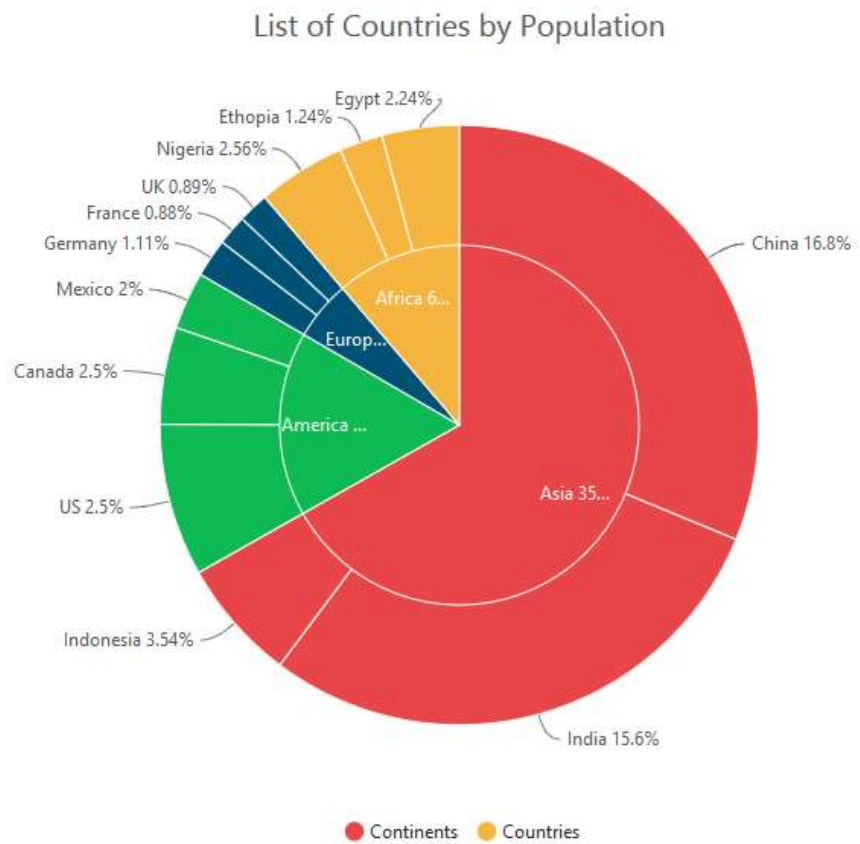
### Multiple Pie Chart

EjChart provides support to render more than one series in pie and in doughnut chart. Radius of each series is calculated based on the radius of the previous series. And in addition legend is displayed according to the list of chart series.

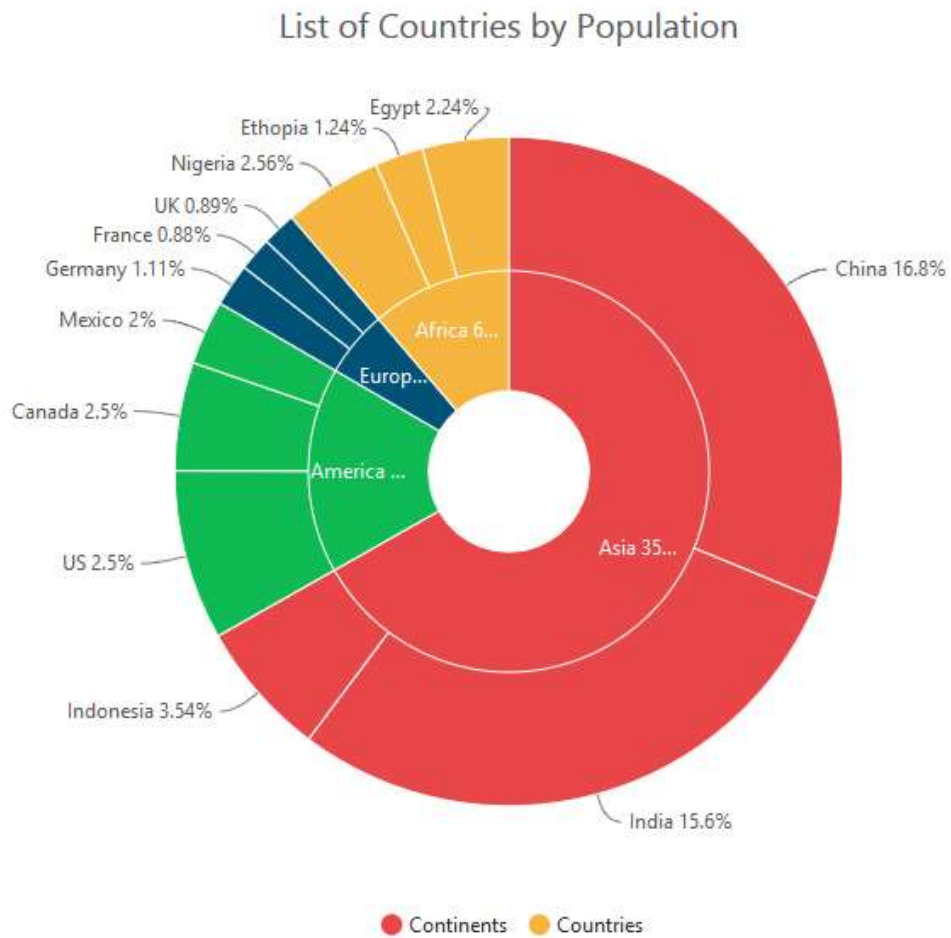
### JAVASCRIPT

```
"use strict";
// ...
var series= [{
  //Adding multiple pie series
  type: "pie",
  //...
}, {
  //Adding multiple pie series
  type: "pie",
  //...
}];
// ...
ReactDOM.render(
  <EJ.Chart id="default_chart_sample_0"
  series={series}
  >
</EJ.Chart>,
  document.getElementById('chart')
);
```

### Multiple Pie



### Multiple Doughnut



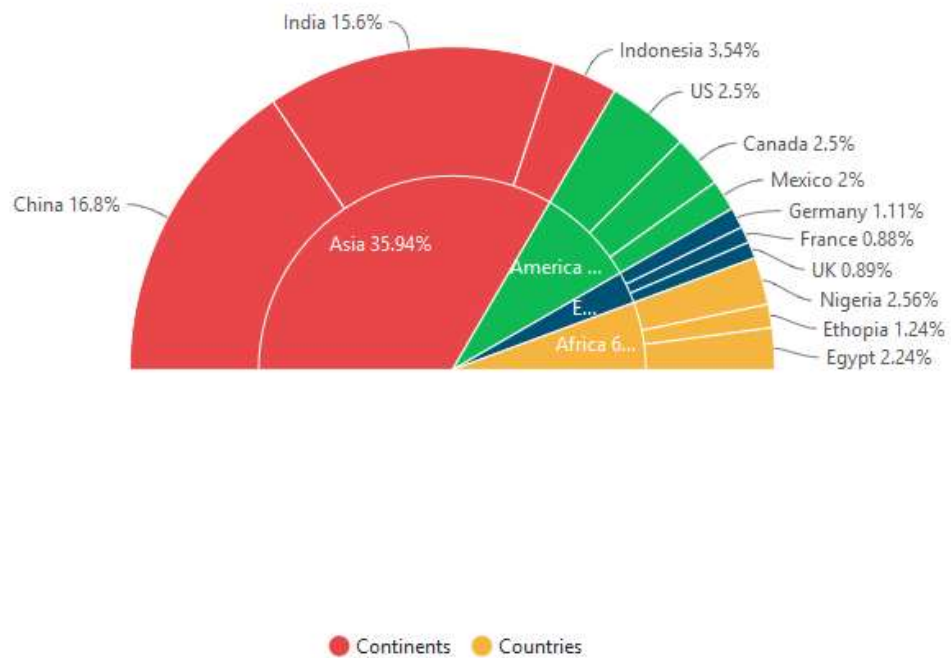
[Click](#) here to view the Multiple Pie chart online demo sample.

*Start and End Angle Support*

In the Multiple Pie chart, the start and end angle property is also supported.

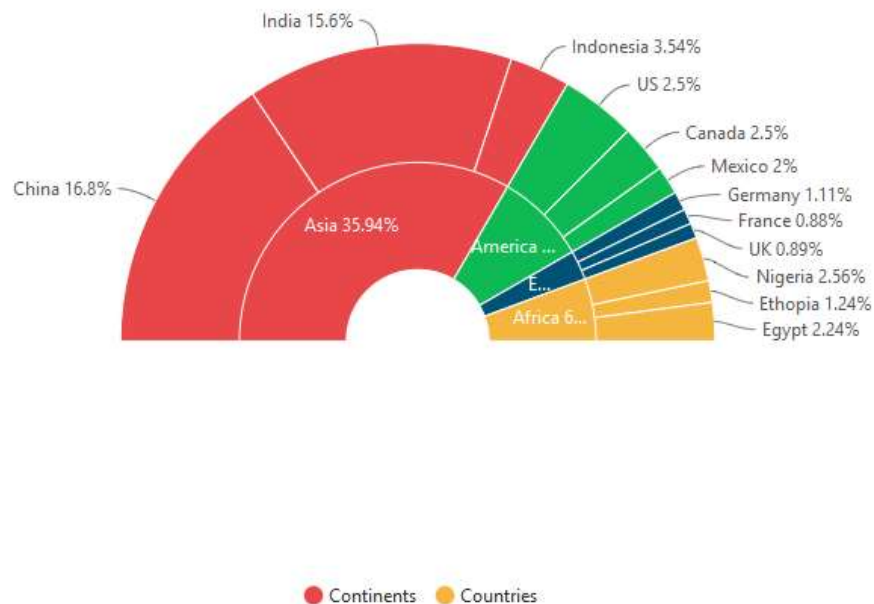
### Sector of Multiple Pie

## List of Countries by Population



## Sector of Multiple Doughnut

List of Countries by Population

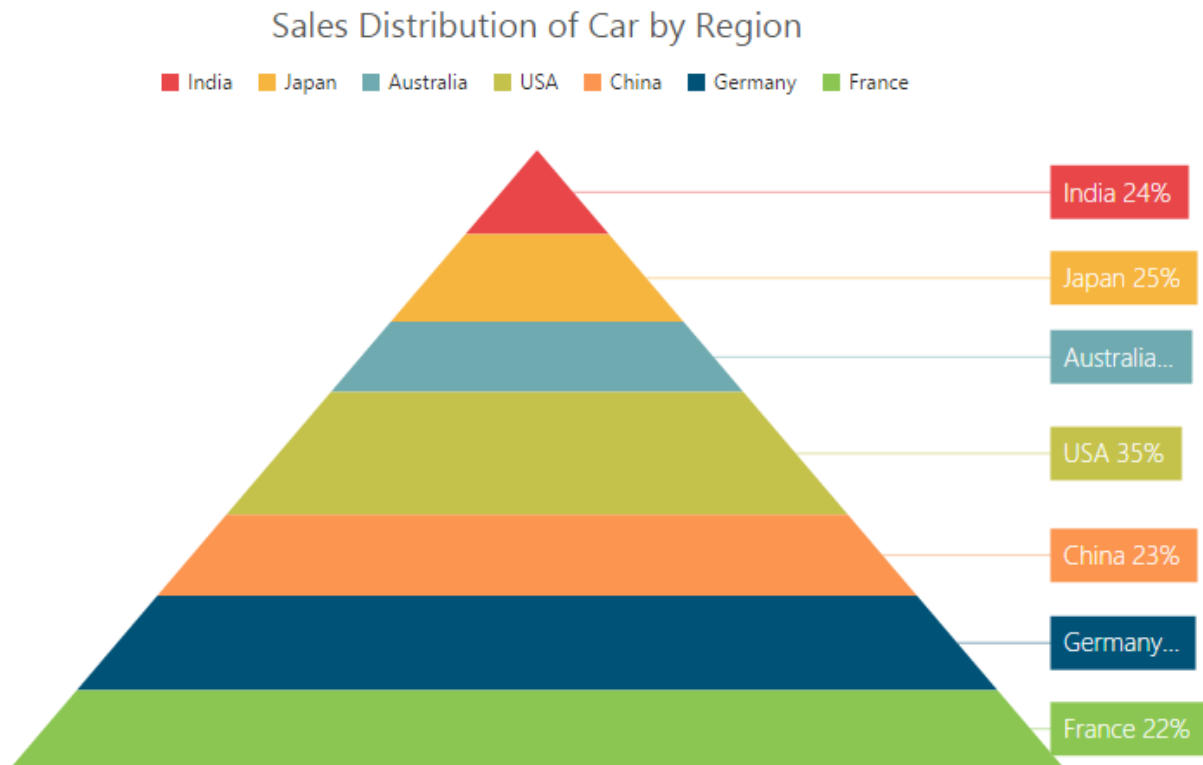


### Pyramid Chart

To create a Pyramid chart, you can specify the series [type](#) as “**pyramid**” in the chart series.

#### JAVASCRIPT

```
"use strict";
// ...
var series= [{
//Set chart type to series
type: 'pyramid',
// ...
}];
// ...
ReactDOM.render(
<EJ.Chart id="default_chart_sample_0"
series={series}
>
</EJ.Chart>,
document.getElementById('chart')
);
```



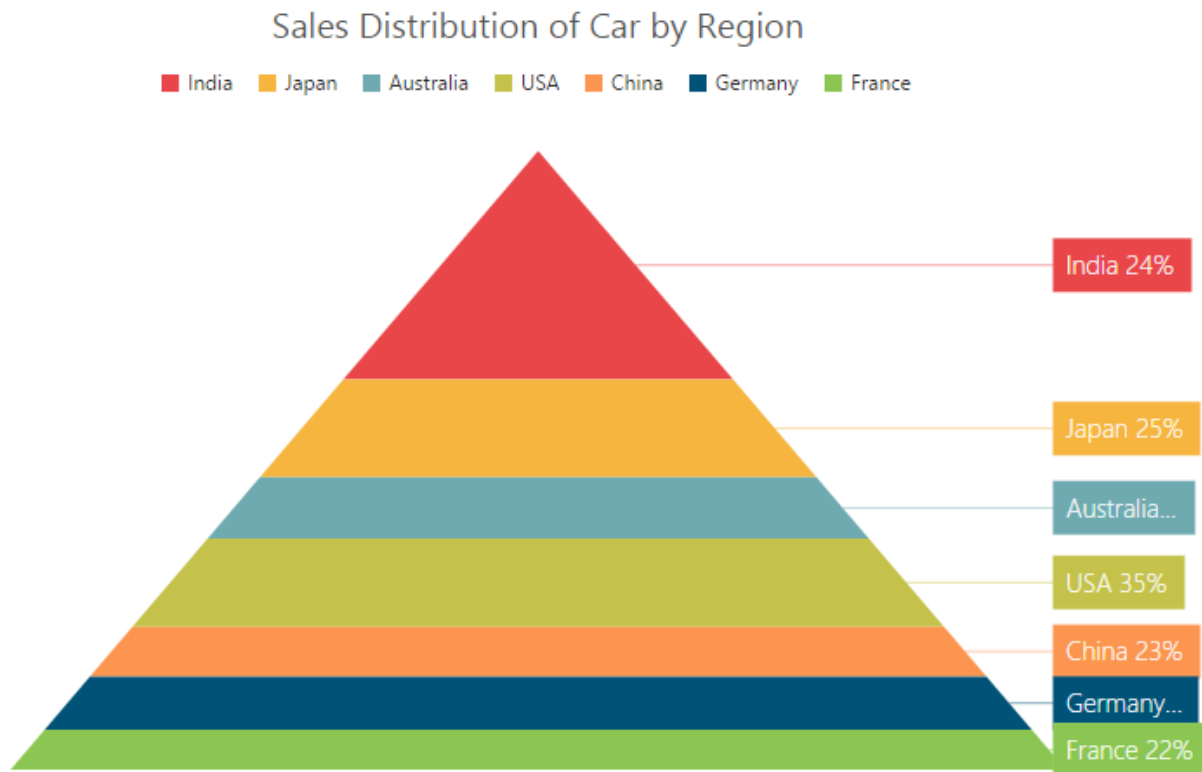
[Click](#) here to view the Pyramid Chart online demo sample.

#### Pyramid Mode

Pyramid mode has two types, *linear* and *surface* respectively. The default “**pyramidMode**” type is “linear”.

#### JAVASCRIPT

```
"use strict";
// ...
var series= [{
  //Change pyramid mode
  pyramidMode: 'surface',
  // ...
}];
// ...
ReactDOM.render(
  <EJ.Chart id="default_chart_sample_0"
  series={series}
  >
  </EJ.Chart>,
  document.getElementById('chart')
);
```



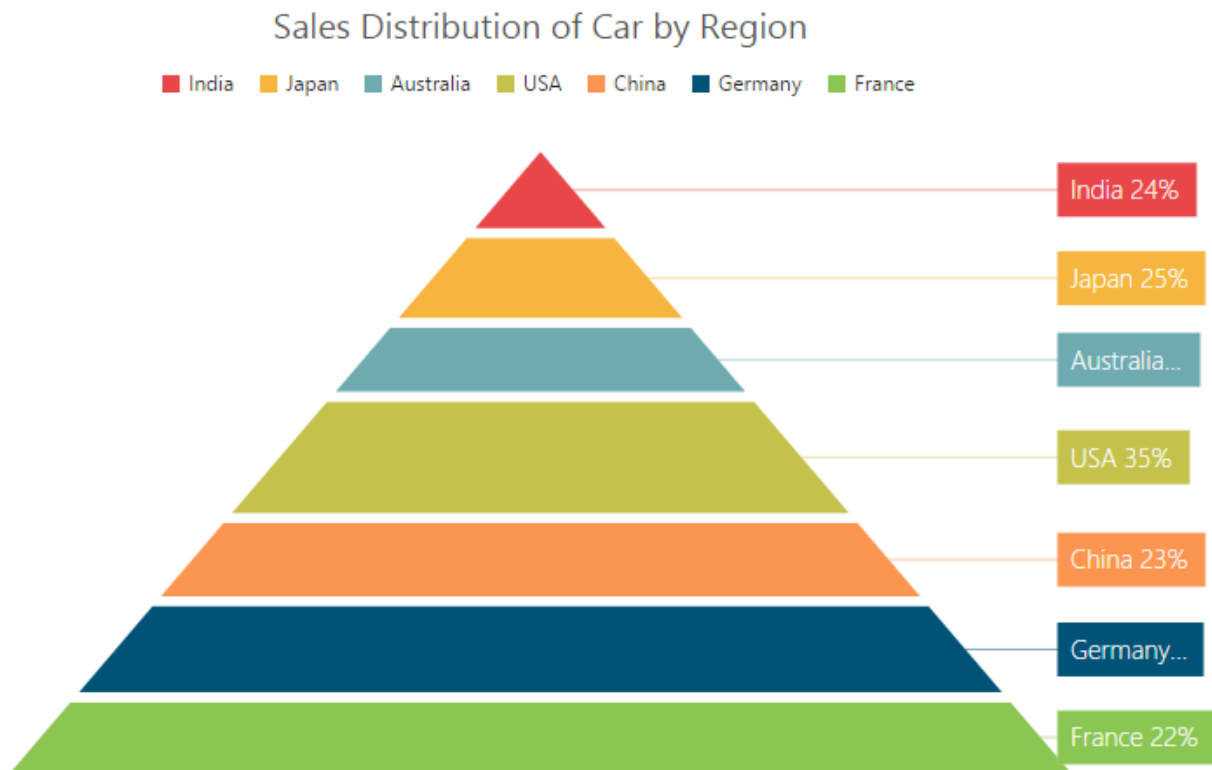
#### Gap between the segments

You can control the gap between the segments by using the [gapRatio](#) option of the series. Its ranges from 0 to 1.

#### JAVASCRIPT

```
"use strict";
// ...
var series= [{
//Set gapRatio value to pyramid chart
gapRatio: 0.1,
// ...
}];
// ...
ReactDOM.render(
<EJ.Chart id="default_chart_sample_0"
series={series}
>
</EJ.Chart>,
document.getElementById('chart')
);
```



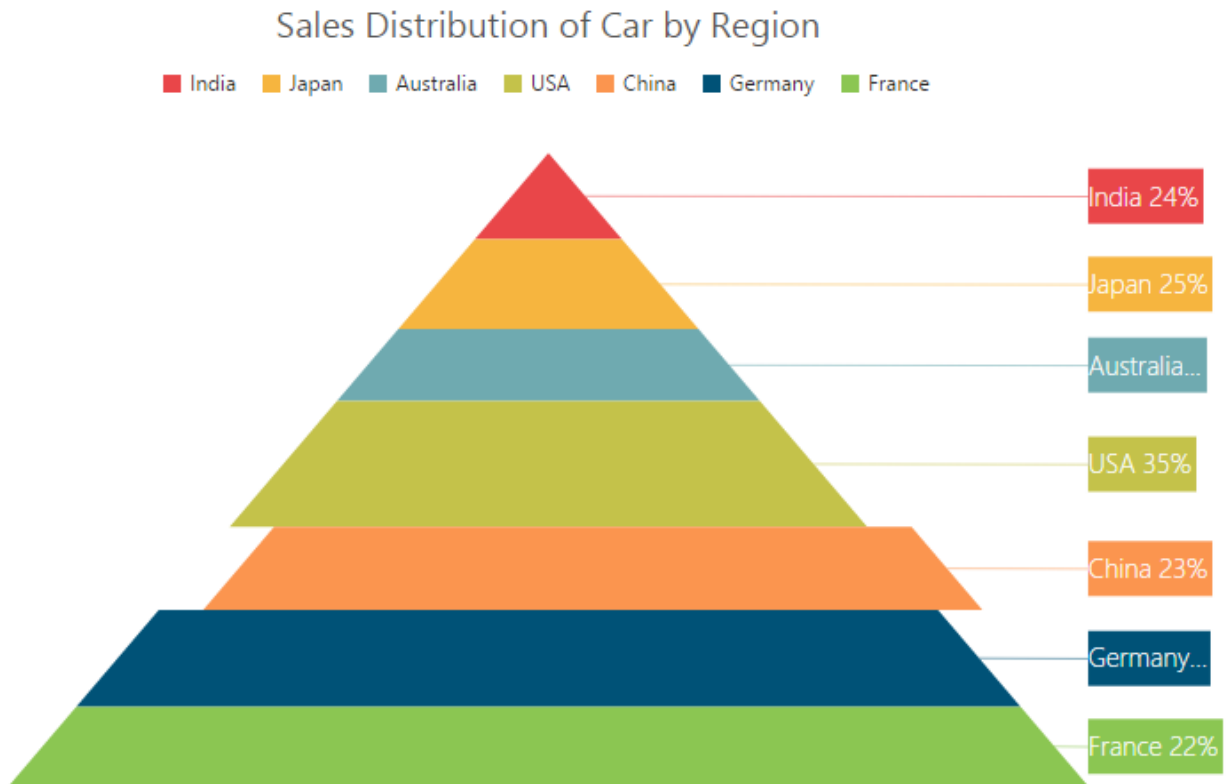


#### Explode a pyramid segment

You can explode a pyramid segment on the chart load by using the [explodeIndex](#) of the series.

#### JAVASCRIPT

```
"use strict";
// ...
var series= [{
  //Set point index value to explode the pyramid segment.
  explodeIndex: 4,
  // ...
}];
// ...
ReactDOM.render(
  <EJ.Chart id="default_chart_sample_0"
  series={series}
  >
</EJ.Chart>,
  document.getElementById('chart')
);
```

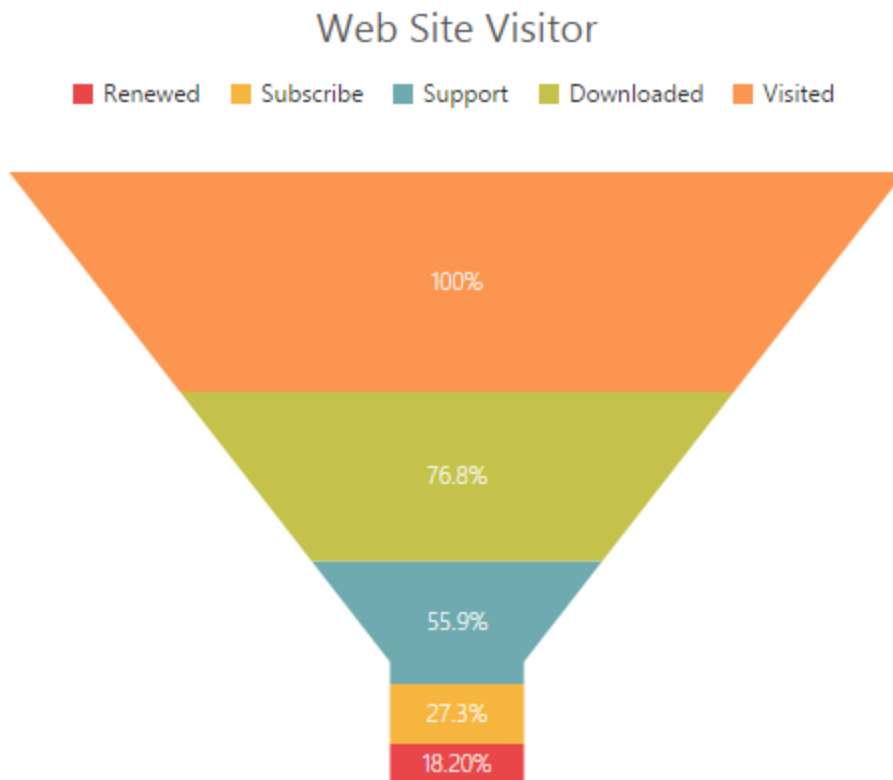


#### Funnel Chart

You can create a funnel chart by setting the series [type](#) as “**funnel**” in the chart series.

#### JAVASCRIPT

```
"use strict";
// ...
var series= [{
  //Set chart type to series
  type: 'funnel',
  // ...
}];
// ...
ReactDOM.render(
  <EJ.Chart id="default_chart_sample_0"
  series={series}
  >
  </EJ.Chart>,
  document.getElementById('chart')
);
```



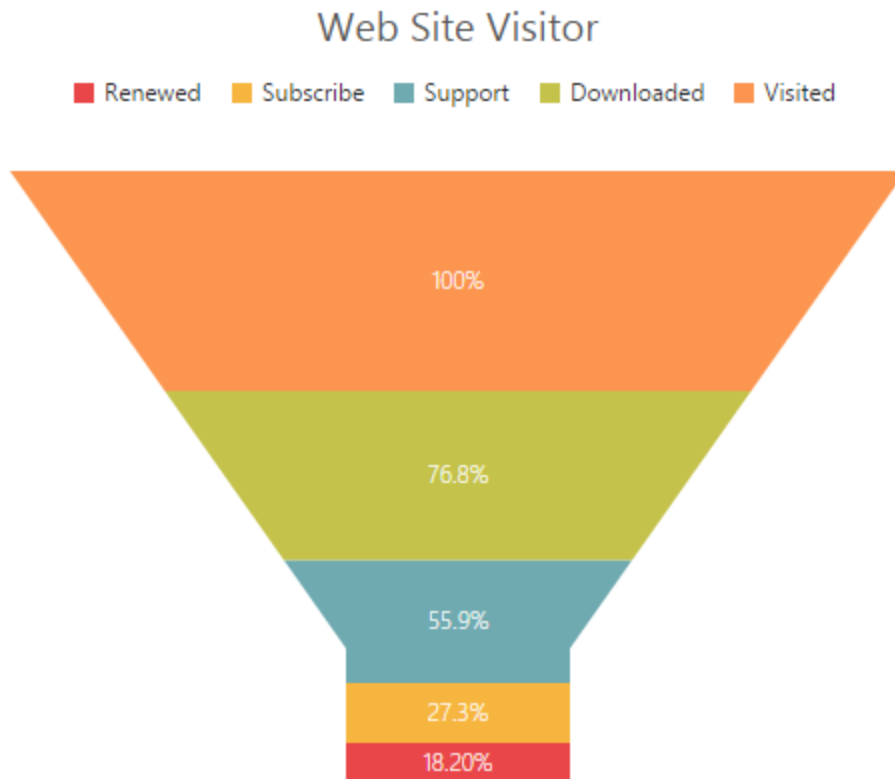
[Click](#) here to view the Funnel Chart online demo sample.

#### *Change the funnel width and height*

Funnel segments height and width is calculated from the chart size, by default. You can change this height and width directly without changing the chart size by using the [funnelHeight](#) and [funnelWidth](#) property of the series.

#### **JAVASCRIPT**

```
"use strict";
// ...
var series= [{
//Change funnel height and width
funnelHeight:"22%",
funnelWidth:"25%",
// ...
}];
// ...
ReactDOM.render(
<EJ.Chart id="default_chart_sample_0"
series={series}
>
</EJ.Chart>,
document.getElementById('chart')
);
```

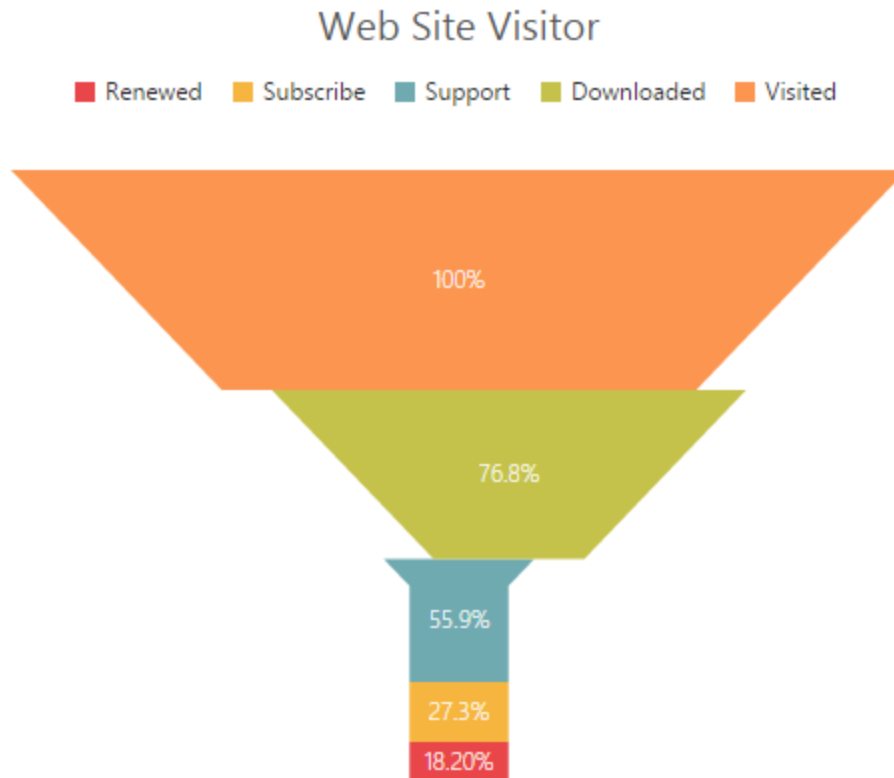


#### *Explode a funnel segment*

You can explode a funnel segment on the chart load by using the [explodeIndex](#) of the series.

#### **JAVASCRIPT**

```
"use strict";
// ...
var series= [{
  //Set point index value to explode the funnel segment.
  explodeIndex: 3,
  // ...
}];
// ...
ReactDOM.render(
  <EJ.Chart id="default_chart_sample_0"
  series={series}
  >
</EJ.Chart>,
  document.getElementById('chart')
);
```



### Bubble Chart

To create a Bubble chart, you can set the series [type](#) as “**bubble**” in the chart series. Bubble chart requires 3 fields (*x*, *y* and *size*) to plot a point. Here, **size** is used to specify the size of each bubble segment.

### JAVASCRIPT

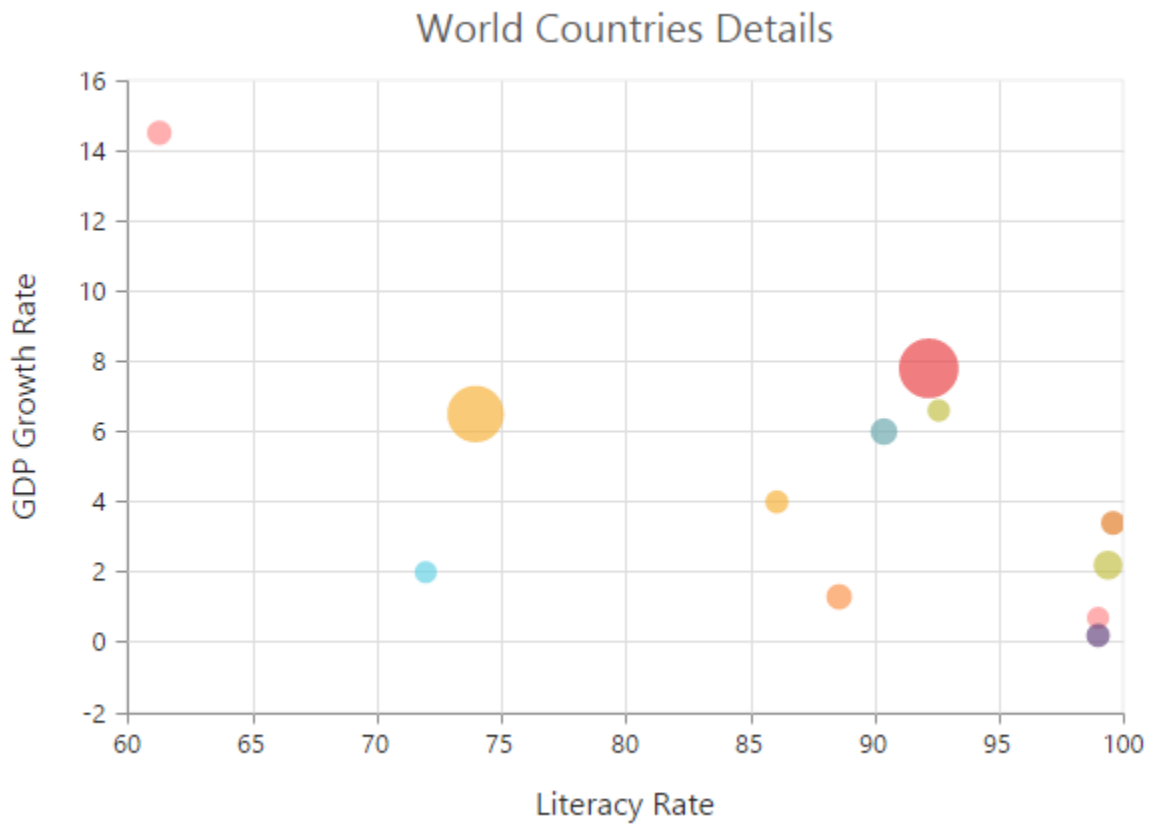
```

"use strict";
var chartData = [
{ month: 'Jan', sales: 35, profit:1.34 },
{ month: 'Feb', sales: 28, profit:1.05 },
{ month: 'Mar', sales: 34, profit:0.45 },
{ month: 'Apr', sales: 32, profit:1.10 },
// ...
];
// ...
var series= [{
//Set chart type to series
type: 'bubble',
//Add datasource and set xName, yName and size to bubble chart
dataSource: chartData,
xName: "month",
yName: "sales",
size: "profit" ,
}];
// ...
ReactDOM.render(
<EJ.Chart id="default_chart_sample_0"
  
```

```

series={series}
>
</EJ.Chart>,
document.getElementById('chart')
);

```



[Click](#) here to view the Bubble Chart online demo sample.

#### Scatter

To create a Scatter chart, you can set the series [type](#) as “**scatter**” in the chart series.

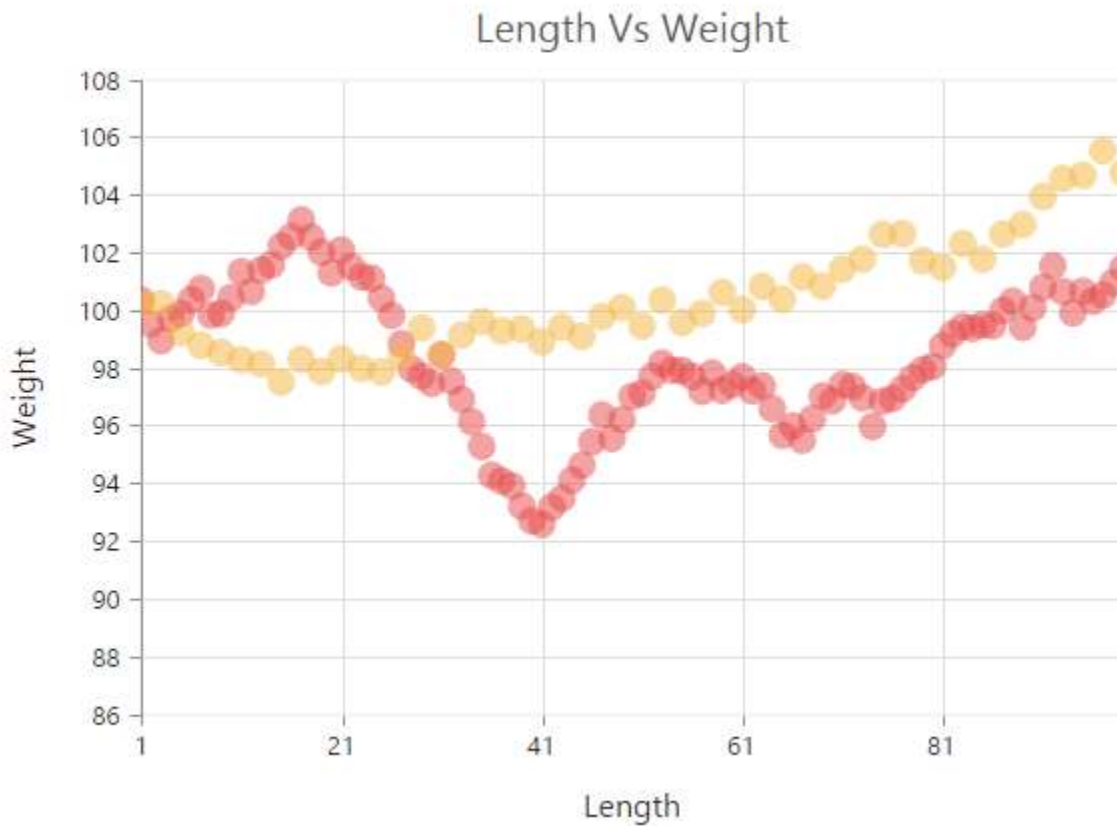
#### JAVASCRIPT

```

"use strict";
// ...
var series= [{
//Set chart type to series
type: 'scatter',
// ...
}];
// ...
ReactDOM.render(
<EJ.Chart id="default_chart_sample_0"
series={series}
>

```

```
</EJ.Chart>,
document.getElementById('chart')
);
```



[Click](#) here to view the Scatter Chart online demo sample.

*Customize the scatter chart*

You can change the scatter size by using the [size](#) property of the series marker. And you can change the scatter color by using the series [fill](#) property.

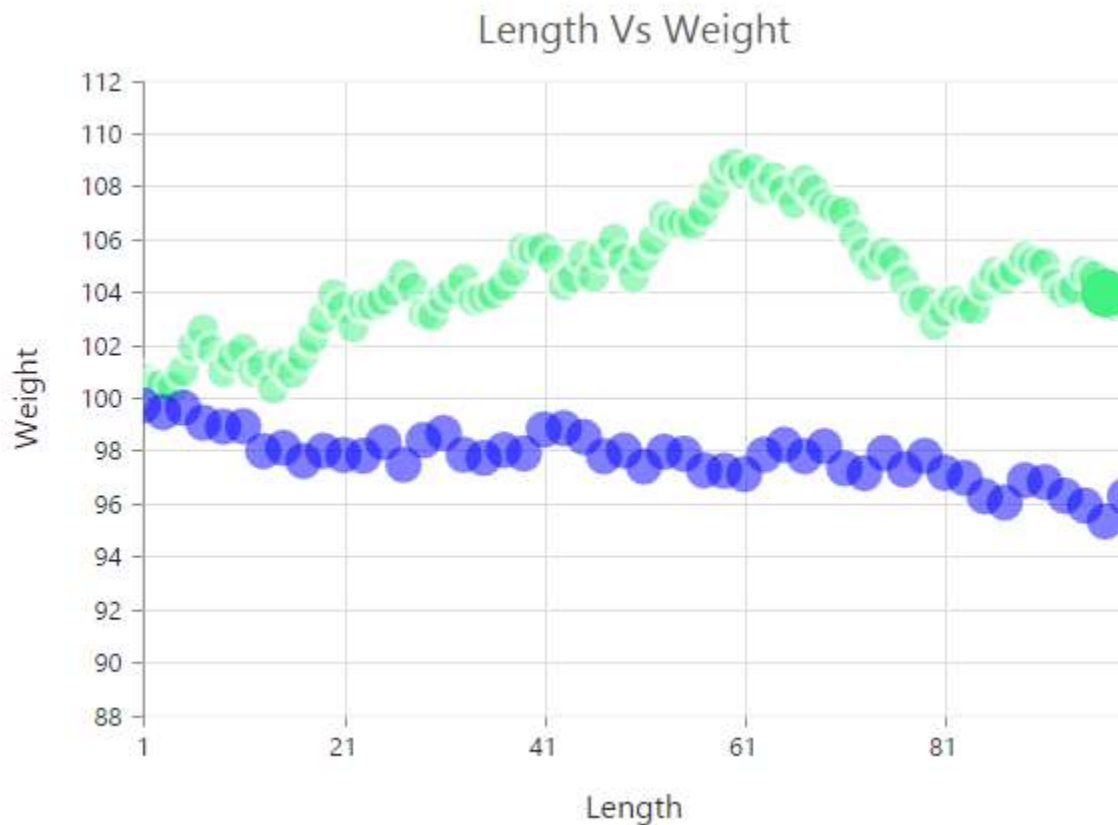
#### JAVASCRIPT

```
"use strict";
// ...
var series= [{
// ...
marker: {
//Change the scatter size
size: { height: 13, width: 13 }
},
//Set fill color to scatter series
fill: "#41F282",
// ...
}];
// ...
ReactDOM.render(
<EJ.Chart id="default_chart_sample_0"
```

```

series={series}
>
</EJ.Chart>,
document.getElementById('chart')
);

```



### HiLoOpenClose Chart

To create a HiLoOpenClose chart, you can set the series [type](#) as “**hiloopenclose**” in the chart series. HiLoOpenClose chart requires 5 fields ([x](#), [high](#), [low](#), [open](#) and [close](#)) to plot a segment.

### JAVASCRIPT

```

"use strict";
var chartData = [
{ month: 'Jan', high: 38, low: 10, open: 38, close: 29 },
{ month: 'Feb', high: 28, low: 15, open: 18, close: 27 },
{ month: 'Mar', high: 54, low: 35, open: 38, close: 49 },
{ month: 'Apr', high: 52, low: 21, open: 35, close: 29 },
// ...
];
// ...
var series= [{
//Set chart type to series
type: 'hiloopenclose',
//Add datasource and set xName, high and low to hilo chart
dataSource: chartData,

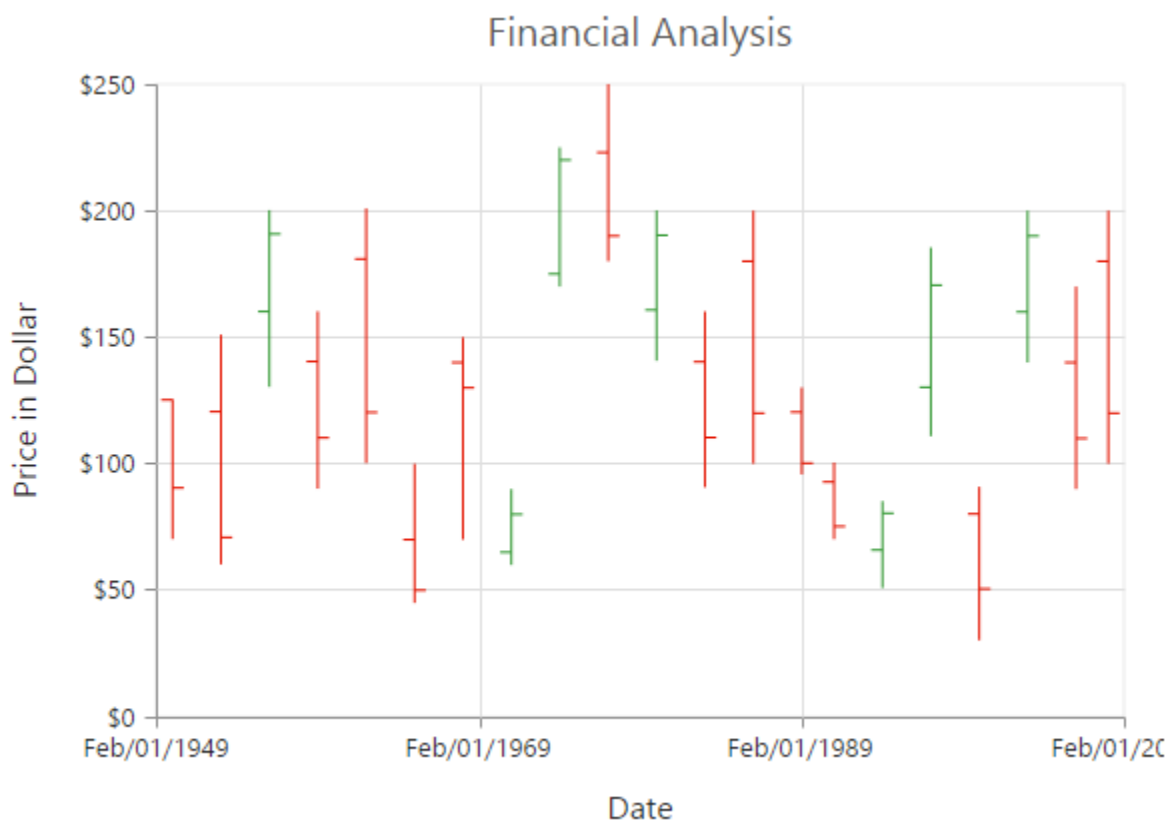
```



```

xName: "month",
high: "high",
low: "low",
open: 'open',
close: 'close',
});
// ...
ReactDOM.render(
  <EJ.Chart id="default_chart_sample_0"
    series={series}
  >
    </EJ.Chart>,
    document.getElementById('chart')
  );

```



[Click](#) here to view the HiLoOpenClose Chart online demo sample.

#### *DrawMode*

You can change the HiLoOpenClose chart [drawMode](#) to [open](#), [close](#) or *both*. The default value of [drawMode](#) is **"both"**.

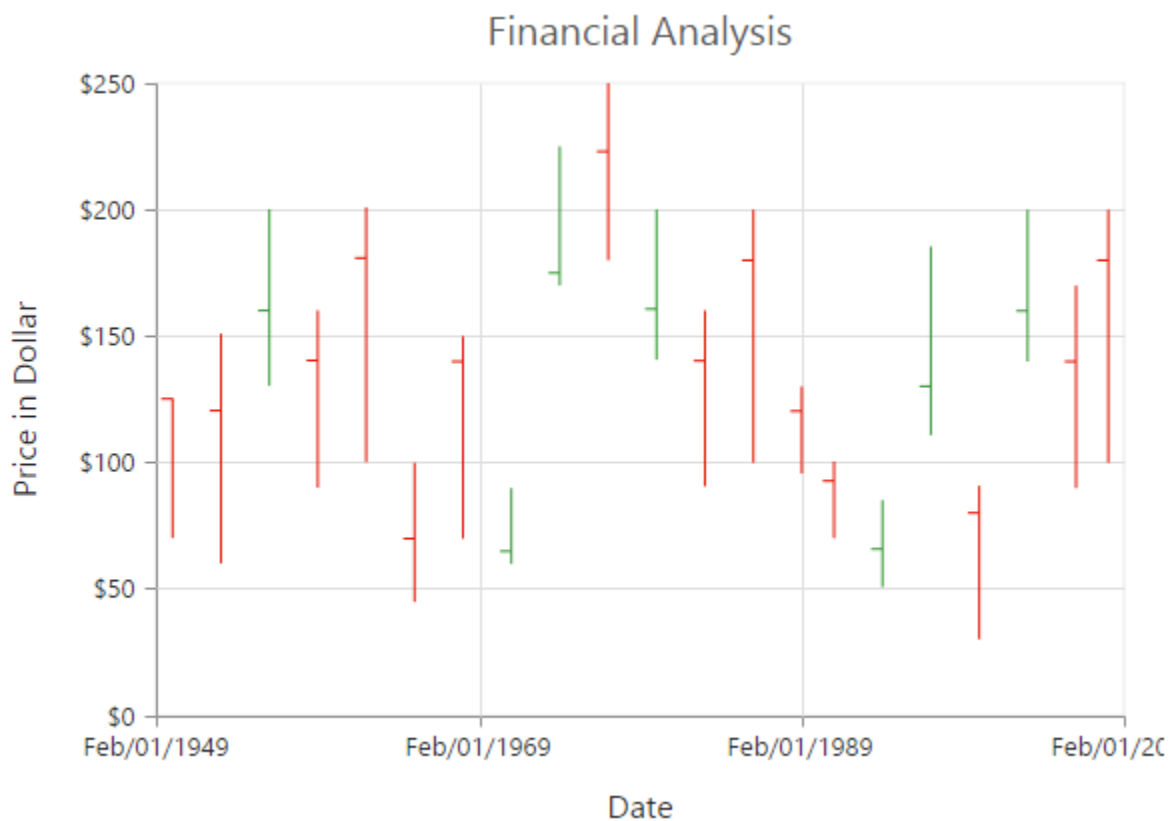
#### **JAVASCRIPT**

```

"use strict";
// ...
var series= [{
// ...

```

```
//Change the OHLC drawMode type
drawMode: 'open',
}];
// ...
ReactDOM.render(
  <EJ.Chart id="default_chart_sample_0"
  series={series}
  >
  </EJ.Chart>,
  document.getElementById('chart')
);
```



#### Bull and Bear Color

HiLoOpenClose chart [bullFillColor](#) is used to specify a fill color for the segments that indicates an increase in stock price in the measured time interval and [bearFillColor](#) is used to specify a fill color for the segments that indicates a decrease in the stock price in the measured time interval.

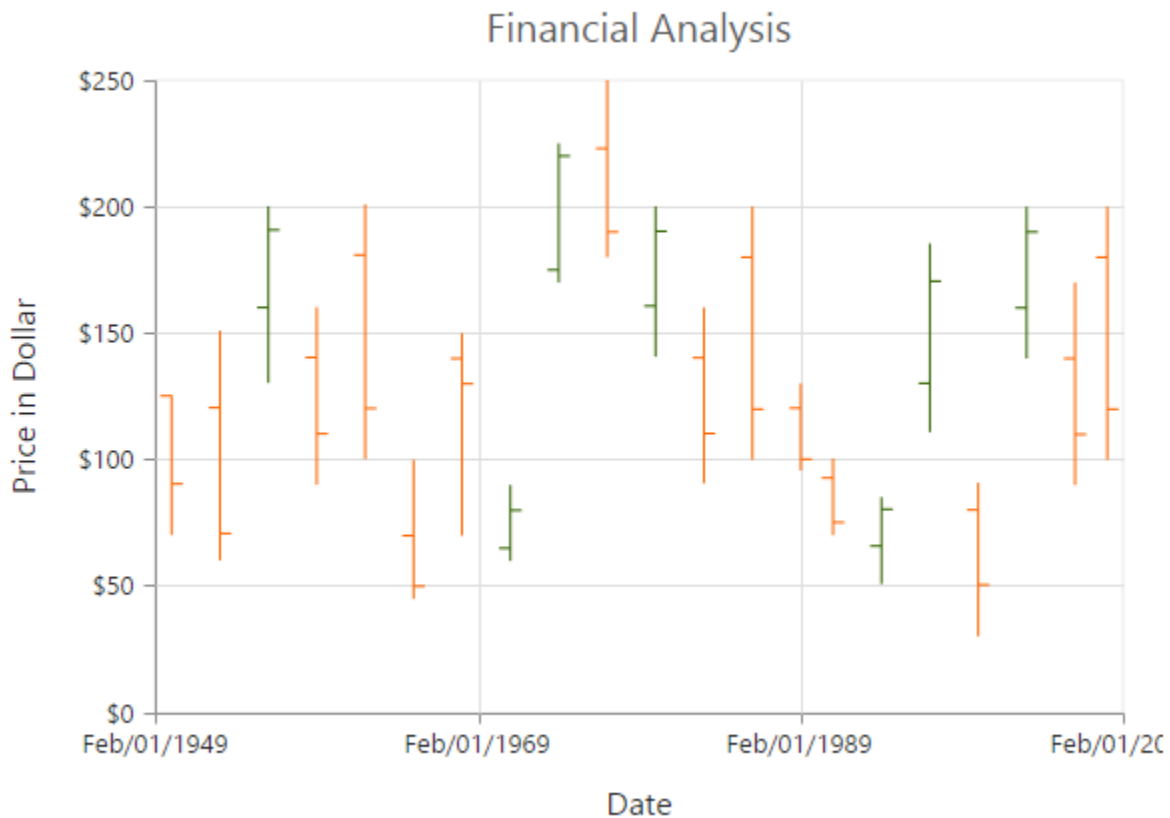
#### JAVASCRIPT

```
"use strict";
// ...
var series= [{
  //Change bullFill and bearFill color of hiloopenclose chart
  bullFillColor: '#FF6600',
  bearFillColor: '#336600',
  // ...
}];
```

```

    }];
    // ...
    ReactDOM.render(
      <EJ.Chart id="default_chart_sample_0"
        series={series}
      >
      </EJ.Chart>,
      document.getElementById('chart')
    );

```



## Candle

You can create a Candle chart by specifying the series [type](#) as “**candle**” in the chart series. Candle chart requires 5 fields ([x](#), [high](#), [low](#), [open](#) and [close](#)) to plot a segment.

## JAVASCRIPT

```

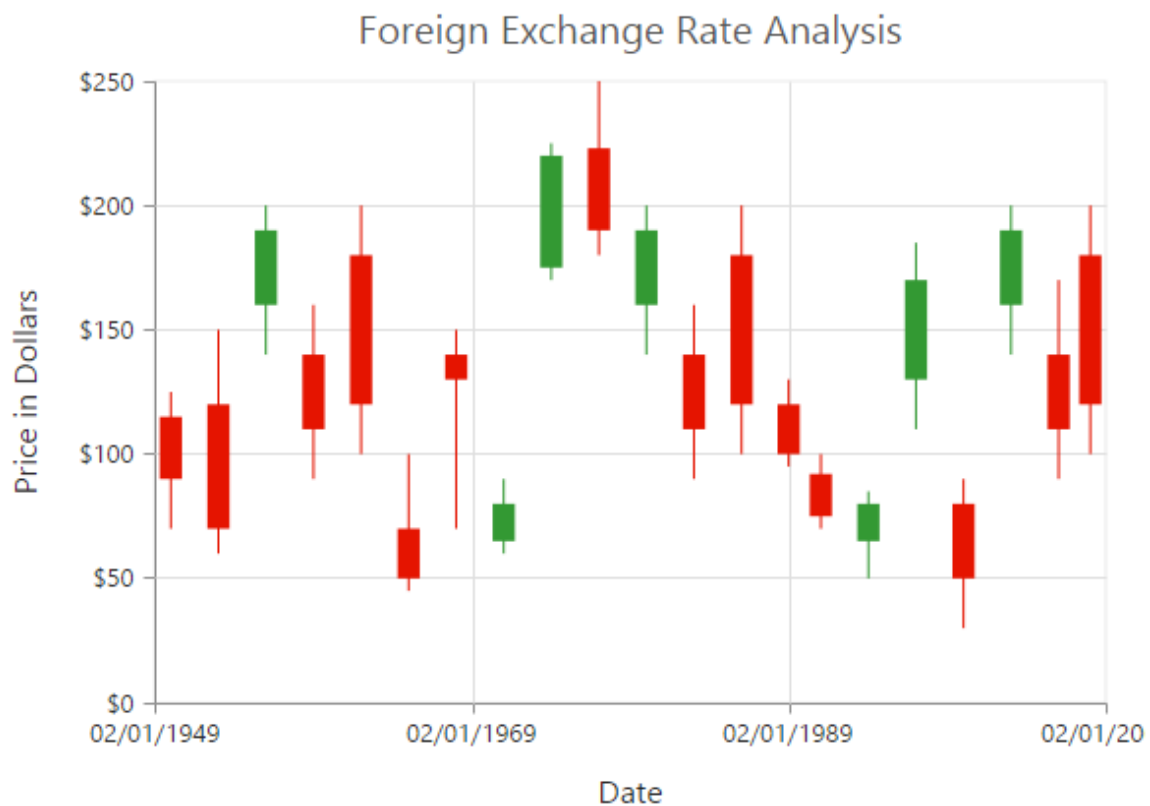
"use strict";
var chartData = [
  { month: 'Jan', high: 38, low: 10, open: 38, close: 29 },
  { month: 'Feb', high: 28, low: 15, open: 18, close: 27 },
  { month: 'Mar', high: 54, low: 35, open: 38, close: 49 },
  { month: 'Apr', high: 52, low: 21, open: 35, close: 29 },
  // .....
];
// ...
var series= [{
  //Set chart type to series

```

```

type: 'candle',
//Add datasource and set xName, high and low to hilo chart
dataSource: chartData,
xName: "month",
high: "high",
low: "low",
open: 'open',
close: 'close'
}];
// ...
ReactDOM.render(
<EJ.Chart id="default_chart_sample_0"
series={series}
>
</EJ.Chart>,
document.getElementById('chart')
);

```



[Click](#) here to view the Candle Chart online demo sample.

#### *Bull and Bear Color*

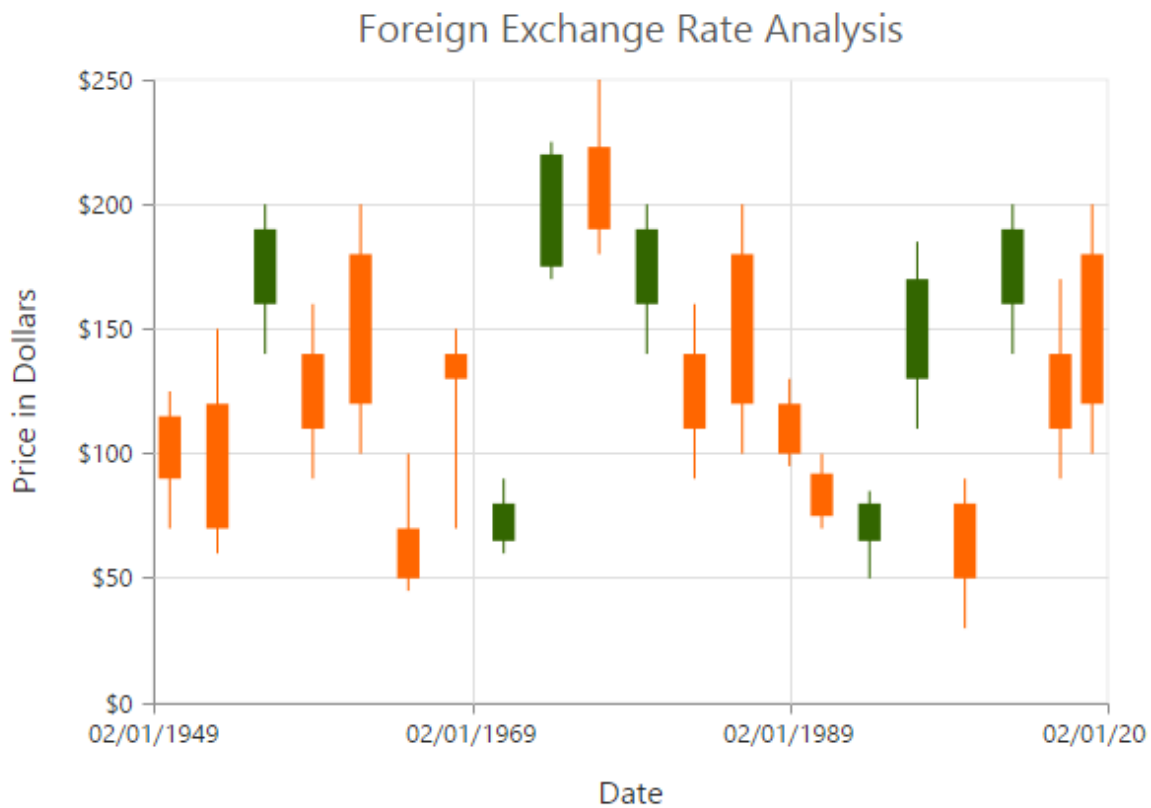
Candle chart [bullFillColor](#) is used to specify a fill color for the segments that indicates an increase in the stock price in the measured time interval and [bearFillColor](#) is used to specify a fill color for the segments that indicates a decrease in the stock price in the measured time interval.

**JAVASCRIPT**

```

"use strict";
// ...
var series= [{
  //Change bullFill and bearFill color of candle chart
  bullFillColor: '#FF6600',
  bearFillColor: '#336600',
  // ...
}];
// ...
ReactDOM.render(
  <EJ.Chart id="default_chart_sample_0"
  series={series}
  >
  </EJ.Chart>,
  document.getElementById('chart')
);

```

**HiLo**

HiLo chart is created by setting the series [type](#) as “**hilo**” in the chart series. HiLo chart requires 3 fields ([x](#), [high](#) and [low](#)) to plot a segment.

**JAVASCRIPT**

```

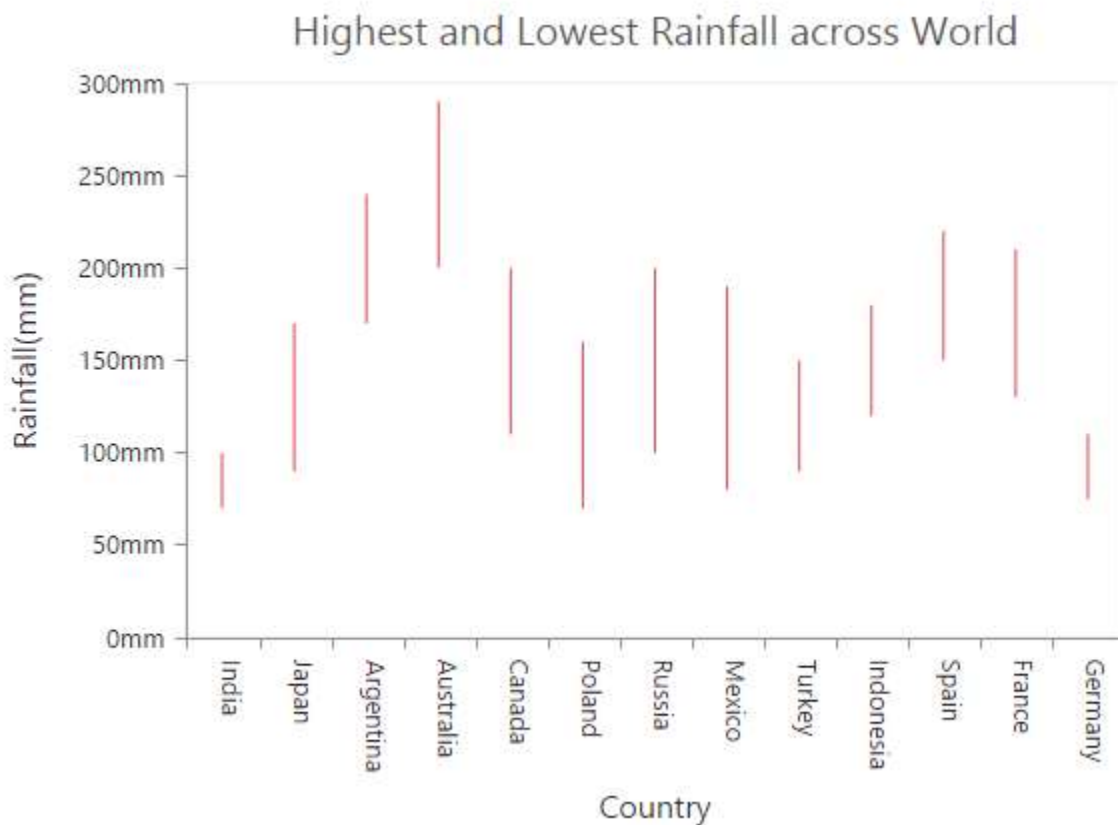
"use strict";
var chartData = [

```

```

{ month: 'Jan', high: 38, low: 34 },
{ month: 'Feb', high: 28, low: 15 },
{ month: 'Mar', high: 54, low: 45 },
{ month: 'Apr', high: 32, low: 21 },
// ...
];
// ...
var series= [{
//Set chart type to series
type: 'hilo',
//Add datasource and set xName, high and low to hilo chart
dataSource: chartData,
xName: "month",
high: "high",
low: "low",
}];
// ...
ReactDOM.render(
<EJ.Chart id="default_chart_sample_0"
series={series}
>
</EJ.Chart>,
document.getElementById('chart')
);

```



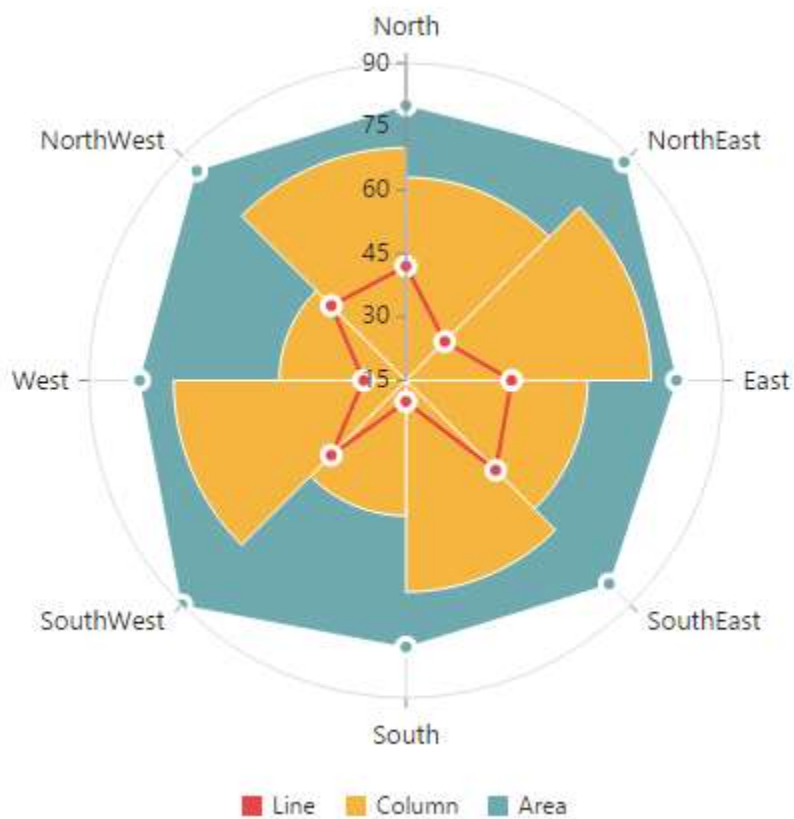
[Click](#) here to view the HiLo Chart online demo sample.

## Polar

Polar chart is created by setting the series [type](#) as **polar** in the chart series.

**JAVASCRIPT**

```
"use strict";
// ...
var series= [{
  //Set chart type to series
  type: 'polar'
}];
// ...
ReactDOM.render(
  <EJ.Chart id="default_chart_sample_0"
  series={series}
  >
  </EJ.Chart>,
  document.getElementById('chart')
);
```



[Click](#) here to view the Polar Chart online demo sample.

*DrawType*

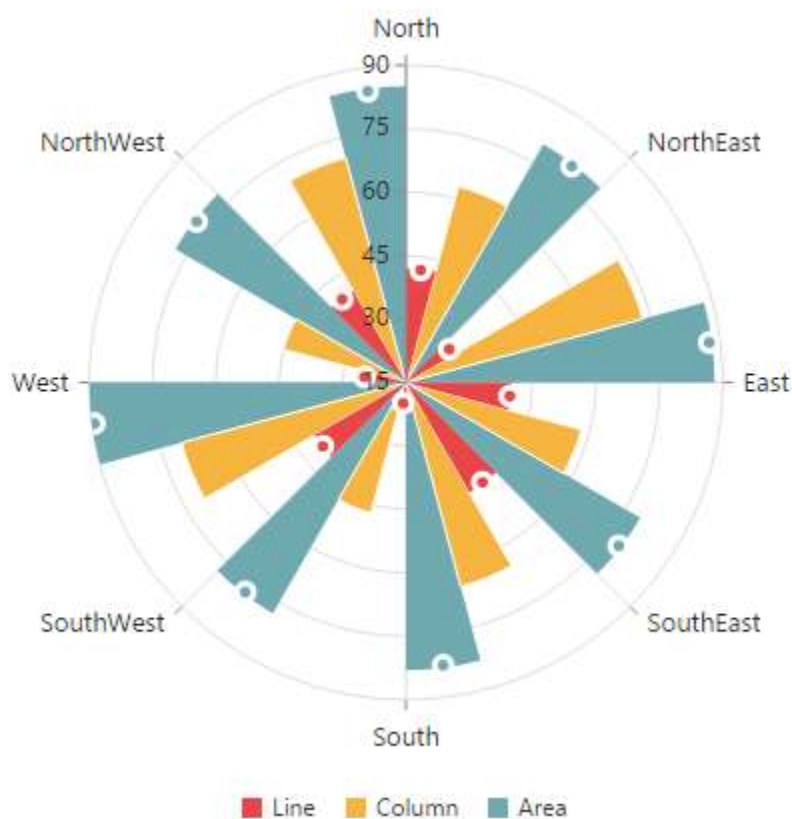
Polar **DrawType** property is used to change the series plotting type to *Line*, *scatter*, *rangeColumn*, *stackingArea*, *spline*, *Column* or *Area*. The default value of DrawType is **Line**.

**JAVASCRIPT**

```

"use strict";
// ...
var series= [{
//Change polar series drawType
drawType: 'column',
// ...
}];
// ...
ReactDOM.render(
<EJ.Chart id="default_chart_sample_0"
series={series}
>
</EJ.Chart>,
document.getElementById('chart')
);

```

*Stack columns in Polar chart*

By using the [isStacking](#) property, you can specify whether the column has to be stacked when the [drawType](#) is column. Its default value is **false**.

**JAVASCRIPT**

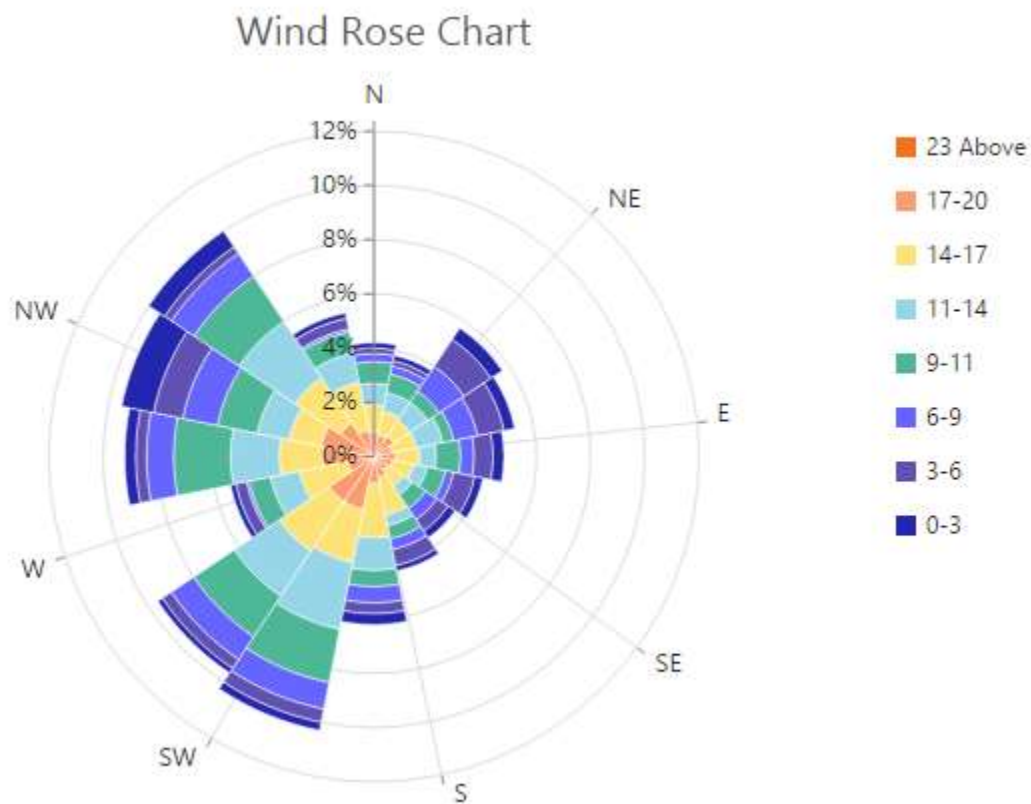
```

"use strict";
// ...
var series= [{

```



```
//Enable isStacking property for stacked column polar chart
isStacking: true
// ...
}];
// ...
ReactDOM.render(
  <EJ.Chart id="default_chart_sample_0"
  series={series}
  >
</EJ.Chart>,
  document.getElementById('chart')
);
```



[Click](#) here to view the Polar Wind Rose Chart online demo sample.

### Radar Chart

Radar **DrawType** property is used to change the series plotting type to *Line*, *scatter*, *rangeColumn*, *stackingArea*, *spline*, *Column* or *Area*. The default value of DrawType is **Line**.

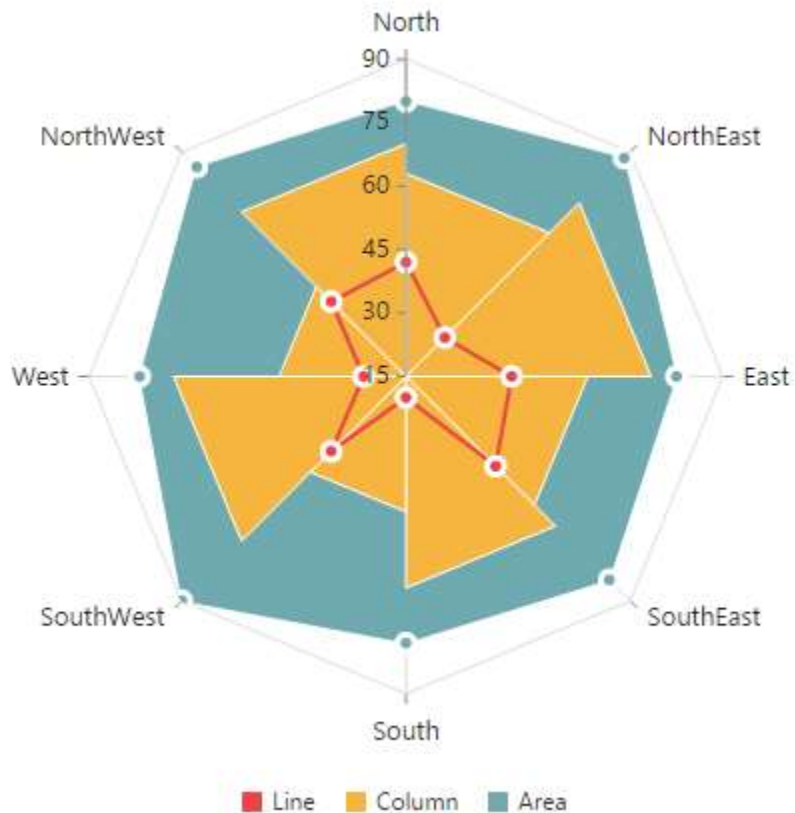
### JAVASCRIPT

```
"use strict";
// ...
var series= [{
  //Set chart type to series
  type: 'radar',
  // ...
}
```

```

    }];
    // ...
    ReactDOM.render(
      <EJ.Chart id="default_chart_sample_0"
        series={series}
      >
    </EJ.Chart>,
    document.getElementById('chart')
    );

```



[Click](#) here to view the Radar Chart online demo sample.

#### DrawType

Radar [drawType](#) property is used to change the series plotting type to *line*, *column* or *area*. The default value of [drawType](#) is “**line**”.

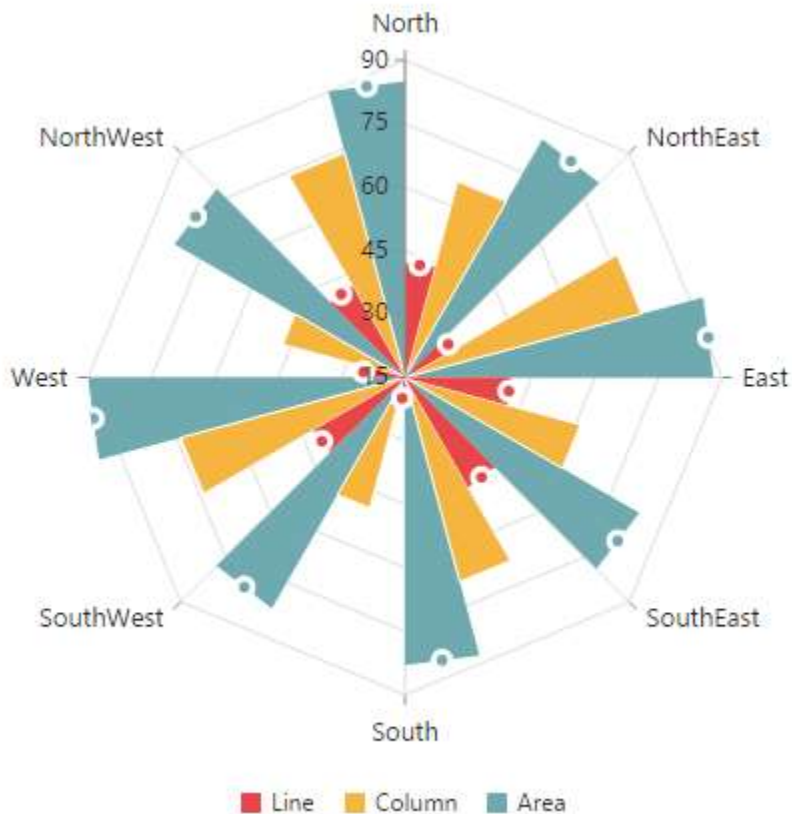
#### JAVASCRIPT

```

"use strict";
// ...
var series= [{
  //Change radar series drawType
  drawType: 'column',
  // ...
}];
// ...

```

```
ReactDOM.render(
  <EJ.Chart id="default_chart_sample_0"
    series={series}
  >
  </EJ.Chart>,
  document.getElementById('chart')
);
```



#### Stack columns in Radar chart

By using the [isStacking](#) property, you can specify whether the column has to be stacked when the [drawType](#) is set as *column*. Its default value is set to **false**.

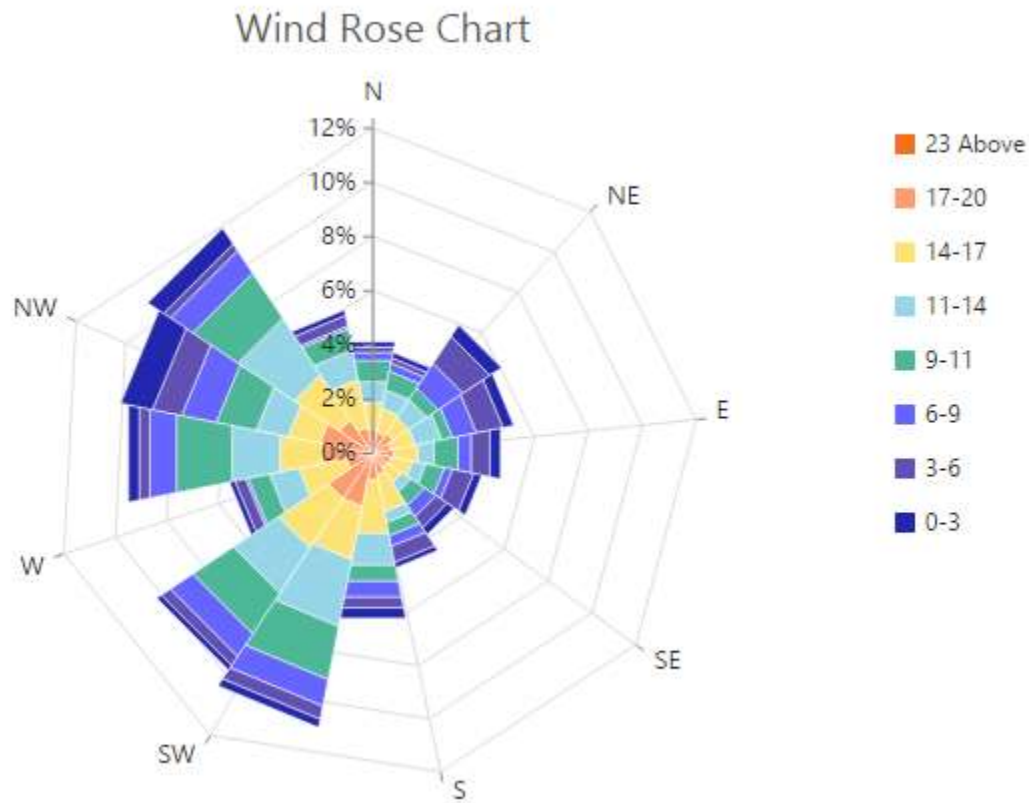
#### JAVASCRIPT

```
"use strict";
// ...
var series= [{
  //Enable isStacking property for stacked column radar chart
  isStacking: true
  // ...
}];
// ...
ReactDOM.render(
  <EJ.Chart id="default_chart_sample_0"
    series={series}
  >
```

```

</EJ.Chart>,
document.getElementById('chart')
);

```



### Waterfall Chart

For rendering a Waterfall chart, set series [type](#) as “**waterfall**” in the chart series. To change the waterfall series segment color use [fill](#) option of series and use [positiveFill](#) property to differentiate the positive segments.

**Note:** The inline property of the **series.positiveFill** has the first priority and override the **series.fill**.

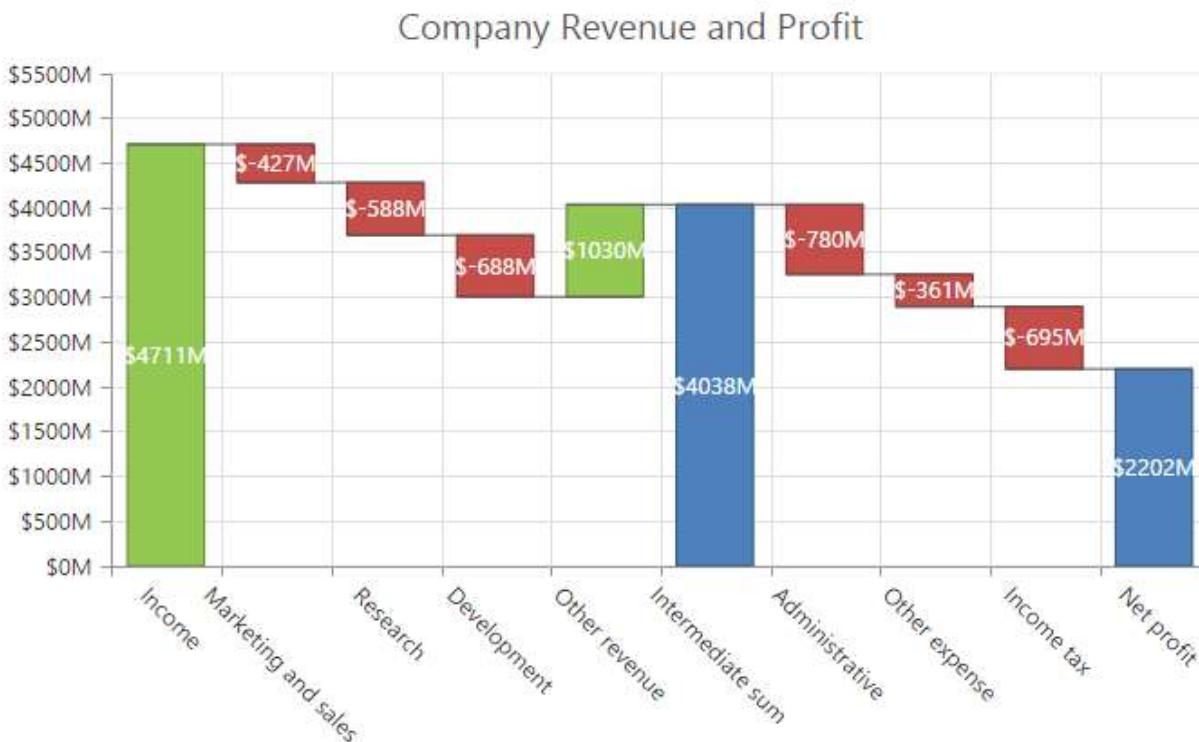
### JAVASCRIPT

```

"use strict";
// ...
var series= [{
  //Change type and color of the series.
  type: waterfall,
  fill: "#C64E4A",
  positiveFill: "#C64E4A"
  // ...
}];
// ...
ReactDOM.render(
  <EJ.Chart id="default_chart_sample_0"
  series={series}

```

```
>
</EJ.Chart>,
document.getElementById('chart')
);
```



[Click](#) here to view the Waterfall Chart online demo sample.

### ShowIntermediateSum

To display the summary of values since the last intermediate point of the waterfall series, set **showIntermediateSum** property as true in the specific point.

### JAVASCRIPT

```
"use strict";
// ...
var series= [{
  points: [
    //Enable showIntermediateSum in to a point.
    // ...
    { x: "Intermediate sum", showIntermediateSum: true }
    // ...
  ],
  // ...
}];
// ...
ReactDOM.render(
  <EJ.Chart id="default_chart_sample_0"
  series={series}
  >
```

```

</EJ.Chart>,
document.getElementById('chart')
);

```

### ShowTotalSum

The sum of all previous point in the waterfall series is displayed on enabling the **showTotalSum** property for a specific point.

#### JAVASCRIPT

```

"use strict";
// ...
var series= [{
  points: [
    //Enable showTotalSum in to a point.
    // ...
    { x: "Total sum", showTotalSum: true }
    // ...
  ],
  // ...
}];
// ...
ReactDOM.render(
<EJ.Chart id="default_chart_sample_0"
series={series}
>
</EJ.Chart>,
document.getElementById('chart')
);

```

### ConnectorLine

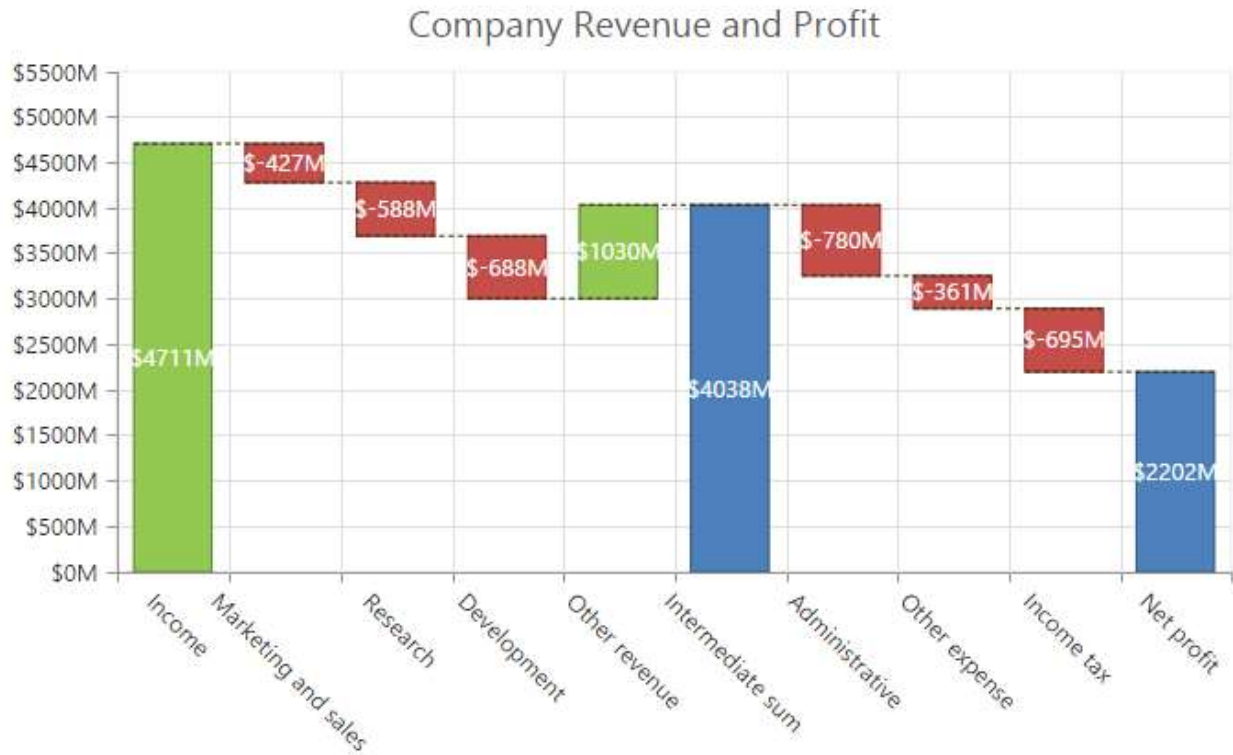
To customize the connector line color, width, opacity and dashArray of the waterfall series, you can use [connectorLine](#) option of series.

#### JAVASCRIPT

```

"use strict";
// ...
var series= [{
  //customize waterfall series connector line styles
  connectorLine: {color: "#333000", width: 1, opacity: 1, dashArray: "3,2"},
  // ...
}];
// ...
ReactDOM.render(
<EJ.Chart id="default_chart_sample_0"
series={series}
>
</EJ.Chart>,
document.getElementById('chart')
);

```

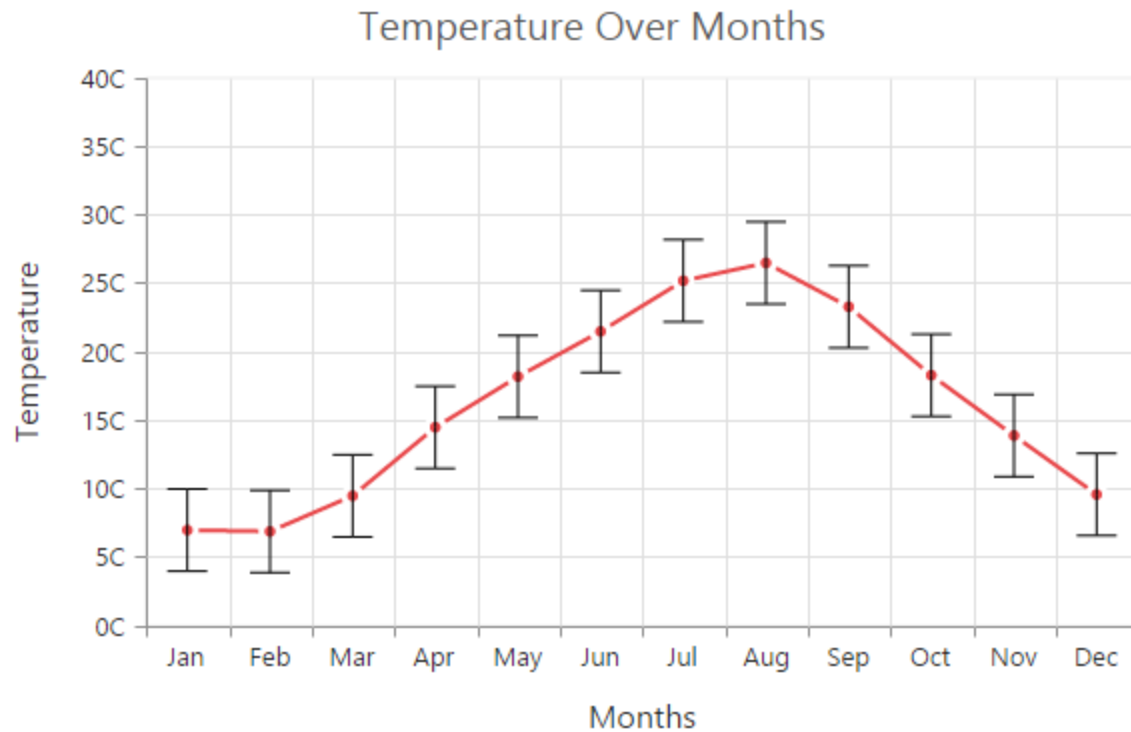


### Error bar Chart

EjChart can generate Error bar for Cartesian type series (*Line, Column, Bar, Scatter, Area, Candle, HiLo, etc.*). To render the Error bar for the series, set [visibility](#) as "visible" to [errorBar](#) in the series.

### JAVASCRIPT

```
"use strict";
// ...
var series= [{
//...
//To toggle the error bar visibility
errorBar: {
visibility: "visible"
}
}];
// ...
ReactDOM.render(
<EJ.Chart id="default_chart_sample_0"
series={series}
>
</EJ.Chart>,
document.getElementById('chart')
);
```



[Click](#) here to view the Error bar Chart online demo sample.

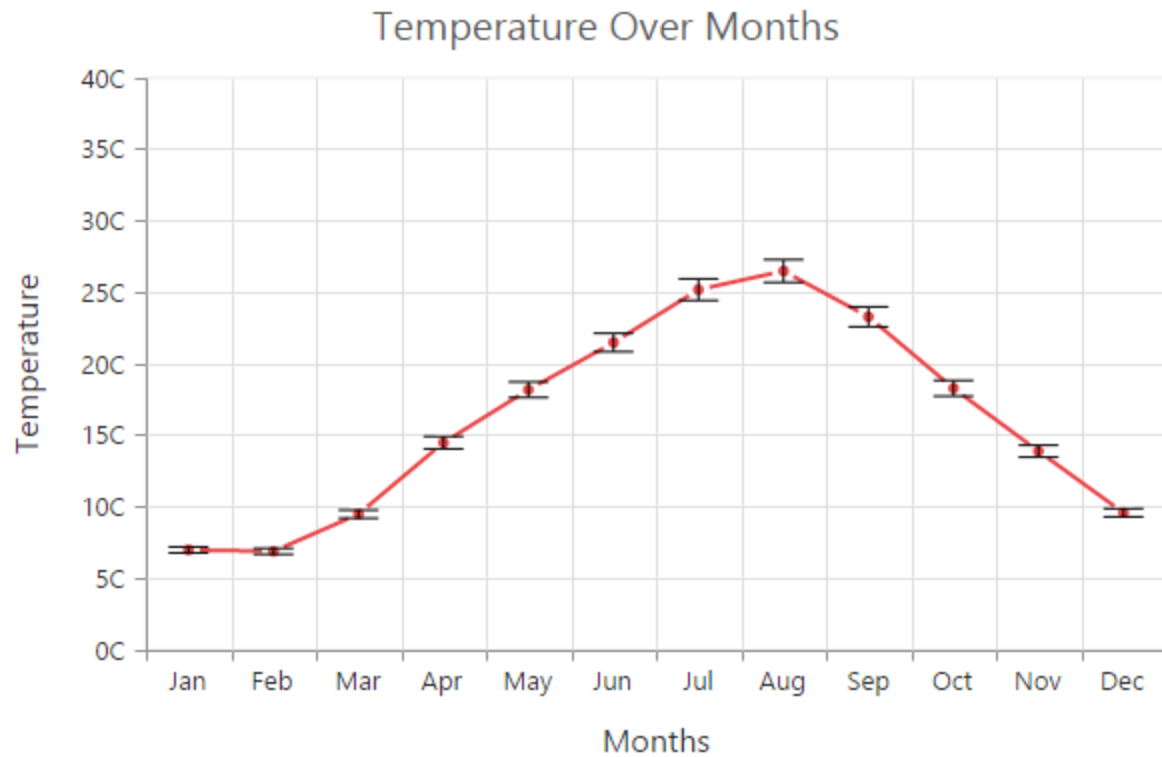
#### Changing Error Bar Type

You can change the error bar rendering type using [type](#) (like *fixedValue*, *percentage*, *standardDeviation*, *standardError* and *custom*) option of `errorBar`. To change the error bar line length you can use [verticalErrorValue](#) property.

#### JAVASCRIPT

```
"use strict";
// ...
var series= [{
  //...
  //Change the error bar type
  errorBar: {
    type: "percentage",
    verticalErrorValue:3
  }
}];
// ...
ReactDOM.render(
  <EJ.Chart id="default_chart_sample_0"
  series={series}
  >
</EJ.Chart>,
  document.getElementById('chart')
);
```



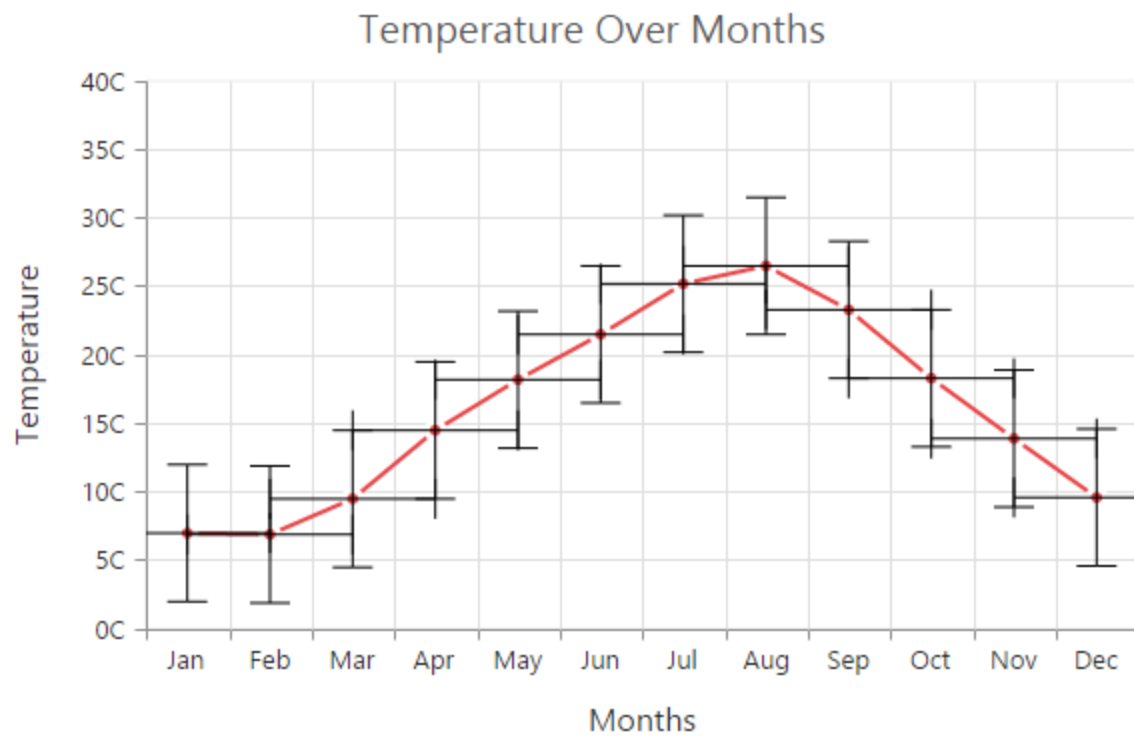


#### Customizing error bar type

To customize the error bar type, set error bar [type](#) as **"custom"** and then change the horizontal/vertical positive and negative value of error bar.

#### JAVASCRIPT

```
"use strict";
// ...
var series= [{
//...
//Change the error bar type
errorBar: {
type: "custom",
verticalPositiveErrorValue:5,
horizontalPositiveErrorValue:1,
verticalNegativeErrorValue:5,
horizontalNegativeErrorValue:1
}
}];
// ...
ReactDOM.render (
<EJ.Chart id="default_chart_sample_0"
series={series}
>
</EJ.Chart>,
document.getElementById('chart')
);
```

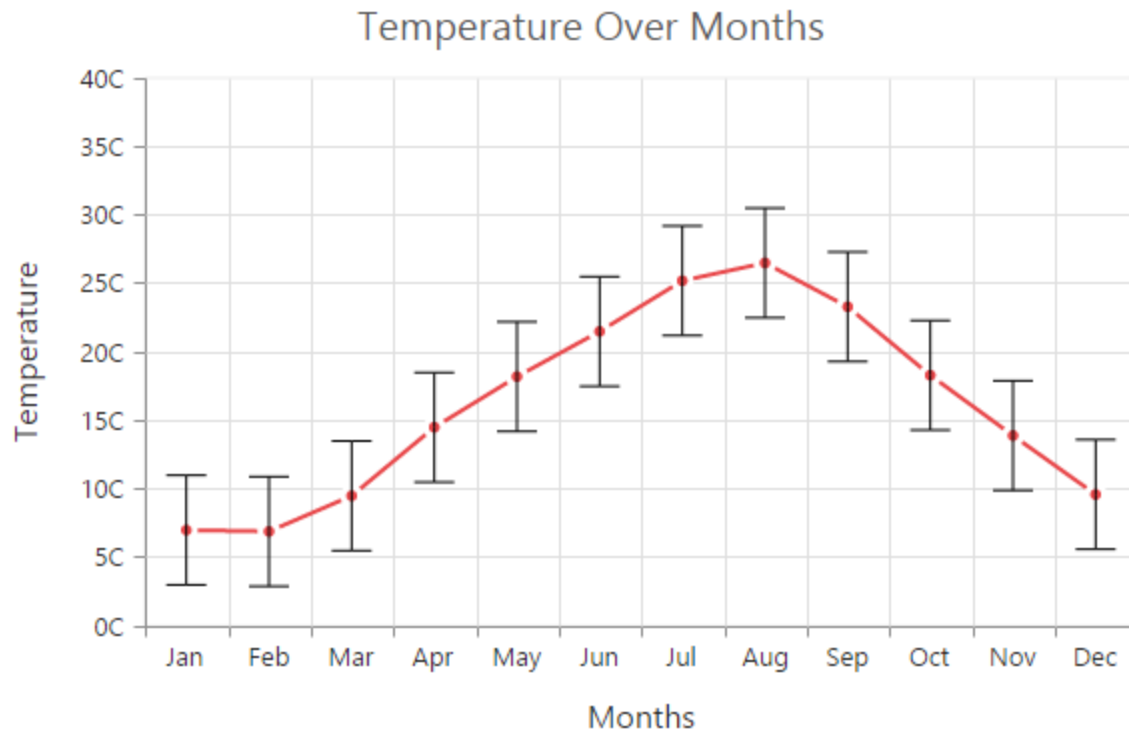


#### Changing Error Bar Mode

Error bar mode is used to define whether the error bar line has to be drawn *horizontally*, *vertically* or in *both* side. To change the error bar mode use [errorBar.mode](#) option.

#### JAVASCRIPT

```
"use strict";
// ...
var series= [{
//...
//Change the error bar mode
errorBar: {
type: "fixedValue",
mode: "vertical"
}
}];
// ...
ReactDOM.render(
<EJ.Chart id="default_chart_sample_0"
series={series}
>
</EJ.Chart>,
document.getElementById('chart')
);
```

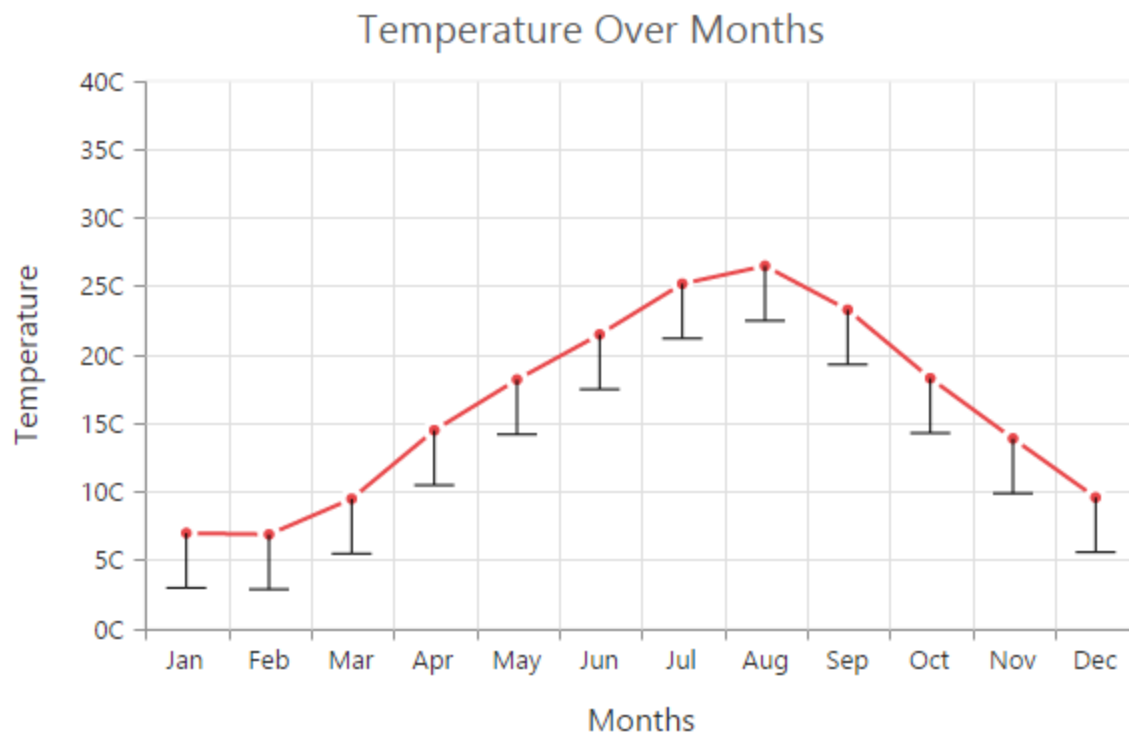


#### Changing Error Bar Direction

You can change the error bar direction to plus, minus or both side using [errorBar.directions](#) option.

#### JAVASCRIPT

```
"use strict";
// ...
var series= [{
//...
//Change the error bar direction
errorBar: {
type: "fixedValue",
mode: "vertical",
direction: "minus"
}
}];
// ...
ReactDOM.render(
<EJ.Chart id="default_chart_sample_0"
series={series}
>
</EJ.Chart>,
document.getElementById('chart')
);
```

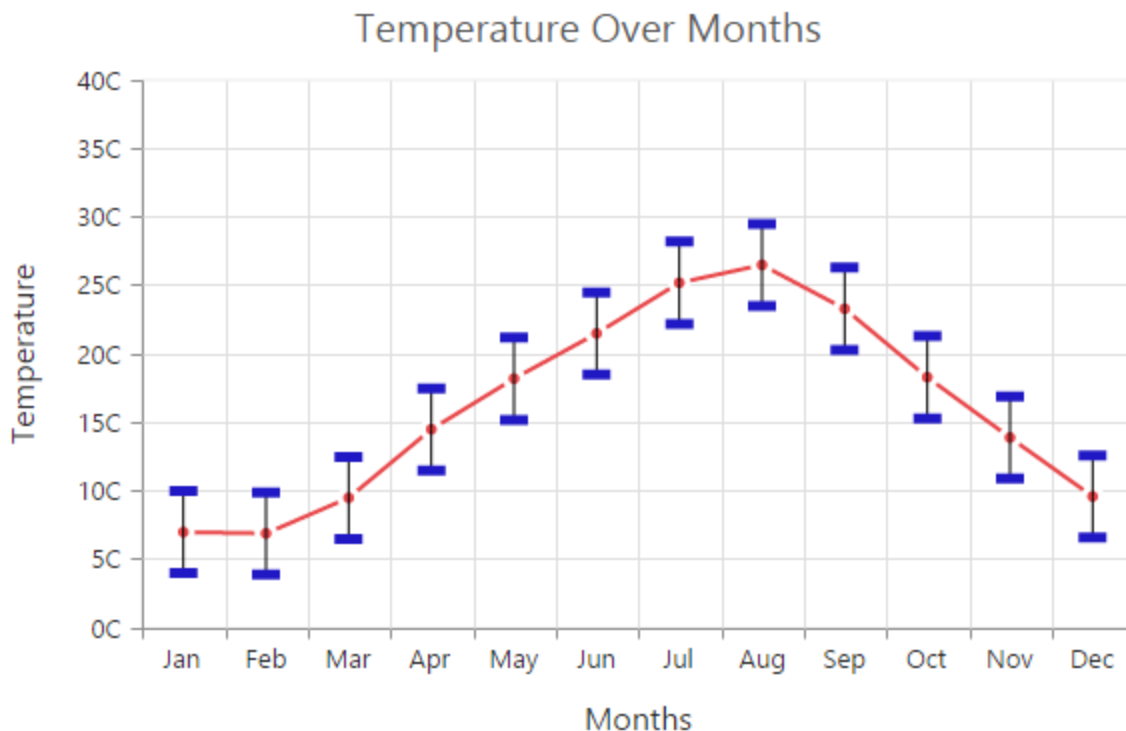


#### Customizing Error bar cap

To customize the error bar cap *visibility*, *length*, *width* and *fill* color, you can use [cap](#) option in the **series.errorBar**.

#### JAVASCRIPT

```
"use strict";
// ...
var series= [{
  //...
  errorBar: {
    //To customize the error bar cap
    cap:{
      visible: true,
      length: 20,
      width: 1,
      fill : "#000000"
    }
  }
}];
// ...
ReactDOM.render(
  <EJ.Chart id="default_chart_sample_0"
  series={series}
  >
  </EJ.Chart>,
  document.getElementById('chart')
);
```



#### Box and Whisker Chart

To render a Box and Whisker Chart, set the series [type](#) as “**boxAndWhisker**”.

Box and Whisker chart requires 2 fields (x and y) to plot a segment.

The field y requires n number of data or it should contains minimum of five values to plot a segment.

#### JAVASCRIPT

```
"use strict";
//...
var series=[{
//...
points:[ { x: "Development", y:
[22,22,23,25,25,25,26,27,27,28,28,29,30,32,34,32,34,36,35,38]},
{ x: "Testing", y: [22,33,23,25,26,28,29,30,34,33,32,31,50]},
{ x: "HR", y: [22,24,25,30,32,34,36,38,39,41,35,36,40,56]},
{ x: "Finance", y: [26,27,28,30,32,34,35,37,35,37,45]},
{ x: "R&D", y: [26,27,29,32,34,35,36,37,38,39,41,43,58] }
],
type: 'boxAndWhisker',
}];
//...
ReactDOM.render(
<EJ.Chart id="default_chart_sample_0"
series={series}
>
</EJ.Chart>,
document.getElementById('chart')
);
```



### BoxPlotMode

You can change the rendering mode of the Box and Whisker series using the [boxPlotMode](#) property. The default [boxPlotMode](#) is “**exclusive**”. The other boxPlotModes available are [inclusive](#) and [normal](#).

### JAVASCRIPT

```
"use strict";
//...
var series=[{
//...
boxPlotMode : 'inclusive',
}];
//...
ReactDOM.render(
<EJ.Chart id="default_chart_sample_0"
series={series}
>
</EJ.Chart>,
document.getElementById('chart')
);
```

### ShowMedian

Box and Whisker [showMedian](#) property is used to show the box and whisker average value. The default value of [showMedian](#) is “**false**”.

**JAVASCRIPT**

```

"use strict";
//...
var series=[{
//...
showMedian : true,
}];
//...
ReactDOM.render(
<EJ.Chart id="default_chart_sample_0"
series={series}
>
</EJ.Chart>,
document.getElementById('chart')
);

```

*Customize the Outlier*

Outlier symbol, width and height can be customized using outlierSettings through [outlierSettings](#) property. By default Outlier symbol is displayed as circle with a height and width of 6 pixels.

**JAVASCRIPT**

```

"use strict";

```

```
//...
var series=[{
//...
outlierSettings:{
  shape: 'triangle',
  size:
  {
    width:10,
    height:10
  }
}
}];
//...
ReactDOM.render(
<EJ.Chart id="default_chart_sample_0"
series={series}
>
</EJ.Chart>,
document.getElementById('chart')
);
```



[Click](#) here to view the Box and Whisker Chart online demo sample.

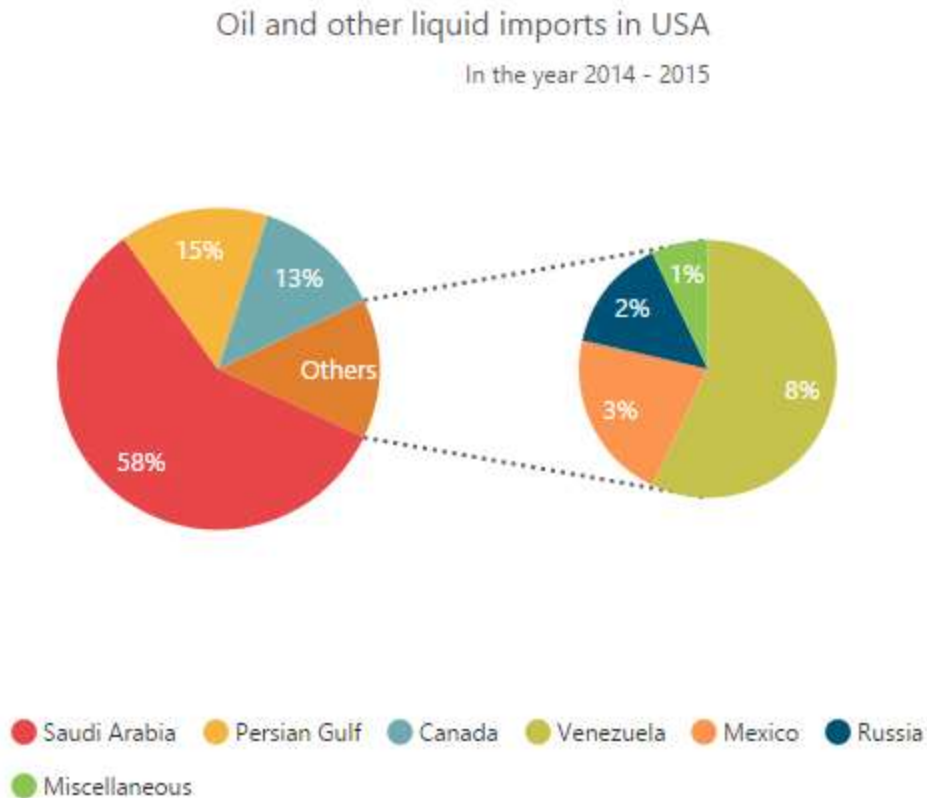


### Pie Of Pie Chart

To render the pie of pie chart, set the series [type](#) as **pieofpie**. Pie of pie chart is used for displaying the data of a pie slice as another pie chart. The values in the second pie is displayed based on the **splitMode** property.

#### JAVASCRIPT

```
//..
var series= [{
  points: [
    {x: 'Saudi Arabia', y: 58, text: '58%'},
    {x: 'Persian Gulf', y: 15, text: '15%'},
    {x: 'Canada', y: 13, text: '13%'},
    {x: 'Venezuela', y: 8, text: '8%'},
    {x: 'Mexico', y: 3, text: '3%'},
    {x: 'Russia', y: 2, text: '2%'},
    {x: 'Miscellaneous', y: 1, text: '1%'}
  ],
  type: 'pieofpie',
  splitValue:"10"
}]
//..
ReactDOM.render(
  <EJ.Chart id="default_chart_sample_0" series={series} >
    </EJ.Chart>,
  document.getElementById('chart')
);
```



[Click](#) here to view the Pie Of Pie Chart online demo sample.

#### *Split Mode and Split Value*

The points to be displayed in the second pie is decided based on the [splitMode](#) property. **SplitMode** property takes the following values.

- Position – Have to split the data points based on its position
- Value – Have to split the data points based on its Y value
- Percentage – Have to split the points based on the percentage value
- Indexes – The data points with the specified indexes are split separately

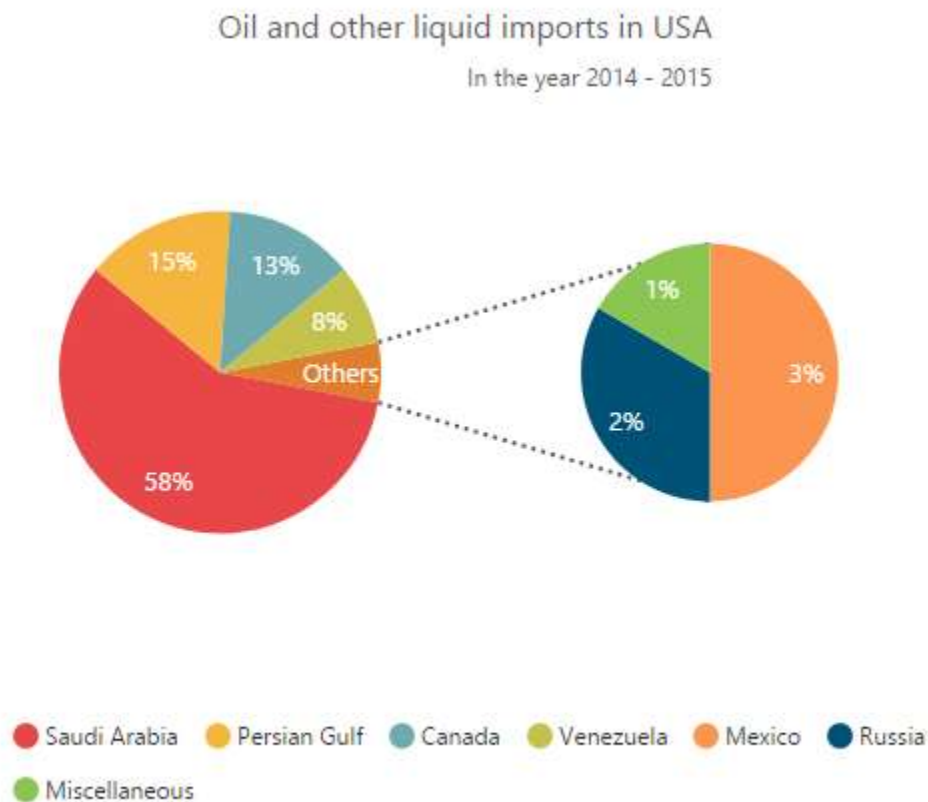
By default, the splitMode is set to **Value**.

#### JAVASCRIPT

```
//..
var series= [{
// ..
splitMode:"Position",
splitValue:"3"
}]
//..
ReactDOM.render(
<EJ.Chart id="default_chart_sample_0" series={series} >
</EJ.Chart>,

```

```
document.getElementById('chart')
);
```



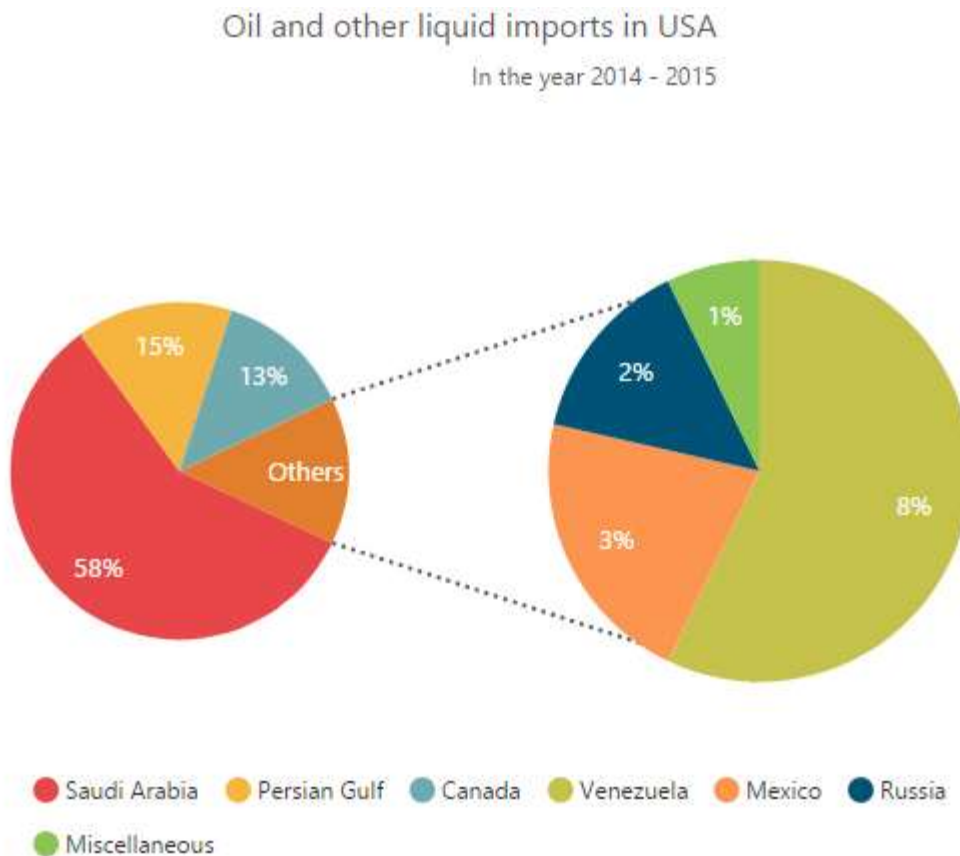
### Changing Pie Of Pie Size

The size of the second Pie can be customized by using the [pieOfPieCoefficient](#) property. The default value of pieOfPieCoefficient is **0.6**. Its value ranges from 0 to 1.

### JAVASCRIPT

```
//...
var series= [{
// ..
pieOfPieCoefficient : 1
}]
//...
ReactDOM.render(
<EJ.Chart id="default_chart_sample_0" series={series} >
</EJ.Chart>,
document.getElementById('chart-default')
);
```

The following screenshot represents the pie of pie series with pieOfPieCoefficient as 1

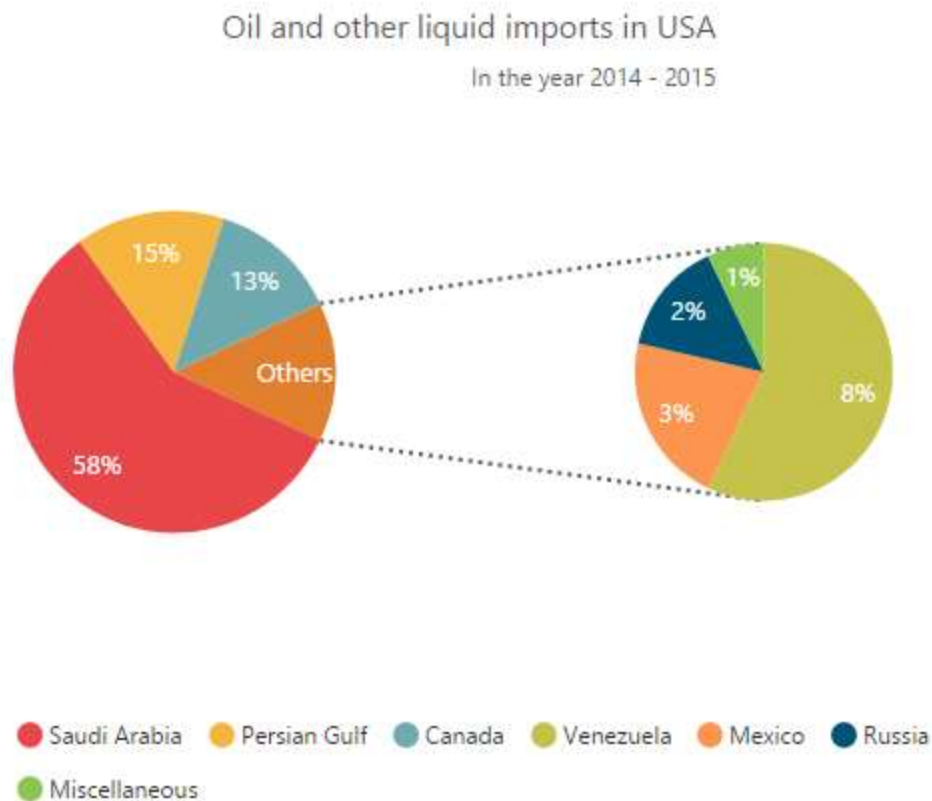


### Customizing the Gap

The distance between the two pies in the pie of pie chart can be controlled by using the [gapWidth](#) property. The default value is **50**.

### JAVASCRIPT

```
//...
var series= [{
// ..
gapWidth:150
}]
//...
ReactDOM.render(
<EJ.Chart id="default_chart_sample_0" series={series} >
</EJ.Chart>,
document.getElementById('chart-default')
);
```



## Chart Series

### Multiple Series

In EjChart, you can add multiple series object in the [series](#) options. The series are rendered in the order it is added to the [series](#) option, by default. You can change this order by using the [zOrder](#) option.

### JAVASCRIPT

```
"use strict";
// ...
//Adding Multiple Series
var series = [{ // Add first series
  points: [{ x: "USA", y: 50 }],
  // ...
},
type: 'column',
// ...
},
{ // Add second series
  points: [{ x: "USA", y: 70 }],
  // ...
},
type: 'column'
// ...
},
{ // Add third series
  points: [{ x: "USA", y: 45 }],
```

```
// ...
],
type: 'column'
// ...
}]
// ...
ReactDOM.render(
<EJ.Chart id="default_chart_sample_0"
series={series}
>
</EJ.Chart>,
document.getElementById('chart')
);
```



[Click](#) here to view the multiple series online demo sample.

#### *Customizing all series together*

By using the [commonSeriesOptions](#), you can customize the series options for all the series commonly, instead of setting the options directly on each series object.

**Note:** The inline properties of the series has the first priority and override the commonSeriesOptions.

The following code example explains on how to enable marker, tooltip and animation for the chart series by using the commonSeriesOptions.

#### **JAVASCRIPT**

```
"use strict";
// ...
//Initializing Common Properties for all the series
var commonSeriesOptions = {
type: 'line',
enableAnimation: true,
tooltip: {
visible: true,
template: 'Tooltip'
},
marker: {
shape: 'circle',
size: {
height: 10, width: 10
},
visible: true
},
border: { width: 2 }
};
var series= [{
// ...
},{
// ...
}],
// ...
ReactDOM.render(
<EJ.Chart id="default_chart_sample_0"
```

```

commonSeriesOptions= {commonSeriesOptions}
series={series}
>
</EJ.Chart>,
document.getElementById('chart')
);

```



### Combination Series

EJChart allows you to render the combination of different series in the chart.

#### JAVASCRIPT

```

"use strict";
// ...
var series= [{
//Set chart type to series1
type: 'column',
// ...
},{
//Set chart type to series2
type: 'line',
// ...
}];
// ...
ReactDOM.render(
<EJ.Chart id="default_chart_sample_0"
series={series}
>
</EJ.Chart>,
document.getElementById('chart')
);

```



[Click](#) here to view the combination series online demo sample.

#### Limitation of combination chart

- [Bar](#), [StackingBar](#), and [StackingBar100](#) cannot be combined with the other Cartesian type series.
- Cartesian type series cannot be combined with the accumulation series ([pie](#), [doughnut](#), [funnel](#), and [pyramid](#)).
- [Polar](#) and [Radar](#) series cannot be combined with the accumulation and Cartesian type series.

When the combination of Cartesian and accumulation series types are added to the series option, the series that are similar to the first series are rendered and other series are ignored. The following code example illustrates this,

#### JAVASCRIPT

```

"use strict";
// ...
//Adding Multiple Series
var series= [{ // Add line series

```

```

points: [{ x: "Jan", y: 45 },
// ...
],
type: 'line',
// ...
},
{
    // Add [Pie](chart-types#pie-chart) series
points: [{ x: "Jan", y: 70 },
// ...
],
type: 'pie'
// ...
}];
ReactDOM.render(
<EJ.Chart id="default_chart_sample_0"
series={series}
>
</EJ.Chart>,
document.getElementById('chart')
);

```



## Data Markers

Data markers are used to provide information about the data point to the user. You can add a shape and label to adorn each data point.

### Add Shapes

You can add shapes to any chart types but they are often used with line, area and spline series to indicate each data point. It is highlighted when you hover the mouse on the shape.

Shapes can be added to the chart by enabling the [visible](#) option of the [marker](#) property. There are different shapes you can add to the chart by using the shape option such as rectangle, circle, diamond etc.

The following code example explains on how to enable series marker and add shapes,

### JAVASCRIPT

```

"use strict";
// ...
//Adding shapes to series
var series = [{
// ....
marker: {
shape: 'Diamond',
visible: true
},
{
// ...
marker: {
shape: 'Triangle',
visible: true
},
},

```



```

{
  // ...
  marker: {
    shape: 'Hexagon',
    visible: true
  }
}]
// ....
ReactDOM.render(
  <EJ.Chart id="default_chart_sample_0"
  series={series}
>
</EJ.Chart>,
document.getElementById('chart')
);

```



#### Add image as marker

Apart from the shapes, you can also add images to mark the data point by using the [imageUrl](#) option.

The following code example illustrates this,

#### JAVASCRIPT

```

"use strict";
// ...
var series = [{
  // ...
  marker: {
    // Enable and customize the marker shape and size
    visible: true,
    // In order to set imageUrl, set shape as 'image' .
    shape: "image",
    imageUrl: "sun_annotation.png",
    size: {width: 20, height: 20}
  }
}]
// ....
ReactDOM.render(
  <EJ.Chart id="default_chart_sample_0"
  series={series}
>
</EJ.Chart>,
document.getElementById('chart')
);

```



#### Add labels

Data label can be added to a chart series by enabling the [visible](#) property in the [dataLabel](#) option. The labels appear at the top of the data point, by default.

The following code example shows how to enable data label and set its horizontal and vertical text alignment.

**JAVASCRIPT**

```

"use strict";
// ...
var series = [{
  // ...
  marker: {
    dataLabel: {
      //Set text alignment to datalabel text
      visible: true,
      horizontalTextAlignment: "center",
      verticalTextAlignment: "far"
    }
  }
}]
// ....
ReactDOM.render(
  <EJ.Chart id="default_chart_sample_0"
  series={series}
  >
</EJ.Chart>,
  document.getElementById('chart')
);

```



Label content can be formatted by using the template option. Inside the template, you can add the placeholder text *"point.x"* and *"point.y"* to display corresponding data points x & y value.

You can adorn the labels with background shapes by setting *shape* option.

The following code example shows how to add background shapes and set template to data label.

**HTML**

```

<div id="template">
<div id="left">

</div>
<div id="right">
<div id="point">#point.y#</div>
</div>
</div>

```

**JAVASCRIPT**

```

"use strict";
// ...
var series = [{
  // ...
  marker: {
    dataLabel: {
      visible: true,
      //Set template to data label
      template: 'template'
    }
  }
}, {

```

```
// ...
marker: {
  dataLabel: {
    visible: true,
    //Add background shape to the data label
    shape: 'Rectangle',
    border: { width: 1, color: "red" }
  }
}, {
  // ...
  marker: {
    dataLabel: {
      visible: true
    }
  }
}];
// ...
ReactDOM.render(
  <EJ.Chart id="default_chart_sample_0"
  series={series}
  >
  </EJ.Chart>,
  document.getElementById('chart')
);
```



The appearance of the labels can be customized by using the [font](#) and [offset](#) options. The [offset](#) option is used to move the labels vertically. Also, labels can be rotated by using the [rotate](#) option.

The following code example shows how to rotate datalabel text and customize the font.

#### JAVASCRIPT

```
"use strict";
// ...
var series = [{
  // ...
  marker: {
    dataLabel: {
      visible: true,
      //Rotate data label and customize the font
      angle: "300",
      offset: 15,
      font: { color: "black", size:"13px" }
    }
  }
}];
// ...
ReactDOM.render(
  <EJ.Chart id="default_chart_sample_0"
  series={series}
  >
  </EJ.Chart>,
  document.getElementById('chart')
);
```



You can position the label to the top, center or bottom position of the segment by using the [textPosition](#) option for the chart types such as column, bar, stacked bar, stacked column, 100% stacked bar, 100% stacked column, candle and OHLC.

The following code example shows how to set textPosition to display data label in the middle of the column rectangle.

#### JAVASCRIPT

```
"use strict";
// ...
var series=[{
  // .....
  marker: {
    dataLabel: {
      visible: true,
      // Place the datalabel text position in the centre of the rectangle
      textPosition: "middle"
    }
  }
}];
// .....
ReactDOM.render(
  <EJ.Chart id="default_chart_sample_0"
  series={series}
  >
  </EJ.Chart>,
  document.getElementById('chart')
);
```



The label can be positioned inside or outside the perimeter of the series by using the [labelPosition](#) option for the chart types such as Pie and Doughnut, .

The following code example shows how to set the labelPosition,

#### JAVASCRIPT

```
"use strict";
// ...
var series=[{
  points: [{ x: 'India', y: 24, text: 'India 24%' },
    { x: 'Japan', y: 25, text: 'Japan 25%' },
    { x: 'Australia', y: 20, text: 'Australia 20%' },
    { x: 'USA', y: 35, text: 'USA 35%' },
    { x: 'China', y: 23, text: 'China 23%' },
    { x: 'Germany', y: 27, text: 'Germany 27%' },
    { x: 'France', y: 22, text: 'France 22%' }],
  marker: {
    dataLabel: {
      visible: true,
      shape: 'rectangle',
      font: {color: "white"}
    }
  }
}];
```

```

},
type: 'doughnut',
//Display data label outside position
labelPosition: 'outside'
}],
// ...
ReactDOM.render(
<EJ.Chart id="default_chart_sample_0"
series={series}
>
</EJ.Chart>,
document.getElementById('chart')
);

```



The following screenshot displays the labels when the [labelPosition](#) is set as *inside* position.



The following screenshot displays the labels when the [labelPosition](#) is set as *outsideExtended* position.



The label can be wrapped for pie, doughnut, funnel, and pyramid series by setting the `enableWrap` property.

### JAVASCRIPT

```

"use strict";
// . . .
var series=[
{
//. . .
marker: {
dataLabel: {
// enable the dataLabel
visible: true,
// enable the wrapping option
enableWrap: true,
// set the maximumLabelWidth of the data label
maximumLabelWidth: 32
}
}
}
// . . .
}]
// . . .
ReactDOM.render(
<EJ.Chart id="default_chart_sample_0"
series={series}
>
</EJ.Chart>,
document.getElementById('chart')
);

```



### Contrast Color for the data label

To change the contrast color for the data label, you can set the [enableContrastColor](#) as **true** in the `dataLabel` property of the chart series.

When we enable this property, the data label text will be rendered in contrast color based on the segment on which it is placed.

If the data label is placed inside the data points segment, then that particular point's color is taken. Else the chart area or chart background color is considered for deriving the contrast color.

### JAVASCRIPT

```
"use strict";
// . . .
var series=[
{
//. . .
marker: {
dataLabel: {
// enable the dataLabel
visible: true,
//Set the saturation color to datalabel text
enableContrastColor: true,
//. . .
}
}
// . . .
}]
// . . .
ReactDOM.render(
<EJ.Chart id="default_chart_sample_0"
series={series}
>
</EJ.Chart>,
document.getElementById('chart')
);
```

### Binding label from the datasource

You can bind the text value to the `dataLabel` from the datasource and then you need to map the text value field with the [textMappingName](#) property respectively.

### JAVASCRIPT

```
"use strict";
// . . .
//data source for chart with label
var chartData = [
{ month: 'Jan', sales: 35, Text: "January" },
{ month: 'Feb', sales: 28, Text: "February" }
];
var series=[{
dataSource: chartData,
xName: "month",
yName: "sales",
marker: {
dataLabel: {
```

```

visible: true,
textMappingName: "Text"
},
},
]);
ReactDOM.render(
<EJ.Chart id="default_chart_sample_0"series={series}>
</EJ.Chart>,
document.getElementById('chart-default')
);

```

### Binding fill color to the points from the datasource

You can bind the color value to the points from the datasource and then you need to map the color value field to the [pointColorMappingName](#) property respectively.

#### JAVASCRIPT

```

"use strict";
// . . .
//data source for chart with fill color
var chartData = [
{ month: 'Jan', sales: 35, Color: "Red" },
{ month: 'Feb', sales: 28, Color: "Blue" }
];
//Mapping the color values to the points
var series=[{
dataSource: chartData,
xName: "month",
yName: "sales",
pointColorMappingName: "Color"
}];
ReactDOM.render(
<EJ.Chart id="default_chart_sample_0"series={series}>
</EJ.Chart>,
document.getElementById('chart-default')
);

```

### Customize specific points

By using the ejChart, you can also customize the individual/specific markers with different colors, shapes and also with different images.

There are two ways to achieve this based on how the data is fed to the series.

When the data is provided by using the [points](#) option, you can add marker for each data point or specific point by using the [marker](#) option as illustrated in the following code example.

#### JAVASCRIPT

```

"use strict";
// ...
var series= [{
// ...
points: [ { x: 'Jan', y: 35 }, { x: 'Feb', y: 28 }, { x: 'Mar', y: 34 },
{ x: 'Apr', y: 32 }, { x: 'May', y: 40 }, { x: 'Jun', y: 32 },
{ x: 'Jul', y: 35 }, { x: 'Aug', y: 55 },
marker: {

```

```

//Enable and customize the data label for a point
dataLabel: {
  visible: true,
  offset: -10,
  shape: "upArrow", font: { color: "white" , size: '11px' },
  margin: { left: 15, right: 15, top: 10, bottom: 10 },
  fill: "green"
} } },
{ x: 'Sep', y: 38 }, { x: 'Oct', y: 30 },
{ x: 'Nov', y: 25,
marker: {
  //Enable and customize the data label for a point
  dataLabel: {
    visible: true,
    offset: -22,
    verticalTextAlignment: 'near',
    shape: "downArrow", font: { color: "white", size: '11px' },
    margin: { left: 15, right: 15, top: 10, bottom: 10 },
    fill: "red"
  } } }, { x: 'Dec', y: 32 }],
}];
// ...
ReactDOM.render(
<EJ.Chart id="default_chart_sample_0"
series={series}
>
</EJ.Chart>,
document.getElementById('chart')
);

```



When the data is bound to the series by using the [dataSource](#) option, you can customize the points in the [seriesRendering](#) event as illustrated in the following code example,

#### JAVASCRIPT

```

"use strict";
// ...
var series= [{
  //Add datasource and set xName and yName
  dataSource: chartData,
  xName: "month",
  yName: "sales"
}];
function onSeriesRender(sender) {
  //Enable and customize the dataLabel for a point using event
  sender.data.series.points[7].marker = {
    dataLabel: {
      visible: true,
      offset: -10,
      shape: "upArrow", font: { color: "white", size: '11px' },
      margin: { left: 15, right: 15, top: 10, bottom: 10 },
      fill: "green"
    } };
  sender.data.series.points[10].marker = {

```



```
//Enable and customize the dataLabel for a point using event
dataLabel: {
  visible: true,
  offset: -22,
  verticalTextAlignment: 'near',
  shape: "downArrow", font: { color: "white", size: '11px' },
  margin: { left: 15, right: 15, top: 10, bottom: 10 },
  fill: "red"
  };
};
ReactDOM.render(
  <EJ.Chart id="default_chart_sample_0"
  series={series}
  seriesRendering={onSeriesRender}
  >
  </EJ.Chart>,
  document.getElementById('chart')
);
```



### Connect Line

This feature is used to connect label and data point by using a line. It can be enabled only for Pie, Doughnut, Pyramid and Funnel chart types. Connector line types can be set as *bezier* or *line* by using the [type](#) option.

The following code example illustrates this,

### JAVASCRIPT

```
"use strict";
// ...
var series= [{
  // ...
  marker: {
    dataLabel: {
      visible: true,
      // Set connector line type and customize the color,
      connectorLine: { type: 'bezier', color: 'black' }
      // ...
    },
  },
  // ...
  labelPosition: 'outsideExtended',
  };
};
// ...
ReactDOM.render(
  <EJ.Chart id="default_chart_sample_0"
  series={series}
  >
  </EJ.Chart>,
  document.getElementById('chart')
);
```



### Smart labels

Overlapping of the labels can be avoided by enabling the [enableSmartLabels](#) property. The default value is *true* for *accumulation type series* and *false* for *other series types*.

The following code example shows how to enable smart labels,

#### JAVASCRIPT

```
"use strict";
// ...
//Initializing Series
var series= [{
  points: [{ x: 'Other Personnal', y: 94658, text: 'Other Personal, 88.47%' },
    { x: 'Medical care', y: 9090, text: 'Medical care, 8.49%' },
    { x: 'Housing', y: 2577, text: 'Housing, 2.40%' },
    { x: 'Transportation', y: 473, text: 'Transportation, 0.44%' },
    { x: 'Education', y: 120, text: 'Education, 0.11%' },
    { x: 'Electronics', y: 70, text: 'Electronics, 0.06%' }],
  marker:{
    dataLabel:{
      visible: true,
      shape: 'none',
      connectorLine: { type: 'bezier', color: 'black' },
      font: { size: '14px' }
    },
    border: { width: 2, color: 'white' },
    type: 'pie',
    labelPosition: "outsideExtended",
    //Display data label outside position and enable smartlabels
    enableSmartLabels: true
  }
}];
// ...
ReactDOM.render(
  <EJ.Chart id="default_chart_sample_0"
    series={series}
  >
  </EJ.Chart>,
  document.getElementById('chart')
);
```



### Legend

The legend contains the list of chart series and Trendlines that appear in a chart.

#### Legend Visibility

By default, the legend is enabled in the chart. You can enable or disable it by using the [visible](#) option of the legend.

#### JAVASCRIPT

```
"use strict";
// ...
var legend= {
  //Visible chart legend
  visible: false
};
```

```
};
//...
ReactDOM.render(
  <EJ.Chart id="default_chart_sample_0"
  legend={legend}
  >
</EJ.Chart>,
  document.getElementById('chart')
);
```



[Click](#) here to view the online demo sample for legend position.

### Legend title

To add the title to the legend, you have to specify the [legend.title.text](#) option.

### JAVASCRIPT

```
"use strict";
// ...
var legend= {
  //...
  title: {
    //Add title to the chart legend
    text: "Countries",
  } };
// ...
ReactDOM.render(
  <EJ.Chart id="default_chart_sample_0"
  legend={legend}
  >
</EJ.Chart>,
  document.getElementById('chart')
);
```



### Position and Align the Legend

By using the [position](#) option, you can position the legend at *left*, *right*, *top* or *bottom* of the chart. The legend is positioned at the **bottom** of the chart, by default.

### JAVASCRIPT

```
"use strict";
// ...
var legend= {
  // ...
  //Place the legend at top of the chart
  position: 'top',
};
// ...
ReactDOM.render(
  <EJ.Chart id="default_chart_sample_0"
  legend={legend}
  >
```

```
</EJ.Chart>,
document.getElementById('chart')
);
```



### Legend Alignment

You can align the legend to the *center*, *far* or *near* based on its position by using the [alignment](#) option.

### JAVASCRIPT

```
"use strict";
// ...
var legend= {
//...
//The below two settings will place the legend at the top-right corner of
the chart.
alignment: 'far',
position: 'top',
};
// ...
ReactDOM.render(
<EJ.Chart id="default_chart_sample_0"
legend={legend}
>
</EJ.Chart>,
document.getElementById('chart')
);
```



### Arrange legend items in the rows and columns

You can arrange the legend items horizontally and vertically by using the [rowCount](#) and [columnCount](#) options of the legend.

- When only the [rowCount](#) is specified, the legend items are arranged according to the [rowCount](#) and number of columns may vary based on the number of legend items.
- When only the [columnCount](#) is specified, the legend items are arranged according to the [columnCount](#) and number of rows may vary based on the number of legend items.
- When both the options are specified, then the one which has higher value is given preference. For example, when the [rowCount](#) is 4 and [columnCount](#) is 3, legend items are arranged in 4 rows.
- When both the options are specified and have the same value, the preference is given to the [columnCount](#) when it is positioned at the top/bottom position. The preference is given to the [rowCount](#) when it is positioned at the left/right position.

### JAVASCRIPT

```
"use strict";
// ...
var legend= {
//Arrange legend items in 4 rows and approximately 4 columns. Column
couldn't may vary based on number of items.
```

```

rowCount: 4,
columnCount: 4
};
// ...
ReactDOM.render(
<EJ.Chart id="default_chart_sample_0"
legend={legend}
>
</EJ.Chart>,
document.getElementById('chart')
);

```



### Customization

#### Legend shape

To change the legend icon shape, you have to specify the shape in the [shape](#) property of the legend. When you want the legend icon to display the prototype of the series, you have to set the **seriesType** as shape.

#### JAVASCRIPT

```

"use strict";
// ...
var legend= {
//...
//Change legend shape
shape: 'seriesType',
};
// ...
ReactDOM.render(
<EJ.Chart id="default_chart_sample_0"
legend={legend}
>
</EJ.Chart>,
document.getElementById('chart')
);

```



#### Legend items size and border

You can change the size of the legend items by using the [itemStyle.width](#) and [itemStyle.height](#) options. To change the legend item border, use [border](#) option of the legend itemStyle.

#### JAVASCRIPT

```

"use strict";
// ...
var legend= {
//...
//Change legend items border, height and width
itemStyle: {width: 13, height: 13, border: { color: "#FF0000", width: 1 } },
};
// ...
ReactDOM.render(

```

```
<EJ.Chart id="default_chart_sample_0"
  legend={legend}
>
</EJ.Chart>,
document.getElementById('chart')
);
```



#### Legend size

By default, legend takes 20% of the **height** horizontally when it was placed on the top or bottom position and 20% of the **width** vertically while placing on the left or right position of the chart. You can change this default legend size by using the **size** option of the legend.

#### JAVASCRIPT

```
"use strict";
// ...
var legend= {
  //...
  //Change legend size
  size:{width: '550', height: '100'}
};
// ...
ReactDOM.render(
<EJ.Chart id="default_chart_sample_0"
  legend={legend}
>
</EJ.Chart>,
document.getElementById('chart')
);
```



#### Legend Item Padding

You can control the spacing between the legend items by using the [itemPadding](#) option of the legend. The default value is 10.

#### JAVASCRIPT

```
"use strict";
// ...
var legend= {
  //...
  //Add space between each legend item
  itemPadding: 15,
};
// ...
ReactDOM.render(
<EJ.Chart id="default_chart_sample_0"
  legend={legend}
>
</EJ.Chart>,
document.getElementById('chart')
);
```



#### Legend border

You can customize the legend border by using the [border](#) option in the legend.

#### JAVASCRIPT

```
"use strict";
// ...
var legend= {
//...
//Set border color and width to legend
border: {color: "#FFC342", width: 2},
};
// ...
ReactDOM.render(
<EJ.Chart id="default_chart_sample_0"
legend={legend}
>
</EJ.Chart>,
document.getElementById('chart')
);
```



#### Scrollbar for legend

You can enable or disable the legend scrollbar by using the [enableScrollbar](#) option of the legend. When you disable the scrollbar option, the legend does not consider the [default size](#) and chart draws in the remaining space. If you have specified the **size** to the legend with the scrollbar disabled, then the legends beyond this limit will get clipped. The default value of [enableScrollbar](#) option is **true**.

#### JAVASCRIPT

```
"use strict";
// ...
var legend= {
//...
//Enable scrollbar option in for legend
enableScrollbar: true,
size:{width: '430', height: '80'},
};
// ...
ReactDOM.render(
<EJ.Chart id="default_chart_sample_0"
legend={legend}
>
</EJ.Chart>,
document.getElementById('chart')
);
```



#### Customize the legend text

To customize the legend item text and title you can use the [legend.font](#) and [legend.title](#) options. You can change the legend title alignment by using the [textAlignment](#) option of the legend title.

**JAVASCRIPT**

```

"use strict";
// ...
var legend= {
  //...
  //Customize the legend item text
  font: { fontFamily: 'Segoe UI', fontStyle: 'Normal', fontWeight: 'Bold',
    size: '15px' },
  title: {
    //...
    textAlign: "center",
    //Customize the legend title text
    font: { fontFamily: 'Segoe UI', fontStyle: 'Italic',
      fontWeight: 'Bold', size: '12px' },
    }
  };
// ...
ReactDOM.render(
  <EJ.Chart id="default_chart_sample_0"
  legend={legend}
  >
  </EJ.Chart>,
  document.getElementById('chart')
);

```



*LegendItems Text Overflow***Trim**

You can trim the legend item text when its width exceeds the [legend.textWidth](#), by specifying [textOverflow](#) as **“trim”**. The original text will be displayed on mouse hover.

**JAVASCRIPT**

```

"use strict";
// ...
var legend= {
  //trim the legend text
  textOverflow: 'trim',
  textWidth: 34
};
// ...
ReactDOM.render(
  <EJ.Chart id="default_chart_sample_0"
  legend={legend}
  >
  </EJ.Chart>,
  document.getElementById('chart')
);

```



**Wrap**

By specifying [textOverflow](#) as **“wrap”**, you can wrap the legend text by word.





### WrapAndTrim

You can wrap and trim the legend text by specifying [textOverflow](#) as “**wrapAndTrim**”. The original text will be displayed on mouse hover.



### Handle the legend item clicked

You can get the legend item details such as *index*, *bounds*, *shape* and *series* by subscribing the [legendItemClick](#) event on the chart. When the legend item is clicked, it triggers the event and returns the [legend information](#).

### JAVASCRIPT

```
"use strict";
// ...
var legend= {
  //trim the legend text
  textOverflow: 'trim',
  textWidth: 34
};
// ...
function onLegendClicked(sender) {
  var legendItem = sender.data;
}
ReactDOM.render(
  <EJ.Chart id="default_chart_sample_0"
    legend={legend}
    legendItemClick={onLegendClicked}
  >
  </EJ.Chart>,
  document.getElementById('chart')
);
```

### Series selection on legend item click

You can select a specific series or point while clicking on the corresponding legend item through disabling the [toggleSeriesVisibility](#) option of the legend. The default value of toggleSeriesVisibility option is **true**. To customize the series selection refer to the series [selection](#).

### JAVASCRIPT

```
"use strict";
// ...
var legend= {
  //...
  //Disable series collapsing on legend item clicked
  toggleSeriesVisibility: false,
};
// ...
ReactDOM.render(
  <EJ.Chart id="default_chart_sample_0"
    legend={legend}
  >
  </EJ.Chart>,
  document.getElementById('chart')
);
```

```
);
```



### Collapsing legend item

You can collapse the specific series/point legend item displaying in the chart, by setting the [visibleOnLegend](#) as "hidden" in the point or series.

### JAVASCRIPT

```
"use strict";
//Initializing Series
var series:[{
  points: [{ x: 'Albania', y: 60.1 },
  //...
  //Collapse the point's legend item in the legend collection
  { x: 'New Zealand', y: 82.8, visibleOnLegend:'hidden' }]
}];
var legend={visible:true};
ReactDOM.render(
  <EJ.Chart id="default_chart_sample_0"
  series={series}
  legend={legend}
  >
  </EJ.Chart>,
  document.getElementById('chart')
);
```



### Empty Points

The Data points that uses the **null** or **undefined** as value are considered as empty points. Empty data points are ignored and not plotted in the Chart. When the data is provided by using the [points](#) property, you can set the **isEmpty** to true to specify that the particular point is an empty point.

### JAVASCRIPT

```
"use strict";
var series=[ {
  //Using empty points
  points: [
    { x: 0, y: 210 },
    { x: 1, y: null }, { x: 2, y: 150 },
    { x: 3, y: 180, isEmpty: true },
    { x: 4, y: 170 },
    { x: 5, y: 200 },
    { x: 6, y: 140, isEmpty: true },
    { x: 7, y: 120 }, { x: 8, y: 140 }
  ],
}];
// ...
ReactDOM.render(
  <EJ.Chart id="default_chart_sample_0"
  series={series}
  >
```

```
</EJ.Chart>,
document.getElementById('chart')
);
```



[Click](#) here to view the online demo sample for empty points.

### EmptyPointSettings

You can customize the empty points visibility and change its [displayMode](#) (*gap, zero and average*) using [emptyPointSettings](#) option.

### JAVASCRIPT

```
"use strict";
var series=[ {
//...
emptyPointSettings: {
// visible the empty point settings with displayMode as average
visible: true,
displayMode : "average"
},
}];
// ...
ReactDOM.render(
<EJ.Chart id="default_chart_sample_0"
series={series}
>
</EJ.Chart>,
document.getElementById('chart')
);
```



If the [visible](#) property of [emptyPointSettings](#) is *false*, then the empty points has been dropped and chart will be rendered without empty points.



### Customizing Styles

Empty points color and border can be customized using [style](#) property of [emptyPointSettings](#).

### JAVASCRIPT

```
"use strict";
var series=[ {
//...
emptyPointSettings: {
visible: true,
//Customizing empty points styles
style: {
color: "#ffa000",
border:{
color: "gray",
width:2
}
}
}
}];
```

```

},
}];
// ...
ReactDOM.render(
<EJ.Chart id="default_chart_sample_0"
series={series}
>
</EJ.Chart>,
document.getElementById('chart')
);

```



## Chart Title & Subtitle

### Title

By using the title option, you can add the [text](#) as well as customize its [border](#), [background](#) color and [font](#).

### JAVASCRIPT

```

"use strict";
// ...
var title= {
//Adding text to chart title
text: 'Efficiency of oil-fired power production ',
//Change the title text background color
background : "lightblue",
//Customizing Chart title border
border: {
color: "blue",
width: 2,
opacity: 0.5 ,
cornerRadius : 4
},
//Customizing Chart title font
font:{
opacity: 1,
fontFamily: "Arial",
fontStyle: 'italic',
fontWeight: 'regular',
color: "#E27F2D",
size: '23px'
},
};
// ...
ReactDOM.render(
<EJ.Chart id="default_chart_sample_0"
title={title}
>
</EJ.Chart>,
document.getElementById('chart')
);

```



[Click](#) here to view the Chart Title online demo sample.

We can trim, wrap and wrapAndTrim to the chart title using textOverflow property. The original text will be displayed as tooltip on mouse hover.

#### JAVASCRIPT

```
"use strict";
// ...
var title= {
  text: 'Efficiency of oil-fired power production ',
  //To enable the title trim, wrap and wrapandtrim
  enableTrim: true,
  //Setting maximum width to the title
  maximumWidth: 150,
  //To trim the title
  textOverflow: "trim",
};
// ...
ReactDOM.render(
  <EJ.Chart id="default_chart_sample_0"
  title={title}
  >
</EJ.Chart>,
  document.getElementById('chart')
);
```



#### Title Alignment

You can change the title alignment to *center*, *far* and *near* by using the [textAlignment](#) property of the chart title.

#### JAVASCRIPT

```
"use strict";
// ...
var title= {
  //Change title text alignment
  textAlignment: "near",
  //...
};
// ...
ReactDOM.render(
  <EJ.Chart id="default_chart_sample_0"
  title={title}
  >
</EJ.Chart>,
  document.getElementById('chart')
);
```



#### Add Subtitle to the chart

By using the subTitle option, you can add the [subTitle](#) to the chart title and customize its [border](#), [background](#) color and [font](#).

**JAVASCRIPT**

```

"use strict";
// ...
var title={
  // ...
  subTitle: {
    //Add subtitle to chart title
    text: "( in a week )",
    //Change the title text background color
    background : "lightblue",
    //Customizing Chart subtitle border
    border: {
      color: "blue",
      width: 2,
      opacity: 0.2 ,
      cornerRadius : 4
    },
    //Customizing Chart subtitle font
    font:{
      opacity: 1,
      fontFamily: "Arial",
      fontStyle: 'italic',
      fontWeight: 'regular',
      color: "#E27F2D",
      size: '12px'
    },
  },
}
// ...
ReactDOM.render(
  <EJ.Chart id="default_chart_sample_0"
  title={title}
  >
  </EJ.Chart>,
  document.getElementById('chart')
);

```



We can trim, wrap and wrapAndTrim to the chart subtitle using textOverflow property. The original text will be displayed as tooltip on mouse hover.

**JAVASCRIPT**

```

"use strict";
// ...
var title= {
  // ...
  subTitle:{
    text: '( in a week )',
    //To enable the sub-title trim, wrap and wrap and trim
    enableTrim: true,
    //Setting maximum width to the sub-title
    maximumWidth: 50,
    //To trim the sub-title
    textOverflow: "wrap",
  },
}

```

```

},
};
// ...
ReactDOM.render(
<EJ.Chart id="default_chart_sample_0"
title={title}
>
</EJ.Chart>,
document.getElementById('chart')
);

```



### Subtitle Alignment

You can change the subtitle alignment to *center*, *far* and *near* by using the [textAlignment](#) property of the subtitle.

### JAVASCRIPT

```

"use strict";
// ...
var title= {
// ...
subTitle:{
//Change subtitle to text alignment
textAlignment: "center",
// ...
}
};
// ...
ReactDOM.render(
<EJ.Chart id="default_chart_sample_0"
title={title}
>
</EJ.Chart>,
document.getElementById('chart')
);

```



### Striplines

EjChart supports horizontal and vertical striplines.

#### Horizontal Stripline

You can create horizontal stripline by adding the [stripline](#) in the **vertical axis** and set [visible](#) option to **true**. Striplines are rendered in the specified **start** to **end** range and you can add more than one stripline for an axis.

### JAVASCRIPT

```

"use strict";
//Initializing Primary Y Axis
var primaryYAxis= {
// ...
stripline: [
//Create horizontal Stripline using vertical Axis

```

```

{
  //Enable Stripline
  visible: true,
  start: 30,
  end: 40,
},
// ...
],
};
// ...
ReactDOM.render(
  <EJ.Chart id="default_chart_sample_0"
  primaryYAxis={primaryYAxis}
  >
  </EJ.Chart>,
  document.getElementById('chart')
);

```



[Click](#) here to view the Striplines online demo sample.

### Vertical Stripline

You can create vertical stripline by adding the [stripline](#) in the **horizontal axis** and set [visible](#) option to **true**.

### JAVASCRIPT

```

"use strict";
// ...
//Initializing Primary X Axis
var primaryXAxis: {
  // ...
  stripLine: [
    //Create vertical Stripline using horizontal Axis
    {
      //Enable Stripline
      visible: true,
      start: 3,
      end: 7,
    },
    // ...
  ],
};
// ...
ReactDOM.render(
  <EJ.Chart id="default_chart_sample_0"
  primaryXAxis={primaryXAxis}
  >
  </EJ.Chart>,
  document.getElementById('chart')
);

```





### Customize the Text

To customize the stripLine text, use the [text](#) and [font](#) options.

#### JAVASCRIPT

```
"use strict";
// ...
//Initializing Primary Y Axis
var primaryYAxis= {
// ...
stripLine: [{
//Customize the stripLine text and font styles
text: 'High Temperature',
font: { size: '18px', color: 'white' }
// ...
}],
};
// ...
ReactDOM.render(
<EJ.Chart id="default_chart_sample_0"
primaryYAxis={primaryYAxis}
>
</EJ.Chart>,
document.getElementById('chart')
);
```



### Text Alignment

Stripline text can be aligned by using the [textAlignment](#) property.

#### JAVASCRIPT

```
"use strict";
// ...
//Initializing Primary Y Axis
var primaryYAxis= {
// ...
stripLine: [{
//Set stripLine text alignment to top position
textAlignment: 'middletop',
// ...
}],
};
// ...
ReactDOM.render(
<EJ.Chart id="default_chart_sample_0"
primaryYAxis={primaryYAxis}
>
</EJ.Chart>,
document.getElementById('chart')
);
```



### Customize the Stripline

To customize the stripLine styles, use the [color](#), [opacity](#), [borderWidth](#) and [borderColor](#) properties.

#### JAVASCRIPT

```
"use strict";
//Initializing Primary Y Axis
var primaryYAxis= {
// ...
stripLine: [{
//Customize the StripLine rectangle
color: '#33CCFF',
borderWidth: 2,
opacity: 0.5,
borderColor: 'red',
// ...
}],
};
// ...
ReactDOM.render(
<EJ.Chart id="default_chart_sample_0"
primaryYAxis={primaryYAxis}
>
</EJ.Chart>,
document.getElementById('chart')
)
```



### Change the Z-order of the stripline

Stripline [zIndex](#) property is used to display the stripLine either behind or over the series.

#### JAVASCRIPT

```
"use strict";
// ...
//Initializing Primary Y Axis
var primaryYAxis= {
// ...
stripLine: [{
//Change stripLine zIndex
zindex: 'over',
// ...
}],
};
// ...
ReactDOM.render(
<EJ.Chart id="default_chart_sample_0"
primaryYAxis={primaryYAxis}
>
</EJ.Chart>,
document.getElementById('chart')
);
```



## User Interactions

### Tooltip

#### *Enable tooltip for data point*

Tooltip for the data points can be enabled by using the [visible](#) option of the [tooltip](#) in the series.

#### **JAVASCRIPT**

```
"use strict";
// ...
var series= [{
//...
//Enable tooltip for the series
tooltip: {visible: true}
}];
// ...
ReactDOM.render(
<EJ.Chart id="default_chart_sample_0"
series={series}
>
</EJ.Chart>,
document.getElementById('chart')
);
```



#### *Format the tooltip*

Tooltip displays data specified by the [format](#) option of the tooltip. The default value of the format option is `#point.x# : #point.y#`. Here, `#point.x#` is the placeholder for x value of the point and `#point.y#` is the placeholder for y value of the point.

You can also use `#series.<optionname>#` as placeholder to display the value of an option in corresponding series and use `#point.<optionname>#` as placeholder to display the value of an option in the corresponding point.

#### **JAVASCRIPT**

```
"use strict";
// ...
var series= [{
tooltip: {
//Displaying tooltip in a format
format: "#series.name# <br/> #point.x# : #point.y# (g/kWh)"
// ...
}
}];
// ...
ReactDOM.render(
<EJ.Chart id="default_chart_sample_0"
series={series}
>
</EJ.Chart>,
document.getElementById('chart')
);
```



### Tooltip Template

HTML elements can be displayed in the tooltip by using the [template](#) option of the tooltip. The template option takes the value of the id attribute of the HTML element. You can use the `#point.x#` and `#point.y#` as place holders in the HTML element to display the x and y values of the corresponding data point.

You can also use `#series.<optionname>#` as place holder to display the value of an option in corresponding series of the tooltip and use `#point.<optionname>#` as place holder to display the value of an option in the corresponding point for which the tooltip is displayed.

### HTML

```
<body>
<div id="chartcontainer"></div>
<!-- Create Tooltip template here -->
<div id="Tooltip" style="display: none;">
<div id="icon"> <div id="eficon"> </div> </div>
<div id="value">
<div>
<label id="efpercentage">&#160;#point.y#% </label>
<label id="ef">Efficiency </label>
</div>
</div>
</div>
</body>
```

### JAVASCRIPT

```
"use strict";
// ...
var series= [{
  tooltip: {
    //Set template id to tooltip template
    template: 'Tooltip',
    // ...
  }
}];
// ...
ReactDOM.render(
  <EJ.Chart id="default_chart_sample_0"
  series={series}
  >
</EJ.Chart>,
  document.getElementById('chart')
);
```



[Click](#) here to view the Tooltip template online demo sample.

### Tooltip template animation

You can enable animation by setting the [enableAnimation](#) to true. Tooltip animates when the mouse moves from one data point to another point. The [duration](#) property in tooltip specifies the time taken to animate the tooltip. the duration is set to “500ms”, by default.

**Note:** Tooltip is animated only if the template is specified for tooltip.

### JAVASCRIPT

```
"use strict";
// ...
var series= [{
  tooltip: {
    //Enable tooltip template animation and set duration time
    enableAnimation: true, duaration: "1000ms",
    // ...
  }
}];
// ...
ReactDOM.render(
  <EJ.Chart id="default_chart_sample_0"
  series={series}
  >
</EJ.Chart>,
  document.getElementById('chart')
);
```

### Customize the appearance of tooltip

The [fill](#) and [border](#) options are used to customize the background color and border of the tooltip respectively. The [font](#) option in the tooltip is used to customize the font of the tooltip text.

### JAVASCRIPT

```
"use strict";
// ...
var series= [{
  tooltip: {
    //Change tooltip color and border
    fill: '#FF9933',
    border: { width: 1, color: "#993300" }
    // ...
  }
}];
// ...
ReactDOM.render(
  <EJ.Chart id="default_chart_sample_0"
  series={series}
  >
</EJ.Chart>,
  document.getElementById('chart')
);
```



### Tooltip with rounded corners

The options [rx](#) and [ry](#) are used to customize the corner radius of the tooltip rectangle.

#### JAVASCRIPT

```
"use strict";
// ...
var series= [{
  tooltip: {
    //Customize the corner radius of the tooltip rectangle.
    rx: "50", ry: "50"
    // ...
  }
}];
// ...
ReactDOM.render(
  <EJ.Chart id="default_chart_sample_0"
  series={series}
  >
</EJ.Chart>,
  document.getElementById('chart')
);
```



### Zooming and Panning

#### Enable Zooming

There are two ways you can zoom the chart,

- When the [zooming.enable](#) option is set to true, you can zoom the chart by using the rubber band selection.
- When the [zooming.enableMouseWheel](#) option is set to true, you can zoom the chart on mouse wheel scrolling.
- When [zooming.enablePinching](#) option is set to *true*, you can zoom the chart through pinch gesture.

**Note:** Pinch zooming is supported only in browsers that support multi-touch gestures. Currently IE10, IE11, Chrome and Opera browsers support multi-touch in desktop devices.

#### JAVASCRIPT

```
"use strict";
// ...
//Enable zooming in chart
var zooming= {enable: true};
// ...
ReactDOM.render(
  <EJ.Chart id="default_chart_sample_0"
  zooming={zooming}
  >
</EJ.Chart>,
  document.getElementById('chart')
);
```



After zooming the chart, a zooming toolbar will appear with options to *zoom*, *pan* and *reset*. Selecting the Pan option will allow to pan the chart and selecting the Reset option will reset the zoomed chart.



[Click](#) here to view the Zooming and Panning online demo sample.

#### Types of zooming

The [type](#) option in zooming specifies whether the chart is allowed to scale along the horizontal axis or vertical axis or along both axis. The default value of the [type](#) is “**xy**” (both axis).

#### JAVASCRIPT

```
"use strict";
// ...
var zooming= {
  enable: true,
  //Enable horizontal zooming
  type: 'x'
};
// ...
ReactDOM.render(
  <EJ.Chart id="default_chart_sample_0"
    zooming={zooming}
  >
  </EJ.Chart>,
  document.getElementById('chart')
);
```

#### Customizing zooming toolbar

You can choose the items displayed in the zooming toolbar by specifying the property [toolBarItems](#).

#### JAVASCRIPT

```
"use strict";
// ...
//Customizing zooming toolbar
var zooming={
  enable: true,
  toolbarItems: ["reset", "zoomIn", "zoomOut"]
};
// ...
ReactDOM.render(
  <EJ.Chart id="default_chart_sample_0"
    zooming={zooming}
  >
  </EJ.Chart>,
  document.getElementById('chart')
);
```



### Enable ScrollBar

EjChart provides scrollbar support to view the other portions of chart area which is not shown in the view port when zoomed, by setting true to [enableScrollbar](#) option in [zooming](#).

#### JAVASCRIPT

```
"use strict";
// ...
//Enable zooming scrollbar in chart
var zooming= {enable:true, enableScrollbar: true};
// ...
ReactDOM.render(
<EJ.Chart id="default_chart_sample_0"
zooming={zooming}
>
</EJ.Chart>,
document.getElementById('chart')
);
```



### Crosshair

Crosshair is used to view the value of an axis at mouse position or touch contact point.

#### Enable crosshair and crosshair label

Crosshair can be enabled by using the [visible](#) option in the [crosshair](#). Crosshair label for an axis can be enabled by using the [visible](#) option of [crosshairLabel](#) in the corresponding axis.

#### JAVASCRIPT

```
"use strict";
var primaryXAxis={
//Enable crosshairLabel to X-Axis
crosshairLabel: { visible: true },
};
var primaryYAxis={
//Enable crosshairLabel to Y-Axis
crosshairLabel: { visible: true },
};
//Initializing Crosshair
var crosshair={
visible: true
};
// ...
ReactDOM.render(
<EJ.Chart id="default_chart_sample_0"
primaryXAxis={primaryXAxis} primaryYAxis={primaryYAxis}
crosshair={crosshair}
>
</EJ.Chart>,
document.getElementById('chart')
);
```



[Click](#) here to view the Crosshair online demo sample.



*Customize the crosshair line and crosshair label*

The [fill](#) and [border](#) options of the [crosshairLabel](#) is used to customize the background color and border of the crosshair label respectively. Color and width of the crosshair line can be customized by using the [line](#) option in the [crosshair](#).

**JAVASCRIPT**

```
"use strict";
// ...
var primaryXAxis= {
//...
crosshairLabel: {
visible: true,
//Customizing the crosshair label background color and border
fill: "red",
border: { color: "green", width: 2 }
}
};
var crosshair= {
visible: true,
//Customizing the crosshair line
line: { color: 'gray', width: 2 },
};
/ ...
ReactDOM.render(
<EJ.Chart id="default_chart_sample_0"
primaryXAxis={primaryXAxis}
crosshair={crosshair}
>
</EJ.Chart>,
document.getElementById('chart')
);
```



[Click](#) here to view the Crosshair online demo sample.

**Trackball**

Trackball is used to track a data point close to the mouse position or touch contact point. Trackball marker indicates the closest point and trackball tooltip displays the information about the point.

*Enable Trackball*

Trackball can be enabled by setting the [visible](#) option of the crosshair to *true* and then set the [type](#) as **“trackball”**. The default value of type is **“crosshair”**.

**JAVASCRIPT**

```
"use strict";
var crosshair= {
visible: true,
//Change crosshair type to trackball
type: 'trackball',
};
//.....
ReactDOM.render(
<EJ.Chart id="default_chart_sample_0"
```

```
crosshair={crosshair}
>
</EJ.Chart>,
document.getElementById('chart')
);
```



[Click](#) here to view the Trackball online demo sample.

#### Customize trackball marker and trackball line

Shape and size of the trackball marker can be customized by using the [shape](#) and [size](#) options of the crosshair marker. Color and width of the trackball line can be customized by using the [line](#) option in the crosshair.

#### JAVASCRIPT

```
"use strict";
// ...
var crosshair= {
  visible: true,
  //Customize the trackball line color and width
  line: { color: '#800000', width: 2 },
  //Customize the trackball marker shape size and visibility
  marker: {
    shape: 'pentagon',
    size: {
      height: 9, width: 9
    },
  },
  //Enable/disable trackball marker
  visible: true
}
};
// ...
ReactDOM.render(
  <EJ.Chart id="default_chart_sample_0"
  crosshair={crosshair}
  >
  </EJ.Chart>,
  document.getElementById('chart')
);
```



#### Format Trackball tooltip

X and Y values displayed in the trackball tooltip are formatted based on its axis [labelFormat](#).

#### JAVASCRIPT

```
"use strict";
// ...
var primaryXAxis= {
  labelFormat: 'MMM, yyyy',
  //...
};
var primaryYAxis= {
```

```

labelFormat: '{value}K',
//...
};
var crosshair= {
visible: true,
type: 'trackball',
};
// ...
ReactDOM.render(
<EJ.Chart id="default_chart_sample_0"
primaryXAxis={primaryXAxis}
primaryYAxis={primaryYAxis}
crosshair={crosshair}
>
</EJ.Chart>,
document.getElementById('chart')
);

```



You can able to show the trackball tooltip in two modes, using trackballTooltipSettings.

1. Grouping. 2. Float.

### JAVASCRIPT

```

"use strict";
// ...
var crosshair= {
visible: true,
type: 'trackball',
//Customize the trackball tooltip
trackballTooltipSettings: {
//Trackball mode
mode: 'grouping',
//Customize the trackball border, fill, rx and ry
border:{
width:1,
color: 'grey'
},
rx: 3,
ry: 3,
fill: 'whitesmoke'
}
};
ReactDOM.render(
<EJ.Chart id="default_chart_sample_0"
crosshair={crosshair}
>
</EJ.Chart>,
document.getElementById('chart')
);

```



[Trackball tooltip template:](#)

Trackball tooltip template is used to display the tooltip in customized template format. You can define the desired template in css style. You can enable the **toolTipTemplate** by using the following code snippet.

**JAVASCRIPT**

```
"use strict";
// ...
var crosshair= {
  visible: true,
  type: 'trackball',
  //Customize the trackball tooltip
  trackballTooltipSettings: {
    //Trackball mode
    mode: 'float',
    tooltipTemplate: 'tooltip'
  }
};
ReactDOM.render(
  <EJ.Chart id="default_chart_sample_0"
  crosshair={crosshair}
  >
</EJ.Chart>,
  document.getElementById('chart')
);
```



## Highlight

EJChart provides highlighting support for the series and data points on mouse hover. To enable the highlighting option, set the [enable](#) property to *true* in the [highlightsettings](#) of the series.

**Note:** When hovering mouse on the data points, the corresponding series legend also will be highlighted.

### JAVASCRIPT

```
"use strict";
// ...
var series=[{
  highlightSettings: {
    // enable the highlight settings
    enable: true,
  },
  // ...
}];
// ...
ReactDOM.render(
  <EJ.Chart id="default_chart_sample_0"
  series= {series}
  >
</EJ.Chart>,
  document.getElementById('chart')
);
```

[Click](#) here to view the highlight and selections online demo sample.

### Highlight Mode

You can set three different highlight mode for the highlighting data point and series by using the [mode](#) property of the [highlightsettings](#).

- Series
- Points
- Cluster

### Series mode

To highlight all the data points of the specified series, you can set the “**series**” value to the [mode](#) option in the highlightSettings.

### JAVASCRIPT

```
"use strict";
// ...
var series=[{
  highlightSettings: {
    // enable the highlight settings
    enable: true,
    //Change highlight mode
    mode: 'series'
  },
  // ...
}];
```

```
// ...
ReactDOM.render(
  <EJ.Chart id="default_chart_sample_0"
    series= {series}
  >
  </EJ.Chart>,
  document.getElementById('chart')
);
```



### Point mode

To highlight a single point, you can set the “**point**” value to the [mode](#) option.

### JAVASCRIPT

```
"use strict";
// ...
var series=[{
  highlightSettings: {
    enable: true,
    //Change highlight mode
    mode: 'point'
  },
  // ...
}];
// ...
ReactDOM.render(
  <EJ.Chart id="default_chart_sample_0"
    series= {series}
  >
  </EJ.Chart>,
  document.getElementById('chart')
);
```



### Cluster mode

To highlight the points that corresponds to the same index in all the series, set the “**cluster**” value to the [mode](#) option.

### JAVASCRIPT

```
"use strict";
// ...
var series=[{
  highlightSettings: {
    enable: true,
    //Change highlight mode
    mode: 'cluster'
  },
  // ...
}];
// ...
ReactDOM.render(
```

```
<EJ.Chart id="default_chart_sample_0"
series= {series}
>
</EJ.Chart>,
document.getElementById('chart')
);
```



#### *Customize the highlight styles*

To customize the highlighted series, use the [color](#), [border](#) and [opacity](#) options in the highlightSettings.

#### **JAVASCRIPT**

```
"use strict";
// ...
var series=[{
highlightSettings: {
enable: true,
//Customizing
border: { width: '1.5', color: "red" },
opacity: 0.5, color: "green"
},
// ...
}];
// ...
ReactDOM.render(
<EJ.Chart id="default_chart_sample_0"
series= {series}
>
</EJ.Chart>,
document.getElementById('chart')
);
```



#### *Patterns to highlight*

EjChart provides pattern support for highlighting the data by setting the value to the [pattern](#) property of the highlightSettings. The different types of highlight patterns are as follows.

1. chessboard 2. crosshatch 3. dots 4. pacman 5. grid 6. turquoise 7. star 8. triangle 9. circle 10. tile 11. horizontalDash 12. verticalDash 13. rectangle 14. box 15. verticalStripe 16. horizontalStripe 17. bubble 18. diagonalBackward 19. diagonalForward

#### **JAVASCRIPT**

```
"use strict";
// ...
var series=[{
highlightSettings: {
enable: true,
//Change highlighting pattern
pattern: "chessboard",
},
// ...
}];
```

```
// ...
ReactDOM.render(
  <EJ.Chart id="default_chart_sample_0"
    series= {series}
  >
</EJ.Chart>,
  document.getElementById('chart')
);
```



### Custom pattern

To create a custom pattern for the highlighting data points, set the pattern type as “**custom**” and add the custom pattern **id** in the [customPattern](#) option of the highlightSettings.

### HTML

```
<body>
<div id="container"></div>
<svg>
  <pattern id="dots_a" patternUnits="userSpaceOnUse" width="6" height="6">
    <rect x="0" y="0" width="6" height="6" transform="translate(0,0)"
      fill="black" opacity="1"></rect>
    <path d='M 3 -3 L -3 3 M 0 6 L 6 0 M 9 3 L 3 9' stroke-width="1"
      stroke="white"></path>
  </pattern>
</svg>
</body>
```

### JAVASCRIPT

```
"use strict";
// ...
var series=[{
  highlightSettings: {
    enable: true,
    //Add custom pattern for highlighting data
    pattern: "custom",
    customPattern: 'dots_a',
    // ...
  },
  // ...
}];
// ...
ReactDOM.render(
  <EJ.Chart id="default_chart_sample_0"
    series= {series}
  >
</EJ.Chart>,
  document.getElementById('chart')
);
```



### Selection

EJChart provides selection support for the series and data points on mouse click. To enable the selection option, set the [enable](#) property to *true* in the [selectionSettings](#) of the series.

**Note:** When mouse is clicked on the data points, the corresponding series legend also will be selected.

#### JAVASCRIPT

```
"use strict";
// ...
var series=[{
  selectionSettings: {
    // enable the selection settings
    enable: true,
  },
  // ...
}];
// ...
ReactDOM.render(
  <EJ.Chart id="default_chart_sample_0"
  series= {series}
  >
</EJ.Chart>,
  document.getElementById('chart')
);
```

[Click](#) here to view the highlight and selections online demo sample.

### Selection Mode

You can set four different selection mode for highlighting the data point and series by using the [mode](#) property of the selectionSettings.

- Series
- Points
- Cluster
- Range

#### Series mode

To select all the data points of the specified series, you can set the “**series**” value to the [mode](#) option in the selectionSettings.

#### JAVASCRIPT

```
"use strict";
// ...
var series=[{
  selectionSettings: {
    enable: true,
    //Change selection mode
    mode: 'series',
    pattern: 'chessboard'
  },
  // ...
}];
```

```
// ...
ReactDOM.render(
  <EJ.Chart id="default_chart_sample_0"
    series= {series}
  >
  </EJ.Chart>,
  document.getElementById('chart')
);
```



### Point mode

To highlight a single point, you can set the “**point**” value to the [mode](#) option.

#### JAVASCRIPT

```
"use strict";
// ...
var series=[{
  selectionSettings: {
    enable: true,
    //Change selection mode
    mode: 'point',
    // ...
  },
  // ...
}];
// ...
ReactDOM.render(
  <EJ.Chart id="default_chart_sample_0"
    series= {series}
  >
  </EJ.Chart>,
  document.getElementById('chart')
);
```



### Cluster mode

To select the points that corresponds to the same index in all the series, set the “**cluster**” value to the [mode](#) option.

#### JAVASCRIPT

```
"use strict";
// ...
var series=[{
  selectionSettings: {
    enable: true,
    //Change selection mode
    mode: 'cluster',
    // ...
  },
  // ...
}];
```

```
// ...
ReactDOM.render(
  <EJ.Chart id="default_chart_sample_0"
    series= {series}
  >
  </EJ.Chart>,
  document.getElementById('chart')
);
```



### Range mode

To fetch the selected area data points value, you can set the selectionSettings [mode](#) as **range** in the chart series. The selection rectangle can be drawn as horizontally, vertically or in both direction by using [rangeType](#) property and the selected data's are returned as an array collection in the [rangeSelected](#) event.

### JAVASCRIPT

```
"use strict";
// ...
var series=[{
  selectionSettings: {
    enable: true,
    //Change selection mode
    mode: 'range',
    //Enable both axis selection
    rangeType: 'xy'
  },
  // ...
}];
//event to fetch the selected data point values
function rangeSelection (sender){
  var selectedData = sender.data.selectedDataCollection;
  //...
}
// ...
ReactDOM.render(
  <EJ.Chart id="default_chart_sample_0"
    series= {series}
    rangeSelected= {rangeSelection}
  >
  </EJ.Chart>,
  document.getElementById('chart')
);
```



[Click](#) here to view the Multiple data selection online demo sample.

### Selection Type

You can set two different selection type for selecting the data point and series on mouse click by using the [type](#) property of the selectionSettings.

- Single
- Multiple

### Single Type

To select a data point or a series on mouse click based on the [selectionSettings.mode](#), set [selectionSettings.type](#) as “single” in the series.

#### JAVASCRIPT

```
"use strict";
// ...
var commonSeriesOptions={
  selectionSettings: {
    enable: true,
    //Selection mode is series or point ,cluster
    mode: 'series',
    //Selection type is single
    type: 'single'
  },
};
// ...
ReactDOM.render(
  <EJ.Chart id="default_chart_sample_0"
  commonSeriesOptions= {commonSeriesOptions}
  >
</EJ.Chart>,
  document.getElementById('chart')
);
```



### Multiple Type

For selecting multiple data points or series on mouse click, set [selectionSettings.type](#) as “multiple” in the series.

#### JAVASCRIPT

```
"use strict";
// ...
var commonSeriesOptions={
  selectionSettings: {
    enable: true,
    //Selection mode is series or point ,cluster
    mode: 'series',
    //Selection type is single
    type: 'multiple'
  },
};
// ...
ReactDOM.render(
  <EJ.Chart id="default_chart_sample_0"
  commonSeriesOptions= {commonSeriesOptions}
  >
</EJ.Chart>,
  document.getElementById('chart')
);
```

```
);
```



### Customizing selection styles

To customize the selection styles, use the [color](#), [border](#) and [opacity](#) options in the selectionSettings.

### JAVASCRIPT

```
"use strict";
// ...
var series=[{
  selectionSettings: {
    enable: true,
    //Customizing selection rectangle styles
    border: { width: '1.5', color: "red" },
    opacity: 0.5, color: "red"
  },
  // ...
},
// ...
];
// ...
ReactDOM.render(
  <EJ.Chart id="default_chart_sample_0"
  series= {series}
  >
</EJ.Chart>,
  document.getElementById('chart')
);
```



### Patterns for selection

EjChart provides pattern support for the data selection by setting the value to the [pattern](#) property of the selectionSettings. The different types of selection patterns are as follows.

1. chessboard 2. crosshatch 3. dots 4. pacman 5. grid 6. turquoise 7. star 8. triangle 9. circle 10. tile 11. horizontalDash 12. verticalDash 13. rectangle 14. box 15. verticalStripe 16. horizontalStripe 17. bubble 18. diagonalBackward 19. diagonalForward

### JAVASCRIPT

```
"use strict";
// ...
var series=[{
  selectionSettings: {
    enable: true,
    //Change selection pattern
    pattern: "diagonalForward",
    // ...
  },
  // ...
},
// ...
];
// ...
ReactDOM.render(
  <EJ.Chart id="default_chart_sample_0"
```

```

series= {series}
>
</EJ.Chart>,
document.getElementById('chart')
);

```



#### Custom pattern

To create a custom pattern for selecting the data points, set the [pattern](#) type as “**custom**” and add the custom pattern **id** in the [customPattern](#) option of the selectionSettings.

#### HTML

```

<body>
<div id="container"></div>
<svg>
<pattern id="dots_a" patternUnits="userSpaceOnUse" width="6" height="6">
<rect x="0" y="0" width="6" height="6" transform="translate(0,0)"
fill="black" opacity="1"></rect>
<path d='M 3 -3 L -3 3 M 0 6 L 6 0 M 9 3 L 3 9' stroke-width="1"
stroke="white"></path>
</pattern>
</svg>
</body>

```

#### JAVASCRIPT

```

"use strict";
// ...
var series=[{
selectionSettings: {
enable: true,
//Add custom pattern for selection data
pattern: "custom",
customPattern: 'dots_a',
// ...
},
// ...
}];
// ...
ReactDOM.render(
<EJ.Chart id="default_chart_sample_0"
series= {series}
>
</EJ.Chart>,
document.getElementById('chart')
);

```



#### Handling Series Selection

To get the series information when selecting the specific series, subscribe to the [seriesRegionClick](#) event and set the **selectionSettings.mode** as “**series**”.

**JAVASCRIPT**

```

"use strict";
// ...
var series=[{
  selectionSettings: {
    enable: true,
    //Change selection mode
    mode: "series",
    // ...
  }
}];
// ...
function seriesSelection(sender) {
  //Get Series information on series selection
  var seriesData = sender.series;
}
ReactDOM.render(
  <EJ.Chart id="default_chart_sample_0"
    series= {series}
    seriesRegionClick={seriesSelection}
  >
</EJ.Chart>,
  document.getElementById('chart')
);

```

*Selection on Load*

We can able to select the point/series programmatically on chart load, by setting series and point index in the `selectedDataPointIndexes` property.

**JAVASCRIPT**

```

"use strict";
// ...
//Added selected data point indexes
var selectedpointIndex= [
  { seriesIndex:0 , pointIndex:2 },
  { seriesIndex:1 , pointIndex:4 }
];
// ...
ReactDOM.render(
  <EJ.Chart id="default_chart_sample_0"
    selectedDataPointIndexes={selectedpointIndex}
  >
</EJ.Chart>,
  document.getElementById('chart')
);

```



*Data Editing*

EjChart provides support to change the location of the rendered points. This can be done by dragging the point and dropping it on another location in chart. To enable the data editing, set the `[enable]` (`../api/ejchart#members:series-dragSettings-enable`) property to true in the [dragSettings](#) of the series.

**JAVASCRIPT**

```

"use strict";
//Initializing Series
var series=[{
  dragSettings:{
    enable: true
  }
}];
ReactDOM.render(
  <EJ.Chart id="default_chart_sample_0"
  series={series}
  >
</EJ.Chart>,
  document.getElementById('chart')
);

```



[Click](#) here to view the data editing online demo sample.

*Customize Dragging direction*

To drag the point along x and y axes, you can specify `[type]` (`../api/ejchart#members:series-dragSettings-type`) as `xy` in `dragSettings`. And to drag along x axis alone, specify the type as `x` and to drag along y axis, specify type as `y`.

**JAVASCRIPT**

```

"use strict";
//Initializing Series
var series=[{
  dragSettings:{
    type: 'y'
  }
}];
ReactDOM.render(
  <EJ.Chart id="default_chart_sample_0"
  series={series}
  >
</EJ.Chart>,
  document.getElementById('chart')
);

```



*Performance*

- When there are large number of points to load, you can enable canvas rendering mode in chart. Canvas rendering is faster than SVG because it does not involve manipulating DOM elements as much as SVG rendering.

**JAVASCRIPT**

```

"use strict";
// ...

```



```
//Enable Canvas rendering mode
canvasRendering: true,
// ...
ReactDOM.render(
  <EJ.Chart id="default_chart_sample_0"
  enableCanvasRendering={canvasRendering}
  >
</EJ.Chart>,
  document.getElementById('chart')
);
```

[Click](#) here to view the online demo sample that shows Chart performance with 100,000 data points.

- Instead of enabling data markers and labels when there are large number of data points, you can use **trackball** and **tooltip** to view point information.

### Lazy Loading

Lazy loading feature provides an effective way for loading data on demand by scrolling and viewing a smaller range of data from a larger collection.

### JAVASCRIPT

```
"use strict";
var primaryXAxis=
{
  scrollbarSettings: {
    // enable the scrollbar
    visible: true,
    // enable the resize option
    canResize: true,
    range: {
      min: "2009/1/1",
      max: "2014/1/1"
    }
  }
};
// . . .
ReactDOM.render(
  <EJ.Chart id="default_chart_sample_0"
  primaryXAxis={primaryXAxis}
  >
</EJ.Chart>,
  document.getElementById('chart')
);
```



### Trendlines

EjChart can generate Trendlines for Cartesian type series (*Line, Column, Scatter, Area, Candle, HiLo etc.*) except bar type series. You can add more than one trendline object to the [trendlines](#) option.

### JAVASCRIPT

```
"use strict";
```

```

var series=[{
trendlines: [{
//Enable Trendline to chart series
visibility: "visible", type: "linear"
}],
//...
}];
//...
ReactDOM.render(
<EJ.Chart id="default_chart_sample_0"
series={series}
>
</EJ.Chart>,
document.getElementById('chart')
);

```



[Click](#) here to view the Trendlines online demo sample.

### Customize the trendline styles

A trendline can be customized by using the properties such as [fill](#), [width](#), [dashArray](#) and [opacity](#). The default type of trendline is **"linear"**.

### JAVASCRIPT

```

"use strict";
var series=[{
trendlines: [{
//...
//Customize the Trendline styles
fill: '#99CCFF', width: 3, opacity: 1, dashArray: '2,3'
}],
//...
}];
//...
ReactDOM.render(
<EJ.Chart id="default_chart_sample_0"
series={series}
>
</EJ.Chart>,
document.getElementById('chart')
);

```



### Types of Trendline

EJChart supports the following type of Trendlines.

- Linear
- Exponential
- Logarithmic
- Power
- Polynomial

- MovingAverage

### Linear

To render Linear Trendline, you have to set the [type](#) as “linear”.

#### JAVASCRIPT

```
"use strict";
var series=[{
  trendlines: [{
    //Change Trendline type
    type: "linear"
  }],
  //...
}];
//...
ReactDOM.render(
  <EJ.Chart id="default_chart_sample_0"
  series={series}
  >
</EJ.Chart>,
  document.getElementById('chart')
);
```



### Exponential

Exponential Trendline can be rendered by setting the [type](#) as “exponential”.

#### JAVASCRIPT

```
"use strict";
var series=[{
  trendlines: [{
    //Change Trendline type
    type: "exponential"
  }],
  //...
}];
//...
ReactDOM.render(
  <EJ.Chart id="default_chart_sample_0"
  series={series}
  >
</EJ.Chart>,
  document.getElementById('chart')
);
```



### Logarithmic

Logarithmic Trendline can be rendered by setting the [type](#) as “Logarithmic”.

#### JAVASCRIPT

```
"use strict";
```

```

var series=[{
trendlines: [{
//Change Trendline type
type: "logarithmic"
}],
//...
}];
//...
ReactDOM.render(
<EJ.Chart id="default_chart_sample_0"
series={series}
>
</EJ.Chart>,
document.getElementById('chart')
);

```



#### Power

Power Trendline can be rendered by setting the [type](#) of the trendline as “power”.

#### JAVASCRIPT

```

"use strict";
var series=[{
trendlines: [{
//Change Trendline type
type: "power"
}],
//...
}];
//...
ReactDOM.render(
<EJ.Chart id="default_chart_sample_0"
series={series}
>
</EJ.Chart>,
document.getElementById('chart')
);

```



#### Polynomial

Polynomial Trendline can be rendered by setting the trendline [type](#) as “polynomial”. You can change the polynomial order by using the **polynomialOrder** of the trendlines. It ranges from 2 to 6.

#### JAVASCRIPT

```

"use strict";
var series=[{
trendlines: [{
//Change Trendline type
type: "polynomial"
}],
//...
}];

```

```
//...
ReactDOM.render(
  <EJ.Chart id="default_chart_sample_0"
  series={series}
  >
</EJ.Chart>,
  document.getElementById('chart')
);
```



### MovingAverage

MovingAverage Trendline can be rendered by setting the [type](#) of the trendline as “movingAverage”.

### JAVASCRIPT

```
"use strict";
var series=[{
  trendlines: [{
    //Change Trendline type and set ['period'] (../api/ejchart#members:series-
    trendlines-period) for moving average
    type: "movingAverage", period: 3
  }],
  //...
}];
//...
ReactDOM.render(
  <EJ.Chart id="default_chart_sample_0"
  series={series}
  >
</EJ.Chart>,
  document.getElementById('chart')
);
```



### Forecasting

**Trendline forecasting** is the prediction of future/past situations. **Forecasting** can be classified into two types as follows.

- Forward Forecasting
- Backward Forecasting

### Forward Forecasting

The value set for [forwardForecast](#) is used to determine the distance moving towards the future trend.

### JAVASCRIPT

```
"use strict";
var series=[{
  trendlines: [{
    //...
    //Set forward forecasting value
    forwardForecast: 5
  }],
  //...
```

```
//...
}];
//...
ReactDOM.render(
  <EJ.Chart id="default_chart_sample_0"
  series={series}
  >
</EJ.Chart>,
  document.getElementById('chart')
);
```



### Backward Forecasting

The value set for the [backwardForecast](#) is used to determine the past trends.

### JAVASCRIPT

```
"use strict";
var series=[{
  trendlines: [{
    //...
    //Set backward forecasting value
    backwardForecast: 5
  }],
  //...
}];
//...
ReactDOM.render(
  <EJ.Chart id="default_chart_sample_0"
  series={series}
  >
</EJ.Chart>,
  document.getElementById('chart')
);
```



### Trendlines Legend

To display the legend item for trendline, use the [name](#) property. You can interact with the trendline legends similar to the series legends (*show/hide trendlines on legend click*).

### JAVASCRIPT

```
"use strict";
var series=[{
  trendlines: [{
    //...
    //Set Trendline name to display in the legend
    name: 'Linear'
  }],
  //...
}];
//...
ReactDOM.render(
  <EJ.Chart id="default_chart_sample_0"
```

```

series={series}
>
</EJ.Chart>,
document.getElementById('chart')
);

```



## Technical Indicators

EjChart control supports 10 types of technical indicators.

Bind data to render the indicator

You can bind the series [dataSource](#) to the indicator by setting the specific series name to the indicator by using the [indicators.seriesName](#) property.

### JAVASCRIPT

```

"use strict";
// ...
//Initializing Series
var series=[{
dataSource: chartData,
xName: "xDate",
high: "High",
low: "Low",
open: "Open",
close: "Close",
//Set name to series
name: 'Hilo',
// ...
}];
//Initializing Indicators
var indicators= [{
//Set Hilo series dataSource to indicator using seriesName
seriesName: "Hilo",
// ...
}];
// ...
ReactDOM.render(
<EJ.Chart id="default_chart_sample_0"
series={series}
indicators={indicators}
>
</EJ.Chart>,
document.getElementById('chart')
);

```

Also, you can add data to the indicator directly by using the [dataSource](#) option of the indicator.

### JAVASCRIPT

```

"use strict";
//Initializing Indicators
var indicators= [{
//Add dataSource to indicator directly
dataSource: chartData,

```

```

xName: "xDate",
high: "High",
low: "Low",
open: "Open",
close: "Close",
// ...
}];
// ...
ReactDOM.render(
<EJ.Chart id="default_chart_sample_0"
indicators={indicators}
>
</EJ.Chart>,
document.getElementById('chart')
);

```

## Indicator Types

### *Accumulation Distribution*

To create an Accumulation Distribution indicator, set the [indicators.type](#) as “**accumulationdistribution**”. Accumulation Distribution require ‘**volume**’ field additionally with the [dataSource](#) to calculate the signal line.

### **JAVASCRIPT**

```

"use strict";
// ...
//Initializing Series
var series=[{
name: "Hilo",
dataSource: chartData,
xName: "xDate",
high: "High",
low: "Low",
open: "Open",
close: "Close",
//Add additional volume field to data source for accumulation distribution
volume: "Volume",
// ...
}];
//Initializing Indicators
var indicators= [{
seriesName: "Hilo",
//Set indicator type
type: "accumulationdistribution",
// ...
}];
// ...
ReactDOM.render(
<EJ.Chart id="default_chart_sample_0"
indicators={indicators}
series={series}
>
</EJ.Chart>,
document.getElementById('chart')
);

```





[Click](#) here to view the Accumulation Distribution indicator online demo sample.

#### *Average True Range (ATR)*

You can create an ATR indicator by setting the [indicators.type](#) as “atr” in the [indicators](#).

#### **JAVASCRIPT**

```
"use strict";
//Initializing Indicators
var indicators= [{
//Set indicator type
type: "atr",
// ...
}];
// ...
ReactDOM.render(
<EJ.Chart id="default_chart_sample_0"
indicators={indicators}
>
</EJ.Chart>,
document.getElementById('chart')
);
```



[Click](#) here to view the ATR indicator online demo sample.

#### *Bollinger Band*

Bollinger Band indicator is created by setting the [indicators.type](#) as “bollingerband”. It contains three lines, namely upper band, lower band and signal line. Bollinger Band default value of the period is 14 and standardDeviations is 2.

#### **JAVASCRIPT**

```
"use strict";
//Initializing Indicators
var indicators= [{
//Set indicator type
type: "bollingerband",
// ...
}];
// ...
ReactDOM.render(
<EJ.Chart id="default_chart_sample_0"
indicators={indicators}
>
</EJ.Chart>,
document.getElementById('chart')
);
```



[Click](#) here to view the Bollinger Band indicator online demo sample.

*Exponential Moving Average (EMA)*

To render an EMA indicator, you have to set the [indicators.type](#) as “**ema**”.

**JAVASCRIPT**

```
"use strict";
//Initializing Indicators
var indicators= [{
//Set indicator type
type: "ema",
// ...
}];
// ...
ReactDOM.render(
<EJ.Chart id="default_chart_sample_0"
indicators={indicators}
>
</EJ.Chart>,
document.getElementById('chart')
);
```



[Click](#) here to view the EMA indicator online demo sample.

*Momentum*

Momentum Technical indicator is created by setting the [indicators.type](#) as “**momentum**”. The momentum indicator renders two lines, namely upper band and signal line. Upper band always rendered at the value 100 and the signal line is calculated based on the momentum of the data.

**JAVASCRIPT**

```
"use strict";
//Initializing Indicators
var indicators= [{
//Set indicator type
type: "momentum",
// ...
}];
// ...
ReactDOM.render(
<EJ.Chart id="default_chart_sample_0"
indicators={indicators}
>
</EJ.Chart>,
document.getElementById('chart')
);
```



[Click](#) here to view the Momentum indicator online demo sample.

*Moving Average Convergence Divergence (MACD)*

To render an MACD indicator, you have to set the [indicators.type](#) as “**macd**”. MACD indicator contains MACD line, Signal line and Histogram column. Histogram is used to differentiate MACD and signal line.

**JAVASCRIPT**

```

"use strict";
//Initializing Indicators
var indicators= [{
//Set indicator type
type: "macd",
// ...
}];
// ...
ReactDOM.render(
<EJ.Chart id="default_chart_sample_0"
indicators={indicators}
>
</EJ.Chart>,
document.getElementById('chart')
);

```



[Click](#) here to view the MACD indicator online demo sample.

**macdType**

By using the [macdType](#) enumeration property, you can change the MACD rendering as *line*, *histogram* or *both*.

**JAVASCRIPT**

```

"use strict";
//Initializing Indicators
var indicators= [{
//Set indicator type
type: "macd",
//Set macd draw type
macdType: "histogram",
// ...
}];
// ...
ReactDOM.render(
<EJ.Chart id="default_chart_sample_0"
indicators={indicators}
>
</EJ.Chart>,
document.getElementById('chart')
);

```



**Relative Strength Index (RSI)**

To render the RSI indicator, set the [indicators.type](#) as “rsi”. It contains three lines, namely upper band, lower band and signal line. Upper and lower band always render at value 70 and 30 respectively and signal line is calculated based on the **RSI** formula.

**JAVASCRIPT**

```

"use strict";

```

```
//Initializing Indicators
var indicators= [{
//Set indicator type
type: "rsi",
// ...
}];
// ...
ReactDOM.render(
<EJ.Chart id="default_chart_sample_0"
indicators={indicators}
>
</EJ.Chart>,
document.getElementById('chart')
);
```



[Click](#) here to view the RSI indicator online demo sample.

#### Simple Moving Average (SMA)

To render the SMA indicator, you should specify the [indicators.type](#) as “sma”.

#### JAVASCRIPT

```
"use strict";
//Initializing Indicators
var indicators= [{
//Set indicator type
type: "sma",
// ...
}];
// ...
ReactDOM.render(
<EJ.Chart id="default_chart_sample_0"
indicators={indicators}
>
</EJ.Chart>,
document.getElementById('chart')
);
```



[Click](#) here to view the SMA indicator online demo sample.

#### Stochastic

For the Stochastic indicator, you need to set the [indicators.type](#) as “stochastic”. The Stochastic indicator renders four lines namely, upper line, lower line, stochastic line and the signal line. Upper line always rendered at value 80 and the lower line is rendered at value 20. Stochastic and Signal Lines are calculated based on the stochastic formula.

#### JAVASCRIPT

```
"use strict";
//Initializing Indicators
var indicators= [{
//Set indicator type
```

```

type: "stochastic",
// ...
}];
// ...
ReactDOM.render(
<EJ.Chart id="default_chart_sample_0"
indicators={indicators}
>
</EJ.Chart>,
document.getElementById('chart')
);

```

![[/js/Chart/Technical-Indicatorsimages/Technical-Indicatorsimg10.png)

[Click](#) here to view the stochastic indicator online demo sample.

*Triangular Moving Average (TMA)*

To render the TMA indicator, you should specify the [indicators.type](#) as "tma".

#### JAVASCRIPT

```

"use strict";
//Initializing Indicators
var indicators= [{
//Set indicator type
type: "tma",
// ...
}];
// ...
ReactDOM.render(
<EJ.Chart id="default_chart_sample_0"
indicators={indicators}
>
</EJ.Chart>,
document.getElementById('chart')
);

```

![[/js/Chart/Technical-Indicatorsimages/Technical-Indicatorsimg11.png)

[Click](#) here to view the TMA indicator online demo sample.

#### Enable Tooltip

To display the indicator tooltip, use [visible](#) option of the [indicators.tooltip](#). Also, you can change and customize the tooltip color, border, format and font properties similar to the series tooltip.

#### JAVASCRIPT

```

"use strict";
//Initializing Indicators
var indicators= [{
// ...
tooltip: {
//Enable tooltip for indicator
visible: true
},
}];
// ...

```

```
ReactDOM.render(
  <EJ.Chart id="default_chart_sample_0"
    indicators={indicators}
  >
  </EJ.Chart>,
  document.getElementById('chart')
);
```



## Annotations

annotations are used to mark the specific area of interest in the chart area with texts, shapes or images.

You can add annotations to the chart by using the [annotations](#) option. By using the [content](#) option of annotation object, you can specify the id of the element that needs to be displayed in the chart area.

## HTML

```
<body>
<div id="chartcontainer"></div>
<div id="watermark" style="font-size:100px; display:none">2014</div>
</body>
```

## JAVASCRIPT

```
// ...
annotations: [
  //Add Annotation content here
  { visible: true, content: "watermark", opacity: 0.2, region: "series" }
  // ...
],
// ...
ReactDOM.render(
  <EJ.Chart id="default_chart_sample_0"
    annotations={annotations}
  >
  </EJ.Chart>,
  document.getElementById('chart')
);
```



[Click](#) here to view the Annotations online demo sample.

**Note:** Annotations are not supported in 3D chart types.

## Rotate the annotation template

To rotate the annotation template, you can use the [angle](#) property of the annotations.

## JAVASCRIPT

```
"use strict";
// ...
annotations: [{ visible: true,
  content: "watermark",
  //Rotate the Annotation template
```

```

angle: 270,
},
// ...
],
// ...
ReactDOM.render(
<EJ.Chart id="default_chart_sample_0"
annotations={annotations}
>
</EJ.Chart>,
document.getElementById('chart')
);

```



### Positioning Annotation

You can position annotations either by using the coordinates ([x](#) and [y](#) options) or by using the alignment options ([horizontalAlignment](#) and [verticalAlignment](#)).

By using the [coordinateUnit](#) option, you can specify whether the value provided in the [x](#) and [y](#) options are relative to the chart or axis.

- If the [coordinateUnit](#) is set to none, the annotations are placed relative to the chart/plot area by using the [horizontalAlignment](#) and [verticalAlignment](#) options.
- If the [coordinateUnit](#) is set to points, the x and y values of the annotation are the coordinates relative to the axis and annotation is positioned relative to the axis. By default, the x and y values are associated with the [primaryXAxis](#) and [primaryYAxis](#). In case, when the chart contains multiple axis and you want to associate the annotation with a particular axis, you can specify the [xAxisName](#) and [yAxisName](#) options of the annotation object.
- If the [coordinateUnit](#) is set to pixels, the x and y values are coordinates relative to the top-left corner of the chart/plot area.

**Note:** By using the [region](#) option, you can specify whether the annotation is placed relative to the entire chart or plot area.

### JAVASCRIPT

```

"use strict";
// ...
annotations: [{ visible: true,
content: "lowtemp",
//Change coordinateUnit type to pixels
coordinateUnit: "pixels", x: 170, y: 350,
// ...
}],
// ...
],
// ...
ReactDOM.render(
<EJ.Chart id="default_chart_sample_0"
annotations={annotations}
>
</EJ.Chart>,
document.getElementById('chart')
);

```

```
);
```



### Annotation alignments

When the coordinateUnit is set to pixels or points, you can align the annotation relative to the coordinates by using the [horizontalAlignment](#) and [verticalAlignment](#) options.

### JAVASCRIPT

```
"use strict";
// ...
annotations: [{ visible: true,
content: "hightemp",
//Change alignment of annotation template
verticalAlignment: "middle",
horizontalAlignment: "near",
margin: { right: 40 }
}],
// ...
],
// ...
ReactDOM.render(
<EJ.Chart id="default_chart_sample_0"
annotations={annotations}
>
</EJ.Chart>,
document.getElementById('chart')
);
```



### Appearance

#### Custom Color Palette

The Chart displays different series in different colors by default. You can customize the color of each series by providing a custom color palette of your choice by using the [palette](#) property.

### JAVASCRIPT

```
"use strict";
//Providing a custom palette
var palette= [ "grey", "skyblue", "orange", ];
// ...
ReactDOM.render(
<EJ.Chart id="default_chart_sample_0"
palette={palette}
>
</EJ.Chart>,
document.getElementById('chart')
);
```



**Note:** The Color palette is applied to the points in accumulation type series



### Built-in Themes

Following are the built-in themes available in the Chart

- flatlight
- flatdark
- gradientlight
- gradientdark
- azure
- azuredark
- lime
- limedark
- saffron
- saffrondark
- gradient-azure
- gradient-azuredark
- gradient-lime
- gradient-limedark
- gradient-saffron
- gradient-saffrondark

You can set your desired theme by using the [theme](#) property. Flat light is the default theme used in the Chart.

#### JAVASCRIPT

```
"use strict";
//Using gradient theme
var theme= "gradientlight";
// ...
ReactDOM.render(
<EJ.Chart id="default_chart_sample_0"
theme={theme}
>
</EJ.Chart>,
document.getElementById('chart')
);
```



### Point level customization

Marker, data label and fill color of each point in a series can be customized individually by using the [points](#) collection.

#### JAVASCRIPT

```
"use strict";
var series= [{
//Customizing marker and fill color of a point
points: [
{
x : 0,
y: 210,
fill: "#E27F2D",
```

```

marker: {
  visible: true,
  // ...
},
// ...
],
// ...
}];
// ...
ReactDOM.render(
  <EJ.Chart id="default_chart_sample_0"
  series={series}
  >
</EJ.Chart>,
  document.getElementById('chart')
);

```



### Series border customization

To customize the series border color, width and dashArray, you can use [series.border](#) option.

**Note:** Series border can be applied to all the series (except Line, Spline, HiLo, HiLoOpenClose and StepLine series).

### JAVASCRIPT

```

"use strict";
//...
var series= [{
  //Change the color, width and dashArray to customize the border of series
  border: { color: "blue", width: 2, dashArray: "5,3" }
  //...
}];
//...
ReactDOM.render(
  <EJ.Chart id="default_chart_sample_0"
  series={series}
  >
</EJ.Chart>,
  document.getElementById('chart')
);

```



### Chart area customization

#### Customize chart background

The Chart background can be customized by using the [background](#) property of the Chart. To customize the chart border, use [border](#) option of the chart.

### JAVASCRIPT

```

"use strict";
// ...
//Customizing Chart background

```

```

var background= "skyblue";
//Customize the chart border and opacity
var border= {color: "#FF0000", width: 2, opacity: 0.35};
// ...
ReactDOM.render(
<EJ.Chart id="default_chart_sample_0"
background={background}
border={border}
>
</EJ.Chart>,
document.getElementById('chart')
);

```



### Chart Margin

The Chart [margin](#) property is used to add the margin to the chart area at the left, right, top and bottom position.

#### JAVASCRIPT

```

"use strict";
// ...
//Change chart margin to left, right, top and bottom
var margin= { left: 40, right: 40, top: 40, bottom: 40 };
// ...
ReactDOM.render(
<EJ.Chart id="default_chart_sample_0"
margin={margin}
>
</EJ.Chart>,
document.getElementById('chart')
);

```



### Setting background image

Background image can be added to the chart by using the [backGroundImageUrl](#) property.

#### JAVASCRIPT

```

"use strict";
// ...
//Setting an image as Chart background
var backGroundImage= "images/chart/wheat.png";
// ...
ReactDOM.render(
<EJ.Chart id="default_chart_sample_0"
backGroundImageUrl={backGroundImage}
>
</EJ.Chart>,
document.getElementById('chart')
);

```



[Click](#) here to view our online demo sample for setting Chart background image.

### Chart area background

The Chart area background can be customized by using the [background](#) property in the chart area.

#### JAVASCRIPT

```
"use strict";
var chartArea= {
  //Setting background for Chart area
  background: "skyblue"
};
// ...
ReactDOM.render(
  <EJ.Chart id="default_chart_sample_0"
    chartArea={chartArea}
  >
  </EJ.Chart>,
  document.getElementById('chart')
);
```



### Customize chart area grid bands

You can provide different color for alternate grid rows and columns formed by the grid lines in the chart area by using the [alternateGridBand](#) property of the axis. The properties [odd](#) and [even](#) are used to customize the grid bands at odd and even positions respectively.

#### JAVASCRIPT

```
"use strict";
// ...
//Creating horizontal grid bands in chart area
var primaryYAxis= {
  //Customizing horizontal grid bands at even position
  alternateGridBand: {
    even: {
      fill: "#A7A9AB",
      opacity: 0.1,
    }
  }
  // ...
};
// ...
ReactDOM.render(
  <EJ.Chart id="default_chart_sample_0"
    primaryYAxis={primaryYAxis}
  >
  </EJ.Chart>,
  document.getElementById('chart')
);
```



[Click](#) here to view the alternate grid band online demo sample.

### Animation

You can enable animation by using the [enableAnimation](#) property of the series. This animates the chart series on two occasions – when the chart is loaded for the first time or whenever you change the series type by using the type property.

#### JAVASCRIPT

```
"use strict";
// ...
var series = [{
  //Enabling animation of series
  enableAnimation: true,
  // ...
}];
// ...
ReactDOM.render(
  <EJ.Chart id="default_chart_sample_0"
  series={series}
  >
</EJ.Chart>,
  document.getElementById('chart')
);
```

However, you can force the chart to animate series by calling the animate method as illustrated in the following code example,

#### JAVASCRIPT

```
"use strict";
// ...
var series = [{
  //Enabling animation of series
  enableAnimation: true,
  // ...
}];
// ...
ReactDOM.render(
  <EJ.Chart id="default_chart_sample_0"
  series={series}
  >
</EJ.Chart>,
  document.getElementById('chart')
);
//Dynamically animating Chart
function animateChart(){
  //Calling the animate method for dynamic animation
  $("#chartcontainer").ejChart("animate");
}
```

### Control the Speed of animation

To control the speed of animation, you can use the [animationDuration](#) property in the series.

#### JAVASCRIPT

```
"use strict";
// ...
```

```
var series = [{
  //Enabling animation of series
  enableAnimation: true,
  animationDuration:"2000"
  // ...
}];
// ...
ReactDOM.render(
  <EJ.Chart id="default_chart_sample_0"
  series={series}
  >
  </EJ.Chart>,
  document.getElementById('chart')
);
```

## Rendering Modes

Chart uses following three rendering technologies

- VML
- SVG
- HTML5 Canvas

### VML

VML is used to render chart in IE lower versions such as IE7 and IE8. VML is used by default in these two browsers, as SVG and Canvas are not supported in these browsers.

#### Limitations:

- Label rotation is not supported.
- Animation is not supported.
- Patterns are not supported.
- Zooming and panning features are not supported.

### SVG

SVG is used to render Chart by default for all browsers other than IE7 and IE8.

### Canvas

You can switch between SVG and Canvas rendering by using the [enableCanvasRendering](#) option. The canvas mode rendering is used in the following scenarios,

- Plotting large number of data points.
- Performing high frequency live updates.

The following code example shows how to enable HTML5 Canvas rendering in chart.

#### JAVASCRIPT

```
"use strict";
//Rendering chart in canvas mode
var canvas= true;
// ...
ReactDOM.render(
```

```
<EJ.Chart id="default_chart_sample_0"
enableCanvasRendering={canvas}
>
</EJ.Chart>,
document.getElementById('chart')
);
```



#### Limitations:

- Animation is not supported.
- 3D charts are not supported.

### Real-Time Chart

Chart can be updated dynamically with the real time data. Whenever you add a point or remove a point from the dataSource, you can call the [redraw](#) method to request the chart to redraw its content.

**Note:** You can get the chart **instance** using instance method.

#### JAVASCRIPT

```
"use strict";
window.setInterval(updateChart, 200);
var series= [
{ points: [ ] }
];
//Function that updates chart dynamically
function updateChart(){
//Creating chart instance
var chart = $("#default_chart_sample_0").ejChart("instance");
if (chart.model.series[0].points.length > 10)
chart.model.series[0].points.splice(0, 1);
var point = chart.model.series[0].points;
var xValue = point.length > 0 ? point[point.length - 1].x + 1 : 1;
point[point.length] = { x: xValue, y: getRandomNum( 1000 ) }
//Update Chart dynamically using redraw option
//chart.redraw() can also be used here instead of redraw option
$("#default_chart_sample_0").ejChart("redraw");
}
// ...
ReactDOM.render(
<EJ.Chart id="default_chart_sample_0"
>
</EJ.Chart>,
document.getElementById('chart')
);
```

[Click](#) here to view the online demo sample for real-time Chart.

### Exporting Chart

Exporting a chart can be done in both client-side and in server-side. This can be modified by setting values to the property "mode" in exporting. Default value for mode is client.

### Client-side Exporting

In client-side rendered chart can be exported as PNG image or as SVG file.

- Export as PNG - The chart can be exported to image when it is rendered in HTML5 Canvas. To render a chart in canvas, set the [enableCanvasRendering](#) option to *true*. To export the chart, you can use the [export](#) method of the chart. Refer to the online [KB for exporting](#) to know more about chart exporting.
- Export as SVG - Chart can be exported as SVG if it is rendered as a scalable vector graphics element. By default chart will be rendered as SVG.

### HTML

```
<body>
<!--Chart download link-->
<a id="download" style="cursor: pointer; position: absolute;right:
150px;">ExportChart</a>
<div id="chartcontainer"></div>
<script>
if (document.getElementById('download').addEventListener)
document.getElementById('download').addEventListener('click', download,
false);
else
document.getElementById('download').attachEvent('onclick', download, false);
</script>
</body>
```

### JAVASCRIPT

```
"use strict";
// ...
//Enable Canvas mode to export chart as image
var canvas= true;
//Setting up required exporting options
var exporting= { type: "png", mode: "client", fileName: "ChartSnapshot" };
ReactDOM.render(
<EJ.Chart id="default_chart_sample_0"
enableCanvasRendering={canvas}
exporting={exporting}
>
</EJ.Chart>,
document.getElementById('chart')
);
```

[Click](#) here to view the Export chart online demo sample.

### Server-side Exporting

Server-side operation can be done by using the server-side frameworks such as WebApi, WCF service to achieve exporting.

#### Server side implementation

- To convert the chart data from client to server-side, refer to the following steps.
- Create an MVC application and add a controller.



- Add Syncfusion.EJ, Syncfusion.EJ.MVC and Syncfusion.EJ.Export as references to the application.
- Parse the data from client in MVC controller and export the chart to client.
- Host the MVC application in your server and get the link for exporting action method Example: <http://js.syncfusion.com/ExportingServices/api/JSChartExport/ExcelExport>
- To pass client data to server-side, you need to call the export method and pass export type (either image or excel) and server-side URL as an argument. The third argument of the export method is a Boolean property that specifies whether only the current chart should be exported or all charts in page should be exported.

### HTML

```
<body>
<!--Export Chart-->
<a id="download" style="cursor: pointer; position: absolute;">
<button onclick="download()" value="Export">Export</button>
</a>
<div id="chartcontainer"></div>
</body>
```

### JAVASCRIPT

```
"use strict";
var canvas = true;
var exportSettings = { type: "jpg", action:
"http://js.syncfusion.com/ExportingServices/api/JSChartExport/Export" };
ReactDOM.render(
<EJ.Chart id="default_chart_sample_0"
enableCanvasRendering = { canvas }
exportSettings={exportSettings}
>
</EJ.Chart>,
document.getElementById('chart')
);
```

At the server side, chart can be exported as JPG, PNG, SVG, PDF, word document and as excel documents. For this the following code have to place in the controller.

### CSHARP

```
public void ExportChart(string Data, string ChartModel)
{
    // declaration
    ChartProperties obj = ConvertChartObject(ChartModel);
    string type = obj.ExportSettings.Type.ToString().ToLower();
    string fileName = obj.ExportSettings.FileName;
    string orientation = obj.ExportSettings.Orientation.ToString();
    if (type == "svg") // for svg export
    {
        StringWriter oStringWriter = new StringWriter();
        string data = HttpUtility.HtmlDecode(Data);
        data = HttpUtility.UrlDecode(Data);
        data = System.Uri.UnescapeDataString(Data);
        oStringWriter.WriteLine(System.Uri.UnescapeDataString(Data));
        Response.ContentType = "text/plain";
    }
}
```

```

Response.AddHeader("Content-Disposition",
String.Format("attachment;filename={0}", (obj.ExportSettings.FileName +
".svg")));
Response.Clear();
using (StreamWriter writer = new StreamWriter(Response.OutputStream))
{
    data = oStringWriter.ToString();
    writer.Write(oStringWriter.ToString());
}
Response.End();
}
else if (type == "xlsx")           // to export chart as excel
{
    List<ExportChartData> chartData = new List<ExportChartData>();
    chartData.Add(new ExportChartData("John", 10));
    chartData.Add(new ExportChartData("Jake", 12));
    chartData.Add(new ExportChartData("Peter", 18));
    chartData.Add(new ExportChartData("James", 11));
    chartData.Add(new ExportChartData("Mary", 9.7));
    ExcelExport exp = new ExcelExport();
    exp.Export(obj, (IEnumerable)chartData, fileName + ".xlsx",
    ExcelVersion.Excel2010, null, null);
}
else
{
    Data = Data.Remove(0, Data.IndexOf(',') + 1);
    MemoryStream stream = new MemoryStream(Convert.FromBase64String(Data));
    if (type == "docx")           // to export as word document
    {
        WordDocument document = new WordDocument();
        IWSection section = document.AddSection();
        IWParagraph paragraph = section.AddParagraph();
        if (obj.ExportSettings.Orientation.ToString() == "Landscape")
            section.PageSetup.Orientation = PageOrientation.Landscape;
        else
            section.PageSetup.Orientation = PageOrientation.Portrait;
        paragraph.AppendPicture(Image.FromStream(stream));
        document.Save(fileName + ".doc", Syncfusion.DocIO.FormatType.Doc,
        HttpContext.ApplicationInstance.Response,
        Syncfusion.DocIO.HttpContentDisposition.Attachment);
    }
    else if (type == "pdf")       // to export as PDF
    {
        PdfDocument pdfDoc = new PdfDocument();
        pdfDoc.Pages.Add();
        if (obj.ExportSettings.Orientation.ToString() == "Landscape")
            pdfDoc.Pages[0].Section.PageSettings.Orientation =
            PdfPageOrientation.Landscape;
        else
            pdfDoc.Pages[0].Section.PageSettings.Orientation =
            PdfPageOrientation.Portrait;
        pdfDoc.Pages[0].Graphics.DrawImage(PdfImage.FromStream(stream), new
        PointF(10, 30));
        pdfDoc.Save(obj.ExportSettings.FileName + ".pdf",
        HttpContext.ApplicationInstance.Response, HttpReadType.Save);
        pdfDoc.Close();
    }
}

```

```

else // to export as image
{
    stream.WriteTo(Response.OutputStream);
    Response.ContentType = "application/octet-stream";
    Response.AddHeader("Content-Disposition",
        String.Format("attachment;filename={0}", fileName + "." + type));
    Response.Flush();
    stream.Close();
    stream.Dispose();
}
}
}
private ChartProperties ConvertChartObject(string ChartModel)
{
    JavaScriptSerializer serializer = new JavaScriptSerializer();
    IEnumerable div = (IEnumerable)serializer.Deserialize(ChartModel,
        typeof(IEnumerable));
    ChartProperties chartProp = new ChartProperties();
    foreach (KeyValuePair<string, object> ds in div)
    {
        var property = chartProp.GetType().GetProperty(ds.Key, BindingFlags.Instance
            | BindingFlags.Public | BindingFlags.IgnoreCase);
        if (property != null)
        {
            Type type = property.PropertyType;
            string serialize = serializer.Serialize(ds.Value);
            object value = serializer.Deserialize(serialize, type);
            property.SetValue(chartProp, value, null);
        }
    }
    return chartProp;
}

```

### Excel Exporting

Excel exporting is a server-side operation. In addition have to refer Syncfusion.XlsIO assembly to export as excel.



### Multiple chart excel exporting

EjChart supports exporting more than one charts in a page, with the third argument for the export method.

**Note:** Refer the MultipleExportType.AppendToSheet, MultipleExportType.NewSheet.

### JAVASCRIPT

```

"use strict";
//Render Chart1
ReactDOM.render(
    <EJ.Chart id="chart1">
    </EJ.Chart>,
    document.getElementById('container1')
);
ReactDOM.render(
    <EJ.Chart id="chart2">

```

```

</EJ.Chart>,
document.getElementById('container2')
);
//Export multiple chart to excel
function downloadExcel() {
var chart = $("#container1").ejChart("instance");
chart.export('Excel',
'http://js.syncfusion.com/ExportingServices/api/JSChartExport/ExcelExport',
true);
}

```

Export multiple chart to excel at server-side

### **CSHARP**

```

public class JSChartExportController : ApiController
{
[System.Web.Http.ActionName("ExcelExport")]
[AcceptVerbs("POST")]
public void ExcelExport()
{
string chartModel = HttpContext.Current.Request.Params["ChartModel"];
IWorkbook book = null;
foreach (string chartProperty in ChartModel)
{
if (chartProperty != null)
{
ExcelExport exp = new ExcelExport();
ChartProperties obj = ConvertChartObject(chartProperty);
if (initial)
{
book = exp.Export((obj as ChartProperties), (IEnumerable)data,
"Export1.xlsx",
ExcelVersion.Excel2010, true, null, null);
initial = false;
}
else
{
exp.Export((obj as ChartProperties), (IEnumerable)data, "Export.xlsx",
ExcelVersion.Excel2010, false, book, MultipleExportType.NewSheet, null,
null);
}
}
}
}
}
}

```



### Naming the exported file

ejChart provides options to customize the name of the file to be exported. This can be done by setting the name of the file to the property "fileName" in exporting.

### Rotating the chart

We can also rotate the chart and can export it. Possible angles of rotation are 0, 90, -90 and 180 degree. This can be achieved by setting values to the "angle" property in exporting.

#### JAVASCRIPT

```
"use strict";
var exportSettings={ angle: 180, action:
"http://js.syncfusion.com/ExportingServices/api/JSChartExport/Export" };
ReactDOM.render (
<EJ.Chart id="default_chart_sample_0"
exportSettings={exportSettings}
>
</EJ.Chart>,
document.getElementById('chart')
);
```



### Setting orientation for the document

This is applicable for PDF, excel and word documents. By setting values to the "orientation" property in exporting, we can change the orientation of those documents. By default it will export with portrait orientation.

### Printing Chart

The rendered chart can be printed directly from the browser by calling the public method [print](#). ID of the chart div element must be passed as argument to that method.

#### HTML

```
<body>
<button type="button" onclick="print()" ></button>
<div id="chartcontainer"></div>
<script>
function print() {
var chartObj = $("#chartcontainer").ejChart("instance");
chartObj.print("chartcontainer");
}
</script>
</body>
```

#### HTML

```
ReactDOM.render (
<EJ.Chart id="chartcontainer">
</EJ.Chart>,
document.getElementById('chartcontainer')
);
```

This print method can be called by performing any action on the web page. For example, by clicking a button. While calling the print method in chart, print preview will be displayed in the browser.



[Click](#) here to view the Printing chart online demo sample

### Printing Multiple chart

Multiple charts in a web page can be printed together. For printing multiple charts, ID of the chart div elements have to be passed as shown in the below code

#### JAVASCRIPT

```
//Render Chart1
ReactDOM.render(
  <EJ.Chart id="container1">
  </EJ.Chart>,
  document.getElementById('container1')
);
//Render Chart2
ReactDOM.render(
  <EJ.Chart id="container2">
  </EJ.Chart>,
  document.getElementById('container2')
);
//Print multiple charts
function print() {
  var chartObj = $("#container1").ejChart("instance");
  chartObj.print("container1", "container2");
}
```

The Print preview of multiple Charts is shown below



### Page Setup

Some of print options are not configurable through JavaScript code. You need to customize layout, paper size, margins options through browser's page setup dialog. Please find the following guidelines link to browser page setup.

- [Chrome](#)
- [Firefox](#)
- [Safari](#)
- [IE](#)

### 3D Chart

Essential 3D Chart for JavaScript allows you to view 8 chart types in 3D view such as [Bar](#), [StackingBar](#), [StackingBar100](#), [Column](#), [Stacked Column](#), [100% Stacked Column](#), [Pie](#), [Doughnut](#).

#### 3D Column Chart

For rendering a 3D Column Chart, specify the series [type](#) as “**column**” in the chart series and set [enable3D](#) option as **true** in the chart.

#### JAVASCRIPT

```
"use strict";
// ...
var series= [{
  //Set chart type to series
  type: 'column',
  // ...
}
```

```

    }];
    // Enable 3D Chart
    var enable3D= true;
    // ...
    ReactDOM.render(
    <EJ.Chart id="default_chart_sample_0"
    series={series}
    enable3D={enable3D}
    >
    </EJ.Chart>,
    document.getElementById('chart')
    );

```



[Click](#) here to view the 3D Column Chart online demo sample.

### 3D Bar Chart

You can create a 3D Bar Chart by setting the series [type](#) as “**bar**” in the chart series and enable [enable3D](#) option in the chart.

#### JAVASCRIPT

```

"use strict";
// ...
var series= [{
//Set chart type to series
type: 'bar',
// ...
}];
// Enable 3D Chart
var enable3D= true;
// ...
ReactDOM.render(
<EJ.Chart id="default_chart_sample_0"
series={series}
enable3D={enable3D}
>
</EJ.Chart>,
document.getElementById('chart')
);

```



[Click](#) here to view the 3D Bar Chart online demo sample.

### 3D Stacked Column Chart

Stacking Column 3DChart is rendered by specifying the series [type](#) as “**stackingColumn**” in the chart series and enable [enable3D](#) option in the chart.

#### JAVASCRIPT

```

"use strict";
// ...
var series= [{
//Set chart type to series1

```

```

type: 'stackingColumn',
// ...
},{
//Set chart type to series2
type: 'stackingColumn',
// ...
}];
// Enable 3D Chart
var enable3D= true;
// ...
ReactDOM.render(
<EJ.Chart id="default_chart_sample_0"
series={series}
enable3D={enable3D}
>
</EJ.Chart>,
document.getElementById('chart')
);

```



[Click](#) here to view the Stacked Column 3DChart online demo sample.

### 3D 100% Stacked Column Chart

100% Stacking Column 3DChart is rendered by specifying the series [type](#) as “**stackingColumn100**” in the chart series and enable [enable3D](#) option in the chart.

### JAVASCRIPT

```

"use strict";
// ...
var series= [{
//Set chart type to series1
type: 'stackingColumn100',
// ...
},{
//Set chart type to series2
type: 'stackingColumn100',
// ...
}];
// Enable 3D Chart
var enable3D= true;
// ...
ReactDOM.render(
<EJ.Chart id="default_chart_sample_0"
series={series}
enable3D={enable3D}
>
</EJ.Chart>,
document.getElementById('chart')
);

```



[Click](#) here to view the 100% Stacked Column 3DChart online demo sample.



### 3D Stacked Bar Chart

To create Stacking Bar 3DChart, set the series [type](#) as “**stackingBar**” in the chart series and enable [enable3D](#) option in the chart.

#### JAVASCRIPT

```
"use strict";
// ...
var series= [{
  //Set chart type to series1
  type: 'stackingBar',
  // ...
},{
  //Set chart type to series2
  type: 'stackingBar',
  // ...
}];
// Enable 3D Chart
var enable3D= true;
// ...
ReactDOM.render(
  <EJ.Chart id="default_chart_sample_0"
  series={series}
  enable3D={enable3D}
  >
  </EJ.Chart>,
  document.getElementById('chart')
);
```



[Click](#) here to view the Stacked Bar 3DChart online demo sample.

### 3D 100% Stacked Bar Chart

You can create 100% Stacking Bar 3DChart by setting the series [type](#) as “**stackingbar100**” in the chart series and enable [enable3D](#) option in chart.

#### JAVASCRIPT

```
"use strict";
// ...
var series= [{
  //Set chart type to series1
  type: 'stackingBar100',
  // ...
},{
  //Set chart type to series2
  type: 'stackingBar100',
  // ...
}];
// Enable 3D Chart
var enable3D= true;
// ...
ReactDOM.render(
  <EJ.Chart id="default_chart_sample_0"
  series={series}
  enable3D={enable3D}
```

```
>
</EJ.Chart>,
document.getElementById('chart')
);
```



[Click](#) here to view the 100% Stacking Bar 3DChart online demo sample.

### 3D Pie Chart

To render the Pie Chart in 3D view, enable the **enable3D** option in the chart and set the series [type](#) as “**pie**” in the chart series.

#### JAVASCRIPT

```
"use strict";
// ...
var series= [{
//Set chart type to series
type: 'pie',
// ...
}];
// Enable 3D Chart
var enable3D= true;
// ...
ReactDOM.render(
<EJ.Chart id="default_chart_sample_0"
series={series}
enable3D={enable3D}
>
</EJ.Chart>,
document.getElementById('chart')
);
```



[Click](#) here to view the Pie 3DChart online demo sample.

### 3D Doughnut Chart

To render the Doughnut Chart in 3D view, enable the **enable3D** option in the chart and set the series [type](#) as “**doughnut**” in the chart series.

#### JAVASCRIPT

```
"use strict";
// ...
var series= [{
//Set chart type to series
type: 'doughnut',
// ...
}];
// Enable 3D Chart
var enable3D= true;
// ...
ReactDOM.render(
<EJ.Chart id="default_chart_sample_0"
```

```

series={series}
enable3D={enable3D}
>
</EJ.Chart>,
document.getElementById('chart')
);

```



[Click](#) here to view the Doughnut 3DChart online demo sample.

### Configure 3D Chart

#### 3D View

To render the EJChart in 3D view, set the [enable3D](#) option as *true* in the chart.

#### JAVASCRIPT

```

"use strict";
// Enable 3D Chart
var enable3D= true;
// ...
ReactDOM.render(
<EJ.Chart id="default_chart_sample_0"
enable3D={enable3D}
>
</EJ.Chart>,
document.getElementById('chart')
);

```



[Click](#) here to view the 3DChart online demo sample.

#### Placing Bar / Column kind of series side-by-side

The [sideBySideSeriesPlacement](#) defines the appearance of the different sets of data on the 3D Chart. When this property is enabled, the data is displayed side by side, otherwise it is displayed along the depth of the axis.

#### JAVASCRIPT

```

"use strict";
ReactDOM.render(
<EJ.Chart id="default_chart_sample_0"
//Enable SideBySideSeriesPlacement for 3DChart
sideBySideSeriesPlacement= {true}
>
</EJ.Chart>,
document.getElementById('chart')
);

```



#### Setting Axis Wall Size

In 3DChart, Cartesian axes lines are represented as walls and it defines the width of the 3D wall. 3D Pie and Doughnut Chart does not support [wallSize](#) because they don't have axes.

**JAVASCRIPT**

```
"use strict";
ReactDOM.render(
  <EJ.Chart id="default_chart_sample_0"
    //Change 3DChart axis wall size
    wallSize= {10}
  >
  </EJ.Chart>,
  document.getElementById('chart')
);
```



*3D Depth*

By using the [depth](#) property, you can view the 3D Chart from the front view of the series to the background wall.

**JAVASCRIPT**

```
"use strict";
ReactDOM.render(
  <EJ.Chart id="default_chart_sample_0"
    //Change 3DChart depth value
    depth= {120}>
  </EJ.Chart>,
  document.getElementById('chart')
);
```



*Rotating and Tilting 3D Chart*

To spin the 3D Chart on mouse dragging, enable [enableRotation](#) option in the chart. The [tilt](#) property specifies the angle of the slope of the 3D Chart. The positive and negative values are declared to the side where the slope is present. The [rotation](#) option is used to rotate the 3D chart towards left or right side of the chart. The direction of the chart depends upon the positive and negative values of the angle.

**JAVASCRIPT**

```
"use strict";
ReactDOM.render(
  <EJ.Chart id="default_chart_sample_0"
    enable3D={true}
    //Change 3DChart depth value
    depth= {120}
    //Change 3DChart tilt value
    tilt= {10}
    //Enable 3DChart rotation on mouse dragging
    enableRotation={true}
    //Change 3DChart rotation on chart load
    rotation={40}
  >
  </EJ.Chart>,
  document.getElementById('chart')
);
```



### [PerspectiveAngle](#)

The [perspectiveAngle](#) specifies the appearance of the height, width, depth and wall of the 3D Chart. When the perspectiveAngle is decreased, the 3D object appears very close to the viewer. But when it is increased, the 3D object appears far away from the viewer.

### JAVASCRIPT

```
"use strict";
ReactDOM.render(
  <EJ.Chart id="default_chart_sample_0"
    //Change 3DChart perspective angle
    perspectiveAngle= {150}
  >
  </EJ.Chart>,
  document.getElementById('chart')
);
```



### [Localization](#)

EJChart supports localization for its axis labels and tooltip. To render the chart with specific culture you have to refer the corresponding **globalize** culture script and need to specify the culture name in [locale](#) property of chart.

### HTML

```
<head>
  <!--Refer french globalize culture script-->
  <script src="../../scripts/cultures/globalize.culture.fr-FR.min.js"></script>
</head>
<body>
  <div id="chartcontainer"></div>
</body>
```

### JAVASCRIPT

```
"use strict";
ReactDOM.render(
  <EJ.Chart id="default_chart_sample_0"
    //Render chart in french locale
    locale= {'fr-FR'}
  >
  </EJ.Chart>,
  document.getElementById('chart')
);
```



[Click](#) here to view the localization chart online demo sample.

### [Exporting Service](#)

#### [Description](#)

Export Chart as PNG, JPG, SVG, PDF document, Excel document or Word document.

*URL*

<http://js.syncfusion.com/ExportingServices/api/JSChartExport/ExportChart>

*Parameters*

Name	Value
exportMultipleChart	false

*Request***JS**

```
var chart = $(".e-datavisualization-chart").ejChart("instance");
var exportSettings = chart.model.exportSettings;
exportSettings.fileName = "Chart";
exportSettings.angle = "90";
exportSettings.type = "png";
exportSettings.mode = 'server';
exportSettings.action =
'http://js.syncfusion.com/ExportingServices/api/JSChartExport/ExportChart';
data = chart.export();
```

*Response (PNG, JPG, SVG, PDF, Word or Excel):*

Status Code: 200

Content-Type: application/octet-stream

Browser will prompt a dialog box to save the file (image or document).

*Online Demo*

Our [Exporting Demo](#) sample uses this service to export Chart as images (PNG, JPG, SVG), PDF document, Word document and Excel worksheet.

## Checkbox

### Overview

**CheckBox** component is an input component with trendy look and appearance used to input multiple options from the user.

### Key Features

- **Trendy Look** : Rich appearance with theme support
- **RTL** : Supports for right to left alignment
- **Tri-State**: Supports **CheckBox** with three states.
- **Easy Customization**: The style and appearance of CheckBox component could be easily configured.
- **Themes**: Supports 17 built-in themes (6 – flat and 6 – gradient effects), and also support custom skin options to set user-defined themes.
- **Responsive and Touch support**: Fits in all range of devices and supports touch devices

### GettingStarted

This section discloses the details on how to render and configure CheckBox component in a ReactJS application.

### Create a CheckBox

Create an HTML page and refer the necessary script and CSS dependency files in your application with the help of given [getting started documentation](#)

With ReactJS, the components can be initialized in two ways.

1. Using jsx Template
2. Without using jsx Template

### Using jsx Template

You can render EJ components by using JSX template, wherein this JSX file will be converted to its equivalent JS file.

1. Create an HTML file and add a div element and give it an ID.

### HTML

```
<body>
<div id="checkbox"></div>
</body>
```

2. Create a JSX file and initialize CheckBox component by using the below code snippet.

### HTML

```
ReactDOM.render (
  <div>
    <br />
    Category
    <br />
    <br />
    <table >
      <tr>
        <td>
          <EJ.CheckBox id="music" text="Music" name="music" value="music"
checked={true}></EJ.CheckBox>
        </td>
        <td>
          <EJ.CheckBox id="sports" text="Sports" name="sports" value="sports"
enableTriState={true} checkState="indeterminate"></EJ.CheckBox>
        </td>
        <td>
          <EJ.CheckBox id="bike" text="Bike riding" name="bike"
value="bike"></EJ.CheckBox>
        </td>
      </tr>
    </table>
    <br />
    <br />
    Favorite search engines
    <br />
    <br />
    <table>
```

```

<tr>
<td>
<EJ.CheckBox id="google" text="Google" name="google" value="google"
checked={true}></EJ.CheckBox>
</td>
<td colspan="2">
<EJ.CheckBox id="yahoo" text="yahoo" name="yahoo" value="yahoo"
enableTriState={true} checkState="indeterminate"></EJ.CheckBox>
</td>
<td colspan="2">
<EJ.CheckBox id="bing" text="Bing" name="bing" value="bing"></EJ.CheckBox>
</td>
</tr>
</table>
</div>,
document.getElementById('checkbox')
);

```

3. Refer the JSX file created in last step in the HTML file as given below.

### HTML

```

<body>
<div id="checkbox"></div>
<!-- CheckBox.jsx created in previous step-->
<script type="text/babel" src="CheckBox.jsx">
</script>
</body>

```

Now the jsx file will be compiled into its equivalent Javascript file by means of Babel.

Execute the above code to render CheckBox component.

### Hobbies

☒ Music ☐ Sports ☐ Bike Riding

### Favorite Search Engines

☒ Google ☐ Yahoo ☐ Bing

### Without using jsx Template

The CheckBox component can be initialized without using JSX template.

1. Create an HTML page and render a



element with an ID set to it.

### HTML

```
Hobbies
<table>
<tr>
<td>
<div id="check1"></div>
</td>
<td>
<div id="check2"></div>
</td>
</tr>
</table>
```

2. Render the CheckBox component by using the below mentioned code snippet.

### JAVASCRIPT

```
ReactDOM.render(
  React.createElement(EJ.CheckBox, {
    id: "check_btn1",
    name: "music",
    value: "music",
    text: "Music",
    checked: true
  }),
  document.getElementById('check1')
);
ReactDOM.render(
  React.createElement(EJ.CheckBox, {
    id: "check_btn2",
    name: "sports",
    value: "sports",
    text: "Sports"
  }),
  document.getElementById('check2')
);
```

Run the above code snippet to get the following output.

Hobbies

☒ Music ☐ Sports

## CircularGauge

### Getting Started

- This section encompasses how to configure Circular Gauge. You can provide data to a Circular Gauge and make them to display in a required way.
- The following screen shot displays a Circular Gauge, which visually represents the functions of an Automobile speedometer with RPM (Rotation per Minute), kph (Kilometer per hour) and it denotes the speed level indicators (Safe, Caution and Danger).



### Analog Speedometer

#### Adding Script Reference

Create an **HTML** page and add the scripts references in the order mentioned in the following code example.

- [jQuery](#) 1.10.2 and later versions

The required ReactJS script dependencies as follows. And you can also refer [React](#) to know more about ReactJS.

- react.min.js - <http://cdn.syncfusion.com/js/assets/external/react.min.js>
- react-dom.min.js - <http://cdn.syncfusion.com/js/assets/external/react-dom.min.js>
- ej.web.react.min.js - <http://cdn.syncfusion.com/{{ site.releaseversion }}/js/common/ej.web.react.min.js>

To get started, you can use the `ej.web.all.min.js` file that encapsulates all the `ej` controls and frameworks in one single file.

#### HTML

```
<!DOCTYPE html>
```

```

<html>
<head>
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<meta name="description" content="Essential Studio for React JS">
<meta name="author" content="Syncfusion">
<title>Getting Started for Ribbon React JS</title>
<!-- Essential Studio for JavaScript theme reference -->
<link href="http://cdn.syncfusion.com/{{ site.releaseversion }}/js/web/flat-
azure/ej.web.all.min.css" rel="stylesheet" />
<!-- Essential Studio for JavaScript script references -->
<script src="http://cdn.syncfusion.com/js/assets/external/jquery-
3.0.0.min.js"></script>
<script
src="http://cdn.syncfusion.com/js/assets/external/react.min.js"></script>
<script src="http://cdn.syncfusion.com/js/assets/external/react-
dom.min.js"></script>
<script src="http://cdn.syncfusion.com/{{ site.releaseversion
}}/js/web/ej.web.all.min.js"></script>
<script src="http://cdn.syncfusion.com/{{ site.releaseversion
}}/js/common/ej.web.react.min.js"></script>
<!-- Add your custom scripts here -->
</head>
<body>
</body>
</html>

```

**Note:** 1. In production, we highly recommend you to use our [custom script generator](#) to create custom script file with required controls and its dependencies only. Also to reduce the file size further please use [GZip compression](#) in your server.

2. For themes, you can use the `ej.web.all.min.css` CDN link from the code snippet given. To add the themes in your application, please refer to [this link](#).

### Control Initialization

Control can be initialized in two ways.

- Using jsx Template
- Without using jsx Template

### Using jsx Template

By using the jsx template, we can create the HTML file and jsx file. The `.jsx` file can be convert to `.js` file and it can be referred in HTML page.

#### Create your circularGauge

In this tutorial, you will learn how to create a simple circular gauge.

#### 1.Create a

tag.

#### HTML

```

<!DOCTYPE html>
<html>
<body>

```

```
<div id="circularGauge-default" style="height:99%;"></div>
<script src="app/circulargauge/default.js"></script>
</body>
</html>
```

2. Initialize the CircularGauge by using the `EJ.CircularGauge` tag.

#### JAVASCRIPT

```
"use strict";
ReactDOM.render(
  <div className="default">
    <EJ.CircularGauge id="circulargauge1"></EJ.CircularGauge>,
  </div>,
  document.getElementById('circulargauge-default')
);
```

Run the above code example to get a default CircularGauge with default values.



#### Set Height and Width

Pointers have different height and width so you can set the height and width of the gauge according to your requirements. Set the basic values of the gauge such as height and width of the canvas element values that are to be rendered.

#### JAVASCRIPT

```
<script type="text/babel">
<!DOCTYPE html>
<html>
<body>
<script type="text/babel">
ReactDOM.render(
  <div className="default">
    <EJ.CircularGauge id="circulargauge1" height={500}
    width={500}></EJ.CircularGauge>,
  </div>,
  document.getElementById('circularGauge-default')
);
</script>
```

```
</body>  
</html>
```

Run the above code example and you will see the following output.



### Set Background Color

The speedometer must have some dark color as background so that its value is clearly visible and you can vary the speed of the pointer by setting `ReadOnly` as `False` for user interaction.

### JAVASCRIPT

```
<script type="text/babel">  
<!DOCTYPE html>  
<html>  
<body>  
<script type="text/babel">  
  ReactDOM.render(  
    <div className="default">  
      <EJ.CircularGauge  
        id="circulargauge1"  
        height={500}  
        width={500}  
        backgroundColor="#3D3F3D"  
        readOnly={false} ></EJ.CircularGauge>,  
    </div>,  
    document.getElementById('circularGauge-default')  
  );  
</script>  
</body>  
</html>
```

Run the above code example and you will see the following output.



### Provide Scale Values

- The pointer cap can be customized with the following options. Cap radius, cap border color, cap background color, pointer cap border width are some of the properties that are customizable.
- The speed limit in the gauge has maximum value of 200 kph. So you can set maximum value for the gauge as 200.
- Major Ticks have the interval value of 20 and minor ticks have the interval value of 5. Show ranges and show indicators are used to display the ranges and indicators in their respective positions.

### JAVASCRIPT

```
<script type="text/babel">
var scales = [{
  showRanges: true,
  showIndicators: true,
  pointerCap: {
    radius: 15,
    borderWidth: 0,
    backgroundColor: "#797C79",
    borderColor: "#797C79"
  },
  maximum: 200,
  majorIntervalValue: 20,
  minorIntervalValue: 5
}];
<!DOCTYPE html>
<html>
<body>
<script type="text/babel">
ReactDOM.render(
<div className="default">
<EJ.CircularGauge
id="circulargauge1"
height={500}
width={500}
```

```
backgroundColor="#3D3F3D"
readOnly={false}
scales={scales} ></EJ.CircularGauge>,
</div>,
document.getElementById('circularGauge-default')
);
</script>
</body>
</html>
```

Run the above code example and you will see the following output.



### Add Label Customization

To display the value around the scales, labels are used. By customizing the label color it displays as specified.

#### JAVASCRIPT

```
<script type="text/babel">
var scales = [{
  //Add the labels customization code here
  labels: [{
    color: "#ffffff"
  }],
}];
<!--DOCTYPE html>
<html>
<body>
<script type="text/babel">
ReactDOM.render(
<div className="default">
<EJ.CircularGauge
id="circulargauge1"
height={500}
width={500}
backgroundColor="#3D3F3D"
```

```
readOnly={false}
scales={scales} </EJ.CircularGauge>,
</div>,
document.getElementById('circularGauge-default')
);
</script>
</body>
</html>
```

Run the above code example and you will see the following output.



### Add Pointers

Here, you have three pointers that denote the kilometer value, rotation per minute value and torque value. The torque value pointer needs not be similar to the other two pointers. You can set torque pointer as marker pointer. And you can set other attributes for pointer such as background color, border color, length, width and distance from scale.

### JAVASCRIPT

```
<script type="text/babel">
var scales = [{
  //Add the labels customization code here
  //Add the pointers customization code here
  pointers: [{
    value: 140,
    distanceFromScale: 60,
    showBackNeedle: false,
    length: 20,
    type: "marker",
    markerType: "triangle",
    width: 10,
    radius: 10,
    backgroundColor: "#FF940A",
    border: {
      color: "#FF940A"
    },
  },
```



```
    },
    {
      value: 110,
      showBackNeedle: false,
      length: 150,
      width: 2,
      radius: 10,
      needleType: "rectangle",
      backgroundColor: "#05AFFE",
      border: {
        color: "#05AFFE"
      },
    }, {
      value: 67,
      showBackNeedle: false,
      length: 100,
      width: 15,
      radius: 10,
      backgroundColor: "#FC5D07",
      border: {
        color: "#FC5D07"
      },
    },
  ]];
<!DOCTYPE html>
<html>
<body>
<script type="text/babel">
ReactDOM.render(
  <div className="default">
    <EJ.CircularGauge
      id="circulargauge1"
      height={500}
      width={500}
      backgroundColor="#3D3F3D"
      readOnly={false}
      scales={scales}
      //Add the ticks customization code here
      //Add the ranges customization code here
      //Add the indicators customization code here
      //Add the Custom labels customization code here ></EJ.CircularGauge>,
    </div>,
    document.getElementById('circularGauge-default')
  );
</script>
</body>
</html>
```

Run the above code example and you will see the following output.



### Add Tick Details

- You can set Major ticks with their width and height equal to Minor ticks.
- You can set Color according to your preference for better visibility in dark backgrounds.
- To display and customize the tick value add the following code example.

### JAVASCRIPT

```
<script type="text/babel">
var scales = [{
  //Add the labels customization code here
  //Add the pointers customization code here
  //Add the ticks customization code here
  ticks: [{
    type: "major",
    distanceFromScale: 70,
    height: 20,
    width: 3,
    color: "#ffffff"
  }, {
    type: "minor",
    height: 12,
    width: 1,
    distanceFromScale: 70,
    color: "#ffffff"
  }]
}];
<!DOCTYPE html>
<html>
<body>
<script type="text/babel">
ReactDOM.render(
  <div className="default">
    <EJ.CircularGauge
      id="circulargauge1"
      height={500}
      width={500}
```

```
backgroundColor="#3D3F3D"  
readOnly={false}  
scales={scales}  
<</EJ.CircularGauge>,  
</div>,  
document.getElementById('circularGauge-default')  
);  
</script>  
</body>  
</html>
```

Run the above code example and you will see the following output.



### Add Range Values

- Ranges denote the property of the scale value in the speedometer. The color values of the ranges denote speed variation. Set ShowRanges as true for showing the ranges in the Circular Gauge.
- For Low speed, you can mention it as safe zone; for moderate speed, you can call it as caution zone and for high speed, you can mark it as high speed.
- You can customize the range with properties such as start value, end value, start width, end width, background color , border color, etc.,

### JAVASCRIPT

```
<script type="text/babel">  
var scales = [{  
  ranges: [{  
    distanceFromScale: 30,  
    startValue: 0,  
    endValue: 70,  
    backgroundColor: "#5DF243",  
    border: {
```

```
color: "#FFFFFF"
},
}, {
distanceFromScale: 30,
startValue: 70,
endValue: 140,
backgroundColor: "#F6FF0A",
border: {
color: "#FFFFFF"
},
},
{
distanceFromScale: 30,
startValue: 140,
endValue: 200,
backgroundColor: "#FF1807",
border: {
color: "#FFFFFF"
},
}]
}]]];
<!DOCTYPE html>
<html>
<body>
<script type="text/babel">
ReactDOM.render(
<div className="default">
<EJ.CircularGauge
id="circulargauge1"
height={500}
width={500}
backgroundColor="#3D3F3D"
readOnly={false}
scales={scales}
></EJ.CircularGauge>,
</div>,
document.getElementById('circularGauge-default')
);
</script>
</body>
</html>
```

Run the above code example and you will see the following output.



### Add Indicator Details

- Indicators denote whether the pointers values are in their respective zones or not. Positioning the indicator on the respective range value gives you the required changes.
- By using Position property, you can set the location of the indicator. StateRanges defines how the indicator should behave when the pointer is in certain values.

### JAVASCRIPT

```
<script type="text/babel">
var scales = [{
  indicators: [
    {
      height: 10,
      width: 10,
      type: "circle",
      position: { x: 210, y: 300 },
      stateRanges: [{
        endValue: 70,
        startValue: 0,
        backgroundColor: "#5DF243",
        borderColor: "#5DF243",
        text: "",
        textColor: "#870505"
      }, {
        endValue: 200,
        startValue: 70,
        backgroundColor: "#145608",
        borderColor: "#145608",
        text: "",
        textColor: "#870505"
      }]
    },
    {
      height: 10,
```

```

width: 10,
type: "circle",
position: { x: 255, y: 200 },
stateRanges: [{
  endValue: 140,
  startValue: 70,
  backgroundColor: "#F6FF0A",
  borderColor: "#F6FF0A",
  text: "",
}, {
  endValue: 70,
  startValue: 0,
  backgroundColor: "#969B0C",
  borderColor: "#969B0C",
  text: "",
}, {
  endValue: 200,
  startValue: 140,
  backgroundColor: "#969B0C",
  borderColor: "#969B0C",
  text: "",
}],
}, {
  height: 10,
  width: 10,
  type: "circle",
  position: { x: 300, y: 300 },
  stateRanges: [{
    endValue: 140,
    startValue: 0,
    backgroundColor: "#890F06",
    borderColor: "#890F06",
    text: "",
  },
  {
    endValue: 200,
    startValue: 140,
    backgroundColor: "#FF1807",
    borderColor: "#FF1807",
    text: "",
  }
  ]},
  ]},
  ]};
<!DOCTYPE html>
<html>
<body>
<script type="text/babel">
ReactDOM.render(
<div className="default">
<EJ.CircularGauge
id="circulargauge1"
height={500}
width={500}
backgroundColor="#3D3F3D"
readOnly={false}
scales={scales}
></EJ.CircularGauge>,
</div>,

```

```
document.getElementById('circularGauge-default')
);
</script>
</body>
</html>
```

Run the above code example and you will see the following output.



### Add Custom Label Details

Custom labels are used to specify the texts that need to be displayed in the gauge. You can customize it through various properties. To display the three range description, custom texts are used here.

#### JAVASCRIPT

```
<script type="text/babel">
var scales = [{
  customLabels: [{
    value: "Safe",
    position: { x: 200, y: 280 },
    color: "#5DF243",
    font:
    {
      size: "12px",
      fontFamily: "Arial",
      fontStyle: "Bold"
    }
  }, {
    value: "Caution",
    position: { x: 253, y: 212 },
    color: "#F6FF0A",
    font:
    {
      size: "12px",
      fontFamily: "Arial",
      fontStyle: "Bold"
    }
  }
]
```

```

    }, {
    value: "Danger",
    position: { x: 290, y: 280 },
    color: "#FF1807",
    font:
    {
    size: "12px",
    fontFamily: "Arial",
    fontStyle: "Bold"
    }
    }
  ]
  });
<!DOCTYPE html>
<html>
<body>
<script type="text/babel">
ReactDOM.render(
<div className="default">
<EJ.CircularGauge
id="circulargauge1"
height={500}
width={500}
backgroundColor="#3D3F3D"
readOnly={false}
scales={scales}
></EJ.CircularGauge>,
</div>,
document.getElementById('circularGauge-default')
);
</script>
</body>
</html>

```

Run the above code example and you will see the following output.





### Without using jsx Template

The Circular Gauge can be created from a HTML DIV element with the HTML id attribute set to it. Refer to the following code example.

#### HTML

```
<div id="circularGauge-default"></div>
```

#### JAVASCRIPT

```
<script type="text/babel">
var scale = [{
  //Add the labels customization code here
  //Add the pointers customization code here
  pointers: [{
    value: 140,
    distanceFromScale: 60,
    showBackNeedle: false,
    length: 20,
    type: "marker",
    markerType: "triangle",
    width: 10,
    radius: 10,
    backgroundColor: "#FF940A",
    border: {
      color: "#FF940A"
    },
  },
  {
    value: 110,
    showBackNeedle: false,
    length: 150,
    width: 2,
    radius: 10,
    needleType: "rectangle",
    backgroundColor: "#05AFFF",
    border: {
      color: "#05AFFF"
    },
  },
  {
    value: 67,
    showBackNeedle: false,
    length: 100,
    width: 15,
    radius: 10,
    backgroundColor: "#FC5D07",
    border: {
      color: "#FC5D07"
    },
  },
  ],
  };
ReactDOM.render(
  React.createElement(EJ.CircularGauge, {id: "default",
    backgroundColor: "#3D3F3D",
    scales: scale,
    width: 500,
    height: 500,
```

```

}
),
document.getElementById('circularGauge-default')
);
</script>

```

Run the above code example and you will see the following output.



## Interaction and Animation

### Interaction

**Circular Gauge** control contains **Interaction** feature. You can use this interaction feature to change the pointer values manually. You can achieve this by clicking and dragging the pointer over the **Gauge** and you can see the value of pointer changes dynamically while dragging. To Enable/Disable the user interaction you can use the Boolean property called **readOnly**. The user interaction option is enabled when you set the value as false for the property **readOnly**. By default it holds the true value. That is by default it does not support interaction.

### JAVASCRIPT

```

"use strict";
ReactDOM.render(
<EJ.CircularGauge id="default" readOnly="false"
>
</EJ.CircularGauge>,
document.getElementById('gauge')
);

```

Execute the above code to render the following output.



### Animations

**Circular Gauge** contains an attractive concept called **Animation**. The animation option enables the pointer to rotate from the minimum value to the current value with animation effects. By using this

animation you can change the pointer value dynamically. You can apply the animation on pointer either by clockwise or counterclockwise based on the scale direction. You can enable / disable it using the property **enableAnimation**. Animation is enabled when you set **enableAnimation** as 'true'. By default it holds the true value. You can control the speed of the pointer during animating by using the property **animationSpeed**. It is a numerical value that holds the time in milliseconds. That is when the value is given as 1000, it is considered as 1 second.

#### JAVASCRIPT

```
"use strict";
ReactDOM.render(
  <EJ.CircularGauge id="default"
    //For enabling animation
    enableAnimation={true}
    //For setting animation speed
    animationSpeed={1000}
  >
  </EJ.CircularGauge>,
  document.getElementById('gauge')
);
```

Execute the above code to render the following output.

![[/js/CircularGauge/Interaction-and-Animationimages/Interaction-and-Animationimg2.png]

#### Gradient

You can change the interior gradient of **Circular Gauge** by using **InteriorGradient** property. The **isRadialGradient** property is used to check whether the gradient is circular or not.

#### JAVASCRIPT

```
"use strict";
ReactDOM.render(
  <EJ.CircularGauge id="default"
    isRadialGradient={true}
  >
  </EJ.CircularGauge>,
  document.getElementById('gauge')
);
```

#### Distance From Corner

You can display the circular gauge from distance apart from the corner by specifying value for **distanceFromCorner** property.

#### JAVASCRIPT

```
"use strict";
ReactDOM.render(
  <EJ.CircularGauge id="default"
    distanceFromCorner={5}
  >
  </EJ.CircularGauge>,
  document.getElementById('gauge')
);
```

### Resize

Circular gauge can be responsive while resizing by specifying `enableResize` property as true.

#### JAVASCRIPT

```
"use strict";
ReactDOM.render(
  <EJ.CircularGauge id="default"
    enableResize={true}
  >
</EJ.CircularGauge>,
  document.getElementById('gauge')
);
```

### Localization

The circular gauge can be localized based on name of culture specified in `Locale` property.

#### JAVASCRIPT

```
"use strict";
ReactDOM.render(
  <EJ.CircularGauge id="default"
    locale="en-fr"
  >
</EJ.CircularGauge>,
  document.getElementById('gauge')
);
```

### Themes

**CircularGauge Theme** is a set of pre-defined options that are applied to the control before **CircularGauge** is instantiated. Following predefined themes are available in JavaScript **CircularGauge**.

1. flat light
2. flat dark
3. gradient light
4. gradient dark
5. azure
6. azure dark
7. lime
8. lime dark
9. saffron
10. saffron dark
11. gradient azure
12. gradient azure dark
13. gradient lime
14. gradient lime dark
15. gradient saffron
16. gradient saffron dark

The theme for circular gauge can be specified using `Theme` property.

### JAVASCRIPT

```
"use strict";
ReactDOM.render(
  <EJ.CircularGauge id="default"
    theme="saffron"
  >
  </EJ.CircularGauge>,
  document.getElementById('gauge')
);
```

### Circular Gauge Values

The **minimum**, **maximum**, **radius** and **value** attributes of circular gauge are used to render the circular gauge with specified location.

### JAVASCRIPT

```
"use strict";
ReactDOM.render(
  <EJ.CircularGauge id="default"
    minimum={10}
    maximum={100}
    radius={50}
    value={20}
  >
  </EJ.CircularGauge>,
  document.getElementById('gauge')
);
```

### Scales

**Scales** are the basic functional block of the **Circular Gauge**. By customizing the scales, the appearance of the **Gauge** can be improved. The functional blocks of Circular Gauge are

- Pointers
- Labels
- CustomLabels
- Indicators
- Ticks
- Ranges
- Sub gauges.

### Adding Scale Collection

**Scale collection** is directly added to the **Gauge** object. Refer the following code example to add scale collection in **Gauge** control.

### JAVASCRIPT

```
"use strict";
var scales = [{
  radius: 150,
  ranges: ranges,
}];
var ranges = [
```

```
{
  distanceFromScale: -30,
  startValue: 0,
  endValue: 70
}];
ReactDOM.render(
  <ej.circulargauge id="circulargauge1" scales={scales}></ej.circulargauge>,
  document.getElementById('circulargauge')
);
```

Execute the above code to render the following output.



### Scale Customization

#### Colors and Border

The Scale border is modified with the object called **border**. It has two border property namely **color** and **width** which are used to customize the border color of the scale and border width of the scale. Setting the background color improves the look and feel of the **Circular Gauge**. You can customize the background color of the scale using **backgroundColor**.

#### JAVASCRIPT

```
"use strict";
var scales = [{
  showScaleBar: true,
  radius: 150,
  backgroundColor: "Red",
  border: {
    //For scale border color
    color: "Blue",
    //For scale border width
    width: 3
  },
  pointers: [{ length: 100 }]
}];
ReactDOM.render(
  <ej.circulargauge id="circulargauge1" scales={scales}></ej.circulargauge>,
  document.getElementById('circulargauge')
);
```

Execute the above code to render the following output.



#### Pointer Cap

**Pointer cap** is a circular shape element that is located at the center of the **Circular Gauge**. The pointer cap is one of the cynosure of the **Circular Gauge**. By customizing the pointer cap, Gauge style is improved. The pointer cap is modified with the object **pointerCap**. It contains radius, borderColor, borderWidth, interiorGradient and backgroundColor properties. The property **radius** is used to set the radius for the pointer cap. **interiorGradient** is used to provide the gradient effects to the pointer cap.

#### JAVASCRIPT

```

"use strict";
var scales = [{
  pointerCap: {
    // For setting pointer cap radius
    radius: 10,
    // For setting pointer cap border width
    borderWidth: 4,
    // For setting pointer cap border color
    borderColor: "Blue",
    // For setting pointer cap background color
    backgroundColor: "Red"
  }
}];
ReactDOM.render(
  <ej.circulargauge id="circulargauge1" scales={scales}></ej.circulargauge>,
  document.getElementById('circulargauge')
);

```

Execute the above code to render the following output.



### Appearance

**Circular Gauge** contains two types of scale direction such as clockwise and counter clockwise. You can set them by enumerable property called **direction**. And you can set the minimum and maximum values for the scale with the properties **minimum** and **maximum**. The two properties **minorIntervalValue** and **majorIntervalValue** are the values used to set interval value for the ticks and labels. The **radius** property is used to set the radius value for the circular scale and the **size** property is used to set the scale bar width. You can also adjust the Opacity of the scale with the property **opacity**. The value for opacity lies between 0 and 1. You can also give some shadow effects for the scale by using the property **shadowOffset**. The property **startAngle** is used to set starting position of the scale at certain angle and **sweepAngle** is used to shrink or expand the scale to certain angle.

### JAVASCRIPT

```

"use strict";
var scales = [{
  // For setting scale bar size
  size: 30,
  // For setting scale radius
  scaleRadius: 130,
  // For setting scale minimum value
  minimum: 20,
  // For setting scale maximum value
  maximum: 120,
  // For setting scale major interval value
  majorIntervalValue: 20,
  // For setting scale minor interval
  minorIntervalValue: 5,
  // For setting scale direction
  direction: ej.datavisualization.CircularGauge.Directions.CounterClockwise,
  // For setting scale background color
  backgroundColor: "red",
  // For setting scale bar opacity
  opacity: 0.5,
}];

```

```
// For setting scale bar shadow offset
shadowOffset: 20
}];
ReactDOM.render(
  <ej.circulargauge id="circulargauge1" scales={scales}></ej.circulargauge>,
  document.getElementById('circulargauge')
);
```

Execute the above code to render the following output.



### Enable/Disable properties

You can enable / disable properties in Circular Gauge using some properties in scale collection. The **showIndicators** property is used to enable/disable the indicators. **ShowLabels**, **showTicks**, **showRanges**, **showPointers** and **showScaleBar** are used to enable/ disable labels, ticks, ranges, pointers and scale bar respectively.

### Multiple Scales

You can set **Multiple scales** for a single **Circular Gauge** control by using an array of scale objects. Each scale object is independent of each other. The following code example refers to two scale objects in a **Gauge**.

### JAVASCRIPT

```
"use strict";
var scales = [
  // For setting first scale
  {
    size: 10,
    showScaleBar: true,
    scaleRadius: 150,
    minimum: 20,
    maximum: 120,
    majorIntervalValue: 20,
    minorIntervalValue: 5,
    direction: ej.datavisualization.CircularGauge.Directions.Clockwise,
    shadowOffset: 20,
    pointers: [{ value: 50, length: 120 }]
  },
  // For setting second scale
  {
    size: 10,
    showScaleBar: false,
    scaleRadius: 80,
    majorIntervalValue: 10,
    direction: ej.datavisualization.CircularGauge.Directions.CounterClockwise,
    opacity: 0.5,
    shadowOffset: 5,
    pointers: [{ value: 40, length: 50 }],
    ticks: [{ color: "red", distanceFromScale: 80 }],
    labels: [{ distanceFromScale: 40, color: "red" }]
  }
];
ReactDOM.render(
  <ej.circulargauge id="circulargauge1" scales={scales}></ej.circulargauge>,
  document.getElementById('circulargauge')
);
```



```
document.getElementById('circulargauge')
);
```

Execute the above code to render the following output.



## Pointers

**Pointer** value points out the actual value set in the **Circular Gauge**. You can customize the pointers to improve the appearance of **Gauge**.

### Adding Pointer Collection

**Pointer collection** is directly added to the scale object. To add pointer collection in a **Gauge** control refer the following code example.

#### JAVASCRIPT

```
"use strict";
var scales = [{
  pointers: [{
    value: 30
  }]
}];
ReactDOM.render(
  <ej.circulargauge id="circulargauge1" scales={scales}></ej.circulargauge>,
  document.getElementById('circulargauge')
);
```

Execute the above code to render the following output.



### Adding Pointer Value

**Pointer value** is the important element in the **Circular Gauge** that indicates the **Gauge** value. Real purpose of the **Circular Gauge** is based on the pointer value. You can set the pointer value either directly during rendering the control or it can be achieved by public method too.

#### JAVASCRIPT

```
"use strict";
var scales = [{
  showRanges: true,
  showScaleBar: true,
  radius: 150, size: 2,
  ranges: [{
    startValue: 20,
    endValue: 80,
    backgroundColor: "Green",
  }],
  pointers: [{
    value: 30
  }]
}];
ReactDOM.render(
  <ej.circulargauge id="circulargauge1" scales={scales}></ej.circulargauge>,
  document.getElementById('circulargauge')
);
```

```
);
```

Execute the above code to render the following output.



### Pointer Styles

#### Colors and Border

The Pointers border is modified with the object called **border** as in scales. It has two border property called **color** and **width** which are used to customize the border color of the pointer and border width of the pointer. You can set the background color to improve the look of the **Circular Gauge** and you can customize the background color of the scale using **backgroundColor**.

#### JAVASCRIPT

```
"use strict";
var scales = [{
  showScaleBar: true,
  width: 10, radius: 150,
  pointers: [{
    // For setting pointer border
    border: { color: "green", width: 2 },
    // For setting pointer background
    backgroundColor: "yellow",
    // For setting pointer value
    value: 45,
    // For setting pointer length
    length: 100,
    // For setting pointer width
    width: 16,
    // For setting pointer opacity
    opacity: 0.6
  }]
}];
ReactDOM.render(
  <ej.circulargauge id="circulargauge1" scales={scales}></ej.circulargauge>,
  document.getElementById('circulargauge')
);
```

Execute the above code to render the following output.



#### Appearance

Based on the value, the **pointer** point out the label value. You can set the pointer length and width using **length** and **width** property respectively. And you can also adjust the opacity of the pointer using the property **opacity** which holds the value between 0 and 1. You can add the gradient effects to the pointer using **gradient** object.

#### JAVASCRIPT

```
"use strict";
var scales = [{
  showScaleBar: true,
```

```

backgroundColor: "orange",
border: { width: 2, color: "Red" },
width: 10, radius: 150,
pointers: [{
  // For setting pointer border
  border: { color: "red", width: 2 },
  // For setting pointer background
  backgroundColor: "orange",
  // For setting pointer value
  value: 45,
  // For setting pointer length
  length: 100,
  // For setting pointer width
  width: 16,
  // For setting pointer opacity
  opacity: 0.6
}]
}];
ReactDOM.render(
  <ej.circulargauge id="circulargauge1" scales={scales}></ej.circulargauge>,
  document.getElementById('circulargauge')
);

```

Execute the above code to render the following output.



### Position the pointer

Pointer can be positioned with the help of two properties such as **distanceFromScale** and **placement**. **distanceFromScale** property defines the distance between the scale and pointer. **Placement** property is used to locate the pointer with respect to scale either inside the scale or outside the scale or along the scale. It is an enumerable data type. Both the property is applied only if pointer type is marker. For needle type marker, it renders with default position that is unchangeable.

### JAVASCRIPT

```

"use strict";
var scales = [{
  showScaleBar: true,
  backgroundColor: "orange",
  border: {
    width: 2,
    color: "red"
  },
  width: 10,
  radius: 150,
  pointers: [{
    type: "marker",
    // For setting marker type
    markerType: "triangle",
    // For setting pointer position
    placement: "near",
    // For setting pointer distance from scale
    distanceFromScale: 10,
    // For setting pointer border

```

```

border: {
  color: "red",
  width: 2
},
// For setting pointer background
backgroundColor: "orange",
// For setting pointer value
value: 40,
// For setting pointer length
length: 20,
// For setting pointer width
width: 20,
// For setting pointer opacity
opacity: 0.6
}]
}];
ReactDOM.render(
  <ej.circulargauge id="circulargauge1" scales={scales}></ej.circulargauge>,
  document.getElementById('circulargauge')
);

```

Execute the above code to render the following output.



### Types

- Circular gauge pointer has two types such as,
- Needle
- Marker
- Needle type pointers are the default pointers that cannot be positioned and that is located at the center of the gauge. There are four different shapes of needle pointers such as
  - Rectangle
  - Triangle
  - Trapezoid
  - Arrow
  - Image
- For marker pointer, the available dimensions are
  - Rectangle
  - Triangle
  - Ellipse
  - Diamond
  - Pentagon
  - Circle
  - Slider
  - Pointer
  - Wedge
  - Trapezoid
  - Rounded Rectangle
  - Image

### Pointer Images

In JavaScript circular gauge, it is possible to replace the pointer with images. We can use image instead of rendering the pointer.

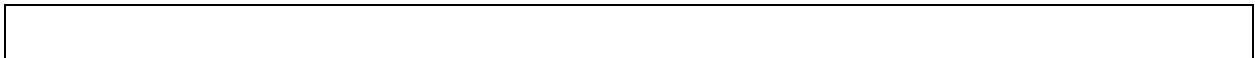
#### Image with URL

- To implement the pointer image we need to give the API called `ImageUrl`. It is a string data type.
- Image type pointer is applicable for both marker as well as needle type pointers and it is possible with combine the normal marker pointer type with image type. The three possibilities are
- Needle Image
- Marker Image
- Marker pointer with Image

#### Needle Image

In this type, needle pointer is completely replaced by image. We can implement it with the help of following snippet.

##### HTML



Execute the above code to render the following output.



#### Marker Image

In this type, marker pointer is completely replaced by image. We can implement it with the help of following snippet.

##### HTML

```
<div id="circulargauge"></div>
```

##### JAVASCRIPT

```
"use strict";
// To set scale options
var scales= [{
// set basic appearance
showRanges: true,
showLabels: false,
startAngle: 180,
sweepAngle: 180,
radius: 130,
showScaleBar: false,
// To set pointer option
pointers: [{
// To set pointer type as marker
type: "marker",
// To set needle type as image
markerType: "image",
```

```

// To set image url for pointer image
ImageUrl: "ball.png",
// To set pointer value
value: 60,
// To set pointer dimension
length: 30,
width: 100,
}],
// To set tick options
ticks: [{
height: 0,
width: 0
}],
// To set range options
ranges: [{
distanceFromScale: -30,
startValue: 0,
endValue: 70,
size: 40,
}, {
distanceFromScale: -30,
startValue: 70,
endValue: 110,
backgroundColor: "#fc0606",
border: {
color: "#fc0606"
},
size: 40,
}]
}];
// To set frame type as half circle
var frame= {
frameType: "halfcircle"
};
ReactDOM.render(
<EJ.CircularGauge id="default" scales={scales} frame={frame}
>
</EJ.CircularGauge>,
document.getElementById('circulargauge')
);

```

Execute the above code to render the following output.



### Marker Pointer with Image

In this type, marker pointer is drawn first and then image also loaded. We can implement it with the help of following snippet.

#### HTML

```
<div id="circulargauge"></div>
```

#### JAVASCRIPT

```
"use strict";
```

```

// To set scale options
var scales= [{
// set basic apperance
showRanges: true, showLabels: false,
startAngle: 180, sweepAngle: 180, radius: 130,
showScaleBar: false,
// To set pointer option
pointers: [{
// To set pointer type as marker
type: "marker",
// To set needle type as rectangle
markerType: "rectangle",
// To set image url for pointer image
ImageUrl: "ball.png",
// To set pointer value
value: 50,
// To set pointer dimension
length: 30,
width: 100,
border: { color: "Black", width: 3 }
}],
// To set tick options
ticks: [{ height: 0, width: 0 }],
// To set range options
ranges: [{
distanceFromScale: -30,
startValue: 0,
endValue: 70, size: 40,
}, {
distanceFromScale: -30,
startValue: 70,
endValue: 110,
backgroundColor: "#fc0606",
border: { color: "#fc0606" }, size: 40,
}]
}];
// To set frame type as half circle
var frame= {
frameType: "halfcircle"
};
ReactDOM.render(
<EJ.CircularGauge id="default" scales={scales} frame={frame}
>
</EJ.CircularGauge>,
document.getElementById('circulargauge')
);

```

Execute the above code to render the following output.



### Multiple Pointers

**Circular Gauge** can have multiple pointers on it. You can use any combination and any number of pointers in a **Gauge**. That is, a Gauge can contain any number of marker pointer and any number of needle pointers. Refer the following code example containing two pointers.

**JAVASCRIPT**

```

"use strict";
var scales = [{
  showScaleBar: true,
  backgroundColor: "#DCEBF9",
  border: { width: 2, color: "Green" },
  width: 10, radius: 150,
  pointers: [
    // For setting pointer1
    {
      border: { color: "Green", width: 2 },
      backgroundColor: "#DCEBF9",
      value: 40,
      length: 100,
      width: 16,
      opacity: 0.6
    },
    // For setting pointer2
    {
      distancFromScale: 20,
      placement: "near",
      type: "marker",
      markerType: "triangle",
      length: 20,
      width: 20,
      value: 60,
      backgroundColor: "#DCEBF9",
      border: { color: "Green", width: 2 },
    }
  ]
}];
ReactDOM.render(
  <ej.circulargauge id="circulargauge1" scales={scales}></ej.circulargauge>,
  document.getElementById('circulargauge')
);

```

Execute the above code to render the following output.



**Pointer Value Text**

Gauge **Pointer value text** is used to display the current value of the pointer in the **Circular Gauge** control.

**Positioning the text**

You can position the **Circular Gauge** pointer value with the gauge as center by using the **API** called **distance**. You can Disable/ Enable these pointers value by using the API **showValue**.

**JAVASCRIPT**

```

"use strict";
var scales = [{
  showRanges: true,
  // Setting tick properties
  ticks: [{ height: 0, width: 0 }],
  // Setting range properties

```



```

ranges: [
  { size: 40, startValue: 0, endValue: 50, backgroundColor: "#1B4279", border:
  { color: "#1B4279" } },
  { size: 40, startValue: 50, endValue: 100, backgroundColor: "#91B8F3",
  border: { color: "#91B8F3" } }
],
// Setting pointer properties
pointers: [{
  // Setting pointer value text properties
  pointerValueText: {
    // enable showValue property
    showValue: true,
    // setting distance property
    distance: 0,
    color: "#8c8c8c"
  }
}],
});
ReactDOM.render(
  <EJ.CircularGauge id="default" radius={100} value={55}
  backgroundColor="transparent" scales={scales}
  >
  </EJ.CircularGauge>,
  document.getElementById('circulargauge')
);

```

Run the above code to render the output as follows.



### Appearance

Appearance of the **Circular Gaugepointer value text** is adjusted by using four properties. Such as **color**, **angle**, **autoAngle** and **opacity**.

- **Color** property is used to set the color of the pointer value text.
- **Angle** property is used to set the angle in which the text is displayed.
- **Auto Angle** is used to display the text in certain angle based on pointer position angle.
- **Opacity** is used to customize the brightness of the text.

### JAVASCRIPT

```

"use strict";
var scales = [{
  showRanges: true,
  // Setting tick properties
  ticks: [{ height: 0, width: 0 }],
  // Setting range properties
  ranges: [
    { size: 40, startValue: 0, endValue: 50, backgroundColor: "#1B4279", border:
    { color: "#1B4279" } },
    { size: 40, startValue: 50, endValue: 100, backgroundColor: "#91B8F3",
    border: { color: "#91B8F3" } }
  ],
  // Setting pointer properties
  pointers: [{

```

```
// Setting pointer value text properties
pointerValueText: {
  showValue: true,
  distance: 0,
  // Setting color property
  color: "Red",
  // Setting opacity property
  opacity: 0.7,
  // Setting angle property
  angle: 20
}
}],
});
ReactDOM.render(
  <ej.circulargauge id="circulargauge1" radius={100}, value={55},
  backgroundColor="transparent" , scales={scales} ></ej.circulargauge>,
  document.getElementById('circulargauge')
);
```

Run the above code to render the output as follows.



### Font Options

Similar to other collection, font option is also available in this pointer value text such as size, fontFamily and fontStyle.

### JAVASCRIPT

```
"use strict";
var scales = [{
  showRanges: true,
  // Setting tick properties
  ticks: [{ height: 0, width: 0 }],
  // Setting range properties
  ranges: [
    { size: 40, startValue: 0, endValue: 50, backgroundColor: "#1B4279", border:
    { color: "#1B4279" } },
    { size: 40, startValue: 50, endValue: 100, backgroundColor: "#91B8F3",
    border: { color: "#91B8F3" } }
  ],
  // Setting pointer properties
  pointers: [{
    // Setting pointer value text properties
    pointerValueText: {
      showValue: true,
      distance: 0,
      color: "Red",
      opacity: 0.7,
      angle: 20,
      //setting font option
      font: {
        size: "15px",
        fontStyle: "Normal",
        fontFamily: "Arial",
      }
    }
  }
  ]
}];
```

```

    }
  }],
  }];
ReactDOM.render(
  <ej.circulargauge id="circulargauge1" radius={100}, value={55},
  backgroundColor="transparent" , scales={scales}></ej.circulargauge>,
  document.getElementById('circulargauge')
);

```

Run the above code to render the output as follows.



## Labels

**Labels** are units that are used to display the values in the scales. You can customize Labels with the properties like angle, color, font, opacity, etc.

### Adding Label Collection

**Label collection** is directly added to the scale object. Refer the following code example to add label collection in a **Gauge**.

#### JAVASCRIPT

```

"use strict";
var scales = [{
  labels: [{
    angle: 30
  }]
}];
ReactDOM.render(
  <ej.circulargauge id="circulargauge1" scales={scales}></ej.circulargauge>,
  document.getElementById('circulargauge')
);

```

Execute the above code to render the following output.



## Label Customization

### Appearance

The **attribute** angle is used to display the labels in the specified angles and **color** attribute is used to display the labels in specified color. You can adjust the opacity of the label with the property **opacity** and the values of it lies between 0 and 1. You can adjust the labels based on the tick's direction by setting **autoAngle** as true. **includeFirstValue** is an special property especially used in some special scenarios such as in clock, where the value 0 needs to be replaced with that of 12. By enabling this property the first value of the label is not rendered.

Font option is also available on the labels. The basic three properties of fonts such as size, family and style can be achieved by **size**, **fontStyle** and **fontFamily**. Labels are two types such as major and minor. Major types labels are for major interval values and minor types labels are for minor interval values.

#### JAVASCRIPT

```

"use strict";
var scales = [{
  showScaleBar: true,
  backgroundColor: "#FAF4B5",
  border: { width: 2, color: "Yellow" },
  width: 10, radius: 150,
  pointers: [{
    border: { color: "Yellow", width: 2 },
    backgroundColor: "#FAF4B5",
    value: 40, length: 100,
    width: 16,
    opacity: 0.6
  }],
  labels: [{
    // For setting label angle
    angle: 10,
    // For setting label opacity
    opacity: 0.8,
    // For disable the include first value property
    includeFistValue: false,
    // For setting label color
    color: "Yellow",
    // For setting label font
    font: {
      size: "15px",
      fontFamily: "Arial",
      fontStyle: "bold"
    }
  }]
}];
ReactDOM.render(
  <ej.circulargauge id="circulargauge1" scales={scales}></ej.circulargauge>,
  document.getElementById('circulargauge')
);

```

Execute the above code to render the following output.



### Unit text and Position

**unitText** is used to add some text along with the labels. For example, in speedometer, you need to mention the units in kph. You can also add the unit text in front of the labels. You can achieve this by using an enumerable property **unitTextPosition**. With this you can position the unit text in front or back.

Labels can be positioned with the help of two properties such as **distanceFromScale** and **placement**. **distanceFromScale** property defines the distance between the scale and labels. Placement property is used to locate the labels with respect to scale either inside the scale or outside the scale or along the scale. It is an enumerable data type.

### JAVASCRIPT

```

"use strict";
var scales = [{
  showRanges: true,
  showScaleBar: true,
  radius: 150, size: 2,

```

```

pointers: [{
  value: 40,
  showBackNeedle: true,
  length: 100
}],
labels: [{
  // For setting unit text
  unitText: "kmpH",
  // For setting unit text position
  unitTextPosition: "back"
}],
ranges: [{ startValue: 0, endValue: 50, backgroundColor: "Green", placement:
"far", distanceFromScale: -30 },
{ startValue: 50, endValue: 80, backgroundColor: "yellow", placement: "far",
distanceFromScale: -30 },
{ startValue: 80, endValue: 100, backgroundColor: "red", placement: "far",
distanceFromScale: -30 }]]
});
ReactDOM.render(
<ej.circulargauge id="circulargauge1" scales={scales}></ej.circulargauge>,
document.getElementById('circulargauge')
);

```

Execute the above code to render the following output.

![[/js/CircularGauge/Labelsimages/Labelsimg3.png)

### Multiple Labels

You can achieve multiple labels such as minor and major in a **Gauge** sample scale. Refer the following code example for multiple labels variation.

### JAVASCRIPT

```

"use strict";
var scales = [{
  showRanges: true, minorIntervalValue: 5,
  backgroundColor: "yellow",
  border: { width: 1.5, color: "Red" },
  showScaleBar: true, radius: 150, size: 5,
  pointerCap: {
    backgroundColor: "yellow",
    borderColor: "Red", borderWidth: 1.5
  },
  labels: [
    // For setting label1
    { type: "minor", color: "yellow" },
    // For setting label2
    { type: "major", color: "Red" }],
  pointers: [{
    backgroundColor: "yellow",
    border: { width: 1.5, color: "Red" },
    length: 110
  }]
}];
ReactDOM.render(
<ej.circulargauge id="circulargauge1" scales={scales}></ej.circulargauge>,

```

```
document.getElementById('circulargauge')
);
```

Execute the above code to render the following output.



### Ticks

Ticks are used to mark some values on the scale. Based on the tick's value you can set the labels on the required position.

### Adding Tick Collection

Tick collection is directly added to the scale object. Refer the following code example to add tick collection in a **Gauge** control.

#### JAVASCRIPT

```
"use strict";
var scales = [{
  ticks: [{
    value: 30
  }]
}];
ReactDOM.render(
  <ej.circulargauge id="circulargauge1" scales={scales}></ej.circulargauge>,
  document.getElementById('CircularGauge1')
);
```

Execute the above code to render the following output.



### Tick Customization

Height and width of the ticks can be applied by using the properties **height** and **width**. You can customize ticks with the properties such as angle, color, etc. **angle** attribute is used to display the labels in the specified angles and **color** attribute is used to display the labels in specified color. Ticks are two types such as major and minor.

Major type ticks are for major interval values and minor type ticks are for minor interval values. You can position ticks with the help of two properties such as **distanceFromScale** and **placement**.

**distanceFromScale** property defines the distance between the scale and ticks. **Placement** property is used to locate the ticks with respect to scale either inside the scale or outside the scale or along the scale. It is an enumerable data type.

#### JAVASCRIPT

```
"use strict";
var scales = [{
  ticks: [
    // For setting tick1
    { type: "major", color: "Red" },
    // For setting tick2
    {
      // For setting tick type
      type: "minor",
```

```
// For setting tick color
color: "yellow",
// For setting tick height
height: 8,
// For setting tick placement
placement: "near",
// For setting tick distance from scale
distanceFromScale: 5
}]
}];
ReactDOM.render(
<ej.circulargauge id="circulargauge1" scales={scales}></ej.circulargauge>,
document.getElementById('CircularGauge1')
);
```

Execute the above code to render the following output.



## Indicators

Indicators simply indicates the current status of the pointer. Indicators are in several formats such as in shape format, textual format and image format.

### Adding Indicator Collection

Indicators collection is directly added to the scale object. Refer the following code to add indicator collection in a **Gauge** control.

### JAVASCRIPT

```
"use strict";
var scales = [{
  showIndicators: true,
  indicators: [{
    // For setting indicator height
    height: 10,
    // For setting indicator width
    width: 10,
    // For setting indicator type
    type: "circle",
    // For setting indicator value
    value: 0,
    // For setting indicator position
    position: { x: 185, y: 300 },
  }]
}];
ReactDOM.render(
<ej.circulargauge id="circulargauge1" scales={scales}></ej.circulargauge>,
document.getElementById('circulargauge')
);
```

Execute the above code to render the following output.



### Basic Customization

You can enable indicators by setting **showIndicators** to 'true'. The **height** and **width** property for the indicators are used to specify the area allocated to the indicator for the width and height respectively. You can use the position collection to position the indicators along **x** and **y** axis.

Indicators are of several types such as, circle, rectangle, rounded rectangle, text and image. By using the **type** property you can avail those shapes. For image type **imageUrl** property is used.

### JAVASCRIPT

```
"use strict";
var scales = [{
  showIndicators: true, minorIntervalValue: 5,
  backgroundColor: "#5DF243",
  border: { width: 1.5, color: "black" },
  showScaleBar: true, radius: 150, size: 5,
  pointers: [{
    backgroundColor: "#5DF243",
    border: { width: 1.5, color: "black" },
    length: 110
  }],
  indicators: [{
    // For setting indicator height
    height: 10,
    // For setting indicator width
    width: 10,
    // For setting indicator type
    type: "circle",
    // For setting indicator value
    value: 0,
    // For setting indicator position
    position: { x: 185, y: 300 },
  }],
}];
ReactDOM.render(
  <ej.circulargauge id="circulargauge1" scales={scales}></ej.circulargauge>,
  document.getElementById('circulargauge')
);
```

Execute the above code to render the following output.



### State Ranges

State ranges are used to specify the indicator behavior in the specified region. Use **startValue** and **endValue** to set the range bound for the pointer. Whenever the pointer cross the specified region, the indicator attributes are applied for ranges.

The **backgroundColor** and **borderColor** sets the appearance behavior for the indicators. For text type indicators you can give value for text. And **text** can be changed whenever the pointer crosses its state range area. There are many basic font options available for the text in the state range such as size, **fontStyle** and **fontFamily**.

### JAVASCRIPT

```
"use strict";
```



```

var scales = [{
  showIndicators: true, minorIntervalValue: 5,
  backgroundColor: "#5DF243",
  border: { width: 1.5, color: "black" },
  showScaleBar: true, radius: 150, size: 5,
  pointers: [{
    backgroundColor: "#5DF243",
    border: { width: 1.5, color: "black" },
    length: 110
  }],
  indicators: [{
    // For setting indicator height
    height: 10,
    // For setting indicator width
    width: 10,
    // For setting indicator type
    type: "circle",
    // For setting indicator value
    value: 0,
    // For setting indicator position
    position: { x: 185, y: 300 },
    // For setting indicator state range collection
    stateRanges: [{
      // For setting state range end value height
      endValue: 100,
      // For setting state range start value
      startValue: 0,
      // For setting indicator background color
      backgroundColor: "#5DF243",
      // For setting indicator border color
      borderColor: "Black",
      // For setting indicator text
      text: "",
      // For setting indicator text color
      textColor: "#870505"
    }]
  }],
  }];
ReactDOM.render(
  <ej.circulargauge id="circulargauge1" scales={scales}></ej.circulargauge>,
  document.getElementById('circulargauge')
);

```

Execute the above code to render the following output.



### Multiple Indicators

You can use multiple indicators for a single **Gauge**. Each indicator have a list of state ranges. Refer the following code example for multiple Indicators.

#### JAVASCRIPT

```

"use strict";
var scales = [{
  readOnly: false,

```

```
showIndicators: true, showRanges: true,
minorIntervalValue: 5,
showScaleBar: true, radius: 150, size: 5,
pointers: [{
  length: 110, value: 70
}],
ranges: [{
  startValue: 0, endValue: 50,
  backgroundColor: "Green",
  placement: "far", distanceFromScale: -30
},
{
  startValue: 50, endValue: 100,
  backgroundColor: "red",
  placement: "far", distanceFromScale: -30
}],
indicators: [
  //Indicator1
  {
    height: 10,
    width: 10,
    type: "circle",
    value: 0,
    position: { x: 165, y: 300 },
    stateRanges: [{
      endValue: 50,
      startValue: 0,
      backgroundColor: "#24F92F",
      borderColor: "Black"
    }, {
      endValue: 50,
      startValue: 100,
      backgroundColor: "#322C04",
      borderColor: "Black"
    }]
  },
  //Indicator2
  {
    height: 10,
    width: 10,
    type: "circle",
    value: 0,
    position: { x: 215, y: 300 },
    stateRanges: [{
      endValue: 50,
      startValue: 0,
      backgroundColor: "#600000",
      borderColor: "Black"
    }, {
      endValue: 100,
      startValue: 50,
      backgroundColor: "#FF4F2A",
      borderColor: "Black"
    }]
  },
];
ReactDOM.render(
```

```
<ej.circulargauge id="circulargauge1" scales={scales}></ej.circulargauge>,
document.getElementById('circulargauge')
);
```

Execute the above code to render the following output.

![[/js/CircularGauge/Indicatorsimages/Indicatorsimg4.png)

## Ranges and Frames

Ranges are used to specify or group the scale values. By using ranges, you can describe the values in the pointers.

### Adding Range Collection

Range collection is directly added to the scale object. Refer the following code example to add range collection in a **Gauge** control.

#### JAVASCRIPT

```
"use strict";
var scales = [{
  showRanges: true,
  ranges: [{
    startValue: 20,
    endValue: 80
  }]
}];
ReactDOM.render(
<ej.circulargauge id="circulargauge" scales={scales}></ej.circulargauge>,
document.getElementById('circulargauge')
);
```

## Range Customization

### Appearance

The API **size** is used to specify the width of the ranges. The major attributes for ranges are **startValue** and **endValue**. **startValue** defines the start position of the ranges and **endValue** defines the end position of the ranges.

**StartWidth** and **endWidth** are used to specify the range width at the starting and ending position of the ranges. You can add the gradient effects to the ranges by using **gradient** object.

#### JAVASCRIPT

```
"use strict";
var scales = [{
  showRanges: true,
  showScaleBar: true,
  radius: 150, size: 2,
  ranges: [{
    //For setting range start value
    startValue: 20,
    //For setting range end value
    endValue: 80,
    //For setting range background color
    backgroundColor: "Green",
  }]
}];
```

```

    });
    ReactDOM.render(
      <ej.circulargauge id="circulargauge" scales={scales}></ej.circulargauge>,
      document.getElementById('circulargauge')
    );

```

Execute the above code to render the following output.

![[/js/CircularGauge/Ranges-and-Framesimages/Ranges-and-Framesimg1.png]

### Colors and Border

By customizing the ranges, the appearance of the **Gauge** can be improved. The range border is modified with the object called **border**. It has two border property such as **color** and **width**. These are used to customize the border color of the ranges and border width of the ranges.

You can set the background color to improve the look and feel of the **Circular Gauge**. For customizing the background color of the ranges, **backgroundColor** is used.

### JAVASCRIPT

```

"use strict";
var scales = [{
  showRanges: true,
  showScaleBar: true,
  radius: 150, size: 2,
  ranges: [{
    //For setting range start value
    startValue: 20,
    //For setting range end value
    endValue: 80,
    //For setting range background color
    backgroundColor: "yellow",
    //For setting range border
    border: { color: "green", width: 2 },
  }]
}];
ReactDOM.render(
  <ej.circulargauge id="circulargauge" scales={scales}></ej.circulargauge>,
  document.getElementById('circulargauge')
);

```

Execute the above code to render the following output.

![[/js/CircularGauge/Ranges-and-Framesimages/Ranges-and-Framesimg2.png]

### Positioning the ranges

You can position ranges using two properties such as **distanceFromScale** and **placement**.

**distanceFromScale** property defines the distance between the scale and range. **Placement** property is used to locate the pointer with respect to scale either inside the scale or outside the scale or along the scale. It is an enumerable data type.

### JAVASCRIPT

```

"use strict";
var scales = [{
  showRanges: true,

```

```

showScaleBar: true,
radius: 150, size: 2,
ranges: [{
  //For setting range start value
  startValue: 0,
  //For setting range end value
  endValue: 100,
  //For setting range background color
  backgroundColor: "Green",
  //For setting range placement
  placement: "far",
  //For setting distance between scale and ranges
  distanceFromScale: -30,
  //For setting range border
  border: { color: "Black", width: 2 },
}]
});
ReactDOM.render(
  <ej.circulargauge id="circulargauge" scales={scales}></ej.circulargauge>,
  document.getElementById('circulargauge')
);

```

Execute the above code to render the following output.

![[/js/CircularGauge/Ranges-and-Framesimages/Ranges-and-Framesimg3.png]

### Multiple Ranges

You can set multiple ranges by adding an array of ranges objects. Refer the following code example for multiple ranges functionality.

#### HTML

```
<div id="circulargauge"></div>
```

#### JAVASCRIPT

```

"use strict";
var scales = [{
  showRanges: true,
  showScaleBar: true,
  radius: 150, size: 2,
  pointers: [{
    value: 40,
    showBackNeedle: true,
    length: 100
  }],
  ranges: [
    //For setting range1
    {
      startValue: 0, endValue: 50,
      backgroundColor: "Green",
      placement: "far", distanceFromScale: -30
    },
    //For setting range2
    {
      startValue: 50, endValue: 80,

```

```

backgroundColor: "yellow",
placement: "far", distanceFromScale: -30
},
//For setting range3
{
startValue: 80, endValue: 100,
backgroundColor: "red",
placement: "far", distanceFromScale: -30
}]
}];
ReactDOM.render(
<ej.circulargauge id="circulargauge" scales={scales}></ej.circulargauge>,
document.getElementById('circulargauge')
);

```

Execute the above code to render the following output.



### Frames

Frame is the element that decides the appearance of the **Circular Gauge**. You can customize it using the object called **frame**. It has the properties such as **frameType**, **backGroundUrl**, **halfCircleFrameStartAngle** and **halfCircleFrameEndAngle**.

**frameType** is used to specify whether frame is a half circle frame or full circle frame.

**halfCircleFrameStartAngle** and **halfCircleFrameEndAngle** are used to specify the angle for **Gauge** with frame type as half circle. **backgroundUrl** is used to set the background image for the frame.

### JAVASCRIPT

```

"use strict";
var scales = [{
startAngle: 180, sweepAngle: 180,
pointers: [{
needleType: "rectangle",
width: 1, length: 120, value: 40
}],
pointerCap: { radius: 50 },
}];
var frame = {
frameType: "halfcircle",
//For setting half circle frame start angle
halfCircleFrameStartAngle: 205,
//For setting half circle frame end angle
halfCircleFrameEndAngle: 335,
};
ReactDOM.render(
<ej.circulargauge id="circulargauge" frame={frame} backgroundColor="#FFCCCC"
scales={scales}></ej.circulargauge>,
document.getElementById('circulargauge')
);

```

Execute the above code to render the following output.



## Legend

The legend contains the list of the ranges that appear in the circular gauge

### Legend Visibility

By default, the legend will not be displayed in the circular gauge. You can enable or disable it by using the [visible](#) property of the legend.

#### JAVASCRIPT

```
var legend = {
  visible: true
}
ReactDOM.render(
  <EJ.CircularGauge id="default" legend={legend}
>
</EJ.CircularGauge>,
document.getElementById('gauge')
);
```



[Click](#) here to view the online demo sample for legend in the circular Gauge.

### Legend Text

The text displayed in the legend can be customized by using the **legendText** property present in the ranges of the circular gauge. When the legendText is not specified in the ranges, then the legend item for that particular range will not displayed. By default the legendText value is **null**.

#### JAVASCRIPT

```
// ...
var ranges= [{
  legendText:"Light air",
}]
//...
// ...
ReactDOM.render(
  <EJ.CircularGauge id="default" ranges={ranges}
>
</EJ.CircularGauge>,
document.getElementById('gauge')
);
```

### Legend Fill and Opacity

You can change the opacity and fill color of legend text using **Opacity** and **Fill** property of legend.

#### JAVASCRIPT

```
var legend = {
  visible: true,
  fill:'blue',
  opacity:0.5
}
ReactDOM.render(
  <EJ.CircularGauge id="default" legend={legend}
>
```

```

</EJ.CircularGauge>,
document.getElementById('gauge')
);

```

### Position and Align the Legend

By using the [position](#) property, you can position the legend at *left*, *right*, *top* or *bottom* of the CircularGauge. The legend is positioned at the **bottom** of the circular gauge, by default.

#### JAVASCRIPT

```

// ...
var legend= {
// ...
//Place the legend at top of the circular gauge
position: 'top',
}
// ...
ReactDOM.render(
<EJ.CircularGauge id="default" legend={legend}
>
</EJ.CircularGauge>,
document.getElementById('gauge')
);

```



### Legend Alignment

You can align the legend to the *center*, *far* or *near* based on its position by using the [alignment](#) property.

#### JAVASCRIPT

```

// ...
var legend= {
//...
//The below two settings will place the legend at the top-right corner of
the circular gauge.
alignment: 'far',
position: 'top',
}
// ...
ReactDOM.render(
<EJ.CircularGauge id="default" legend={legend}
>
</EJ.CircularGauge>,
document.getElementById('gauge')
);

```



### Customization

#### Legend shape

To change the legend item shape, you have to specify the desired shape in the [shape](#) property of the legend. By default, the shape of the legend is **circle**. It also supports rectangle, diamond, triangle, slider, line, pentagon, trapezoid and wedge shapes.



**JAVASCRIPT**

```
// ...
var legend= {
//...
//Change legend shape
shape: 'slider',
}
// ...
ReactDOM.render(
<EJ.CircularGauge id="default" legend={legend}
>
</EJ.CircularGauge>,
document.getElementById('gauge')
);
```



*Legend Item Size and Border*

You can change the size of the legend items by using the [width](#) and [height](#) properties in the **itemStyle**. To change the legend item border, use [border](#) property of the legend itemStyle.

**JAVASCRIPT**

```
// ...
var legend= {
//...
//Change legend items border, height and width
itemStyle: {width: 13, height: 13, border: { color: "#FF0000", width: 2 } },
}
// ...
ReactDOM.render(
<EJ.CircularGauge id="default" legend={legend}
>
</EJ.CircularGauge>,
document.getElementById('gauge')
);
```



*Legend size*

You can change the default legend size by using the **size** property of the legend.

**JAVASCRIPT**

```
// ...
var legend= {
//...
//Change legend size
size:{width: '350', height: '100'}
}
// ...
ReactDOM.render(
<EJ.CircularGauge id="default" legend={legend}
>
</EJ.CircularGauge>,
);
```

```
document.getElementById('gauge')
);
```



#### Legend Item Padding

You can control the spacing between the legend items by using the [itemPadding](#) option of the legend. The default value is 20.

#### JAVASCRIPT

```
"use strict";
// ...
var legend= {
//...
//Add space between each legend item
itemPadding: 30,
}
// ...
ReactDOM.render(
<EJ.CircularGauge id="default" legend={legend}
>
</EJ.CircularGauge>,
document.getElementById('gauge')
);
```



#### Legend border

You can customize the legend border by using the [border](#) option in the legend.

#### JAVASCRIPT

```
"use strict";
// ...
var legend= {
//...
//Set border color and width to legend
border: {color: "#FFC342", width: 2},
}
// ...
ReactDOM.render(
<EJ.CircularGauge id="default" legend={legend}
>
</EJ.CircularGauge>,
document.getElementById('gauge')
);
```



#### Font of the legend text

The font of the legend item text can be customized by using the [font](#) property in legend.

#### JAVASCRIPT

```
// ...
```

```

var legend= {
  //...
  //Customize the legend item text
  font: { fontFamily: 'Segoe UI', fontStyle: 'Normal', fontWeight: 'Bold',
    size: '15px' },
}
// ...
ReactDOM.render(
  <EJ.CircularGauge id="default" legend={legend}
  >
  </EJ.CircularGauge>,
  document.getElementById('gauge')
);

```



## Events

### Legend Item Render

[LegendItemRender](#) event triggers before rendering the legend items. This event is triggered for each legend item in Circular gauge. You can use this event to customize legend item shape or add custom text to legend item dynamically

### JAVASCRIPT

```

function onLegendRender(sender) {
  //Get legend item details before rendering
  var legendItem = sender.data;
}
ReactDOM.render(
  <EJ.CircularGauge id="default"
  //Subscribe the legendItemRender event
  legendItemRender={onLegendRender}
  >
  </EJ.CircularGauge>,
  document.getElementById('gauge')
);

```

### Legend Item Click

You can get the legend item details such as *RangeIndex*, *bounds* and *shape* by subscribing the [legendItemClick](#) event of the circular gauge. When the legend item is clicked, it triggers the event and returns the legend information

### JAVASCRIPT

```

$("#CircularGauge1").ejCircularGauge({
  function onLegendClicked(sender) {
    //Get legend item details on legend item click.
    var legendItem = sender.data;
  }
  ReactDOM.render(
    <EJ.CircularGauge id="default"
    //Subscribe the legend item click event
    legendItemClick={onLegendClicked}
    >
    </EJ.CircularGauge>,

```

```
document.getElementById('gauge')
);
```

## Custom labels

Custom labels are the texts that you can use them in any location of the **Gauge**.

### Adding Custom Label Collection

Custom labels collection is directly added to the scale object. Refer the following code to add custom labels collection in a **Gauge** control.

#### JAVASCRIPT

```
"use strict";
Adding Custom Label Collection
var scales = [{
  showCustomLabels: true,
  customLabels: [{
    color: "Red"
  }]
}];
ReactDOM.render(
  <ej.circulargauge id="circulargauge1" scales={scales}></ej.circulargauge>,
  document.getElementById('circulargauge')
);
```

## Basic Customization

You can customize custom labels using the properties such as **textAngle**, **color** and **font**. **textAngle** attribute is used to display the custom labels in the specified angles and **color** attribute is used to display the custom labels in specified color.

You can use **Value** attribute to set the text value in the custom labels. To display the custom labels, set **showCustomLabels** as 'true'. To set the location of the custom label in **Circular Gauge**, **location** property is used. By using **x** and **y** axis you can adjust the position of the custom labels.

Font option is also available on custom labels. The basic three properties of fonts such as size, family and style can be achieved by **size**, **fontStyle** and **fontFamily** attributes.

#### JAVASCRIPT

```
"use strict";
var scales = [{
  size: 10,
  shadowOffset: 10,
  showCustomLabels: true,
  showRanges: true,
  showScaleBar: true,
  radius: 150, size: 2,
  customLabels: [{
    //For setting custom label text angle
    textAngle: 10,
    //For setting custom label color
    color: "Red",
    //For setting custom label value
    value: "CustomLabel1",
```

```
//For setting custom label font option
font: {
  size: "18px",
  fontFamily: "Arial",
  fontStyle: "bold"
},
position: { x: 180, y: 100 }
}]
}];
ReactDOM.render(
  <ej.circulargauge id="circulargauge1" scales={scales}></ej.circulargauge>,
  document.getElementById('circulargauge')
);
```

Execute the above code to render the following output.



### Multiple Custom Labels

You can set multiple custom labels in a single **Circular Gauge** by adding an array of custom label objects. Refer the following code example for multiple custom label functionality.

#### JAVASCRIPT

```
"use strict";
var scales = [{
  size: 10,
  shadowOffset: 10,
  showCustomLabels: true,
  showRanges: true,
  showScaleBar: true,
  radius: 150, size: 2,
  customLabels: [
    //custom label1
    {
      textAngle: 10,
      color: "Red",
      value: "CustomLabel1",
      font: {
        size: "18px",
        fontFamily: "Arial",
        fontStyle: "bold"
      },
      position: { x: 180, y: 100 }
    },
    //custom label2
    {
      textAngle: 10,
      color: "Green",
      value: "CustomLabel2",
      font: {
        size: "18px",
        fontFamily: "Arial",
        fontStyle: "bold"
      },
      position: { x: 180, y: 250 }
    }
  ]
},
```

```

    ]]
  }];
  ReactDOM.render(
    <ej.circulargauge id="circulargauge1" scales={scales}></ej.circulargauge>,
    document.getElementById('circulargauge')
  );

```

Execute the above code to render the following output.

![[/js/CircularGauge/Custom-labelsimages/Custom-labelsimg2.png)

### Outer Custom Label

**Outer Custom Label** is used to show custom labels outside the **gauge** control. The **Outer Custom Label** can be positioned with API called **outerCustomLabelPosition**. The value for this API is enumerable type and its possible values are,

- Right
- Left
- Top
- Bottom

When a custom label is to be displayed as an **Outer Custom Label**, set the API **customLabelType** as Outer. Refer to the following code example to get the **Outer Custom Label**.

### JAVASCRIPT

```

"use strict";
var scales = [{
  showLabels: true,
  radius: 150,
  // Customizes the custom label options.
  customLabels: [{
    value: "Average Speed",
    position: { x: 360, y: 30 },
    color: "Red",
    font: {
      size: "18px",
      fontFamily: "Arial",
      fontStyle: "bold"
    },
    positionType: "outer",
  }],
  pointers: [{
    value: 60,
    length: 100,
  }]
}];
var tooltip = {
  // Enables the custom label tooltip.
  showCustomLabelTooltip: true,
};
ReactDOM.render(
  <ej.circulargauge id="circulargauge1" tooltip={tooltip}
  outercustomlabelposition="right"
  scales={scales}></ej.circulargauge>,

```

```
document.getElementById('circulargauge')
);
```

Execute the above code to render the following output.

![[/js/CircularGauge/Custom-labelsimages/Custom-labelsimg3.png)

## Tooltip

**Tooltip** feature has been added to the **Circular Gauge**. **Circular Gauge** has several elements such as pointers, label, customLabel, scales, etc. There is a need for **Tooltip** feature in the **Circular Gauge** control because whenever the text hides or overrides with other gauge elements, it may not be fully visible. For resolving those problems **Tooltip** feature has been implemented in the **Circular Gauge** control.

## Default Tooltip

**Tooltip** has three attributes in it. The first two attributes such as **showLabelTooltip** and **showCustomLabelTooltip** are for enabling the **Tooltip** for label as well as custom label in default appearance.

**ShowLabelTooltip** is to enable the **Tooltip** for labels and **showCustomLabelTooltip** is for enabling the **Tooltip** option for customLabels.

## JAVASCRIPT

```
"use strict";
var tooltip = {
  //Enables the label tooltip.
  showLabelTooltip: true,
  //Enables the custom label tooltip.
  showCustomLabelTooltip: true,
};
var scales = [{
  showLabels: true,
  radius: 150,
  //Customizes the custom label options.
  customLabels: [{
    value: "095345",
    font: {
      size: "18px",
      fontFamily: "Arial",
      fontStyle: "bold"
    },
    position: { x: 180, y: 220 }
  }],
  //Customizes the pointers options.
  pointers: [{
    value: 60,
    length: 100,
  }]
}];
ReactDOM.render(
  <ej.circulargauge id="circulargauge1" tooltip={tooltip}
  scales={scales}></ej.circulargauge>,
  document.getElementById('CircularGauge1')
);
```

Execute the above code to render the following output.

![[/js/CircularGauge/Tooltipimages/Tooltipimg1.png]

### Tooltip Template

In **Tooltip** option, you can customize the Tooltip window by adding the tooltip template on that page with the help of API **TemplateID**. Refer to the following code example to know more about Tooltip template.

#### HTML

```
<div id="Tooltip" style="height: 60px; display: none;">
  <div id="icon">
    <div id="eficon"></div>
  </div>
  <div id="value">
    <div>
      <label id="efpercentage">&#160;#label#</label>
    </div>
  </div>
</div>
<div id="tooltipGauge"></div>
```

#### JAVASCRIPT

```
"use strict";
//Defines the tooltip object.
var tooltip = {
  // Enables the label tooltip.
  showLabelTooltip: true,
  // Enables the custom label tooltip.
  showCustomLabelTooltip: true,
  // Adds tooltip template.
  templateID: "Tooltip"
};
// Customizes the scale options.
var scales = [{
  showLabels: true,
  radius: 150,
  // Customizes the custom label options.
  customLabels: [{
    value: "0 9 5 3 4 5",
    font: {
      size: "18px",
      fontFamily: "Arial",
      fontStyle: "bold"
    },
    position: { x: 180, y: 220 }
  }],
  // Customizes the pointers options.
  pointers: [{
    value: 60,
    length: 100,
  }]
}];
ReactDOM.render(
```



```
<EJ.CircularGauge id="default" radius={100} value={55}
  backgroundColor="transparent" scales={scales}
>
</EJ.CircularGauge>,
document.getElementById('CircularGauge1')
);
```

### HTML

```
<style type="text/css">
<!-- Adds the necessary styles here. -->.
</style>
```

Execute the above code to render the following output.

![[/js/CircularGauge/Tooltipimages/Tooltipping2.png]

### Sub Gauges

A **Circular Gauge** containing another circular gauge is said to be **Sub Gauges**. In order to make a sample like watch that has second gauge, minute gauge and hour gauge, sub gauges are used.

### Adding SubGauges

Sub gauge collection is directly added to the scale object. Refer the following code example to add custom sub gauge collection in a **Gauge** control.

### HTML

```
<div id="CircularGauge1"></div>
```

### JAVASCRIPT

```
var scales: [{ showSubGauges: true,
  subGauges: [{
    gaugeID: "Gauge1"
  }]
}];
ReactDOM.render(
  <EJ.CircularGauge id="default" scales={scales}
  >
  </EJ.CircularGauge>,
  document.getElementById('CircularGauge1')
);
```

### Basic Customization

Basic attributes such as **height** and **width** property are used to set height and width of the sub gauge. You can easily position the gauge in another gauge using the **position** object and by giving the **X** and **Y** Coordinates value. **controlID** attribute is used to specify the sub gauge ID.

### HTML

```
<div id=" SubGauge1"></div>
<div id="CircularGauge1"> </div>
```

**JAVASCRIPT**

```

"use strict";
var scales= [{radius:190,
subGauges:[{
//For setting sub gauge control ID
controlID: "SubGauge1",
//For setting sub gauge height
height:250,
//For setting sub gauge width
width: 250,
//For setting sub gauge position
position: { x: 150, y: 100 }
}]
}]
var scales1=[{
radius: 110,
}]
ReactDOM.render(
<EJ.CircularGauge id="default" radius={100} value={55}
backgroundColor="transparent" scales={scales}
>
</EJ.CircularGauge>,
document.getElementById('CircularGauge1')
);
ReactDOM.render(
<EJ.CircularGauge id="default2" scales={scales1} backgroundColor="Blue"
value={50} radius={110}
>
</EJ.CircularGauge>,
document.getElementById('SubGauge1')
);

```

Execute the above code to render the following output.



**Multiple SubGauges**

You can set multiple sub gauges in a single **Circular Gauge** by adding an array of sub gauge objects. Refer the following code example for multiple sub gauges functionality.

**HTML**

```

<div id="CircularGauge1"></div>
<div id=" SubGauge1"> </div>
<div id=" SubGauge2"> </div>

```

**JAVASCRIPT**

```

"use strict";
var scales= [{
radius:250,
subGauges:[
//Sub gauge1
{
controlID: "SubGauge1",

```

```

height:200,
width: 200,
position: { x: 200, y: 150 }
},
//Sub gauge2
{
controlID: "SubGauge2",
height: 200,
width: 200,
position: { x: 50, y: 200 }
}]
}]
var scales1=[{
radius: 150
}]
var scales2=[{
radius: 150
}]
ReactDOM.render(
<EJ.CircularGauge id="default" radius={100} value={55}
backgroundColor="transparent" scales={scales}
>
</EJ.CircularGauge>,
document.getElementById('CircularGauge1')
);
ReactDOM.render(
<EJ.CircularGauge id="default2" scales={scales1} backgroundColor="#f5b43f"
>
</EJ.CircularGauge>,
document.getElementById('SubGauge1')
);
ReactDOM.render(
<EJ.CircularGauge id="default3" scales={scales3} backgroundColor="#f5b43f"
>
</EJ.CircularGauge>,
document.getElementById('SubGauge2')
);

```

Execute the above code to render the following output.



## Gauge Position

**Semi-circular Gauge** can be positioned within the canvas element which provides better appearance for the gauge in the canvas.

### Positioning

Semi-circular Gauge can be positioned with the help of the attribute called gaugePosition. It is an enumerable value. You can position the gauge away from the corner with the help of the distanceFromCorner attribute.

The possible enum values for the gaugePosition are as follows:

- Topleft
- Topcenter

- Topright
- Middleleft
- Center
- Middleright
- Bottomleft
- Bottomcenter
- Bottomright

### HTML

```
<div style="float: left" id="gaugel"></div>
<div id=" CoreCircularGaugehalfright "></div>
```

### JAVASCRIPT

```
"use strict";
var scales= [{
  startAngle: 270,
  sweepAngle: 180,
  radius: 160,
  showScaleBar: true,
  size: 1,
  maximum: 120,
  majorIntervalValue: 20,
  minorIntervalValue: 10,
  border: {
    width: 0.5,
  }
}];
var frame= {
  frameType: 'halfcircle',
  halfCircleFrameStartAngle: 270,
  halfCircleFrameEndAngle: 90
};
ReactDOM.render(
  <EJ.CircularGauge id="default" width={800} height={500} radius={120}
  value={60} backgroundColor="transparent" scales={scales}
  distanceFromCorner={30} frame={frame}
  // To set the gauge position.
  gaugePosition="center"
  >
  </EJ.CircularGauge>,
  document.getElementById('circulargauge')
);
```

Execute the above code to render the following output.

![[/js/CircularGauge/Gauge-Positionimages/Gauge-Positionimg1.png]

### Exporting

**Circular Gauge** has an exporting feature that converts **Gauge** control into image format and then export in client side. The method API **exportImage** is used to export the **Circular Gauge**. It has two arguments such as **file name** and **file format** to specify the file name and file formats. For exporting refer the following code example.

**HTML**

```
<input type="submit" value="Export Image" id="btnExportImage">
<div id="circulargauge"></div>
<div id="txtFileName">FileName </div>
<div id="ddlFileType">FileFormat </div>
</input>
<select id="Select1">
<option value="JPEG">JPEG</option>
<option value="PNG">PNG</option>
</select>
```

**JAVASCRIPT**

```
ReactDOM.render(
  <EJ.CircularGauge id="circulargauge">
  </EJ.CircularGauge>,
  document.getElementById('circulargauge')
);
ReactDOM.render(
  <EJ.Button id="btnExportImage" width={100} click={buttonclickevent}>
  </EJ.Button>
)
function buttonclickevent() {
  var FileName = $("#txtFileName").val();
  var FileFormat = $("#ddlFileType").val();
  $("#circulargauge").ejCircularGauge("exportImage", FileName, FileFormat);
}
```

Execute the above code to render the following output.



**Methods***destroy()*

destroy the circular gauge widget. all events bound using this.\_on will be unbind automatically and bring the control to pre-init state.

**HTML**

```
<div id="circular"></div>
```

**JAVASCRIPT**

```
ReactDOM.render(
  <EJ.CircularGauge id="default"></EJ.CircularGauge>,
  document.getElementById('circular')
);
function CircularGaugeMethod() {
  var circularObj = $("#default").data("ejCircularGauge");
  circularObj.destroy();
};
```

*exportImage()*

To export Image

**HTML**

```
<div id="circular"></div>
```

**JAVASCRIPT**

```
ReactDOM.render(  
  <EJ.CircularGauge id="default"></EJ.CircularGauge>,  
  document.getElementById('circular')  
);  
function CircularGaugeMethod(){  
  var circularObj = $("#default").data("ejCircularGauge");  
  circularObj.exportImage();  
};
```

*getBackNeedleLength()*

To get BackNeedleLength

**HTML**

```
<div id="circular"></div>
```

**JAVASCRIPT**

```
ReactDOM.render(  
  <EJ.CircularGauge id="default"></EJ.CircularGauge>,  
  document.getElementById('circular')  
);  
function CircularGaugeMethod(){  
  var circularObj = $("#default").data("ejCircularGauge");  
  circularObj.getBackNeedleLength();  
};
```

*getCustomLabelAngle()*

To get CustomLabelAngle

**HTML**

```
<div id="circular"></div>
```

**JAVASCRIPT**

```
ReactDOM.render(  
  <EJ.CircularGauge id="default"></EJ.CircularGauge>,  
  document.getElementById('circular')  
);  
function CircularGaugeMethod(){  
  var circularObj = $("#default").data("ejCircularGauge");  
  circularObj.getCustomLabelAngle();  
};
```

*getCustomLabelValue()*

To get CustomLabelValue

**HTML**

```
<div id="circular"></div>
```

**JAVASCRIPT**

```
ReactDOM.render(  
  <EJ.CircularGauge id="default"></EJ.CircularGauge>,  
  document.getElementById('circular')  
)  
;  
function CircularGaugeMethod(){  
  var circularObj = $("#default").data("ejCircularGauge");  
  circularObj.getCustomLabelValue();  
};
```

*getLabelAngle()*

To get LabelAngle

**HTML**

```
<div id="circular"></div>
```

**JAVASCRIPT**

```
ReactDOM.render(  
  <EJ.CircularGauge id="default"></EJ.CircularGauge>,  
  document.getElementById('circular')  
)  
;  
function CircularGaugeMethod(){  
  var circularObj = $("#default").data("ejCircularGauge");  
  circularObj.getLabelAngle();  
};
```

*getLabelDistanceFromScale()*

To get LabelDistanceFromScale

**HTML**

```
<div id="circular"></div>
```

**JAVASCRIPT**

```
ReactDOM.render(  
  <EJ.CircularGauge id="default"></EJ.CircularGauge>,  
  document.getElementById('circular')  
)  
;  
function CircularGaugeMethod(){  
  var circularObj = $("#default").data("ejCircularGauge");  
  circularObj.getLabelDistanceFromScale();  
};
```

*getLabelPlacement()*

To get LabelPlacement

**HTML**

```
<div id="circular"></div>
```

**JAVASCRIPT**

```
ReactDOM.render(  
  <EJ.CircularGauge id="default"></EJ.CircularGauge>,  
  document.getElementById('circular')  
)  
;  
function CircularGaugeMethod(){  
  var circularObj = $("#default").data("ejCircularGauge");  
  circularObj.getLabelPlacement();  
};
```

*getLabelStyle()*

To get LabelStyle

**HTML**

```
<div id="circular"></div>
```

**JAVASCRIPT**

```
ReactDOM.render(  
  <EJ.CircularGauge id="default"></EJ.CircularGauge>,  
  document.getElementById('circular')  
)  
;  
function CircularGaugeMethod(){  
  var circularObj = $("#default").data("ejCircularGauge");  
  circularObj.getLabelStyle();  
};
```

*getMajorIntervalValue()*

To get MajorIntervalValue

**HTML**

```
<div id="circular"></div>
```

**JAVASCRIPT**

```
ReactDOM.render(  
  <EJ.CircularGauge id="default"></EJ.CircularGauge>,  
  document.getElementById('circular')  
)  
;  
function CircularGaugeMethod(){  
  var circularObj = $("#default").data("ejCircularGauge");  
  circularObj.getMajorIntervalValue();  
};
```



*getMarkerDistanceFromScale()*

To get MarkerDistanceFromScale

**HTML**

```
<div id="circular"></div>
```

**JAVASCRIPT**

```
ReactDOM.render(  
  <EJ.CircularGauge id="default"></EJ.CircularGauge>,  
  document.getElementById('circular')  
)  
;  
function CircularGaugeMethod(){  
  var circularObj = $("#default").data("ejCircularGauge");  
  circularObj.getMarkerDistanceFromScale();  
};
```

*getMarkerStyle()*

To get MarkerStyle

**HTML**

```
<div id="circular"></div>
```

**JAVASCRIPT**

```
ReactDOM.render(  
  <EJ.CircularGauge id="default"></EJ.CircularGauge>,  
  document.getElementById('circular')  
)  
;  
function CircularGaugeMethod(){  
  var circularObj = $("#default").data("ejCircularGauge");  
  circularObj.getMarkerStyle();  
};
```

*getMaximumValue()*

To get MaximumValue

**HTML**

```
<div id="circular"></div>
```

**JAVASCRIPT**

```
ReactDOM.render(  
  <EJ.CircularGauge id="default"></EJ.CircularGauge>,  
  document.getElementById('circular')  
)  
;  
function CircularGaugeMethod(){  
  var circularObj = $("#default").data("ejCircularGauge");  
  circularObj.getMaximumValue();  
};
```

*getMinimumValue()*

To get MinimumValue

**HTML**

```
<div id="circular"></div>
```

**JAVASCRIPT**

```
ReactDOM.render(  
  <EJ.CircularGauge id="default"></EJ.CircularGauge>,  
  document.getElementById('circular')  
)  
;  
function CircularGaugeMethod(){  
  var circularObj = $("#default").data("ejCircularGauge");  
  circularObj.getMinimumValue();  
};
```

*getMinorIntervalValue()*

To get MinorIntervalValue

**HTML**

```
<div id="circular"></div>
```

**JAVASCRIPT**

```
ReactDOM.render(  
  <EJ.CircularGauge id="default"></EJ.CircularGauge>,  
  document.getElementById('circular')  
)  
;  
function CircularGaugeMethod(){  
  var circularObj = $("#default").data("ejCircularGauge");  
  circularObj.getMinorIntervalValue();  
};
```

*getNeedleStyle()*

To get NeedleStyle

**HTML**

```
<div id="circular"></div>
```

**JAVASCRIPT**

```
ReactDOM.render(  
  <EJ.CircularGauge id="default"></EJ.CircularGauge>,  
  document.getElementById('circular')  
)  
;  
function CircularGaugeMethod(){  
  var circularObj = $("#default").data("ejCircularGauge");  
  circularObj.getNeedleStyle();  
};
```

*getPointerCapBorderWidth()*

To get PointerCapBorderWidth

**HTML**

```
<div id="circular"></div>
```

**JAVASCRIPT**

```
ReactDOM.render(  
  <EJ.CircularGauge id="default"></EJ.CircularGauge>,  
  document.getElementById('circular')  
)  
;  
function CircularGaugeMethod(){  
  var circularObj = $("#default").data("ejCircularGauge");  
  circularObj.getPointerCapBorderWidth();  
};
```

*getPointerCapRadius()*

To get PointerCapRadius

**HTML**

```
<div id="circular"></div>
```

**JAVASCRIPT**

```
ReactDOM.render(  
  <EJ.CircularGauge id="default"></EJ.CircularGauge>,  
  document.getElementById('circular')  
)  
;  
function CircularGaugeMethod(){  
  var circularObj = $("#default").data("ejCircularGauge");  
  circularObj.getPointerCapRadius();  
};
```

*getPointerLength()*

To get PointerLength

**HTML**

```
<div id="circular"></div>
```

**JAVASCRIPT**

```
ReactDOM.render(  
  <EJ.CircularGauge id="default"></EJ.CircularGauge>,  
  document.getElementById('circular')  
)  
;  
function CircularGaugeMethod(){  
  var circularObj = $("#default").data("ejCircularGauge");  
  circularObj.getPointerLength();  
};
```

*getPointerNeedleType()*

To get PointerNeedleType

**HTML**

```
<div id="circular"></div>
```

**JAVASCRIPT**

```
ReactDOM.render(  
  <EJ.CircularGauge id="default"></EJ.CircularGauge>,  
  document.getElementById('circular')  
)  
;  
function CircularGaugeMethod(){  
  var circularObj = $("#default").data("ejCircularGauge");  
  circularObj.getPointerNeedleType();  
};
```

*getPointerPlacement()*

To get PointerPlacement

**HTML**

```
<div id="circular"></div>
```

**JAVASCRIPT**

```
ReactDOM.render(  
  <EJ.CircularGauge id="default"></EJ.CircularGauge>,  
  document.getElementById('circular')  
)  
;  
function CircularGaugeMethod(){  
  var circularObj = $("#default").data("ejCircularGauge");  
  circularObj.getPointerPlacement();  
};
```

*getPointerValue()*

To get PointerValue

**HTML**

```
<div id="circular"></div>
```

**JAVASCRIPT**

```
ReactDOM.render(  
  <EJ.CircularGauge id="default"></EJ.CircularGauge>,  
  document.getElementById('circular')  
)  
;  
function CircularGaugeMethod(){  
  var circularObj = $("#default").data("ejCircularGauge");  
  circularObj.getPointerValue();  
};
```

*getPointerWidth()*

To get PointerWidth

**HTML**

```
<div id="circular"></div>
```

**JAVASCRIPT**

```
ReactDOM.render(  
  <EJ.CircularGauge id="default"></EJ.CircularGauge>,  
  document.getElementById('circular')  
)  
;  
function CircularGaugeMethod(){  
  var circularObj = $("#default").data("ejCircularGauge");  
  circularObj.getPointerWidth();  
};
```

*getRangeBorderWidth()*

To get RangeBorderWidth

**HTML**

```
<div id="circular"></div>
```

**JAVASCRIPT**

```
ReactDOM.render(  
  <EJ.CircularGauge id="default"></EJ.CircularGauge>,  
  document.getElementById('circular')  
)  
;  
function CircularGaugeMethod(){  
  var circularObj = $("#default").data("ejCircularGauge");  
  circularObj.getRangeBorderWidth();  
};
```

*getRangeDistanceFromScale()*

To get RangeDistanceFromScale

**HTML**

```
<div id="circular"></div>
```

**JAVASCRIPT**

```
ReactDOM.render(  
  <EJ.CircularGauge id="default"></EJ.CircularGauge>,  
  document.getElementById('circular')  
)  
;  
function CircularGaugeMethod(){  
  var circularObj = $("#default").data("ejCircularGauge");  
  circularObj.getRangeDistanceFromScale();  
};
```

*getRangeEndValue()*

To get RangeEndValue

**HTML**

```
<div id="circular"></div>
```

**JAVASCRIPT**

```
ReactDOM.render(  
  <EJ.CircularGauge id="default"></EJ.CircularGauge>,  
  document.getElementById('circular')  
)  
;  
function CircularGaugeMethod(){  
  var circularObj = $("#default").data("ejCircularGauge");  
  circularObj.getRangeEndValue();  
};
```

*getRangePosition()*

To get RangePosition

**HTML**

```
<div id="circular"></div>
```

**JAVASCRIPT**

```
ReactDOM.render(  
  <EJ.CircularGauge id="default"></EJ.CircularGauge>,  
  document.getElementById('circular')  
)  
;  
function CircularGaugeMethod(){  
  var circularObj = $("#default").data("ejCircularGauge");  
  circularObj.getRangePosition();  
};
```

*getRangeSize()*

To get RangeSize

**HTML**

```
<div id="circular"></div>
```

**JAVASCRIPT**

```
ReactDOM.render(  
  <EJ.CircularGauge id="default"></EJ.CircularGauge>,  
  document.getElementById('circular')  
)  
;  
function CircularGaugeMethod(){  
  var circularObj = $("#default").data("ejCircularGauge");  
  circularObj.getRangeSize();  
};
```

*getRangeStartValue()*

To get RangeStartValue

**HTML**

```
<div id="circular"></div>
```

**JAVASCRIPT**

```
ReactDOM.render(  
  <EJ.CircularGauge id="default"></EJ.CircularGauge>,  
  document.getElementById('circular')  
)  
;  
function CircularGaugeMethod(){  
  var circularObj = $("#default").data("ejCircularGauge");  
  circularObj.getRangeStartValue();  
};
```

*getScaleBarSize()*

To get ScaleBarSize

**HTML**

```
<div id="circular"></div>
```

**JAVASCRIPT**

```
ReactDOM.render(  
  <EJ.CircularGauge id="default"></EJ.CircularGauge>,  
  document.getElementById('circular')  
)  
;  
function CircularGaugeMethod(){  
  var circularObj = $("#default").data("ejCircularGauge");  
  circularObj.getScaleBarSize();  
};
```

*getScaleBorderWidth()*

To get ScaleBorderWidth

**HTML**

```
<div id="circular"></div>
```

**JAVASCRIPT**

```
ReactDOM.render(  
  <EJ.CircularGauge id="default"></EJ.CircularGauge>,  
  document.getElementById('circular')  
)  
;  
function CircularGaugeMethod(){  
  var circularObj = $("#default").data("ejCircularGauge");  
  circularObj.getScaleBorderWidth();  
};
```

*getScaleDirection()*

To get ScaleDirection

**HTML**

```
<div id="circular"></div>
```

**JAVASCRIPT**

```
ReactDOM.render(  
  <EJ.CircularGauge id="default"></EJ.CircularGauge>,  
  document.getElementById('circular')  
)  
;  
function CircularGaugeMethod(){  
  var circularObj = $("#default").data("ejCircularGauge");  
  circularObj.getScaleDirection();  
};
```

*getScaleRadius()*

To get ScaleRadius

**HTML**

```
<div id="circular"></div>
```

**JAVASCRIPT**

```
ReactDOM.render(  
  <EJ.CircularGauge id="default"></EJ.CircularGauge>,  
  document.getElementById('circular')  
)  
;  
function CircularGaugeMethod(){  
  var circularObj = $("#default").data("ejCircularGauge");  
  circularObj.getScaleRadius();  
};
```

*getStartAngle()*

To get StartAngle

**HTML**

```
<div id="circular"></div>
```

**JAVASCRIPT**

```
ReactDOM.render(  
  <EJ.CircularGauge id="default"></EJ.CircularGauge>,  
  document.getElementById('circular')  
)  
;  
function CircularGaugeMethod(){  
  var circularObj = $("#default").data("ejCircularGauge");  
  circularObj.getStartAngle();  
};
```



*getSubGaugeLocation()*

To get SubGaugeLocation

**HTML**

```
<div id="circular"></div>
```

**JAVASCRIPT**

```
ReactDOM.render(  
  <EJ.CircularGauge id="default"></EJ.CircularGauge>,  
  document.getElementById('circular')  
)  
;  
function CircularGaugeMethod(){  
  var circularObj = $("#default").data("ejCircularGauge");  
  circularObj.getSubGaugeLocation();  
};
```

*getSweepAngle()*

To get SweepAngle

**HTML**

```
<div id="circular"></div>
```

**JAVASCRIPT**

```
ReactDOM.render(  
  <EJ.CircularGauge id="default"></EJ.CircularGauge>,  
  document.getElementById('circular')  
)  
;  
function CircularGaugeMethod(){  
  var circularObj = $("#default").data("ejCircularGauge");  
  circularObj.getSweepAngle();  
};
```

*getTickAngle()*

To get TickAngle

**HTML**

```
<div id="circular"></div>
```

**JAVASCRIPT**

```
ReactDOM.render(  
  <EJ.CircularGauge id="default"></EJ.CircularGauge>,  
  document.getElementById('circular')  
)  
;  
function CircularGaugeMethod(){  
  var circularObj = $("#default").data("ejCircularGauge");  
  circularObj.getTickAngle();  
};
```

*getTickDistanceFromScale()*

To get TickDistanceFromScale

**HTML**

```
<div id="circular"></div>
```

**JAVASCRIPT**

```
ReactDOM.render(  
  <EJ.CircularGauge id="default"></EJ.CircularGauge>,  
  document.getElementById('circular')  
)  
;  
function CircularGaugeMethod(){  
  var circularObj = $("#default").data("ejCircularGauge");  
  circularObj.getTickDistanceFromScale();  
};
```

*getTickHeight()*

To get TickHeight

**HTML**

```
<div id="circular"></div>
```

**JAVASCRIPT**

```
ReactDOM.render(  
  <EJ.CircularGauge id="default"></EJ.CircularGauge>,  
  document.getElementById('circular')  
)  
;  
function CircularGaugeMethod(){  
  var circularObj = $("#default").data("ejCircularGauge");  
  circularObj.getTickHeight();  
};
```

*getTickPlacement()*

To get TickPlacement

**HTML**

```
<div id="circular"></div>
```

**JAVASCRIPT**

```
ReactDOM.render(  
  <EJ.CircularGauge id="default"></EJ.CircularGauge>,  
  document.getElementById('circular')  
)  
;  
function CircularGaugeMethod(){  
  var circularObj = $("#default").data("ejCircularGauge");  
  circularObj.getTickPlacement();  
};
```

*getTickStyle()*

To get TickStyle

**HTML**

```
<div id="circular"></div>
```

**JAVASCRIPT**

```
ReactDOM.render(  
  <EJ.CircularGauge id="default"></EJ.CircularGauge>,  
  document.getElementById('circular')  
)  
;  
function CircularGaugeMethod(){  
  var circularObj = $("#default").data("ejCircularGauge");  
  circularObj.getTickStyle();  
};
```

*getTickWidth()*

To get TickWidth

**HTML**

```
<div id="circular"></div>
```

**JAVASCRIPT**

```
ReactDOM.render(  
  <EJ.CircularGauge id="default"></EJ.CircularGauge>,  
  document.getElementById('circular')  
)  
;  
function CircularGaugeMethod(){  
  var circularObj = $("#default").data("ejCircularGauge");  
  circularObj.getTickWidth();  
};
```

*includeFirstValue()*

To set includeFirstValue

**HTML**

```
<div id="circular"></div>
```

**JAVASCRIPT**

```
ReactDOM.render(  
  <EJ.CircularGauge id="default"></EJ.CircularGauge>,  
  document.getElementById('circular')  
)  
;  
function CircularGaugeMethod(){  
  var circularObj = $("#default").data("ejCircularGauge");  
  circularObj.includeFirstValue();  
};
```

*redraw()*

Switching the redraw option for the gauge

**HTML**

```
<div id="circular"></div>
```

**JAVASCRIPT**

```
ReactDOM.render(  
  <EJ.CircularGauge id="default"></EJ.CircularGauge>,  
  document.getElementById('circular')  
);  
function CircularGaugeMethod(){  
  var circularObj = $("#default").data("ejCircularGauge");  
  circularObj.redraw();  
};
```

*setBackNeedleLength()*

To set BackNeedleLength

**HTML**

```
<div id="circular"></div>
```

**JAVASCRIPT**

```
ReactDOM.render(  
  <EJ.CircularGauge id="default"></EJ.CircularGauge>,  
  document.getElementById('circular')  
);  
function CircularGaugeMethod(){  
  var circularObj = $("#default").data("ejCircularGauge");  
  circularObj.setBackNeedleLength();  
};
```

*setCustomLabelAngle()*

To set CustomLabelAngle

**HTML**

```
<div id="circular"></div>
```

**JAVASCRIPT**

```
ReactDOM.render(  
  <EJ.CircularGauge id="default"></EJ.CircularGauge>,  
  document.getElementById('circular')  
);  
function CircularGaugeMethod(){  
  var circularObj = $("#default").data("ejCircularGauge");  
  circularObj.setCustomLabelAngle();  
};
```

*setCustomLabelValue()*

To set CustomLabelValue

**HTML**

```
<div id="circular"></div>
```

**JAVASCRIPT**

```
ReactDOM.render(  
  <EJ.CircularGauge id="default"></EJ.CircularGauge>,  
  document.getElementById('circular')  
)  
;  
function CircularGaugeMethod(){  
  var circularObj = $("#default").data("ejCircularGauge");  
  circularObj.setCustomLabelValue();  
};
```

*setLabelAngle()*

To set LabelAngle

**HTML**

```
<div id="circular"></div>
```

**JAVASCRIPT**

```
ReactDOM.render(  
  <EJ.CircularGauge id="default"></EJ.CircularGauge>,  
  document.getElementById('circular')  
)  
;  
function CircularGaugeMethod(){  
  var circularObj = $("#default").data("ejCircularGauge");  
  circularObj.setLabelAngle();  
};
```

*setLabelDistanceFromScale()*

To set LabelDistanceFromScale

**HTML**

```
<div id="circular"></div>
```

**JAVASCRIPT**

```
ReactDOM.render(  
  <EJ.CircularGauge id="default"></EJ.CircularGauge>,  
  document.getElementById('circular')  
)  
;  
function CircularGaugeMethod(){  
  var circularObj = $("#default").data("ejCircularGauge");  
  circularObj.setLabelDistanceFromScale();  
};
```

*setLabelPlacement()*

To set LabelPlacement

**HTML**

```
<div id="circular"></div>
```

**JAVASCRIPT**

```
ReactDOM.render(  
  <EJ.CircularGauge id="default"></EJ.CircularGauge>,  
  document.getElementById('circular')  
);  
function CircularGaugeMethod(){  
  var circularObj = $("#default").data("ejCircularGauge");  
  circularObj.setLabelPlacement();  
};
```

*setLabelStyle()*

To set LabelStyle

**HTML**

```
<div id="circular"></div>
```

**JAVASCRIPT**

```
ReactDOM.render(  
  <EJ.CircularGauge id="default"></EJ.CircularGauge>,  
  document.getElementById('circular')  
);  
function CircularGaugeMethod(){  
  var circularObj = $("#default").data("ejCircularGauge");  
  circularObj.setLabelStyle();  
};
```

*setMajorIntervalValue()*

To set MajorIntervalValue

**HTML**

```
<div id="circular"></div>
```

**JAVASCRIPT**

```
ReactDOM.render(  
  <EJ.CircularGauge id="default"></EJ.CircularGauge>,  
  document.getElementById('circular')  
);  
function CircularGaugeMethod(){  
  var circularObj = $("#default").data("ejCircularGauge");  
  circularObj.setMajorIntervalValue();  
};
```

*setMarkerDistanceFromScale()*

To set MarkerDistanceFromScale

**HTML**

```
<div id="circular"></div>
```

**JAVASCRIPT**

```
ReactDOM.render(  
  <EJ.CircularGauge id="default"></EJ.CircularGauge>,  
  document.getElementById('circular')  
);  
function CircularGaugeMethod(){  
  var circularObj = $("#default").data("ejCircularGauge");  
  circularObj.setMarkerDistanceFromScale();  
};
```

*setMarkerStyle()*

To set MarkerStyle

**HTML**

```
<div id="circular"></div>
```

**JAVASCRIPT**

```
ReactDOM.render(  
  <EJ.CircularGauge id="default"></EJ.CircularGauge>,  
  document.getElementById('circular')  
);  
function CircularGaugeMethod(){  
  var circularObj = $("#default").data("ejCircularGauge");  
  circularObj.setMarkerStyle();  
};
```

*setMaximumValue()*

To set MaximumValue

**HTML**

```
<div id="circular"></div>
```

**JAVASCRIPT**

```
ReactDOM.render(  
  <EJ.CircularGauge id="default"></EJ.CircularGauge>,  
  document.getElementById('circular')  
);  
function CircularGaugeMethod(){  
  var circularObj = $("#default").data("ejCircularGauge");  
  circularObj.setMaximumValue();  
};
```

*setMinimumValue()*

To set MinimumValue

**HTML**

```
<div id="circular"></div>
```

**JAVASCRIPT**

```
ReactDOM.render(  
  <EJ.CircularGauge id="default"></EJ.CircularGauge>,  
  document.getElementById('circular')  
)  
;  
function CircularGaugeMethod(){  
  var circularObj = $("#default").data("ejCircularGauge");  
  circularObj.setMinimumValue();  
};
```

*setMinorIntervalValue()*

To set MinorIntervalValue

**HTML**

```
<div id="circular"></div>
```

**JAVASCRIPT**

```
ReactDOM.render(  
  <EJ.CircularGauge id="default"></EJ.CircularGauge>,  
  document.getElementById('circular')  
)  
;  
function CircularGaugeMethod(){  
  var circularObj = $("#default").data("ejCircularGauge");  
  circularObj.setMinorIntervalValue();  
};
```

*setNeedleStyle()*

To set NeedleStyle

**HTML**

```
<div id="circular"></div>
```

**JAVASCRIPT**

```
ReactDOM.render(  
  <EJ.CircularGauge id="default"></EJ.CircularGauge>,  
  document.getElementById('circular')  
)  
;  
function CircularGaugeMethod(){  
  var circularObj = $("#default").data("ejCircularGauge");  
  circularObj.setNeedleStyle();  
};
```



*setPointerCapBorderWidth()*

To set PointerCapBorderWidth

**HTML**

```
<div id="circular"></div>
```

**JAVASCRIPT**

```
ReactDOM.render(  
  <EJ.CircularGauge id="default"></EJ.CircularGauge>,  
  document.getElementById('circular')  
)  
;  
function CircularGaugeMethod(){  
  var circularObj = $("#default").data("ejCircularGauge");  
  circularObj.setPointerCapBorderWidth();  
};
```

*setPointerCapRadius()*

To set PointerCapRadius

**HTML**

```
<div id="circular"></div>
```

**JAVASCRIPT**

```
ReactDOM.render(  
  <EJ.CircularGauge id="default"></EJ.CircularGauge>,  
  document.getElementById('circular')  
)  
;  
function CircularGaugeMethod(){  
  var circularObj = $("#default").data("ejCircularGauge");  
  circularObj.setPointerCapRadius();  
};
```

*setPointerLength()*

To set PointerLength

**HTML**

```
<div id="circular"></div>
```

**JAVASCRIPT**

```
ReactDOM.render(  
  <EJ.CircularGauge id="default"></EJ.CircularGauge>,  
  document.getElementById('circular')  
)  
;  
function CircularGaugeMethod(){  
  var circularObj = $("#default").data("ejCircularGauge");  
  circularObj.setPointerLength();  
};
```

*setPointerNeedleType()*

To set PointerNeedleType

**HTML**

```
<div id="circular"></div>
```

**JAVASCRIPT**

```
ReactDOM.render(  
  <EJ.CircularGauge id="default"></EJ.CircularGauge>,  
  document.getElementById('circular')  
);  
function CircularGaugeMethod(){  
  var circularObj = $("#default").data("ejCircularGauge");  
  circularObj.setPointerNeedleType();  
};
```

*setPointerPlacement()*

To set PointerPlacement

**HTML**

```
<div id="circular"></div>
```

**JAVASCRIPT**

```
ReactDOM.render(  
  <EJ.CircularGauge id="default"></EJ.CircularGauge>,  
  document.getElementById('circular')  
);  
function CircularGaugeMethod(){  
  var circularObj = $("#default").data("ejCircularGauge");  
  circularObj.setPointerPlacement();  
};
```

*setPointerValue()*

To set PointerValue

**HTML**

```
<div id="circular"></div>
```

**JAVASCRIPT**

```
ReactDOM.render(  
  <EJ.CircularGauge id="default"></EJ.CircularGauge>,  
  document.getElementById('circular')  
);  
function CircularGaugeMethod(){  
  var circularObj = $("#default").data("ejCircularGauge");  
  circularObj.setPointerValue();  
};
```

*setPointerWidth()*

To set PointerWidth

**HTML**

```
<div id="circular"></div>
```

**JAVASCRIPT**

```
ReactDOM.render(  
  <EJ.CircularGauge id="default"></EJ.CircularGauge>,  
  document.getElementById('circular')  
)  
;  
function CircularGaugeMethod(){  
  var circularObj = $("#default").data("ejCircularGauge");  
  circularObj.setPointerWidth();  
};
```

*setRangeBorderWidth()*

To set RangeBorderWidth

**HTML**

```
<div id="circular"></div>
```

**JAVASCRIPT**

```
ReactDOM.render(  
  <EJ.CircularGauge id="default"></EJ.CircularGauge>,  
  document.getElementById('circular')  
)  
;  
function CircularGaugeMethod(){  
  var circularObj = $("#default").data("ejCircularGauge");  
  circularObj.setRangeBorderWidth();  
};
```

*setRangeDistanceFromScale()*

To set RangeDistanceFromScale

**HTML**

```
<div id="circular"></div>
```

**JAVASCRIPT**

```
ReactDOM.render(  
  <EJ.CircularGauge id="default"></EJ.CircularGauge>,  
  document.getElementById('circular')  
)  
;  
function CircularGaugeMethod(){  
  var circularObj = $("#default").data("ejCircularGauge");  
  circularObj.setRangeDistanceFromScale();  
};
```

*setRangeEndValue()*

To set RangeEndValue

**HTML**

```
<div id="circular"></div>
```

**JAVASCRIPT**

```
ReactDOM.render(  
  <EJ.CircularGauge id="default"></EJ.CircularGauge>,  
  document.getElementById('circular')  
)  
;  
function CircularGaugeMethod(){  
  var circularObj = $("#default").data("ejCircularGauge");  
  circularObj.setRangeEndValue();  
};
```

*setRangePosition()*

To set RangePosition

**HTML**

```
<div id="circular"></div>
```

**JAVASCRIPT**

```
ReactDOM.render(  
  <EJ.CircularGauge id="default"></EJ.CircularGauge>,  
  document.getElementById('circular')  
)  
;  
function CircularGaugeMethod(){  
  var circularObj = $("#default").data("ejCircularGauge");  
  circularObj.setRangePosition();  
};
```

*setRangeSize()*

To set RangeSize

**HTML**

```
<div id="circular"></div>
```

**JAVASCRIPT**

```
ReactDOM.render(  
  <EJ.CircularGauge id="default"></EJ.CircularGauge>,  
  document.getElementById('circular')  
)  
;  
function CircularGaugeMethod(){  
  var circularObj = $("#default").data("ejCircularGauge");  
  circularObj.setRangeSize();  
};
```

*setRangeStartValue()*

To set RangeStartValue

**HTML**

```
<div id="circular"></div>
```

**JAVASCRIPT**

```
ReactDOM.render(  
  <EJ.CircularGauge id="default"></EJ.CircularGauge>,  
  document.getElementById('circular')  
)  
;  
function CircularGaugeMethod(){  
  var circularObj = $("#default").data("ejCircularGauge");  
  circularObj.setRangeStartValue();  
};
```

*setScaleBarSize()*

To set ScaleBarSize

**HTML**

```
<div id="circular"></div>
```

**JAVASCRIPT**

```
ReactDOM.render(  
  <EJ.CircularGauge id="default"></EJ.CircularGauge>,  
  document.getElementById('circular')  
)  
;  
function CircularGaugeMethod(){  
  var circularObj = $("#default").data("ejCircularGauge");  
  circularObj.setScaleBarSize();  
};
```

*setScaleBorderWidth()*

To set ScaleBorderWidth

**HTML**

```
<div id="circular"></div>
```

**JAVASCRIPT**

```
ReactDOM.render(  
  <EJ.CircularGauge id="default"></EJ.CircularGauge>,  
  document.getElementById('circular')  
)  
;  
function CircularGaugeMethod(){  
  var circularObj = $("#default").data("ejCircularGauge");  
  circularObj.setScaleBorderWidth();  
};
```

*setScaleDirection()*

To set ScaleDirection

**HTML**

```
<div id="circular"></div>
```

**JAVASCRIPT**

```
ReactDOM.render(  
  <EJ.CircularGauge id="default"></EJ.CircularGauge>,  
  document.getElementById('circular')  
)  
;  
function CircularGaugeMethod(){  
  var circularObj = $("#default").data("ejCircularGauge");  
  circularObj.setScaleDirection();  
};
```

*setScaleRadius()*

To set ScaleRadius

**HTML**

```
<div id="circular"></div>
```

**JAVASCRIPT**

```
ReactDOM.render(  
  <EJ.CircularGauge id="default"></EJ.CircularGauge>,  
  document.getElementById('circular')  
)  
;  
function CircularGaugeMethod(){  
  var circularObj = $("#default").data("ejCircularGauge");  
  circularObj.setScaleRadius();  
};
```

*setStartAngle()*

To set StartAngle

**HTML**

```
<div id="circular"></div>
```

**JAVASCRIPT**

```
ReactDOM.render(  
  <EJ.CircularGauge id="default"></EJ.CircularGauge>,  
  document.getElementById('circular')  
)  
;  
function CircularGaugeMethod(){  
  var circularObj = $("#default").data("ejCircularGauge");  
  circularObj.setStartAngle();  
};
```

*setSubGaugeLocation()*

To set SubGaugeLocation

**HTML**

```
<div id="circular"></div>
```

**JAVASCRIPT**

```
ReactDOM.render(  
  <EJ.CircularGauge id="default"></EJ.CircularGauge>,  
  document.getElementById('circular')  
);  
function CircularGaugeMethod(){  
  var circularObj = $("#default").data("ejCircularGauge");  
  circularObj.setSubGaugeLocation();  
};
```

*setSweepAngle()*

To set SweepAngle

**HTML**

```
<div id="circular"></div>
```

**JAVASCRIPT**

```
ReactDOM.render(  
  <EJ.CircularGauge id="default"></EJ.CircularGauge>,  
  document.getElementById('circular')  
);  
function CircularGaugeMethod(){  
  var circularObj = $("#default").data("ejCircularGauge");  
  circularObj.setSweepAngle();  
};
```

*setTickAngle()*

To set TickAngle

**HTML**

```
<div id="circular"></div>
```

**JAVASCRIPT**

```
ReactDOM.render(  
  <EJ.CircularGauge id="default"></EJ.CircularGauge>,  
  document.getElementById('circular')  
);  
function CircularGaugeMethod(){  
  var circularObj = $("#default").data("ejCircularGauge");  
  circularObj.setTickAngle();  
};
```

*setTickDistanceFromScale()*

To set TickDistanceFromScale

**HTML**

```
<div id="circular"></div>
```

**JAVASCRIPT**

```
ReactDOM.render(  
  <EJ.CircularGauge id="default"></EJ.CircularGauge>,  
  document.getElementById('circular')  
);  
function CircularGaugeMethod(){  
  var circularObj = $("#default").data("ejCircularGauge");  
  circularObj.setTickDistanceFromScale();  
};
```

*setTickHeight()*

To set TickHeight

**HTML**

```
<div id="circular"></div>
```

**JAVASCRIPT**

```
ReactDOM.render(  
  <EJ.CircularGauge id="default"></EJ.CircularGauge>,  
  document.getElementById('circular')  
);  
function CircularGaugeMethod(){  
  var circularObj = $("#default").data("ejCircularGauge");  
  circularObj.setTickHeight();  
};
```

*setTickPlacement()*

To set TickPlacement

**HTML**

```
<div id="circular"></div>
```

**JAVASCRIPT**

```
ReactDOM.render(  
  <EJ.CircularGauge id="default"></EJ.CircularGauge>,  
  document.getElementById('circular')  
);  
function CircularGaugeMethod(){  
  var circularObj = $("#default").data("ejCircularGauge");  
  circularObj.setTickPlacement();  
};
```



*setTickStyle()*

To set TickStyle

**HTML**

```
<div id="circular"></div>
```

**JAVASCRIPT**

```
ReactDOM.render(  
  <EJ.CircularGauge id="default"></EJ.CircularGauge>,  
  document.getElementById('circular')  
);  
function CircularGaugeMethod(){  
  var circularObj = $("#default").data("ejCircularGauge");  
  circularObj.setTickStyle();  
};
```

*setTickWidth()*

To set TickWidth

**HTML**

```
<div id="circular"></div>
```

**JAVASCRIPT**

```
ReactDOM.render(  
  <EJ.CircularGauge id="default"></EJ.CircularGauge>,  
  document.getElementById('circular')  
);  
function CircularGaugeMethod(){  
  var circularObj = $("#default").data("ejCircularGauge");  
  circularObj.setTickWidth();  
};
```

*Events**drawCustomLabel*

Triggers while the custom labels are being drawn on the gauge.

**HTML**

```
<div id="circular"></div>
```

**JAVASCRIPT**

```
ReactDOM.render(  
  <EJ.CircularGauge id="default" drawCustomLabel = {DrawCustomLabel}  
  ></EJ.CircularGauge>,  
  document.getElementById('circular')  
);  
function DrawCustomLabel(){  
  // Do Something  
};
```

### *drawIndicators*

Triggers while the indicators are being started to drawn on the gauge.

#### **HTML**

```
<div id="circular"></div>
```

#### **JAVASCRIPT**

```
ReactDOM.render(  
  <EJ.CircularGauge id="default" drawIndicators = {DrawIndicators}  
  ></EJ.CircularGauge>,  
  document.getElementById('circular')  
)  
;  
function DrawIndicators() {  
  // Do Something  
};
```

### *drawLabels*

Triggers while the labels are being drawn on the gauge.

#### **HTML**

```
<div id="circular"></div>
```

#### **JAVASCRIPT**

```
ReactDOM.render(  
  <EJ.CircularGauge id="default" drawLabels = {DrawLabels}  
  ></EJ.CircularGauge>,  
  document.getElementById('circular')  
)  
;  
function DrawLabels() {  
  // Do Something  
};
```

### *drawPointerCap*

Triggers while the pointer cap is being drawn on the gauge.

#### **HTML**

```
<div id="circular"></div>
```

#### **JAVASCRIPT**

```
ReactDOM.render(  
  <EJ.CircularGauge id="default" drawPointerCap = {DrawPointerCap}  
  ></EJ.CircularGauge>,  
  document.getElementById('circular')  
)  
;  
function DrawPointerCap() {  
  // Do Something  
};
```

### *drawPointers*

Triggers while the pointers are being drawn on the gauge.

#### **HTML**

```
<div id="circular"></div>
```

#### **JAVASCRIPT**

```
ReactDOM.render(  
  <EJ.CircularGauge id="default" drawPointers = {DrawPointers}  
  ></EJ.CircularGauge>,  
  document.getElementById('circular')  
);  
function DrawPointers() {  
  // Do Something  
};
```

### *drawRange*

Triggers when the ranges begin to be getting drawn on the gauge.

#### **HTML**

```
<div id="circular"></div>
```

#### **JAVASCRIPT**

```
ReactDOM.render(  
  <EJ.CircularGauge id="default" drawRange = {DrawRange} ></EJ.CircularGauge>,  
  document.getElementById('circular')  
);  
function DrawRange() {  
  // Do Something  
};
```

### *drawTicks*

Triggers while the ticks are being drawn on the gauge.

#### **HTML**

```
<div id="circular"></div>
```

#### **JAVASCRIPT**

```
ReactDOM.render(  
  <EJ.CircularGauge id="default" drawTicks = {DrawTicks} ></EJ.CircularGauge>,  
  document.getElementById('circular')  
);  
function DrawTicks() {  
  // Do Something  
};
```

### *load*

Triggers while the gauge start to Load.

#### **HTML**

```
<div id="circular"></div>
```

#### **JAVASCRIPT**

```
ReactDOM.render(  
  <EJ.CircularGauge id="default" load = {Load} ></EJ.CircularGauge>,  
  document.getElementById('circular')  
);  
function Load() {  
  // Do Something  
};
```

### *mouseClick*

Triggers when the left mouse button is clicked.

#### **HTML**

```
<div id="circular"></div>
```

#### **JAVASCRIPT**

```
ReactDOM.render(  
  <EJ.CircularGauge id="default" mouseClick = {MouseClicked}  
  ></EJ.CircularGauge>,  
  document.getElementById('circular')  
);  
function MouseClick() {  
  // Do Something  
};
```

### *mouseClickMove*

Triggers when clicking and dragging the mouse pointer over the gauge pointer.

#### **HTML**

```
<div id="circular"></div>
```

#### **JAVASCRIPT**

```
ReactDOM.render(  
  <EJ.CircularGauge id="default" drawCustomLabel = {MouseClickedMove}  
  ></EJ.CircularGauge>,  
  document.getElementById('circular')  
);  
function MouseClickMove() {  
  // Do Something  
};
```

### *mouseClickUp*

Triggers when the mouse click is released.

#### **HTML**

```
<div id="circular"></div>
```

#### **JAVASCRIPT**

```
ReactDOM.render(  
  <EJ.CircularGauge id="default" mouseClickUp = {MouseClickUp}  
  ></EJ.CircularGauge>,  
  document.getElementById('circular')  
);  
function MouseClickUp() {  
  // Do Something  
};
```

### *renderComplete*

Triggers when the rendering of the gauge is completed.

#### **HTML**

```
<div id="circular"></div>
```

#### **JAVASCRIPT**

```
ReactDOM.render(  
  <EJ.CircularGauge id="default" renderComplete = {RenderComplete}  
  ></EJ.CircularGauge>,  
  document.getElementById('circular')  
);  
function RenderComplete() {  
  // Do Something  
};
```

## ColorPicker

### Overview

The **ColorPicker** control provides you a rich visual interface for color selection. You can select the color from the professionally designed palettes or custom color. By clicking a point on the color, you can change the active color to the color that is located under the pointer.

You can also choose colors in different specifications; red-green-blue-alpha (opacity) (RGBA), hue-saturation-value (HSV) and hexadecimal (HEX). The **ColorPicker** provides a selection of basic colors, standard presets, custom colors and color swatches.

### Getting Started

Using the following steps, you can create a React ColorPicker component. The basic rendering of React ColorPicker can be achieved with default functionality.

### Create an ColorPicker

You can create a React application and add necessary scripts and styles with the help of the given [React Getting Started Documentation](#).

Create a JSX file for rendering ColorPicker component using <EJ.ColorPicker> syntax. Add required properties to it in <EJ.ColorPicker> tag element

#### HTML

```
ReactDOM.render(  
  <EJ.ColorPicker>  
  </EJ.ColorPicker>,  
  document.getElementById('colorpick')  
) ;
```

Define an HTML element for adding ColorPicker in the application and refer the JSX file created.

#### HTML

```
<div id="colorpick"></div>  
<script type="text/babel" src="sample.jsx">
```

This will render an empty ColorPicker component on executing.

### Configure Properties

In the JSX, need to declare the ColorPicker properties. Refer to the following code,

#### HTML

```
ReactDOM.render(  
  <EJ.ColorPicker value="#278787">  
  </EJ.ColorPicker>,  
  document.getElementById('colorpick')  
) ;
```

Run the above code to render the following output,



*Note: You can find the ColorPicker properties from the [API reference](#) document.*

## CurrencyTextbox

### Getting Started

This section helps to get started of the CurrencyTextbox component in a React application

#### Create a CurrencyTextbox

Refer the common React Getting Started Documentation to create an application and add necessary scripts and styles for rendering our ReactJS components.

Create a JSX file and use <EJ.CurrencyTextbox> syntax to render React CurrencyTextbox component. Add required properties to <EJ.CurrencyTextbox> tag element.

#### HTML

```
ReactDOM.render(  
  <EJ.CurrencyTextbox>  
  </EJ.CurrencyTextbox>,  
  document.getElementById('currency')  
) ;
```

Define an HTML element for adding CurrencyTextbox in the application and refer the JSX file created.

#### HTML

```
<div id="currency"></div>  
<script type="text/babel" src="sample.jsx">
```

This will render an empty CurrencyTextbox component on executing.

#### Configure Properties

In the JSX, need to declare the CurrencyTextbox properties. Refer to the following code,.

## HTML

```
ReactDOM.render(  
  <EJ.CurrencyTextbox value={10} width="250px">  
  </EJ.CurrencyTextbox >,  
  document.getElementById('currency')  
)
```

Run the above code to render the following output,



Note: You can find the CurrencyTextbox properties from the [API reference](#) document.

## DatePicker

### OverView

The ReactJS DatePicker control provides an intuitive visual interface for selecting a date. Its rich feature set includes functionalities like highlighting special dates, globalization, responsive rendering, accessibility and much more.

### Key features

- Formatting the date value.
- Globalization.
- Range the date value to pick.
- Quick picking date by drill down or up.
- State persistence.
- Responsive dimension.
- Flexible customization.
- Custom Styling.
- Built-in jQuery validation.
- Accessibility (keyboard and ARIA).

### Getting Started

This section helps to get start with the DatePicker component in a ReactJS application

#### Create a DatePicker

Refer the common ReactJS Getting Started Documentation to create an application and add necessary scripts and styles for rendering our ReactJS components.

#### Using JSX template

Create a JSX file and add the EJ component, by using DatePicker tag like <EJ.DatePicker>. This JSX file will be converted to JS file which can be referred in html file.

**Below code example explain to render the EJ components,**

## JAVASCRIPT

```
ReactDOM.render(  
  <EJ.DatePicker id="dtp" value={new Date()} >
```



```
</EJ.DatePicker>,  
document.getElementById('dtp')  
);
```

Define an HTML element for adding DatePicker in the application and refer the JSX file that you have created.

### HTML

```
<div id="container"></div>
```

Configure properties of the DatePicker control

### JAVASCRIPT

```
ReactDOM.render(  
  <EJ.DatePicker  
    value = {new Date()}  
    minDate={new Date("11/11/2011")}  
    maxDate= {new Date("11/11/2018")}  
  >  
    </EJ.DatePicker>,  
  document.getElementById('container')  
);
```

*Without using JSX template*

You can also render the DatePicker component without using JSX template. Here the DatePicker component can be created with use of script section lie below.

### JAVASCRIPT

```
<script>  
ReactDOM.render(  
  React.createElement(EJ.DatePicker, { id: "container", }  
),  
  document.getElementById('container')  
);  
</script>
```

This will render an empty DatePicker component on executing.

### Configure Properties

In the JSX file, you can define all required APIs of DatePicker based on needs. Refer below code example to know the adding the required APIs.

### JAVASCRIPT

```
<script>  
ReactDOM.render(  
  React.createElement(EJ.DatePicker, {id: "container",  
    value:new Date("11/12/2018"),  
    minDate: new Date("12/12/2015"),  
    maxDate: new Date("1/12/2019")  
  }  
),  
  document.getElementById('container')
```

```
);  
</script>
```

Run the above code to render the following output,



## DateTimePicker

### Overview

**DateTimePicker** control used to input the date and time with a specific format. The DateTimePicker control combines the DatePicker and TimePicker controls so that users can select the date and time in their desired format.

### Key Features

- **DateTime format:** Supports all valid date and time formats.
- **Localization:** Supports localization to different cultures.
- **Persist:** Supports state maintenance during page refresh.
- **RTL:** Support for Right to Left alignment of content in DateTimePicker control.
- **Themes:** Supports 17 built-in themes (6 – flat and 6 – gradient effects), and also support custom skin options to set user-defined themes.

As the DateTimePicker inherits the functionalities of the DatePicker and TimePicker controls, it supports the basic functionalities of both controls.

### GettingStarted

This section discloses the details on how to render and configure DateTimePicker component in a ReactJS application.

### Create a DateTimePicker

Create an HTML page and refer the necessary script and CSS dependency files in your application with the help of given [getting started documentation](#).

With ReactJS, the components can be initialized in two ways.

1. Using jsx Template
2. Without using jsx Template

#### Using jsx Template

You can render EJ components by using JSX template, wherein this JSX file will be converted to its equivalent JS file.

1. Create a div element in the HTML file and give it an ID.

#### HTML

```
<body>
<div id="datetimepicker"></div>
</body>
```

2. Create a JSX file and initialize DateTimePicker component by using the below code snippet.

#### HTML

```
ReactDOM.render (
  <EJ.DateTimePicker id="datetimepicker" value={new Date()}>
</EJ.DateTimePicker>,
document.getElementById('datetimepicker')
);
```

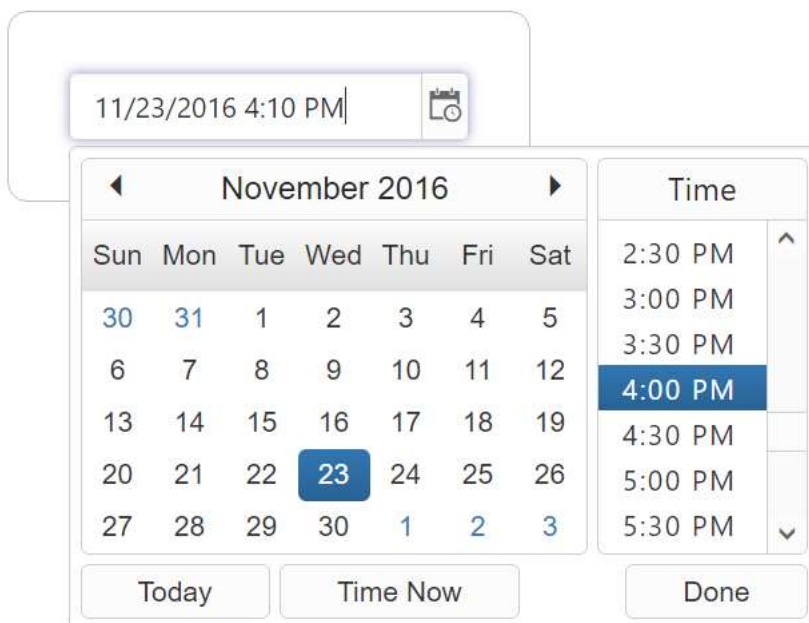
3. Refer the JSX file created in last step in the HTML file as given below.

#### HTML

```
<body>
<div id="datetimepicker"></div>
<!-- Datetimepicker.jsx created in previous step-->
<script type="text/babel" src="Datetimepicker.jsx">
</script>
</body>
```

Now the jsx file will be compiled into its equivalent Javascript file by means of Babel.

Execute the above code to render Datetimepicker component.



### Configuring DateTimePicker

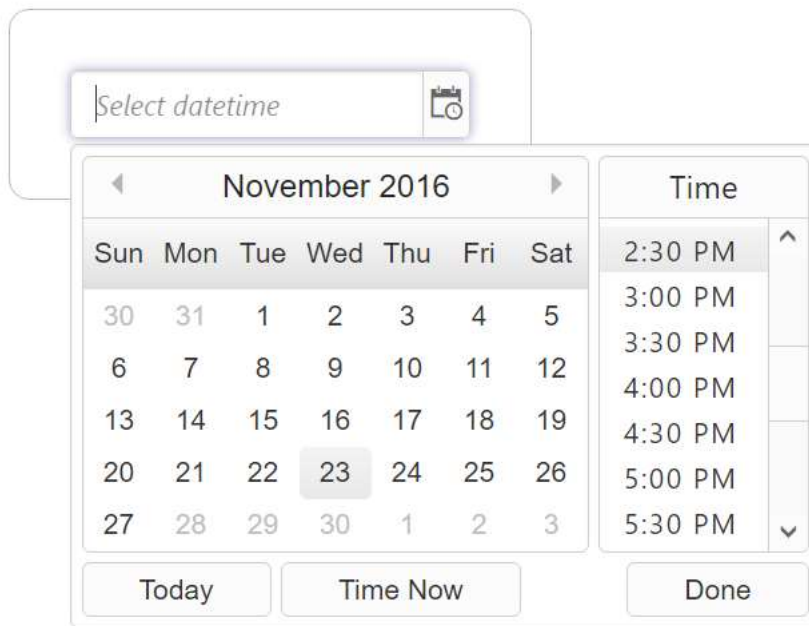
#### Set minDateTime and maxDateTime

EJ DateTimePicker provides API through which you can set the maximum and minimum allowed date and time values. Min and Max date and time values can be set at initialization as show below.

#### JAVASCRIPT

```
ReactDOM.render(  
  <EJ.DateTimePicker id="datetimepicker" minDateTime={new Date("11/1/2016  
10:00 AM")} maxDateTime={new Date("11/27/2016 10:00 PM")} >  
    </EJ.DateTimePicker>,  
  document.getElementById('datetimepicker')  
)
```

The following screenshot illustrates the output of above code.



*Note: You need to refer browser.min.js file in the script section and specify the type attribute of script tag to text/babel for compiling the JSX template.*

*Note: You can find the DateTimePicker properties from the [API reference](#) document.*

#### Without using jsx Template

The DateTimePicker component can be initialized without using JSX template.

1. Create an HTML page and render a  
element with an ID set to it.

#### HTML

```
<div id="datetimepicker"></div>
```

2. Render the DateTimePicker component by using the below mentioned code snippet.

#### JAVASCRIPT

```
ReactDOM.render(
  React.createElement(EJ.DateTimePicker, {
    id: "datetimepicker-0",
    value: new Date()
  })
),
document.getElementById('datetimepicker')
```

Run the above code snippet to get the following output.

### Configuring DateTimePicker

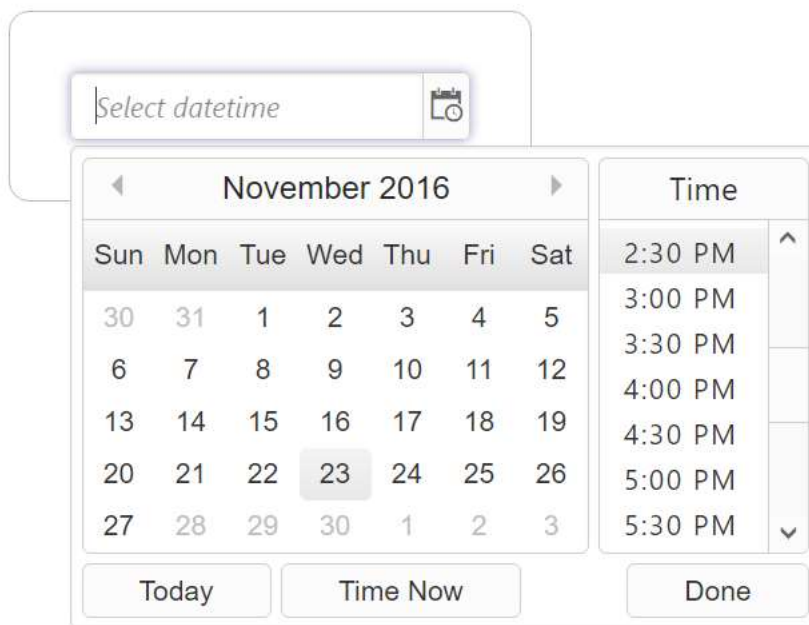
#### Set minDateTime and maxDateTime

EJ DateTimePicker provides API through which you can set the maximum and minimum allowed date and time values. Min and Max date and time values can be set at initialization as show below.

#### JAVASCRIPT

```
ReactDOM.render(
  React.createElement(EJ.DateTimePicker, {
    id: "datetimepicker-0",
    minDateTime: new Date("11/1/2016 10:00 AM"),
    maxDateTime: new Date("11/27/2016 10:00 PM")
  }
),
document.getElementById('datetimepicker')
);
```

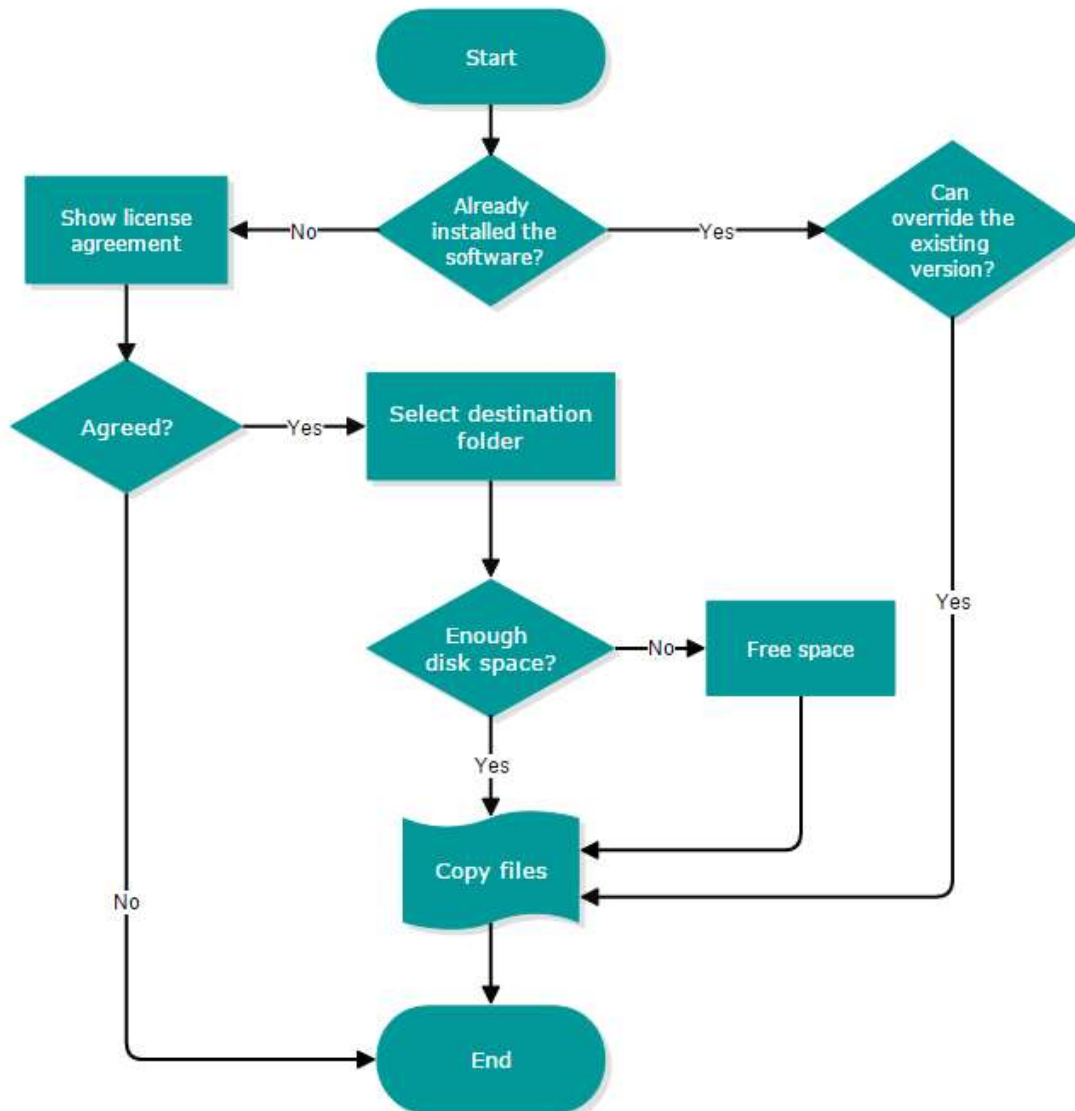
The following screenshot illustrates the output of above code.



## Diagram

### Overview

**Essential Diagram React JS** creates rich Visio-like applications. Its Framework comprises of many Elements that helps you to create an application easily. The rich feature set of the Diagram control includes Snapping, Guidelines, Gridlines, Serialization and Zooming.



The list of rich features of Diagram control in React JS is as follows.

- **Node, Connector, Group, Port, Label:** Element used to compose diagram.
- **Symbol Palette:** It holds a list of symbols that is dropped over diagram.
- **Clipboard Commands:** Performs Cut, Copy and Paste operations.
- **Undo/Redo:** Performs correction in recent change.
- **Serialization:** Save current state of Diagram, and load them back when needed.
- **Snapping:** Snap the diagram elements towards the nearest elements.
- **Gridlines:** Visual horizontal/vertical lines that helps to align elements on diagram.
- **Interaction:** Zoom, pan, multiple selections, keyboard shortcuts, snapping.
- **Layout:** Arranges nodes in a tree-like structure based on relationship between the Nodes.

### Getting started

This section explains briefly about how to create a **Diagram** control in your application with **reactjs**.

## Adding Script Reference

Create an **HTML** page and add the scripts references in the order mentioned in the following code example.

- [jQuery](#) 1.10.2 and later versions

The required ReactJS script dependencies as follows. And you can also refer [React](#) to know more about react js.

- react.min.js - <http://cdn.syncfusion.com/js/assets/external/react.min.js>
- react-dom.min.js - <http://cdn.syncfusion.com/js/assets/external/react-dom.min.js>
- ej.web.react.min.js - <http://cdn.syncfusion.com/{{ site.releaseversion }}/js/common/ej.web.react.min.js>

To get started, you can use the `ej.web.all.min.js` file that encapsulates all the `ej` controls and frameworks in one single file.

## HTML

```
<!DOCTYPE html>
<html>
<head>
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<meta name="description" content="Essential Studio for React JS">
<meta name="author" content="Syncfusion">
<title>Getting Started for Ribbon React JS</title>
<!-- Essential Studio for JavaScript theme reference -->
<link href="http://cdn.syncfusion.com/{{ site.releaseversion }}/js/web/flat-
azure/ej.web.all.min.css" rel="stylesheet" />
<!-- Essential Studio for JavaScript script references -->
<script src="http://cdn.syncfusion.com/js/assets/external/jquery-
3.0.0.min.js"></script>
<script
src="http://cdn.syncfusion.com/js/assets/external/react.min.js"></script>
<script src="http://cdn.syncfusion.com/js/assets/external/react-
dom.min.js"></script>
<script src="http://cdn.syncfusion.com/{{ site.releaseversion
}}/js/web/ej.web.all.min.js"></script>
<script src="http://cdn.syncfusion.com/{{ site.releaseversion
}}/js/common/ej.web.react.min.js"></script>
<!-- Add your custom scripts here -->
</head>
<body>
</body>
</html>
```

**Note:** 1. In production, we highly recommend you to use our [custom script generator](#) to create custom script file with required controls and its dependencies only. Also to reduce the file size further please use [GZip compression](#) in your server.

2. For themes, you can use the `ej.web.all.min.css` CDN link from the code snippet given. To add the themes in your application, please refer to [this link](#).



### Control Initialization

Control can be initialized in two ways.

- Using jsx Template
- Without using jsx Template

### Using jsx Template

By using the jsx template, we can create the html file and jsx file. The .jsx file can be convert to .js file and it can be referred in html page.

### Initialize Diagram

Add a `div` container to render the Diagram.

#### HTML

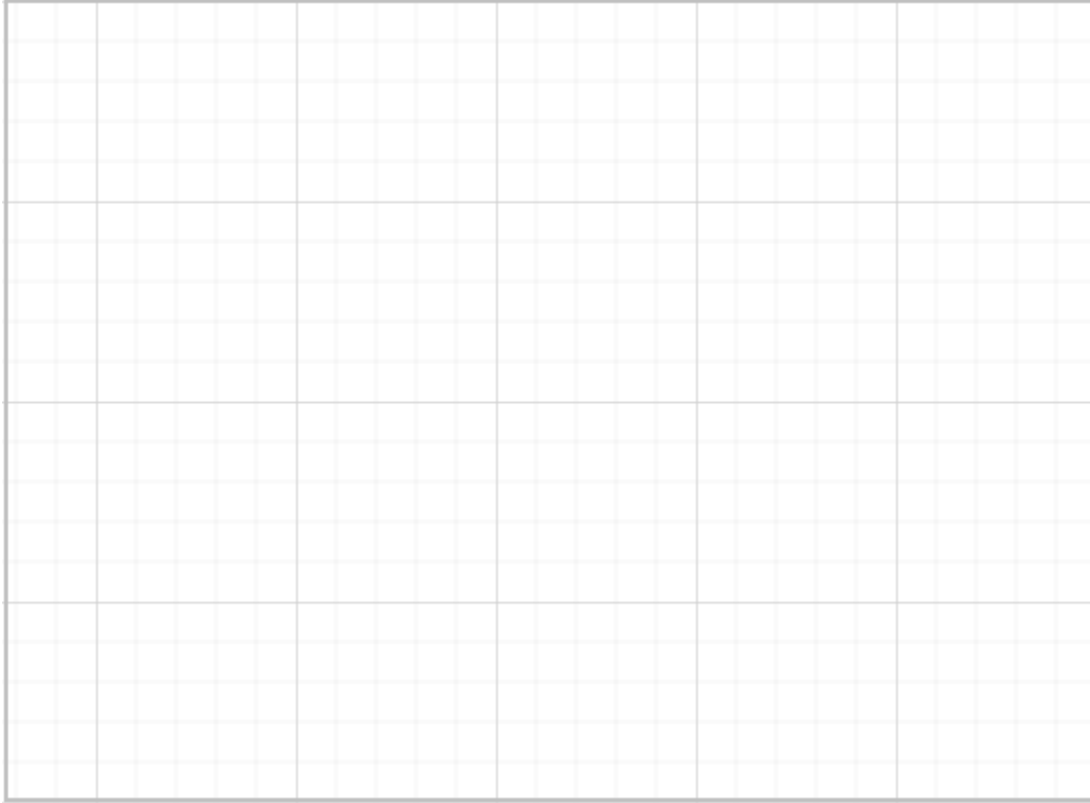
```
<!DOCTYPE html>
<html>
<body>
<div id="diagram-default" style="height:99%;"></div>
<script src="app/diagram/default.js"></script>
</body>
</html>
```

Initialize the Diagram by using the `EJ.Diagram` tag. The Diagram is rendered based on default `width` and `height`. You can also customize the Diagram dimension by setting the `width` and `height` attribute in `scrollSettings`.

#### JAVASCRIPT

```
"use strict";
ReactDOM.render(
  <div className="default">
    <EJ.Diagram id="diagram1" width="100%" height="100%" ></EJ.Diagram>,
  </div>,
  document.getElementById('diagram-default')
);
```

This creates an empty diagram as shown in image



#### *Populate Diagram with nodes and connectors*

Now, this section explains how to populate JSON data to the Diagram.

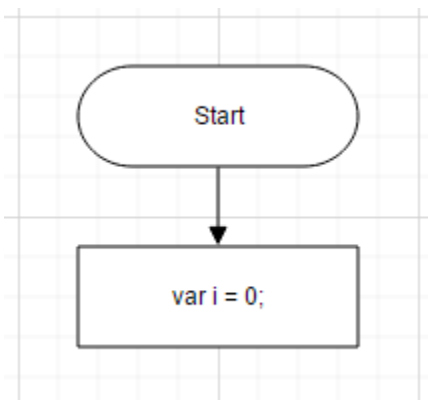
#### **JAVASCRIPT**

```
<script type="text/babel">
var def_nodes = [{
  // Unique name for the node
  name: "Start",
  // Position of the node
  offsetX: 300,
  offsetY: 50,
  // Size of the node
  width: 140,
  height: 50,
  // Text(label) added to the node
  labels: [{
    text: "Start"
  }],
  // Shape for the node
  type: "flow",
  shape: "terminator"
},
{
  name: "Init",
  offsetX: 300,
  offsetY: 140,
  width: 140,
  height: 50,
  labels: [{
```

```

text: "var i = 0;"
}],
type: "flow",
shape: "process"
}
];
var def_connectors = [{
// Unique name for the connector
name: "connector1",
// Source and Target node's name to which connector needs to be connected.
sourceNode: "Start",
targetNode: "Init",
// An empty orthogonal segment
segments: [{ type: "orthogonal" }]
}];
<!DOCTYPE html>
<html>
<body>
<script type="text/babel">
ReactDOM.render(
<div className="default">
<EJ.Diagram id="diagram1" width="100%" height="100%" nodes: {def_nodes},
connectors: {def_connectors}, </EJ.Diagram>,
</div>,
document.getElementById('diagram-default')
);
</script>
</body>
</html>

```



*Business object (Employee information)*

- Define Employee Information as JSON data. The following code example shows an employee array whose,
- Name is used as a unique identifier and
- ReportingPerson is used to identify the person to whom an employee report to, in the organization.

### JAVASCRIPT

```
var data = [
```

```
{ Name: "Elizabeth", Role: "Director" },
{ Name: "Christina", ReportingPerson: "Elizabeth", Role: "Manager" },
{ Name: "Yoshi", ReportingPerson: "Christina", Role: "Lead" },
{ Name: "Philip", ReportingPerson: "Christina", Role: "Lead" },
{ Name: "Yang", ReportingPerson: "Elizabeth", Role: "Manager" },
{ Name: "Roland", ReportingPerson: "Yang", Role: "Lead" },
{ Name: "Yvonne", ReportingPerson: "Yang", Role: "Lead" }
];
```

### Without using jsx Template

The Diagram can be created from a HTML **DIV** element with the HTML **id** attribute set to it. Refer to the following code example.

- You can configure data mapping using "Employee Information" with Diagram, so that the node and connector are automatically generated using mapping properties. DefaultSettings can define the default appearance of node and connector.
- The NodeTemplate is used to update each node based on employee data.
- The following code examples show how dataSourceSetting is used to map id and parent with property name identifiers for employee information.

### HTML

```
<div id="diagram-organizationalchart" style="height:99%;">
</div>
```

### JAVASCRIPT

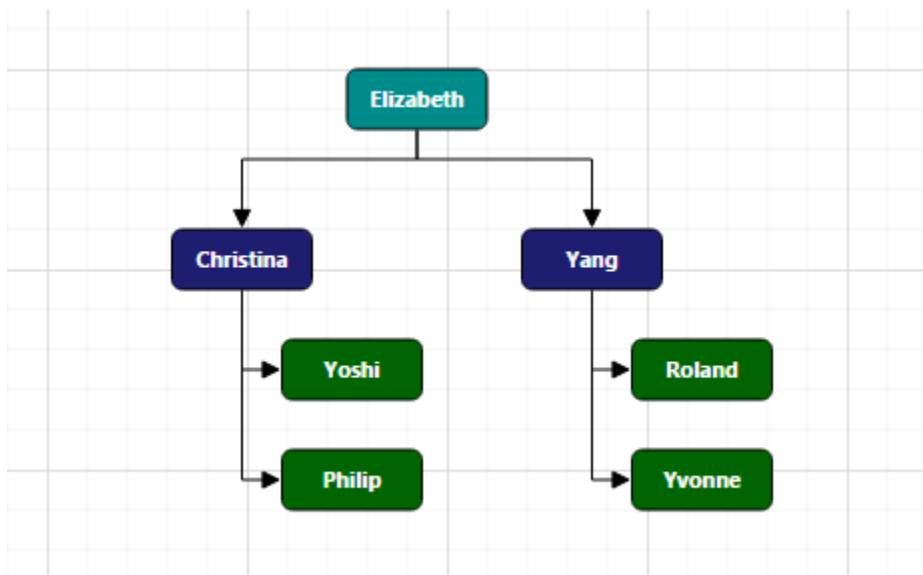
```
//To represent the roles
var codes = {
  Director: "rgb(0, 139,139)",
  Manager: "rgb(30, 30,113)",
  Lead: "rgb(0, 100,0)"
}
//Binds custom data with node
function nodeTemplate(diagram, node) {
  node.labels[0].text = node.Name;
  node.fillColor = codes[node.Role];
}
var layout= { type: "organizationalchart", orientation: "toptobottom",
horizontalSpacing: 25, verticalSpacing: 35, marginX: 3, marginY: 3};
var defaultSettings= {
  node: { constraints: ej.datavisualization.Diagram.NodeConstraints.Select |
ej.datavisualization.Diagram.NodeConstraints.PointerEvents, width: 100,
height: 40, borderColor: "black", labels: [{ fontColor: "#ffffff" } ] },
  connector: {
    lineColor: "#000000", segments: [{ type: "orthogonal" }], targetDecorator: {
      shape: "none" },
    constraints: ej.datavisualization.Diagram.ConnectorConstraints.None
  }
};
var dataSourceSettings = { id: "Name", parent: "ReportingPerson",
dataSource: data }
function diagram5Create(args)
```

```

{
  var height=$(( "#diagram4" ).height() - $( ".e-box.e-addborderbottom.e-
header" ).height()
  $( "#diagram4" ).ejDiagram({ height : height });
}
React.createElement(EJ.Diagram, {
  id: "diagram5",
  height: "100%",
  width: "100%",
  layout: layout,
  defaultSettings: defaultSettings,
  dataSourceSettings: dataSourceSettings,
  nodeTemplate: nodeTemplate,
  create: diagram5Create
}), document.getElementById('diagram-organizationalchart')

```

- The Employee details are displayed in the Diagram as follows.



## Dialog

### Overview

Our Essential ReactJS Dialog component displays a Dialog window within a web page. The Dialog enables a message to be displayed, such as supplementary content like images and text, and an interactive content like forms.

### Key Features:

- **Modal dialog support:** Displays the content in a modal dialog, disabling interaction with other items on the page
- **AJAX Load:** Load AJAX content in dialog content panel.
- **Drag support:** Drag the Dialog within the page.
- **Customized dialog position:** By default, the dialog is shown in the center of the container. If given a position, the dialog is displayed at the particular position.

- **Keyboard navigation:** Users can interact with the dialog by using the keyboard.

## Getting Started

This section helps you to understand the getting started of the Dialog component with the step-by-step instructions.

### Create a simple Dialog

Refer the common React's Getting Started Documentation to create an application and add necessary scripts and styles for rendering our React JS components.

Create a JSX file for rendering Dialog component using `<EJ.Dialog>` syntax. Add required properties to it in `<EJ.Dialog>` tag element

#### JAVASCRIPT

```
ReactDOM.render(  
  <EJ.Dialog id="basicDialog">  
  </EJ.Dialog>,  
  document.getElementById('Dialog-default')  
);
```

Define an HTML element for adding Dialog in the application and refer the JSX file created with script type "text/babel".

#### HTML

```
<div id="Dialog-default"></div>  
<script type="text/babel" src="sample.jsx">
```

This will render an empty Dialog component on executing.

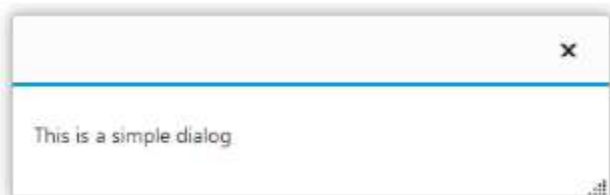
### Add content to Dialog

Add the below code to render Dialog Component with content

#### JAVASCRIPT

```
ReactDOM.render(  
  <EJ.Dialog id="basicDialog">  
    <p>This is a simple dialog</p>  
  </EJ.Dialog>,  
  document.getElementById('Dialog-default')  
);
```

Any content can be added inside of `<EJ.Dialog>` `</EJ.Dialog>` which will be available in Dialog content area on rendering.



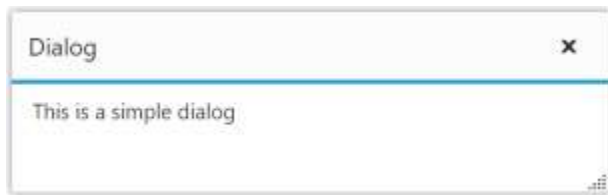
### Set header text

You can set Dialog component header using the **title** property.

#### JAVASCRIPT

```
ReactDOM.render(
  <EJ.Dialog id="basicDialog" title="Dialog">
    <p>This is a simple dialog</p>
  </EJ.Dialog>,
  document.getElementById('Dialog-default')
);
```

Run the above code and your output will be,



### Open Dialog dynamically

In most cases, the Dialog components are needed only in dynamic actions like showing some messages on clicking a button, to provide alert, etc. So the Dialog component provides “open” and “close” methods to open/close the dialogs dynamically.

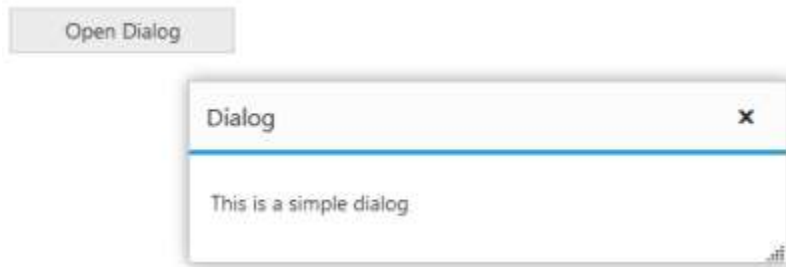
The Dialog Component can be hidden on initialize using **showOnInit** property which should be set to false.

Define click action for the button and the dialog will be opened on clicking the Button component.

#### JAVASCRIPT

```
var DefaultDialog = React.createClass({
  onOpen: function (e) {
    $("#basicDialog").ejDialog("open");
  },
  onDialogClose: function (e) {
    $("#btnOpen").show();
  },
  render: function () {
    return (
      <div id="dialog_default">
        <EJ.Button id="btnOpen" size="medium" type="button" height={30} width={150}
          text="Open Dialog" click={this.onOpen}>
        </EJ.Button>
        <EJ.Dialog id="basicDialog" title="Dialog" showOnInit={false}
          allowDraggable={true} target="#Dialog-default" close={this.onDialogClose}>
          <p>This is a simple dialog</p>
        </EJ.Dialog>
      </div>
    );
  }
});
ReactDOM.render(<DefaultDialog />, document.getElementById('Dialog-
default'));
```

Your output will be,



*Note: You can find the Dialog properties from the [API reference](#) document.*

## DigitalGauge

### Getting Started

- This section encompasses the details on how to configure DigitalGauge. Here you will learn how to provide data for a DigitalGauge and display the data in the required way.
- In addition, you will learn how to customize the default DigitalGauge appearance according to your requirements. As a result, you will get a DigitalGauge that shows it as Digital thermometer.
- You can use this DigitalGauge in advertisements, decorative purposes, displaying share details in share market, game score boards, token systems, etc.



#### Digital Thermometer

##### Adding Script Reference

Create an **HTML** page and add the scripts references in the order mentioned in the following code example.

- [jQuery](#) 1.10.2 and later versions

The required ReactJS script dependencies as follows. And you can also refer [React](#) to know more about react.js.

- react.min.js - <http://cdn.syncfusion.com/js/assets/external/react.min.js>
- react-dom.min.js - <http://cdn.syncfusion.com/js/assets/external/react-dom.min.js>
- ej.web.react.min.js - <http://cdn.syncfusion.com/{{ site.releaseversion }}/js/common/ej.web.react.min.js>

To get started, you can use the `ej.web.all.min.js` file that encapsulates all the `ej` controls and frameworks in one single file.

#### HTML



```

<!DOCTYPE html>
<html>
<head>
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<meta name="description" content="Essential Studio for React JS">
<meta name="author" content="Syncfusion">
<title>Getting Started for Ribbon React JS</title>
<!-- Essential Studio for JavaScript theme reference -->
<link href="http://cdn.syncfusion.com/{{ site.releaseversion }}/js/web/flat-
azure/ej.web.all.min.css" rel="stylesheet" />
<!-- Essential Studio for JavaScript script references -->
<script src="http://cdn.syncfusion.com/js/assets/external/jquery-
3.0.0.min.js"></script>
<script
src="http://cdn.syncfusion.com/js/assets/external/react.min.js"></script>
<script src="http://cdn.syncfusion.com/js/assets/external/react-
dom.min.js"></script>
<script src="http://cdn.syncfusion.com/{{ site.releaseversion
}}/js/web/ej.web.all.min.js"></script>
<script src="http://cdn.syncfusion.com/{{ site.releaseversion
}}/js/common/ej.web.react.min.js"></script>
<!-- Add your custom scripts here -->
</head>
<body>
</body>
</html>

```

**Note:** 1. In production, we highly recommend you to use our [custom script generator](#) to create custom script file with required controls and its dependencies only. Also to reduce the file size further please use [GZip compression](#) in your server.

2. For themes, you can use the `ej.web.all.min.css` CDN link from the code snippet given. To add the themes in your application, please refer to [this link](#).

### Control Initialization

Control can be initialized in two ways.

- Using jsx Template
- Without using jsx Template

### Using jsx Template

By using the jsx template, we can create the html file and jsx file. The `.jsx` file can be convert to `.js` file and it can be referred in html page.

#### Create a Digital Gauge

1. Create a

tag.

### HTML

```

<!DOCTYPE html>
<html>
<body>
<div id="digitalgauge-default" ></div>

```

```
<script src="app/digitalgauge/default.js"></script>
</body>
</html>
```

2. Initialize the DigitalGauge in ts file by using the `EJ.DigitalGauge` tag

#### JAVASCRIPT

```
"use strict";
ReactDOM.render(
  <div className="default">
    <EJ.DigitalGauge id="digitalgauge1"></EJ.DigitalGauge>,
  </div>,
  document.getElementById('digitalgauge-default')
);
```

Run the above code example and you will get a default Digital Gauge as follows.

text

Digital Gauge

#### Set Height and Width values

Basic attributes of each canvas elements are height and width. You can set the height and width of the gauge.

#### JAVASCRIPT

```
<script type="text/babel">
<!DOCTYPE html>
<html>
<body>
<script type="text/babel">
ReactDOM.render(
  <div className="default">
    <EJ.DigitalGauge id="digitalgauge1" height={145}
    width={260}></EJ.DigitalGauge>,
  </div>,
  document.getElementById('digitalgauge-default')
);
</script>
</body>
</html>
```

Run the above code example and you will see a default gauge with the specified height and width values.

text

Digital Gauge with Height and Width

### Set Items Property

Items have different properties to customize the Digital Gauge.

#### *Add Segment and Character Properties*

- In the Welcome Board, the text color must be attentive in nature. You can give some segment properties such as segment spacing, segment width, segment color, segment length and segment opacity.
- Character type is to define the Digital representation of the character. The five types of character representation available are,
  1. EightCrossEightDotMatrix
  2. SevenSegment
  3. FourteenSegment
  4. SixteenSegment
  5. EightCrossEightSquareMatrix.

### JAVASCRIPT

```
<script type="text/babel">
var items=[{
segmentSettings: { width: 2, length: 20 },
characterSettings: { type: "sevensegment", spacing: 12, },
value: "102",
}];
<!DOCTYPE html>
<html>
<body>
<script type="text/babel">
ReactDOM.render(
<div className="default">
<EJ.DigitalGauge id="digitalgauge1" height={145} width={260}
items={items}></EJ.DigitalGauge>,
</div>,
document.getElementById('digitalgauge-default')
);
</script>
</body>
</html>
```

Run the above code example and you will see the following output.



### Digital Gauge Segment Properties

#### Add Background Image

- Add a  
  
element to set the background for the Digital Gauge.

- Add a style tag in the View page to add the background image for the Digital Gauge.
- Add the required properties to show the background image such as position, margin, display, etc.,

### HTML

```
<div id="frameDiv">
<div id="DigitalGauge" style="width:100%"></div>
</div>
<style>
#frameDiv {
align : center;
position : relative;
margin : 0px auto;
display :table;
background-image :url("script/frame.png");
background-repeat :no-repeat;
}
</style>
```

Run the above code example and you will see the following output.



Digital Gauge Background Image

### Add Location

The Location property is used to position the digital letters inside the canvas element.

### JAVASCRIPT

```
var items=[{
//For Displaying Fahrenheit value
segmentSettings: { width: 2, length: 20 },
characterSettings: { type: "sevensegment", spacing: 12, },
value: "102", position: { x: 15, y: 40 }
}]
});
<!DOCTYPE html>
<html>
<body>
<script type="text/babel">
ReactDOM.render(
<div className="default">
<EJ.DigitalGauge id="digitalgaugel" height={145} width={260}
items={items}></EJ.DigitalGauge>,
</div>,
document.getElementById('digitalgauge-default')
);
</script>
</body>
```

```
</html>
```

Run the above code example and you will see the following output.



Digital Gauge with Segment Location

#### Add Items Collection

You can further add the Items Collection to display the temperature value like Digital Thermometer.

#### JAVASCRIPT

```
var items=[{
  //For Displaying Fahrenheit value
  segmentSettings: { width: 2, length: 20, spacing: 0 },
  characterSettings: { type: "sevensegment", spacing: 12, },
  value: "102",
  position: { x: 15, y: 40 }
},
{
  //For displaying degree symbol
  segmentSettings: { width: 2, length: 5, spacing: 0 },
  characterSettings: { type: "sevensegment", spacing: 5, },
  value: "°",
  position: { x: 70, y: 28 }
},
{
  //For displaying Fahrenheit symbol
  segmentSettings: { width: 2, length: 20, spacing: 0 },
  characterSettings: { type: "sevensegment", spacing: 12, },
  value: "F",
  position: { x: 170, y: 40 }
},
{
  //For displaying Celcius value
  segmentSettings: { width: 1, length: 9, spacing: 0, color: "#F5b43f" },
  characterSettings: { type: "sevensegment", spacing: 12, },
  value: "38",
  position: { x: 70, y: 90 },
},
{
  //For displaying degree symbol
  segmentSettings: { width: 1, length: 3, spacing: 0, color: "#F5b43f" },
  characterSettings: { type: "sevensegment", spacing: 12, },
  value: "°",
  position: { x: 90, y: 80 }
},
{
  //For displaying celcius symbol
  segmentSettings: { width: 1, length: 9, spacing: 0, color: "#F5b43f" },
  characterSettings: { type: "sevensegment", spacing: 12, },
```

```

value: "c",
position: { x: 120, y: 90 }
}];
<!DOCTYPE html>
<html>
<body>
<script type="text/babel">
ReactDOM.render(
<div className="default">
<EJ.DigitalGauge id="digitalgauge1" height={145} width={260}
items={items}></EJ.DigitalGauge>,
</div>,
document.getElementById('digitalgauge-default')
);
</script>
</body>
</html>

```

Run the above code example and you will see the following output.



Digital Gauge with Item Collection

#### Without using jsx Template

The Digital Gauge can be created from a HTML **DIV** element with the HTML **id** attribute set to it. Refer to the following code example.

#### HTML

```
<div id="digitalgauge-default"></div>
```

#### JAVASCRIPT

```

<script type="text/babel">
var items=[{
segmentSettings: { width: 2, length: 20 },
characterSettings: { type: "sevensegment", spacing: 12, },
value: "102",
}];
ReactDOM.render(
React.createElement(EJ.DigitalGauge, {id: "digitalgauge",
height: 145,
width:260,
items: items
}
),
document.getElementById('digitalgauge-default')
);
</script>

```

Run the above code example and you will see the following output.



## Basic Settings

### Height and Width Customization

The basic customization for any control is to set the dimension. Here dimension refers to two major attributes such as **height** and **width**. The **height** and **width** assigned in the control will render the canvas element in the given size. The code example to set **height** and **width** is as follow.

#### JAVASCRIPT

```
"use strict";
ReactDOM.render(
  <ej.digitalGauge id="digitalgaugel1" height = {200} width = {500} value
  ="Syncfusion" >
</ej.digitalGauge>,
  document.getElementById('digitalgauge')
);
```

Execute the above code examples to render the **DigitalGauge** as follows.



### Responsive Layout

- For any display devices, the control will be rendered based on the space available in that device. For this purpose, **resizing** property is given to the **Digital Gauge** control. The **Digital Gauge** renders with a given value.
- When the browser resize the canvas element checks the dimension with its parent element. If there are any changes in parent dimension, **Gauge** control will changes the dimension based on its parent element change. This feature is enabled by using the property **isResponsive**.

#### JAVASCRIPT

```
"use strict";
ReactDOM.render(
  <ej.digitalGauge id="digitalgaugel1" isResponsive = {true} width = {800}
  ></ej.digitalGauge>,
  document.getElementById('digitalgauge')
);
```

Execute the above code examples to render the **DigitalGauge** as follows.



### Themes

**Themes** give the good appearance to the control. There are two types of **Themes** available for **DigitalGauge** as follows

- flatlight

- flatdark

### JAVASCRIPT

```
"use strict";
ReactDOM.render(
  <ej.digitalgauge id="digitalgauge1" isResponsive = {true} width = {800}
  themes = "flatdark" value = "LOS ANGELS 40 KM"></ej.digitalgauge>,
  document.getElementById('digitalgauge')
);
```

Execute the above code examples to render the **DigitalGauge** as follows.



### Frames

#### Inner and Outer Width Customization

**Frames** are space that enclose the **Digital Gauge**. The inner width of the **Frame** is the distance between the canvas element and the frame. The outer width is the distance from the frame. The code example to set frame's **innerWidth** and **outerWidth** is as follow.

### JAVASCRIPT

```
"use strict";
var frame = {
  // For setting inner width
  innerWidth: 6,
  // For setting outer width
  outerWidth: 10,
};
ReactDOM.render(
  <ej.digitalgauge id="digitalgauge1" value = "WELCOME" frame =
  {frame}></ej.digitalgauge >,
  document.getElementById('digitalgauge')
);
```

Execute the above code examples to render the **DigitalGauge** as follows.



#### Setting Background Image

For a better appearance, you can set the **backgroundimage** for the **Digital Gauge** using the property **backgroundImageUrl**.

### JAVASCRIPT

```
"use strict";
var frame= {
  // For setting background image
  backgroundImageUrl: "board3.jpg"
};
var items= [{
  position: {
    x: 95,
    y: 10
```



```

}
}];
//For Digital Gauge rendering
ReactDOM.render(
<EJ.DigitalGauge id="digitalgauge"
//For setting text
value="RADAR" frame={frame} height={300} items={items}
>
</EJ.DigitalGauge>,
document.getElementById('digitalgauge')
);

```

Execute the above code examples to render the **DigitalGauge** as follows.



## Digital Elements

### Text Customization

- The attribute **value** refers the text displayed in the **Digital Gauge**. This text is applicable only for that item instead of all items. Text color is changed by using the property **textColor**.
- It is possible to align the text inside the **Digital Gauge** control by using the property **textAlign**. Two possible values for text align are as follows
  - left
  - right

### HTML

```
<div id="DigitalGauge1"></div>
```

### HTML

```

"use strict";
var items = [{
// For setting alignment
textAlign: "right",
// For setting text
value: "STOP",
}];
ReactDOM.render(
<ej.digitalgauge id="digitalgauge1" items = {items} ></ej.digitalgauge >,
document.getElementById('DigitalGauge1')
);

```

Execute the above code examples to render the **DigitalGauge** as follows.



## Segment Settings

### Appearance

- **Digital Gauge** consists of several digital segments. Segment is customized with some properties. Color of the segment is set by using **color** property. Color is either given as string or hexadecimal value.
- You can add gradient effects to the segments with the help of **gradient** attribute. The **opacity** of the segment is also adjustable. The space between two segments are adjusted with **spacing** property.

### HTML

```
"use strict";
<div id="DigitalGauge1"></div>
```

### JAVASCRIPT

```
"use strict";
var items = [{
  // For setting text
  value: "GO AHEAD",
  segmentSettings: {
    // For setting segment color
    color: "Green",
    // For setting segment opacity
    opacity: 0.1,
    // For setting segment spacing
    spacing: 4,
  }
}];
//For Digital Gauge rendering
ReactDOM.render(
  <EJ.DigitalGauge id="digitalgauge"
  items={items} width={800}
  >
</EJ.DigitalGauge>,
  document.getElementById('DigitalGauge1')
);
```

Execute the above code examples to render the **DigitalGauge** as follows.



### Dimension Modification

- **Digital Gauge** consists of several digital segments. Segment is customized with some properties. Color of the segment is set by using **color** property. Color is either given as string or hexadecimal value.
- You can add gradient effects to the segments with the help of **gradient** attribute. The **opacity** of the segment is also adjustable. The space between two segments are adjusted with **spacing** property.

**HTML**

```
<div id="DigitalGauge1"></div>
```

**JAVASCRIPT**

```
"use strict";
var items = [{
  // For setting text
  value: "WELCOME",
  segmentSettings: {
    // For setting segment length
    length: 3,
    // For setting segment width
    width: 3
  }
}];
//For Digital Gauge rendering
ReactDOM.render(
  <EJ.DigitalGauge id="digitalgauge"
  items={items} width={800}
  >
</EJ.DigitalGauge>,
  document.getElementById('DigitalGauge1')
);
```

Execute the above code examples to render the **DigitalGauge** as follows.



**Character Settings****Appearance**

The opacity of the character is adjustable with the help of **opacity** property. The space between two characters are adjusted with **spacing** property as like in the segment settings.

**HTML**

```
<div id="DigitalGauge1"></div>
```

**JAVASCRIPT**

```
"use strict";
var items = [{
  // For setting text
  // For setting text
  value: " Syncfusion ",
  characterSettings: {
    // For setting character opacity
    opacity: 0.3,
    // For setting character spacing
    spacing: 3
  }
}];
ReactDOM.render(
```

```
<ej.digitalgauge id="digitalgauge1" width={800} items = {items}
></ej.digitalgauge >,
document.getElementById('DigitalGauge1')
);
```

Execute the above code examples to render the **DigitalGauge** as follows.



### Count and Type

The number of text to be displayed can be limited by the attribute called **count**. In **Digital Gauge** five different types of characters are supported. They are as follows,

- EightCrossEightDotMatrix
- SevenSegment
- FourteenSegment
- SixteenSegment
- EightCrossEightSquareMatrix.

### HTML

```
<div id="DigitalGauge1"></div>
```

### JAVASCRIPT

```
"use strict";
var items = [{
  // For setting text
  value: "1234567890",
  segmentSettings: {
    // For setting segment length
    length: 8,
    // For setting segment width
    width: 1
  },
  characterSettings: {
    // For setting character count
    count: 10,
    // For setting segment spacing
    spacing: 10,
    // For setting character type
    type: "sevensegment",
  }
}];
ReactDOM.render(
  <ej.digitalgauge id="digitalgauge1" width={800} items = {items}
  ></ej.digitalgauge>,
  document.getElementById('DigitalGauge1')
);
```

Execute the above code examples to render the **DigitalGauge** as follows.



### Text Positioning

The text in the **DigitalGauge** is positioned with position object. This object contains two attributes such as **x** and **y**. The **x** variable positions the text in the horizontal axis and the **y** variable positions the text in the vertical axis.

#### HTML

```
<div id="DigitalGauge1"></div>
```

#### JAVASCRIPT

```
"use strict";
var items=[{
  // For setting text
  value: "YELLOW",
  // For setting segment color
  segmentSettings: { color: "Yellow" },
  position:{
    // For setting segment x location
    x:80,
    // For setting segment y location
    y:10
  }
}];
var frame={
  backgroundImageUrl: "Board1.jpg"
}
//For Digital Gauge rendering
ReactDOM.render(
  <EJ.DigitalGauge id="digitalgauge"
  items={items} width={800} height={300} frame={frame}
  >
  </EJ.DigitalGauge>,
  document.getElementById('DigitalGauge1')
);
```

Execute the above code examples to render the **DigitalGauge** as follows.



### Shadow Effects

The text in the **Digital Gauge** is positioned with position object. This object contains two attributes such as **x** and **y**. The **x** variable positions the text in the horizontal axis and **y** variable positions the text in the vertical axis.

#### HTML

```
<div id="DigitalGauge1"></div>
```

#### JAVASCRIPT

```
"use strict";
var items = [{
  //For setting Text
  value: "WELCOME",
```

```
//For setting segment length and width
segmentSettings: {
length: 3,
width: 3
},
//For setting shadow color
shadowColor: "yellow",
//For setting shadow Blur
shadowBlur: 20,
//For setting horizontal offset
shadowOffsetX: 15,
//For setting vertical offset
shadowOffsetY: 15,
}];
ReactDOM.render(
<ej.digitalgauge id="digitalgauge1" width={800} items =
{items}></ej.digitalgauge>,
document.getElementById('DigitalGauge1')
);
```

Execute the above code examples to render the **DigitalGauge** as follows.

![[/js/DigitalGauge/Character-Settingsimages/Character-Settingsimg4.png]

### Font Customization

You can customize the **font** of the text as per your requirement. To customize the font, you have to set **enableCustomFont**. Following font customization options are available.

**Font-family**- used to set the font-family of the text.

**Font-style**- used to set the font-style of the text.

**Font-size**- used to set the font-size of the text.

### JAVASCRIPT

```
"use strict";
var items = [{
//For setting Text
value: "WELCOME",
//For setting segment length and width
segmentSettings: {
length: 3,
width: 3
},
font:{
fontFamily:'Arial',
fontStyle:'Italic',
size:18,
opacity:0.5
}
}];
ReactDOM.render(
<ej.digitalgauge id="digitalgauge1" width={800} items =
{items}></ej.digitalgauge>,
document.getElementById('DigitalGauge1')
);
```

## Multiple Items

The text in the **Digital Gauge** is positioned with position object. This object contains two attributes such as **x** and **y**. The **x** variable positions the text in the horizontal axis and **y** variable positions the text in the vertical axis.

### HTML

```
<div id="DigitalGauge1"></div>
```

### JAVASCRIPT

```
"use strict";
var items = [
  // For Item 1
  {
    // For setting text
    value: "BLUE",
    segmentSettings: {
      color: "blue"
    },
    position: {
      x: 90,
      y: 0
    }
  },
  // For Item 2
  {
    // For setting text
    value: "RED",
    segmentSettings: {
      color: "red"
    },
    position: {
      x: 90,
      y: 50
    }
  },
  // For Item 3
  {
    // For setting text
    value: "PINK",
    segmentSettings: {
      color: "pink"
    },
    position: {
      x: 90,
      y: 100
    }
  }
];
var frame = {
  backgroundImageUrl: "Board1.jpg"
};
ReactDOM.render(
```

```
<ej.digitalgauge id="digitalgauge1" width={1350} height={400} items =
{items} frame = {frame} >
</ej.digitalgauge >,
document.getElementById('DigitalGauge1')
);
```

Execute the above code example to render the **DigitalGauge** as follows.

![[/js/DigitalGauge/Multiple-Itemsimages/Multiple-Itemsimg1.png]

## Exporting the Digital Gauge

**Digital Gauge** has an exporting feature where **Gauge** control is converted into image format and then exported to client-side. The method API **exportImage** exports the **Digital Gauge**. It has two arguments such as **filename** and **file format**. For exporting, you can refer the following code example.

### HTML

```
<div id="DigitalGauge1"></div>
<button id="btnSubmit">Export</button>
<div id=" fileName ">FileName </div>
<div id=" fileFormat ">FileFormat </div>
<select id="fileFormat">
<option value="JPEG">JPEG</option>
<option value="PNG">PNG</option>
</select>
```

### JAVASCRIPT

```
"use strict"
ReactDOM.render(
<EJ.DigitalGauge id="default" value="Syncfusion">
</EJ.DigitalGauge>,
document.getElementById('DigitalGauge1')
);
ReactDOM.render(
<EJ.Button id="btnExportImage" width={100} click={buttonclickevent}>
</EJ.Button>
)
function buttonclickevent() {
var FileName = $("#txtFileName").val();
var FileFormat = $("#ddlFileType").val();
var flag = $("#default").ejDigitalGauge("exportImage", FileName,
FileFormat);
if(!flag)
alert("Sorry for the inconvenience. Export is currently not supported in
Internet Explorer 9 and below version");
}
```

Execute the above code examples to render the **DigitalGauge** as follows.

![[/js/DigitalGauge/Exporting-the-Digital-Gaugeimages/Exporting-the-Digital-Gaugeimg1.png]

## Methods

### destroy()

To destroy the digital gauge



**HTML**

```
<div id="digital"></div>
```

**JAVASCRIPT**

```
ReactDOM.render(  
  <EJ.DigitalGauge id="default"></EJ.DigitalGauge>,  
  document.getElementById('digital')  
)  
;  
function DigitalGaugeMethod(){  
  var digitalObj = $("#default").data("ejDigitalGauge");  
  digitalObj.destroy();  
};
```

*exportImage(fileName, fileType)*

To export Digital Gauge as Image

**HTML**

```
<div id="digital"></div>
```

**JAVASCRIPT**

```
ReactDOM.render(  
  <EJ.DigitalGauge id="default"></EJ.DigitalGauge>,  
  document.getElementById('digital')  
)  
;  
function DigitalGaugeMethod(){  
  var digitalObj = $("#default").data("ejDigitalGauge");  
  digitalObj.exportImage();  
};
```

*getPosition(itemIndex)*

Gets the location of an item that is displayed on the gauge.

**HTML**

```
<div id="digital"></div>
```

**JAVASCRIPT**

```
ReactDOM.render(  
  <EJ.DigitalGauge id="default"></EJ.DigitalGauge>,  
  document.getElementById('digital')  
)  
;  
function DigitalGaugeMethod(){  
  var digitalObj = $("#default").data("ejDigitalGauge");  
  digitalObj.getPosition();  
};
```

*getValue(itemIndex)*

ClientSideMethod getValue Gets the value of an item that is displayed on the gauge

**HTML**

```
<div id="digital"></div>
```

**JAVASCRIPT**

```
ReactDOM.render(  
  <EJ.DigitalGauge id="default"></EJ.DigitalGauge>,  
  document.getElementById('digital')  
)  
;  
function DigitalGaugeMethod(){  
  var digitalObj = $("#default").data("ejDigitalGauge");  
  digitalObj.getValue();  
};
```

*refresh()*

Refresh the digital gauge widget

**HTML**

```
<div id="digital"></div>
```

**JAVASCRIPT**

```
ReactDOM.render(  
  <EJ.DigitalGauge id="default"></EJ.DigitalGauge>,  
  document.getElementById('digital')  
)  
;  
function DigitalGaugeMethod(){  
  var digitalObj = $("#default").data("ejDigitalGauge");  
  digitalObj.refresh();  
};
```

*setPosition(itemIndex, value)*

ClientSideMethod Set Position Sets the location of an item to be displayed in the gauge

**HTML**

```
<div id="digital"></div>
```

**JAVASCRIPT**

```
ReactDOM.render(  
  <EJ.DigitalGauge id="default"></EJ.DigitalGauge>,  
  document.getElementById('digital')  
)  
;  
function DigitalGaugeMethod(){  
  var digitalObj = $("#default").data("ejDigitalGauge");  
  digitalObj.setPosition();  
};
```

*setValue(itemIndex, value)*

ClientSideMethod SetValue Sets the value of an item to be displayed in the gauge.

**HTML**

```
<div id="digital"></div>
```

**JAVASCRIPT**

```
ReactDOM.render(  
  <EJ.DigitalGauge id="default"></EJ.DigitalGauge>,  
  document.getElementById('digital')  
);  
function DigitalGaugeMethod(){  
  var digitalObj = $("#default").data("ejDigitalGauge");  
  digitalObj.setValue();  
};
```

**Events***init*

Triggers when the gauge is initialized.

**HTML**

```
<div id="digital"></div>
```

**JAVASCRIPT**

```
ReactDOM.render(  
  <EJ.DigitalGauge id="default" init = {Init}></EJ.DigitalGauge>,  
  document.getElementById('digital')  
);  
function Init(){  
  // Do Something  
};
```

*itemRendering*

Triggers when the gauge item rendering.

**HTML**

```
<div id="digital"></div>
```

**JAVASCRIPT**

```
ReactDOM.render(  
  <EJ.DigitalGauge id="default" itemRendering =  
    {ItemRendering}></EJ.DigitalGauge>,  
  document.getElementById('digital')  
);  
function ItemRendering(){  
  // Do Something  
};
```

*load*

Triggers when the gauge is start to load.

**HTML**

```
<div id="digital"></div>
```

**JAVASCRIPT**

```
ReactDOM.render(
  <EJ.DigitalGauge id="default" load = {Load}></EJ.DigitalGauge>,
  document.getElementById('digital')
);
function Load() {
  // Do Something
};
```

*renderComplete*

Triggers when the gauge render is completed.

**HTML**

```
<div id="digital"></div>
```

**JAVASCRIPT**

```
ReactDOM.render(
  <EJ.DigitalGauge id="default" renderComplete =
  {RenderComplete}></EJ.DigitalGauge>,
  document.getElementById('digital')
);
function RenderComplete() {
  // Do Something
};
```

## DropDownList

### DropDownList

The DropDownList widget displays a single column list of items which enables you to select single or multiple items from the list. By default no selection is been made in the widget, the user has to navigate through the items using mouse or keyboard actions to select an item.

#### Key Features

- **DataSources** - Supports various client-side and remote data sources such as JSON, RESTful services, OData services, WCF services and much more.
- **Sorting** - Sorts both ascending and descending orders.
- **Grouping** - Categorizes the list Items.
- **Searching** - Provides both incremental and Filter Search.
- **Virtual scrolling** - Fetches the data from remote on demand.
- **PopupList Resize** - Resizes the popup list in the needed dimensions.
- **Template** - Designs own layout of list items.
- **Cascading** - Cascades the data sources with multiple DropDownList.
- **State persistence** - Persistent state of properties on page refresh.

- **Responsive** - Suits responsive layouts.
- **Validation** - Built in jQuery validation.
- **Localization** - Supports localization to different cultures.
- **Selection** - Supports single or multi selection with the DropDownList and display the text in two modes, delimiter and visual mode.
- **Accessibility** - Supports keyboard and ARIA accessibility.

## Getting Started

The external script dependencies of the DropDownList widget are,

- [jQuery 1.7.1](#) and later versions.
- [jQuery.easing](#) - to support the animation effects.

And the internal script dependencies of the DropDownList widget are:

File	Description / Usage
ej.core.min.js	Must be referred always before using all the JS controls.
ej.data.min.js	Used to handle data operation and should be used while binding data to JS controls.
ej.dropdownlist.min.js	The dropdownlist's main file
ej.checkbox.min.js	Should be referred when using checkbox functionalities in DropDownList.
ej.scroller.min.js	Should be referred when using scrolling in DropDownList.
ej.draggable.min.js	Should be referred when using popup resize functionality in DropDownList.

For getting started you can use the 'ej.web.all.min.js' file, which encapsulates all the 'ej' controls and frameworks in one single file.

For themes, you can use the 'ej.web.all.min.css' CDN link from the snippet given. To add the themes in your application, please refer [this link](#).

## Creating DropDownList in React JS

The DropDownList can be created from a HTML 'select' element with the HTML 'id' attribute and pre-defined options set to it. Use the below given code to create a DropDownList in React JS.

You can create a React application and add necessary scripts and styles with the help of the given [React Getting Started Documentation](#).

Define an HTML element for adding DropDownList in the application and refer the JSX file.

### HTML

```
<div id="dropdownlist-default"></div>
<script src="app/dropdownlist/default.js"></script>
```

Create a JSX file for rendering DropDownList component using <EJ.DropDownList> syntax. Add required properties to it in <EJ.DropDownList> tag element

**JAVASCRIPT**

```

"use strict";
var list = [
{ empid: "cr1", text: "ListItem 1", value: "ListItem 1" },
{ empid: "cr2", text: "ListItem 2", value: "ListItem 2" },
{ empid: "cr3", text: "ListItem 3", value: "ListItem 3" },
{ empid: "cr4", text: "ListItem 4", value: "ListItem 4" },
{ empid: "cr5", text: "ListItem 5", value: "ListItem 5" },
];
ReactDOM.render(
<EJ.DropDownList dataSource={list} fields-text="text" fields-value="value">
</EJ.DropDownList>,
document.getElementById('dropdownlist-default')
);

```

**Populating data**

The DropDownList can be bounded to both local array and remote data services using [ej.DataManager](#). You can use [DataManager](#) component to serve data from the data services based on the query provided. You can also bind local data using DataManager. To bind data to DropDownList widget, the [dataSource](#) property should be assigned with the instance of 'ej.DataManager'.

**Note:** ODataAdaptor is the default adaptor for DataManager. On binding to other web services, proper [data adaptor](#) needs to be set on 'adaptor' option of DataManager.

**HTML**

```

<div id="dropdownlist-default"></div>
<script src="app/dropdownlist/default.js"></script>

```

**JAVASCRIPT**

```

"use strict";
var customers= [
{ id: "1", text: "ALFKI" }, { id: "2", text: "ANATR" }, { id: "3", text: "ANTON" },
{ id: "4", text: "AROUT" }, { id: "5", text: "BERGS" }, { id: "6", text: "BLAUS" }
];
var dataManager=ej.DataManager(customers);
ReactDOM.render(
<EJ.DropDownList dataSource={dataManager} fields-text="text" >

```

```
</EJ.DropDownList>,
document.getElementById('dropdownlist-default')
);
```



### Setting Dimensions

DropDownList dimensions can be set using width and height API.

#### HTML

```
<div id="dropdownlist-default"></div>
<script src="app/dropdownlist/default.js"></script>
```

#### JAVASCRIPT

```
"use strict";
var list = [
{ empid: "cr1", text: "ListItem 1", value: "ListItem 1" },
{ empid: "cr2", text: "ListItem 2", value: "ListItem 2" },
{ empid: "cr3", text: "ListItem 3", value: "ListItem 3" },
{ empid: "cr4", text: "ListItem 4", value: "ListItem 4" },
{ empid: "cr5", text: "ListItem 5", value: "ListItem 5" },
];
ReactDOM.render(
<EJ.DropDownList dataSource={list} fields-text="text" fields-value='ListItem 1' width="300px" height="40px">
</EJ.DropDownList>,
document.getElementById('dropdownlist-default')
);
```

### Setting dimensions to Popup list

PopupWidth and popupHeight can be used to create a fixed size popup list.

#### HTML

```
<select id="dropdown1" ej-dropdownlist e-datasource="dataList" e-
value="value" e-popupHeight="popupheight" e-popupWidth="popupwidth"/>
```

#### JAVASCRIPT

```
"use strict";
var list = [
```

```

{ empid: "cr1", text: "ListItem 1", value: "ListItem 1" },
{ empid: "cr2", text: "ListItem 2", value: "ListItem 2" },
{ empid: "cr3", text: "ListItem 3", value: "ListItem 3" },
{ empid: "cr4", text: "ListItem 4", value: "ListItem 4" },
{ empid: "cr5", text: "ListItem 5", value: "ListItem 5" },
];
ReactDOM.render(
<EJ.DropDownList dataSource={list} fields-text="text" fields-value='ListItem
1' popupHeight="100px" popupWidth="200px">
</EJ.DropDownList>,
document.getElementById('dropdownlist-default')
);

```

## FileExplorer

### Getting Started

Using the following steps, you can create a React FileExplorer component. The basic rendering of React FileExplorer is achieved with default functionality.

#### Create an FileExplorer

You can create a React application and add necessary scripts and styles with the help of the given [React Getting Started Documentation](#).

Create a JSX file for rendering FileExplorer component using <EJ.FileExplorer> syntax. Add required properties to it in <EJ.FileExplorer> tag element

#### JS

```

var localServ, ajaxDataType;
localServ =
"http://js.syncfusion.com/demos/ejServices/api/FileExplorer/FileOperations"
var browname = /MSIE 8.0/i.test(window.navigator.userAgent) || /MSIE
9.0/i.test(window.navigator.userAgent) ? true : false;
if (browname)
{
localServ =
"http://js.syncfusion.com/demos/ejServices/api/FileExplorer/FileOperations";
ajaxDataType = "json" ;
}
var path =
"http://js.syncfusion.com/demos/ejServices/Content/FileBrowser/";
ReactDOM.render(
<EJ.FileExplorer ajaxDataType={ajaxDataType} ajaxAction={localServ}
isResponsive={true} width="100%" minWidth= "150px" fileTypes= "*.png, *.gif,
*.jpg, *.jpeg, *.docx" layout= "tile" path= {path} >
</EJ.FileExplorer>,
document.getElementById('fileexplorer')
);

```

To perform the server side actions using local web API service, please refer the [link](#).

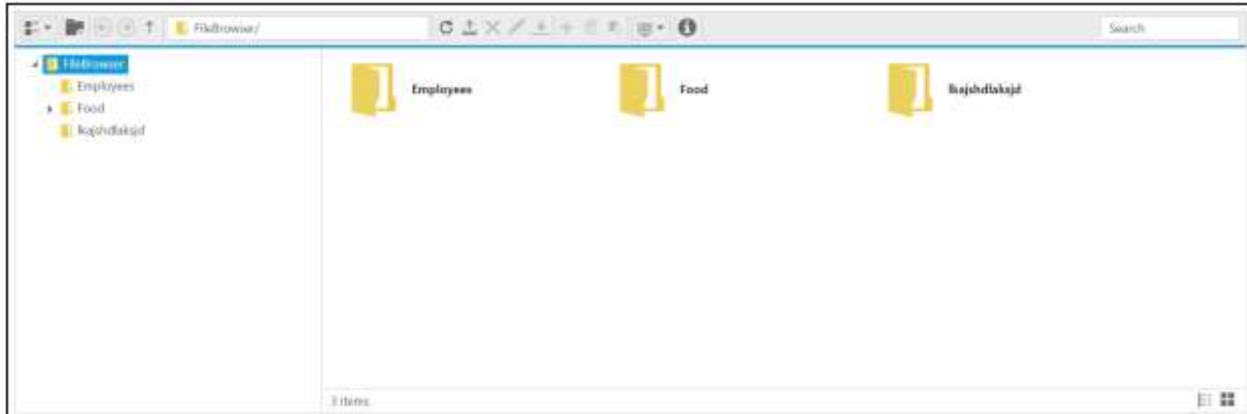
Define an HTML element for adding FileExplorer in the application and refer the JSX file.

#### HTML



```
<div id="fileexplorer"></div>
<script type="text/babel" src="fileexplorer.jsx"></script>
```

Run the above code to render the following output.



**Note:** You can find the FileExplorer properties from the [API reference](#) document

## Grid

### Overview

The Grid control for ReactJS is an efficient display engine for tabular data. It will pull data from a datasource, such as array of JSON objects, OData web services or ej.DataManager; binding data fields to columns and displaying a column header to identify the field. It is a feature-rich control that provides extensive appearance customization options with support for grouped records. This Grid is very useful for generating complex grid-based reports with rich formatting. The most important features available in the Grid control for ReactJS are paging, sorting, filtering, searching, grouping and editing

### Key Features

Some important features of the Grid control are:

- **Data sources** - Bind the Grid control with an array of JSON objects or ej.DataManager.
- **Responsive** - Provides three level of responsive modes which includes redesigned UI for Mobile.
- **Sorting and grouping** - Supports n levels of sorting and grouping.
- **Filtering** - Offers Excel-like filtering for filter data.
- **Editing** - Offers two editing modes for inserting, editing, and deleting records in a grid.
- **Paging** - Provides the option to easily switch between pages using the pager bar.
- **Reordering** - Allows you to drag any column and drop it at any position in the Grid's column header row, allowing columns to be repositioned at the required place.
- **Resize columns** - Grid provides option for resizing the columns.
- **Summary support** - Offers options for specifying summary rows and columns.
- **Detail template** - Offers to render the detail row for the corresponding expanded master row.
- **Unbound columns** - Offers the option to specify unbound columns.
- **RTL support** - Provides a full-fledged right-to-left mode which aligns content in the **Grid** control from right to left.
- **Localization** - Provides inherent support to localize the UI.

## Getting started

### Preparing HTML document

The grid control has the following list of external JavaScript dependencies.

- [jQuery](#) 1.7.1 and later versions
- [jsRender](#) - to render the templates

Refer to the internal dependencies in the following table.

File	Description/Usage
ej.core.min.js	It is referred always before using all the JS controls.
ej.data.min.js	Used to handle data operation and is used while binding data to the JS controls.
ej.touch.min.js	Used to handle touch operations in touch-enabled devices
ej.print.min.js	Used to handle print operation in JS controls.
ej.draggable.min.js	Used for drag and drop an element in JS controls.
ej.grid.min.js	The grid's main file.
ej.pager.min.js	It is referred when paging is used in the Grid.
ej.scroller.min.js	It is referred when scrolling is used in the Grid.
ej.waitingpopup.min.js	It is referred when the remote databinding is used in the Grid. The waiting popup shows while requesting the server for data.
ej.gridresize.min.js	It is referred when resizing is used in the Grid.
ej.dropdownlist.min.js	These files are used while enable the Editing and Filtering feature in the Grid.
ej.dialog.min.js	
ej.button.min.js	
ej.autocomplete.min.js	
ej.datepicker.min.js	
ej.datetimepicker.min.js	
ej.timepicker.min.js	
ej.checkbox.min.js	It is referred when toolbar is enabled in Grid.
ej.editor.min.js	
ej.tooltip.js	
ej.toolbar.min.js	It is referred when excel like filter menu or context menu is enabled.
ej.menu.js	

ej.radiobutton.js	It is referred when filtering is enabled.
ej.excelfilter.js	It is referred when excel like filter menu is enabled.
ej.globalize.min.js	It is referred when using localization in Grid.

The required ReactJS script dependencies as follows. And you can also refer [React](#) to know more about react js.

- react.min.js - <http://cdn.syncfusion.com/js/assets/external/react.min.js>
- react-dom.min.js - <http://cdn.syncfusion.com/js/assets/external/react-dom.min.js>
- browser.min.js - <http://cdnjs.cloudflare.com/ajax/libs/babel-core/5.8.34/browser.min.js>
- ej.web.react.min.js - <http://cdn.syncfusion.com/{{ site.releaseversion }}/js/common/ej.web.react.min.js>

To get started, you can use the `ej.web.all.min.js` file that encapsulates all the `ej` controls and frameworks in one single file. So the complete boilerplate code is

### HTML

```
<!DOCTYPE html>
<html>
<head>
<meta name="viewport" content="width=device-width, initial-scale=1.0"/>
<meta name="description" content="Essential Studio for React JS"/>
<meta name="author" content="Syncfusion"/>
<title>Getting started for Grid React JS</title>
<!-- Essential Studio for React JS theme reference -->
<link rel="stylesheet"
href="http://cdn.syncfusion.com/13.2.0.29/js/web/flat-
azure/ej.web.all.min.css" />
<!-- Essential Studio for React JS script references -->
<script
src="http://cdn.syncfusion.com/js/assets/external/react.min.js"></script>
<script src="http://cdn.syncfusion.com/js/assets/external/react-
dom.min.js"></script>
<script src="https://cdnjs.cloudflare.com/ajax/libs/babel-
core/5.8.34/browser.min.js"></script>
<script src="https://code.jquery.com/jquery-1.10.2.min.js"></script>
<script
src="http://cdn.syncfusion.com/js/assets/external/jsrender.min.js"></script>
<script
src="http://cdn.syncfusion.com/13.2.0.29/js/web/ej.web.all.min.js"></script>
<script src="http://cdn.syncfusion.com/{{ site.releaseversion
}}/js/common/ej.web.react.min.js"></script>
<!-- Add your custom scripts here -->
</head>
<body>
</body>
</html>
```

**Note:** In production, we highly recommend you to use our [custom script generator](#) to create custom script file with required controls and its dependencies only. Also to reduce the file size further please use [GZip compression](#) in your server.

For themes, you can use the `ej.web.all.min.css` CDN link from the code example given. To add the themes in your application, please refer to [this link](#).

### Create a Grid

Control can be initialized in two ways.

- Using jsx Template
- Without using jsx Template

### Using jsx Template

By using the jsx template, we can create the html file and jsx file. The `.jsx` file can be convert to `.js` file and it can be referred in html page.

Please refer to the code of HTML file.

### HTML

```
<div id="Grid-default"></div>
<script type="text/babel" src="app.jsx">
</script>
```

Create a JSX file and paste the following content

### JS

```
var shipDetails = [
  { Name: 'Hanari Carnes', City: 'Brazil' },
  { Name: 'Split Rail Beer & Ale', City: 'USA' },
  { Name: 'Ricardo Adocicados', City: 'Brazil' }
];
ReactDOM.render(
  <EJ.Grid dataSource = {shipDetails} >
  </EJ.Grid>,
  document.getElementById('Grid-default')
);
```

Name	City
Hanari Carnes	Brazil
Split Rail Beer & Ale	USA
Ricardo Adocicados	Brazil

### Data binding

[Data binding](#) in the grid is achieved by assigning an array of JavaScript objects to the [dataSource](#) property. Refer to the following code example.

Please refer to the code of HTML file.

**HTML**

```
<div id="Grid-default"></div>
<script type="text/babel" src="app.jsx">
</script>
```

Create a JSX file and paste the following content

**JS**

```
var cols = ["OrderID", "EmployeeID", "CustomerID", "ShipCountry",
"Freight"];
ReactDOM.render(
//The datasource "window.gridData" is referred from
'http://js.syncfusion.com/demos/web/scripts/jsondata.min.js'
<EJ.Grid dataSource = {window.gridData} columns={cols}>
</EJ.Grid>,
document.getElementById('Grid-default')
);
```

OrderID	EmployeeID	CustomerID	ShipCountry	Freight
10248	5	VINET	France	32.38
10249	6	TOMSP	Germany	11.61
10250	4	HANAR	Brazil	65.83
10251	3	VICTE	France	41.34
10252	4	SUPRD	Belgium	51.3
10253	3	HANAR	Brazil	58.17
10254	5	CHOPS	Switzerland	22.98
10255	9	RICSU	Switzerland	148.33

**Note:** ODataAdaptor is the default adaptor for the DataManager. On binding to other web services, proper [data adaptor](#) needs to be set on **adaptor** option of the DataManager.

Enable Paging

[Paging](#) can be enabled by setting the [allowPaging](#) to true. This adds the pager in the bottom of the grid and page size can be customized by using the [pageSettings.pageSize](#) property

Please refer to the code of HTML file.

**HTML**

```
<div id="Grid-default"></div>
<script type="text/babel" src="app.jsx">
</script>
```

Create a JSX file and paste the following content

### JS

```
var pageSettings = { pageSize: 8 };
var cols = ["OrderID", "EmployeeID", "CustomerID", "ShipCountry",
"Freight"];
ReactDOM.render(
//The datasource "window.gridData" is referred from
'http://js.syncfusion.com/demos/web/scripts/jsondata.min.js'
<EJ.Grid dataSource = {window.gridData} columns={cols} allowPaging = {true}
pageSettings={pageSettings} >
</EJ.Grid>,
document.getElementById('Grid-default')
);
```

**Note:** Pager settings can be customized by using the [pageSettings.pageSize](#) property. When it is not given the default values for [pageSize](#) and [pageCount](#) are 12 and 8 respectively.

OrderID	EmployeeID	CustomerID	ShipCountry	Freight
10248	5	VINET	France	32.38
10249	6	TOMSP	Germany	11.61
10250	4	HANAR	Brazil	65.83
10251	3	VICTE	France	41.34
10252	4	SUPRD	Belgium	51.3
10253	3	HANAR	Brazil	58.17
10254	5	CHOPS	Switzerland	22.98
10255	9	RICSU	Switzerland	148.33

1
2
3
4
5
6
7
8
...

1 of 25 pages (200 items)

### Enable Filtering

[Filtering](#) can be enabled by setting the [allowFiltering](#) to [true](#). By default, the filter bar row is displayed to perform filtering, you can change the filter type by using the [filterSetting.filterType](#) property.

Please refer to the code of HTML file.

### HTML

```
<div id="Grid-default"></div>
<script type="text/babel" src="app.jsx">
</script>
```

Create a JSX file and paste the following content

JS

```

var pageSettings = { pageSize: 8 };
var cols = ["OrderID", "EmployeeID", "CustomerID", "ShipCountry",
"Freight"];
ReactDOM.render(
  //The datasource "window.gridData" is referred from
  'http://js.syncfusion.com/demos/web/scripts/jsondata.min.js'
  <EJ.Grid dataSource = {window.gridData} columns={cols} allowPaging = {true}
  pageSettings={pageSettings} allowFiltering={true} >
  </EJ.Grid>,
  document.getElementById('Grid-default')
);

```

OrderID	EmployeeID	CustomerID	ShipCountry	Freight
10248	5	VINET	France	32.38
10249	6	TOMSP	Germany	11.61
10250	4	HANAR	Brazil	65.83
10251	3	VICTE	France	41.34
10252	4	SUPRD	Belgium	51.3
10253	3	HANAR	Brazil	58.17
10254	5	CHOPS	Switzerland	22.98
10255	9	RICSU	Switzerland	148.33

« ◀ 1 2 3 4 5 6 7 8 ... ▶ »
1 of 25 pages (200 items)

## Enable Grouping

[Grouping](#) can be enabled by setting the [allowGrouping](#) to `true`. Columns can be grouped dynamically by drag and drop the grid column header to the group drop area. The initial grouping can be done by adding required column names in the [groupSettings.groupedColumns](#) property.

Please refer to the code of HTML file.

HTML

```

<div id="Grid-default"></div>
<script type="text/babel" src="app.jsx">
</script>

```

Create a JSX file and paste the following content

JS

```

var pageSettings = { pageSize: 8 };
var cols = ["OrderID", "EmployeeID", "CustomerID", "ShipCountry",
"Freight"];
ReactDOM.render(
//The datasource "window.gridData" is referred from
'http://js.syncfusion.com/demos/web/scripts/jsondata.min.js'
<EJ.Grid dataSource = {window.gridData} columns={cols} allowPaging = {true}
pageSettings={pageSettings} allowGrouping={true} >
</EJ.Grid>,
document.getElementById('Grid-default')
);

```

OrderID	EmployeeID	CustomerID	ShipCountry	Freight
10248	5	VINET	France	32.38
10249	6	TOMSP	Germany	11.61
10250	4	HANAR	Brazil	65.83
10251	3	VICTE	France	41.34
10252	4	SUPRD	Belgium	51.3
10253	3	HANAR	Brazil	58.17
10254	5	CHOPS	Switzerland	22.98
10255	9	RICSU	Switzerland	148.33

1 of 25 pages (200 items)

Refer to the following code example for initial grouping.

Please refer to the code of HTML file.

### HTML

```

<div id="Grid-default"></div>
<script type="text/babel" src="app.jsx">
</script>

```

Create a JSX file and paste the following content

### JS

```

var pageSettings = { pageSize: 8 };
var groupSettings = { groupedColumns: ["ShipCountry", "CustomerID"] };
var cols = ["OrderID", "EmployeeID", "CustomerID", "ShipCountry",
"Freight"];
ReactDOM.render(

```



```
//The datasource "window.gridData" is referred from
'http://js.syncfusion.com/demos/web/scripts/jsondata.min.js'
<EJ.Grid dataSource = {window.gridData} columns={cols} allowPaging = {true}
pageSettings={pageSettings} allowGrouping={true}
groupSettings={groupSettings} >
</EJ.Grid>,
document.getElementById('Grid-default')
);
```

ShipCountry ^ > CustomerID ^					
	OrderID	EmployeeID	CustomerID ^	ShipCountry ^	Freight
ShipCountry: Argentina - 1 item					
CustomerID: OCEAN - 1 item					
	10409	3	OCEAN	Argentina	29.83
ShipCountry: Austria - 2 items					
CustomerID: ERNSH - 10 items					
	10258	1	ERNSH	Austria	140.51
	10263	9	ERNSH	Austria	146.06
	10351	1	ERNSH	Austria	162.33
	10368	2	ERNSH	Austria	101.95
	10382	4	ERNSH	Austria	94.77
	10390	6	ERNSH	Austria	126.38
	10402	8	ERNSH	Austria	67.88
<div> <span>⏪</span> <span>⏩</span> <span>1</span> <span>2</span> <span>3</span> <span>4</span> <span>5</span> <span>6</span> <span>7</span> <span>8</span> <span>...</span> <span>⏪</span> <span>⏩</span> </div> <div>1 of 25 pages (200 items)</div>					

### Add Summaries

[Summaries](#) can be added by setting the [showSummary](#) to true and adding required summary rows and columns in the [summaryRows](#) property. For demonstration, Stock column's sum value is displayed as summary.

### HTML

```
var pageSettings = { pageSize: 8 };
var groupSettings = { groupedColumns: ["CustomerID"] };
var cols = ["OrderID", "EmployeeID", "CustomerID", "ShipCountry",
"Freight"];
var summaryRows = [
```

```
{
  title: "Sum",
  summaryColumns: [{ summaryType: ej.Grid.SummaryType.Sum, displayColumn:
    "Freight", dataMember: "Freight" }]
}
];
ReactDOM.render(
  //The datasource "window.gridData" is referred from
  'http://js.syncfusion.com/demos/web/scripts/jsondata.min.js'
  <EJ.Grid dataSource = {window.gridData} columns={cols} allowPaging = {true}
  pageSettings={pageSettings} allowGrouping={true}
  groupSettings={groupSettings} showSummary={true} summaryRows={summaryRows} >
  </EJ.Grid>,
  document.getElementById('Grid-default')
);
```

CustomerID ^ X

	OrderID	EmployeeID	CustomerID ^	ShipCountry	Freight
CustomerID: ANATR - 1 item					
	10308	7	ANATR	Mexico	1.61
	Sum				1.61
CustomerID: ANTON - 1 item					
	10365	3	ANTON	Mexico	22
	Sum				22.00
CustomerID: AROUT - 2 items					
	10355	6	AROUT	UK	41.95
	10383	8	AROUT	UK	34.24
	Sum				76.19
CustomerID: BERGS - 5 items					
	10278	8	BERGS	Sweden	92.69
	10280	2	BERGS	Sweden	8.98
	10384	3	BERGS	Sweden	168.64
	10444	3	BERGS	Sweden	3.5
	Sum				283.11
	Sum				13,126.33

1

2

3

4

5

6

7

8

...

▶

⏮

1 of 25 pages (200 items)

### Data binding

The Grid control uses [ej.DataManager](#) which supports both RESTful JSON data services binding and local JSON array binding. The [dataSource](#) property can be assigned either with the instance of [ej.DataManger](#) or JSON data array collection. It supports different kinds of data binding methods such as

1. Local data
2. Remote data

### Local Data

To bind local data to the Grid, you can assign a JSON array to the [dataSource](#) property.

The following code example describes the above behavior.

### HTML

```
<div id="Grid"></div>
<script type="text/babel" src="app.jsx">
</script>
```

Create a JSX file and paste the following content

### JAVASCRIPT

```
ReactDOM.render(
  //The datasource "window.gridData" is referred from
  'http://js.syncfusion.com/demos/web/scripts/jsondata.min.js'
  <EJ.Grid dataSource = {window.gridData} allowPaging = {true}>
    <columns>
      <column field="OrderID" />
      <column field="EmployeeID" />
      <column field="ShipCity" />
      <column field="ShipCountry" />
      <column field="Freight" />
    </columns>
  </EJ.Grid>,
  document.getElementById('Grid')
);
```

The following output is displayed as a result of the above code example.

OrderID	EmployeeID	ShipCity	ShipCountry	Freight
10248	5	Reims	France	32.38
10249	6	Münster	Germany	11.61
10250	4	Rio de Janeiro	Brazil	65.83
10251	3	Lyon	France	41.34
10252	4	Charleroi	Belgium	51.3
10253	3	Rio de Janeiro	Brazil	58.17
10254	5	Bern	Switzerland	22.98
10255	9	Genève	Switzerland	148.33
10256	3	Resende	Brazil	13.97
10257	4	San Cristóbal	Venezuela	81.91
10258	1	Graz	Austria	140.51
10259	4	México D.F.	Mexico	3.25

1

2

3

4

5

6

7

8

...

1 of 17 pages (200 items)

**Note:** 1. There is no in-built support to bind the XML data to the grid. But you can achieve this requirement with the help of [custom adaptor] concept.

2. Refer this [Knowledge Base link](#) for bounding XML data to grid using custom adaptor.

### Remote Data

To bind remote data to Grid Control, you can assign a service data as an instance of [ej.DataManager](#) to the [dataSource](#) property.

### OData

OData is a standardized protocol for creating and consuming data. You can provide the [OData service](#) URL directly to the [ej.DataManager](#) class and then you can assign it to Grid [dataSource](#).

The following code example describes the above behavior.

### HTML

```
<div id="Grid"></div>
<script type="text/babel" src="app.jsx">
</script>
```

Create a JSX file and paste the following content

### JAVASCRIPT

```
var data = ej.DataManager({
url: "http://js.syncfusion.com/demos/ejServices/Wcf/Northwind.svc/Orders"
});
```

```
ReactDOM.render(  
  //The datasource "window.gridData" is referred from  
  'http://js.syncfusion.com/demos/web/scripts/jsondata.min.js'  
  <EJ.Grid dataSource = {data} allowPaging = {true}>  
    <columns>  
      <column field="OrderID" />  
      <column field="EmployeeID" />  
      <column field="ShipCity" />  
      <column field="ShipCountry" />  
      <column field="Freight" />  
    </columns>  
  </EJ.Grid>,  
  document.getElementById('Grid')  
) ;
```

The following output is displayed as a result of the above code example.

OrderID	EmployeeID	CustomerID	ShipCountry	Freight
10248	1	VINET	SWEDEN	6.0000
10249	6	TOMSP	RUSSIA	18.0000
10250	9	HANAR	CANADA	8.0000
10251	3	VICTE	UKRAINE	43.0000
10252	4	SUPRD	DENMARK	28.0000
10253	3	HANAR	Venesuela	35.0000
10254	5	CHOPS	SWEDEN	28.0000
10255	9	RICSU	RUSSIA	148.0000
10256	3	WELLI	Brazil	13.9700
10257	4	HILAA	Venezuela	81.9100
10258	1	ERNSH	Austria	140.5100
10259	4	CENTC	NEW ZEASLAND	25.0000

1 of 1108 pages (13294 items)

Elements Sources Network Console Timeline Profiles Resources Audits Security

View: [Icons] Preserve log Disable cache No throttling

Filter [ ] Hide data URLs

All XHR JS CSS Img Media Font Doc WS Manifest Other

Name Path

2 / 21 requests | 12.4 KB / 124...

Headers Preview Response Timing

General

Request URL: http://mvc.syncfusion.com/Services/Northwnd.svc/Orders/?\$inlinecount=allpages&\$skip=0&\$top=12

Request Method: OPTIONS

Status Code: 200 OK

Remote Address: 191.234.40.112:80

Response Headers view source

Access-Control-Allow-Headers: accept, maxdataserviceversion, origin, x-requested-with, dataserviceversion, content-type

**Note:** By default , if no adaptor is specified for ej.DataManager and only the url link is mentioned it will consider as ODataService.

## Columns

Column definitions are used as the [dataSource](#) schema in Grid and it plays vital role in rendering column values in required format. Grid operations such as sorting, filtering, editing would be performed based

on the column definitions. The [field](#) property of the [columns](#) is necessary to map the datasource values in Grid columns.

**Note:** 1. If the column with [field](#) is not in the datasource, then the column values will be displayed as empty.

2. If the [field](#) name contains "dot" operator then it is considered as complex binding.

### Column Template

HTML templates can be specified in the [template](#) definition of the particular column as a string (HTML element) or ID of the template's HTML element.

**Note:** If [field](#) is not specified, you will not able to perform editing, grouping, filtering, sorting, search and summary functionalities in particular column.

The following code example describes the above behavior.

### HTML

```
<div id="Grid"></div>
<script type="text/babel" src="app.jsx">
</script>
```





Create a JSX file and paste the following content

### JAVASCRIPT

```
var pageSettings = {pageSize:4};
ReactDOM.render(
  //The datasource "window.employeeView" is referred from
  'http://js.syncfusion.com/demos/web/scripts/jsondata.min.js'
  <EJ.Grid dataSource = {window.employeeView} allowPaging = {true}
  pageSettings={pageSettings}>
  <columns>
  <column headerText="Photo", template = "<img style='width: 75px; height:
  70px'
  src='/13.2.0.29/themes/web/images/employees/{\"{\":EmployeeID{}}}.png'
  alt='\"{\":EmployeeID{}}' />\" />
  <column field="EmployeeID" />
  <column field="FirstName" />
  <column field="LastName" />
  <column field="Country" />
  </columns>
  </EJ.Grid>,
  document.getElementById('Grid')
);
```

The following output is displayed as a result of the above code example.



Photo	EmployeeID	FirstName	LastName	Country
	1	Nancy	Davolio	USA
	2	Andrew	Fuller	USA
	3	Janet	Leverling	USA
	4	Margaret	Peacock	USA

1

2

3

1 of 3 pages (9 items)

## Row

It represents the record details that are fetched from the datasource.

## Details Template

It provides a detailed view /additional information about each row of the grid. You can render any type of JsRender template and assign the script template id in the [detailsTemplate](#) property. And also you can change HTML elements in detail template row into JavaScript controls using [detailsDataBound](#) event.

On enabling details template, new column will be added in grid with an expander button in it and that can be expanded or collapsed to show or hide the underlying details row respectively.

The following code example describes the above behavior.

## HTML

```
<div id="Grid"></div>
<script type="text/babel" src="app.jsx">
</script>
<script id="tabGridContents" type="text/x-jsrender">
<div class="tabcontrol" id="Test">
<ul>
<li><a>Stock Grid</a></li>
</ul>
<div id="gridTab{{"{{"}}:EmployeeID{{}}}}">
<div id="detailGrid"></div>
</div>
</div>
</script>
```

Create a JSX file and paste the following content

**JAVASCRIPT**

```

var DetailGrid = React.createClass({
  detailGrid: function(e) {
    var filteredData = e.data["EmployeeID"];
    var data =
    ej.DataManager(window.ordersView).executeLocal(ej.Query().where("EmployeeID"
    , "equal", parseInt(filteredData), true).take(5));
    e.detailsElement.find("#detailGrid").ejGrid({
    dataSource: data,
    columns: ["OrderID", "EmployeeID", "ShipCity", "ShipCountry", "Freight"]
    });
    e.detailsElement.find(".tabcontrol").ejTab();
  },
  render: function () {
    return (
      //The datasource "window.employeeView" is referred from
      'http://js.syncfusion.com/demos/web/scripts/jsondata.min.js'
      <EJ.Grid dataSource = {window.employeeView} detailsTemplate =
      "#tabGridContents" detailsDataBound={this.detailGrid}>
      <columns>
      <column field="EmployeeID" />
      <column field="FirstName" />
      <column field="Title" />
      <column field="City" />
      <column field="Country" />
      </columns>
      </EJ.Grid>
    );
  }
});
ReactDOM.render(<DetailGrid />, document.getElementById('Grid'));

```

The following output is displayed as a result of the above code example.

	EmployeeID	FirstName	Title	City	Country
1	Nancy	Sales Representative	Seattle	USA	
<div>Stock Grid</div>					
	OrderID	EmployeeID	ShipCity	ShipCountry	Freight
	10835	1	Berlin	Germany	69.53
	10952	1	Berlin	Germany	40.42
	10677	1	México D.F.	Mexico	4.03
	10558	1	Colchester	UK	72.97
	10453	1	Colchester	UK	25.36
2	Andrew	Vice President, Sales	Tacoma	USA	
3	Janet	Sales Representative	Kirkland	USA	
4	Margaret	Sales Representative	Redmond	USA	
5	Steven	Sales Manager	London	UK	
6	Michael	Sales Representative	London	UK	
7	Robert	Sales Representative	London	UK	
8	Laura	Inside Sales Coordinat	Seattle	USA	
9	Anne	Sales Representative	London	UK	

### Row Template

Row template enables you to set the customized look and behavior to grid all rows. [rowTemplate](#) property can be used bind the `id` of HTML template.

The following code example describes the above behavior.

### HTML

```
<div id="Grid"></div>
<script type="text/babel" src="app.jsx">
</script>
<script id="rowTemp" type="text/x-jsrender">
<tr>
<td class="photo">
 />
</td>
<td class="details">
<table class="CardTable" cellpadding="3" cellspacing="2">
<colgroup>
```

```

<col width="50%">
<col width="50%">
</colgroup>
<tbody>
<tr>
<td class="CardHeader">First Name </td>
<td>{{:FirstName}} </td>
</tr>
<tr>
<td class="CardHeader">Birth Date
</td>
<td>{{:BirthDate}}
</td>
</tr>
<tr>
<td class="CardHeader">Hire Date
</td>
<td>{{:HireDate}}
</td>
</tr>
</tbody>
</table>
</td>
</tr>
</script>

```

Create a JSX file and paste the following content

#### JAVASCRIPT

```

var scroll = { width: 500, height: 380 };
ReactDOM.render(
  <EJ.Grid dataSource = {window.employeeData} allowScrolling = {true}
  scrollSettings={scroll} rowTemplate="#rowTemp">
    <columns>
      <column field="Photo" headerText="Photo" width={30} />
      <column headerText="Employee Details" width={70} />
    </columns>
  </EJ.Grid>,
  document.getElementById('Grid')
);

```

#### CSS

```

<style type="text/css" class="cssStyles">
.photo img
{
width: 130px;
height: 115px;
}
.photo, .details
{
border-color: #c4c4c4;
border-style: solid;
}
.photo





```

```

{
border-width: 1px 0px 0px 0px;
}
.details
{
border-width: 1px 0px 0px 1px;
}
#RowGrid tbody tr td
{
vertical-align: middle;
}
.details > table
{
width: 100%;
}
.CardHeader
{
font-weight: bolder;
}
td
{
padding: 2px 2px 3px 2px;
}
</style>

```

The following output is displayed as a result of the above code example.

Photo	Employee Details	
	<b>First Name</b> Nancy <b>Birth Date</b> 12/08/1948 <b>Hire Date</b> 05/01/1992	^
	<b>First Name</b> Andrew <b>Birth Date</b> 02/19/1952 <b>Hire Date</b> 08/14/1992	
	<b>First Name</b> Janet <b>Birth Date</b> 08/30/1963 <b>Hire Date</b> 04/01/1992	
		v

## Editing

The grid control has support for dynamic insertion, updating and deletion of records. You can start the edit action either by double clicking the particular row or by selecting the required row and clicking on Edit icon in toolbar. Similarly, you can add new record to grid either by clicking on insert icon in toolbar or on an external button which is bound to call [addRecord](#) method of grid. **Save** and **Cancel** while on edit mode is possible using respective toolbar icon in grid.

Deletion of the record is possible by selecting the required row and clicking on Delete icon in toolbar.

The primary key for the data source should be defined in [columns](#) property, for editing to work properly. In [e-columns](#) property, particular primary column's [isPrimaryKey](#) property should be set to **true**. Refer the Knowledge base [link](#) for more information.

**Note:** 1. In grid, the primary key column will be automatically set to read only while editing the row, but you can specify primary key column value while adding a new record.

2. The column which is specified as [isIdentity](#) will be in readonly mode both while editing and adding a record. Also, auto incremented value is assigned to that [isIdentity](#) column.

## Toolbar with edit option

Using toolbar which is rendered at the top of the grid header, you can show all the CRUD related action. To enable toolbar and toolbar items, set [showToolbar](#) property as true and [toolbarItems](#). The default toolbar items are **Add**, **Edit**, **Delete**, **Update** and **Cancel**.

**Note:** For [toolbarItems](#) property you can assign either string value ("add") or enum value (ej.Grid.ToolbarItems.Add).

The following code example describes the above behavior.

## HTML

```
<div id="Grid"></div>
<script type="text/babel" src="app.jsx">
</script>
```

Create a JSX file and paste the following content

## JAVASCRIPT

```
var editSettings = { allowEditing: true, allowAdding: true, allowDeleting: true };
var toolbarItems = { showToolbar: true, toolbarItems: ["add", "edit", "delete", "update", "cancel"] };
ReactDOM.render(
  //The datasource "window.gridData" is referred from
  'http://js.syncfusion.com/demos/web/scripts/jsondata.min.js'
  <EJ.Grid dataSource = {window.gridData} allowPaging = {true}
  editSettings={editSettings} toolbarSettings={toolbarItems}>
  <columns>
  <column field="OrderID" isPrimaryKey={true} />
  <column field="CustomerID" />
  <column field="EmployeeID" />
  <column field="ShipCity" />
  <column field="ShipCountry" />
  </columns>
```

```
</EJ.Grid>,
document.getElementById('Grid')
);
```

The following output is displayed as a result of the above code example.

OrderID	CustomerID	EmployeeID	ShipCity	ShipCountry
10248	VINET	5	Reims	France
10249	TOMSP	6	Münster	Germany
10250	HANAR	4	Rio de Janeiro	Brazil
10251	VICTE	3	Lyon	France
10252	SUPRD	4	Charleroi	Belgium
10253	HANAR	3	Rio de Janeiro	Brazil
10254	CHOPS	5	Bern	Switzerland
10255	RICSU	9	Genève	Switzerland
10256	WELLI	3	Resende	Brazil
10257	HILAA	4	San Cristóbal	Venezuela
10258	ERNSH	1	Graz	Austria
10259	CENTC	4	México D.F.	Mexico

1 of 17 pages (200 items)

### Cell edit type and its params

The edit type of bound column can be customized using [editType](#) property of [columns](#). The following Essential JavaScript controls are supported built-in by [editType](#). You can set the [editType](#) based on specific data type of the column.

- [CheckBox](#) control for boolean data type.
- [NumericTextBox](#) control for integers, double, and decimal data types.
- [\[InputTextBox\]](#) control for string data type.
- [DatePicker](#) control for date data type.
- [DateTimePicker](#) control for date-time data type.
- [DropDownList](#) control for list of data type.

And also you can define the model for all the editTypes controls while editing through [editParams](#) property of [columns](#).

The following table describes [editType](#) and their corresponding [editParams](#) of the specific data type of the column.

EditType	EditParams	Example
CheckBox	<a href="#">ejCheckBox</a>	editParams: { checked: true }
NumericTextBox	<a href="#">ejTextBoxes</a>	editParams: { decimalPlaces: 2, value:5 }
InputTextBox	-	-
DatePicker	<a href="#">ejDatePicker</a>	editParams: { buttonText : "Now" }
DateTimePicker	<a href="#">ejDateTimePicker</a>	editParams: { enabled: true }
DropDownList	<a href="#">ejDropDownList</a>	editParams: { allowGrouping: true }

**Note:** 1. If [editType](#) is not set, then by default it will display the input element ("stringedit") while editing a column.

2. For [editType](#) property you can assign either `string` value ("numericedit") or `enum` value (ej.Grid.EditingType.Numeric).

The following code example describes the above behavior.

#### HTML

```
<div id="Grid"></div>
<script type="text/babel" src="app.jsx">
</script>
```

#### JAVASCRIPT

```
var editSettings = { allowEditing: true, allowAdding: true, allowDeleting: true };
var toolbarItems = { showToolbar: true, toolbarItems: ["add", "edit", "delete", "update", "cancel"] };
var decimalPlaces = {decimalPlaces:3};
var enableAnimation = {enableAnimation:true};
var showRoundedCorner = {showRoundedCorner:true};
ReactDOM.render(
  //The datasource "window.gridData" is referred from
  'http://js.syncfusion.com/demos/web/scripts/jsondata.min.js'
  <EJ.Grid dataSource = {window.gridData} allowPaging = {true}
  editSettings={editSettings} toolbarSettings={toolbarItems}>
  <columns>
  <column field="OrderID" isPrimaryKey={true} />
  <column field="CustomerID" editType="stringedit" />
  <column field="Freight" editType="numericedit" editParams={decimalPlaces} />
  <column field="ShipCity" editType="dropdownedit"
  editParams={enableAnimation} />
  <column field="ShipCountry" />
  <column field="OrderDate" editType="datepicket" format="{0:MM/dd/yyyy}" />
  <column field="Verified" editType="booleanedit"
  editParams={showRoundedCorner} />
  </columns>
  </EJ.Grid>,
  document.getElementById('Grid')
```



```
);
```

The following output is displayed as a result of the above code example.

OrderID	CustomerID	Freight	ShipCity	ShipCountry	OrderDate	Verified
10248	VINET	32.38	Reims	France	07/04/1996	<input checked="" type="checkbox"/>
10249	TOMSP	11.61	Münster	Germany	07/05/1996	<input type="checkbox"/>
10250	HANAR	65.83	Rio de Janeiro	Brazil	07/08/1996	<input checked="" type="checkbox"/>
10251	VICTE	41.34	Lyon	France	07/08/1996	<input checked="" type="checkbox"/>
10252	SUPRD	51.3	Charleroi	Belgium	<div><div><div>July 1996</div></div></div>	
10253	HANAR	58.17	Rio de Janeiro	Brazil	<div><div><div>Su</div><div>Mo</div><div>Tu</div><div>We</div><div>Th</div><div>Fr</div><div>Sa</div></div></div>	
10254	CHOPS	22.98	Bern	Switzerland	<div><div><div>30</div><div>1</div><div>2</div><div>3</div><div>4</div><div>5</div><div>6</div></div></div>	
10255	RICSU	148.33	Genève	Switzerland	<div><div><div>7</div><div>8</div><div>9</div><div>10</div><div>11</div><div>12</div><div>13</div></div></div>	
10256	WELLI	13.97	Resende	Brazil	<div><div><div>14</div><div>15</div><div>16</div><div>17</div><div>18</div><div>19</div><div>20</div></div></div>	
10257	HILAA	81.91	San Cristóbal	Venezuela	<div><div><div>21</div><div>22</div><div>23</div><div>24</div><div>25</div><div>26</div><div>27</div></div></div>	
10258	ERNSH	140.51	Graz	Austria	<div><div><div>28</div><div>29</div><div>30</div><div>31</div><div>1</div><div>2</div><div>3</div></div></div>	
10259	CENTC	3.25	México D.F.	Mexico	07/18/1996	<input type="checkbox"/>

</

### Cell Edit Template

On editing the column values, custom editor can be created by using `editTemplate` property of `columns`. It has three functions, they are

1. **create** - It is used to create the control at time of initialize.
2. **read** - It is used to read the input value at time of save.
3. **write** - It is used to assign the value to control at time of editing.

The following code example describes the above behavior.

### HTML

```
<div id="Grid"></div>
<script type="text/babel" src="app.jsx">
</script>
```

Create a JSX file and paste the following content

### JAVASCRIPT

```

var editSettings = { allowEditing: true, allowAdding: true, allowDeleting:
true };
var toolbarItems = { showToolbar: true, toolbarItems: ["add", "edit",
"delete", "update", "cancel"] };
var postalEditTemplate = {
create : function () {
return "<input>";
},
read : function (args) {
return args.ejMaskEdit("get_UnstrippedValue");
},
write : function (args) {
args.element.ejMaskEdit({
maskFormat : "99-99-9999",
value : args.rowdata["ShipPostalCode"]
});
}
};
ReactDOM.render(
//The datasource "window.gridData" is referred from
'http://js.syncfusion.com/demos/web/scripts/jsondata.min.js'
<EJ.Grid dataSource = {window.gridData} allowPaging = {true}
editSettings={editSettings} toolbarSettings={toolbarItems}>
<columns>
<column field="OrderID" isPrimaryKey={true} />
<column field="CustomerID" />
<column field="EmployeeID" />
<column field="Freight" />
<column field="ShipCountry" />
<column field="ShipPostalCode" editTemplate={postalEditTemplate} />
</columns>
</EJ.Grid>,
document.getElementById('Grid')
);

```

The following output is displayed as a result of the above code example.

OrderID	CustomerID	Freight	ShipCountry	ShipPostalCode
10248	VINET	32.38	France	51100
10249	TOMSP	11.61	Germany	44-08-7
10250	HANAR	65.83	Brazil	05454-876
10251	VICTE	41.34	France	69004
10252	SUPRD	51.3	Belgium	B-6000
10253	HANAR	58.17	Brazil	05454-876
10254	CHOPS	22.98	Switzerland	3012
10255	RICSU	148.33	Switzerland	1204
10256	WELLI	13.97	Brazil	08737-363
10257	HILAA	81.91	Venezuela	5022
10258	ERNSH	140.51	Austria	8010
10259	CENTC	3.25	Mexico	05022

1 of 17 pages (200 items)

## Edit Modes

### Inline

Set `editMode` as `normal`, then the row itself is changed as edited row.

**Note:** For `editMode` property you can assign either `string` value ("normal") or `enum` value (`ej.Grid.EditMode.Normal`).

The following code example describes the above behavior.

### HTML

```
<div id="Grid"></div>
<script type="text/babel" src="app.jsx">
</script>
```

Create a JSX file and paste the following content

### JAVASCRIPT

```
var editSettings = { allowEditing: true, allowAdding: true, allowDeleting:
true, editMode:"normal" };
var toolbarItems = { showToolbar: true, toolbarItems: ["add", "edit",
"delete", "update", "cancel"] };
ReactDOM.render(
//The datasource "window.gridData" is referred from
'http://js.syncfusion.com/demos/web/scripts/jsondata.min.js'
```

```

<EJ.Grid dataSource = {window.gridData} allowPaging = {true}
editSettings={editSettings} toolbarSettings={toolbarItems}>
<columns>
<column field="OrderID" isPrimaryKey={true} />
<column field="CustomerID" />
<column field="Freight" editType="numericedit" />
<column field="ShipCountry" editType="dropdownedit"/>
<column field="OrderDate" editType="datepicker" format="{0:MM/dd/yyyy}"/>
</columns>
</EJ.Grid>,
document.getElementById('Grid')
);

```

The following output is displayed as a result of the above code example.

OrderID	CustomerID	Freight	ShipCountry	OrderDate
10248	VINET	32.38	France	04/07/1996
10249	TOMSP	11.61	Germany	05/07/1996
10250	HANAR	65.83	Brazil	08/07/1996
10251	VICTE	41.34	France	08/07/1996
10252	SUPRD	51	Belgium	09/07/1996
10253	HANAR	58.17	Argentina	10/07/1996
10254	CHOPS	22.98	Austria	11/07/1996
10255	RICSU	148.33	Belgium	12/07/1996
10256	WELLI	13.97	Brazil	15/07/1996
10257	HILAA	81.91	Canada	16/07/1996
10258	ERNSH	140.51	Austria	17/07/1996
10259	CENTC	3.25	Mexico	18/07/1996

1 of 17 pages (200 items)

#### Inline Form

Set `editMode` as `inlineform`, then edit form will be inserted next to the row which is to be edited.

The following code example describes the above behavior.

#### HTML

```

<div id="Grid"></div>
<script type="text/babel" src="app.jsx">
</script>






```

Create a JSX file and paste the following content

#### JAVASCRIPT

```
var editSettings = { allowEditing: true, allowAdding: true, allowDeleting:
true, editMode:"inlineform" };
var toolbarItems = { showToolbar: true, toolbarItems: ["add", "edit",
"delete", "update", "cancel"] };
ReactDOM.render(
//The datasource "window.gridData" is referred from
'http://js.syncfusion.com/demos/web/scripts/jsondata.min.js'
<EJ.Grid dataSource = {window.gridData} allowPaging = {true}
editSettings={editSettings} toolbarSettings={toolbarItems}>
<columns>
<column field="OrderID" isPrimaryKey={true} />
<column field="CustomerID" />
<column field="Freight" editType="numericedit" />
<column field="ShipCountry" editType="dropdownedit"/>
<column field="OrderDate" editType="datepicker" format="{0:MM/dd/yyyy}"/>
</columns>
</EJ.Grid>,
document.getElementById('Grid')
);
```

The following output is displayed as a result of the above code example.

    				
OrderID	CustomerID	Freight	ShipCountry	OrderDate
10248	VINET	32.38	France	04/07/1996
10249	TOMSP	11.61	Germany	05/07/1996
10250	HANAR	65.83	Brazil	08/07/1996
10251	VICTE	41.34	France	08/07/1996
Details of 10251				
OrderID	<input type="text" value="10251"/>	CustomerID	<input type="text" value="VICTE"/>	
Freight	<input type="text" value="41"/>	ShipCountry	<input type="text" value="France"/>	
OrderDate	<input type="text" value="08/07/1996"/>			
				<input type="button" value="Save"/> <input type="button" value="Cancel"/>
10252	SUPRD	51.3	Belgium	09/07/1996
10253	HANAR	58.17	Brazil	10/07/1996
10254	CHOPS	22.98	Switzerland	11/07/1996
10255	RICSU	148.33	Switzerland	12/07/1996
10256	WELLI	13.97	Brazil	15/07/1996
10257	HILAA	81.91	Venezuela	16/07/1996

### Inline Template Form

You can edit any of the fields pertaining to a single record of data and apply it to a template so that the same format is applied to all the other records that you may edit later.

Using this template support, you can edit the fields that are not bound to grid columns.

To edit the records using Inline template form, set `editMode` as `inlineformtemplate` and specify the template ID to `inlineFormTemplateID` of `[editSettings]` property.

While using template form, you can change the HTML elements to appropriate JS controls based on the column type. This can be achieved by using `actionComplete` event of grid.

**Note:** 1. `value` attribute is used to bind the corresponding field value while editing.

2. `name` attribute is used to get the changed field values while saving the edited record.

4. For `editMode` property you can assign either `string` value ("`inlineformtemplate`") or `enum` value (`ej.Grid.EditMode.InlineTemplateForm`)

The following code example describes the above behavior.

### HTML

```
<div id="Grid"></div>
<script type="text/babel" src="app.jsx">
</script>
<script id="template" type="text/x-jsrender">
<table cellpadding="10">
<tr>
<td>Order ID</td>
<td>
<input id="OrderID" name="OrderID" disabled="disabled"
ngModel="{{{"{":{}:OrderID{}}}}}" value="{{{"{":{}:OrderID{}}}}}" />
</td>
<td>Customer ID</td>
<td>
<input id="CustomerID" name="CustomerID" ngModel="{{{"{":{}:CustomerID{}}}}}"
value="{{{"{":{}:CustomerID{}}}}}" class="e-field e-ejinputtext"
style="width: 116px; height: 28px" />
</td>
</tr>
<tr>
<td>Employee ID</td>
<td>
<input type="text" id="EmployeeID" name="EmployeeID"
ngModel="{{{"{":{}:EmployeeID{}}}}}" value="{{{"{":{}:EmployeeID{}}}}}" />
</td>
<td>Ship City</td>
<td>
<select id="ShipCity" name="ShipCity">
<option value="Argentina">Argentina</option>
<option value="Austria">Austria</option>
<option value="Belgium">Belgium</option>
<option value="Brazil">Brazil</option>
<option value="Canada">Canada</option>
<option value="Denmark">Denmark</option>
</select>
</td>
</tr>
</table>
</script>
```

Create a JSX file and paste the following content

### JAVASCRIPT

```
var EditGrid = React.createClass({
  actionComplete: function(e) {
    $("#EmployeeID").ejNumericTextbox();
    $("#Freight").ejNumericTextbox();
    $("#ShipCity").ejDropDownList();
  },
  editSettings: { allowEditing: true, allowAdding: true, allowDeleting: true,
    editMode: "inlineformtemplate", inlineFormTemplateID: "#template" },
  toolbarItems: { showToolbar: true, toolbarItems: ["add", "edit", "delete",
    "update", "cancel"] },
```

```

render: function () {
  return (
    //The datasource "window.gridData" is referred from
    'http://js.syncfusion.com/demos/web/scripts/jsondata.min.js'
    <EJ.Grid dataSource = {window.gridData} allowPaging = {true}
    editSettings={this.editSettings} toolbarSettings={this.toolbarItems}
    actionComplete={this.actionComplete}>
    <columns>
    <column field="OrderID" isPrimaryKey={true} />
    <column field="CustomerID" />
    <column field="ShipCity" />
    </columns>
    </EJ.Grid>
  );
};
ReactDOM.render(<EditGrid />, document.getElementById('Grid'));

```

The following output is displayed as a result of the above code example.

OrderID	CustomerID	ShipCity
10248	VINET	Reims
10249	TOMSP	Münster
10250	HANAR	Rio de Janeiro

**Details of 10250**

Order ID	<input type="text" value="10250"/>	Customer ID	<input type="text" value="HANAR"/>
Employee ID	<input type="text" value="4"/>	Ship City	<input type="text" value="Argentina"/>

10251	VICTE	Lyon
10252	SUPRD	Charleroi
10253	HANAR	Rio de Janeiro
10254	CHOPS	Bern
10255	RICSU	Genève
10256	WELLI	Resende
10257	HILAA	San Cristóbal
10258	ERNSH	Graz



Before the template elements are converted to JS controls

OrderID	CustomerID	ShipCity
10248	VINET	Reims
10249	TOMSP	Münster
10250	HANAR	Rio de Janeiro

**Details of 10250**

Order ID: 
 Customer ID:

Employee ID: 
 Ship City:

10251	VICTE	Lyon
10252	SUPRD	Charleroi
10253	HANAR	Rio de Janeiro
10254	CHOPS	Bern
10255	RICSU	Genève
10256	WELLI	Resende
10257	HILAA	San Cristóbal
10258	ERNSH	Graz

After the template elements are converted to JS controls using `actionComplete` event

#### Dialog

Set `editMode` as `dialog` to edit data using a dialog box, which displays the fields associated with the data record being edited.

The following code example describes the above behavior.

#### HTML

```
<div id="Grid"></div>
<script type="text/babel" src="app.jsx">
</script>
```

Create a JSX file and paste the following content

#### JAVASCRIPT

```

var editSettings = { allowEditing: true, allowAdding: true, allowDeleting:
true, editMode: "dialog" };
var toolbarItems = { showToolbar: true, toolbarItems: ["add", "edit",
"delete", "update", "cancel"] };
ReactDOM.render(
//The datasource "window.gridData" is referred from
'http://js.syncfusion.com/demos/web/scripts/jsondata.min.js'
<EJ.Grid dataSource = {window.gridData} allowPaging = {true}
editSettings={editSettings} toolbarSettings={toolbarItems}>
<columns>
<column field="OrderID" isPrimaryKey={true} />
<column field="CustomerID" />
<column field="Freight" editType="numericedit" />
<column field="ShipCountry" editType="dropdownedit"/>
<column field="OrderDate" editType="datepicker" format="{0:MM/dd/yyyy}"/>
</columns>
</EJ.Grid>,
document.getElementById('Grid')
);

```

The following output is displayed as a result of the above code example.

The screenshot displays a Syncfusion Grid with 5 columns: OrderID, CustomerID, Freight, ShipCountry, and OrderDate. The grid contains 17 records. A dialog box titled 'Details of 10250' is open, allowing editing of the record with OrderID 10250. The dialog fields are: OrderID (10250), CustomerID (HANAR), Freight (66), ShipCountry (Brazil), and OrderDate (08/07/1996). The dialog has 'Save' and 'Cancel' buttons. The grid's footer shows '1 of 17 pages (200 items)'.

OrderID	CustomerID	Freight	ShipCountry	OrderDate
10248	VINET	32.38	France	04/07/1996
10249	TOMSP			05/07/1996
10250	HANAR			08/07/1996
10251	VICTE			08/07/1996
10252	SUPRD			09/07/1996
10253	HANAR			10/07/1996
10254	CHOPS			11/07/1996
10255	RICSU			12/07/1996
10256	WELLI			15/07/1996
10257	HILAA			16/07/1996
10258	ERNSH			17/07/1996
10259	CENTC	3.25	Mexico	18/07/1996

#### Dialog Template Form

You can edit any of the fields pertaining to a single record of data and apply it to a template so that the same format is applied to all the other records that you may edit later.

Using this template support, you can edit the fields that are not bound to grid columns.

To edit the records using Inline template form, set [editMode](#) as dialogtemplate and specify the template id to [dialogEditorTemplateID](#) property of [editSettings](#).

While using template, you can change the elements that are defined in the [template](#), to appropriate JS controls based on the column type. This can be achieved by using [actionComplete](#) event of grid.

**Note:** 1. [value](#) attribute is used to bind the corresponding field value while editing.

2. [name](#) attribute is used to get the changed field values while save the edited record.

3. For [editMode](#) property you can assign either [string](#) value ("dialogtemplate") or [enum](#) value (ej.Grid.EditMode.DialogTemplate).

The following code example describes the above behavior.

### HTML

```
<div id="Grid"></div>
<script type="text/babel" src="app.jsx">
</script>
<script id="template" type="text/x-jsrender">
<table cellpadding="10">
<tr>
<td>Order ID</td>
<td>
<input id="OrderID" name="OrderID" disabled="disabled"
ngModel="{{{"{":{}:OrderID{}}}}}" value="{{{"{":{}:OrderID{}}}}"/>
</td>
<td>Customer ID</td>
<td>
<input id="CustomerID" name="CustomerID" ngModel="{{{"{":{}:CustomerID{}}}}}"
value="{{{"{":{}:CustomerID{}}}}}" class="e-field e-ejinputtext"
style="width: 116px; height: 28px" />
</td>
</tr>
<tr>
<td>Employee ID</td>
<td>
<input type="text" id="EmployeeID" name="EmployeeID"
ngModel="{{{"{":{}:EmployeeID{}}}}}" value="{{{"{":{}:EmployeeID{}}}}"/>
</td>
<td>Ship City</td>
<td>
<select id="ShipCity" name="ShipCity">
<option value="Argentina">Argentina</option>
<option value="Austria">Austria</option>
<option value="Belgium">Belgium</option>
<option value="Brazil">Brazil</option>
<option value="Canada">Canada</option>
<option value="Denmark">Denmark</option>
</select>
</td>
</tr>
</table>
</script>
```

Create a JSX file and paste the following content

### JAVASCRIPT

```
var EditGrid = React.createClass({
  actionComplete: function(e) {
    $("#EmployeeID").ejNumericTextbox();
    $("#Freight").ejNumericTextbox();
    $("#ShipCity").ejDropDownList();
  },
  editSettings: { allowEditing: true, allowAdding: true, allowDeleting: true,
  editMode: "dialogtemplate", dialogEditorTemplateID: "#template" },
  toolbarItems: { showToolbar: true, toolbarItems: ["add", "edit", "delete",
  "update", "cancel"] },
  render: function () {
    return (
      //The datasource "window.gridData" is referred from
      'http://js.syncfusion.com/demos/web/scripts/jsondata.min.js'
      <EJ.Grid dataSource = {window.gridData} allowPaging = {true}
      editSettings={this.editSettings} toolbarSettings={this.toolbarItems}
      actionComplete={this.actionComplete}>
      <columns>
      <column field="OrderID" isPrimaryKey={true} />
      <column field="CustomerID" />
      <column field="ShipCity" />
      </columns>
      </EJ.Grid>
    );
  }
});
ReactDOM.render(<EditGrid />, document.getElementById('Grid'));
```

The following output is displayed as a result of the above code example.

The screenshot shows a data grid with three columns: OrderID, CustomerID, and ShipCity. The grid contains 17 rows of data. The row with OrderID 10250 is selected. An edit dialog box titled 'Details of 10250' is open, showing the current values for the selected row. The dialog has four input fields: Order ID (10250), Customer ID (HANAR), Employee ID (4), and Ship City (Argentina). There are 'Save' and 'Cancel' buttons at the bottom of the dialog. The grid has a footer showing '1 of 17 pages (200 items)'.

OrderID	CustomerID	ShipCity
10248	VINET	Reims
10249	TOMSP	Münster
10250	HANAR	Rio de Janeiro
10251		
10252		
10253		
10254		
10255		
10256		
10257	HILAA	San Cristóbal
10258	ERNSH	Graz
10259	CENTC	México D.F.

Details of 10250

Order ID: 10250 Customer ID: HANAR

Employee ID: 4 Ship City: Argentina ▼

Save Cancel

1 of 17 pages (200 items)

Before the template elements are converted to JS controls

The screenshot shows a grid with columns OrderID, CustomerID, and ShipCity. The row with OrderID 10250 is selected. An external edit form titled 'Details of 10250' is open, displaying the following fields:

Field	Value
Order ID	10250
Customer ID	HANAR
Employee ID	4
Ship City	Belgium

Below the fields are 'Save' and 'Cancel' buttons. The grid below the form shows rows 10251 through 10259. The grid footer indicates '1 of 17 pages (200 items)'.

After the template elements are converted to JS controls using `actionComplete` event

#### [External Form](#)

By setting the `editMode` as `externalform`, the edit form is opened outside the grid content.

The following code example describes the above behavior.

#### HTML

```
<div id="Grid"></div>
<script type="text/babel" src="app.jsx">
</script>
```

Create a JSX file and paste the following content

#### JAVASCRIPT

```
var pageSettings = {pageSize:7};
var editSettings = { allowEditing: true, allowAdding: true, allowDeleting:
true, editMode:"externalform"}
var toolbarItems = { showToolbar: true, toolbarItems: ["add", "edit",
"delete", "update", "cancel"] };
ReactDOM.render(
//The datasource "window.gridData" is referred from
'http://js.syncfusion.com/demos/web/scripts/jsondata.min.js'
```

```

<EJ.Grid dataSource = {window.gridData} allowPaging = {true}
editSettings={editSettings} toolbarSettings={toolbarItems}
pageSettings={pageSettings}>
<columns>
<column field="OrderID" isPrimaryKey={true} />
<column field="CustomerID" />
<column field="Freight" editType="numericedit" />
<column field="ShipCountry" editType="dropdownedit"/>
<column field="OrderDate" editType="datepicker" format="{0:MM/dd/yyyy}"/>
</columns>
</EJ.Grid>,
document.getElementById('Grid')
);

```

The following output is displayed as a result of the above code example.

OrderID	CustomerID	Freight	ShipCountry	OrderDate
10248	VINET	32.38	France	04/07/1996
10249	TOMSP	11.61	Germany	05/07/1996
10250	HANAR	65.83	Brazil	08/07/1996
10251	VICTE	41.34	France	08/07/1996
10252	SUPRD	51.3	Belgium	09/07/1996
10253	HANAR	58.17	Brazil	10/07/1996
10254	CHOPS	22.98	Switzerland	11/07/1996

1 of 29 pages (200 items)

### Details of 10250

OrderID: 
 CustomerID:

Freight: 
 ShipCountry:

OrderDate:

Form Position:

You can position an external edit form in the following two ways.

1. Top-right
2. Bottom left

This can be achieved by setting the [formPosition](#) property of [editSettings](#) as "topright" or "bottomleft".

The following code example describes the above behavior.

#### HTML

```
<div id="Grid"></div>
<script type="text/babel" src="app.jsx">
</script>
```

Create a JSX file and paste the following content

#### JAVASCRIPT

```
var pageSettings = {pageSize:7};
var editSettings = { allowEditing: true, allowAdding: true, allowDeleting:
true, editMode:"externalform", formPosition:"topRight"};
var toolbarItems = { showToolbar: true, toolbarItems: ["add", "edit",
"delete", "update", "cancel"] };
ReactDOM.render(
//The datasource "window.gridData" is referred from
'http://js.syncfusion.com/demos/web/scripts/jsondata.min.js'
<EJ.Grid dataSource = {window.gridData} allowPaging = {true}
editSettings={editSettings} toolbarSettings={toolbarItems}
pageSettings={pageSettings}>
<columns>
<column field="OrderID" isPrimaryKey={true} />
<column field="CustomerID" />
<column field="Freight" editType="numericedit" />
<column field="ShipCountry" editType="dropdownedit"/>
</columns>
</EJ.Grid>,
document.getElementById('Grid')
);
```

The following output is displayed as a result of the above code example.



OrderID	CustomerID	Freight	ShipCountry
10248	VINET	32.38	France
10249	TOMSP	11.61	Germany
10250	HANAR	65.83	Brazil
10251	VICTE	41.34	France
10252	SUPRD	51.3	Belgium
10253	HANAR	58.17	Brazil
10254	CHOPS	22.98	Switzerland
10255	RICSU	148.33	Switzerland
10256	WELLI	13.97	Brazil
10257	HILAA	81.91	Venezuela
10258	ERNSH	140.51	Austria
10259	CENTC	3.25	Mexico

Details of 10251

OrderID:

CustomerID:

Freight:

ShipCountry:

Save Cancel

1 of 17 pages (200 items)

### External Template Form

You can edit any of the fields pertaining to a single record of data and apply it to a template so that the same format is applied to all the other records that you may edit later.

Using this template support, you can edit the fields that are not bound to grid columns.

To edit the records using External template form, set `editMode` as `externalformtemplate` and specify the template id to `externalFormTemplateID` property of `e-edit-settings`.

While using template, you can change the elements that are defined in the template, to appropriate JS controls based on the column type. This can be achieved by using `actionComplete` event of grid.

- Note:**
1. `value` attribute is used to bind the corresponding field value while editing.
  2. `name` attribute is used to get the changed field values while save the edited record.
  3. For `editMode` property you can assign either `string` value ("externalformtemplate") or `enum` value (`ej.Grid.EditMode.ExternalFormTemplate`).

The following code example describes the above behavior.

### HTML

```
<div id="Grid"></div>
<script type="text/babel" src="app.jsx">
</script>
<script id="template" type="text/x-jsrender">
<table cellpadding="10">
<tr>
<td>Order ID</td>
<td>
<input id="OrderID" name="OrderID" disabled="disabled"
ngModel="{{"{{"{{":OrderID{{"{{" value="{{"{{"{{":OrderID{{"{{"}}"/>
</td>
<td>Customer ID</td>
<td>
```

```

<input id="CustomerID" name="CustomerID" ngModel="{{'{}':CustomerID{{}}}}"
value="{{'{}':CustomerID{{}}}}" class="e-field e-ejinputtext"
style="width: 116px; height: 28px" />
</td>
</tr>
<tr>
<td>Employee ID</td>
<td>
<input type="text" id="EmployeeID" name="EmployeeID"
ngModel="{{'{}':EmployeeID{{}}}}" value="{{'{}':EmployeeID{{}}}}" />
</td>
<td>Ship City</td>
<td>
<select id="ShipCity" name="ShipCity">
<option value="Argentina">Argentina</option>
<option value="Austria">Austria</option>
<option value="Belgium">Belgium</option>
<option value="Brazil">Brazil</option>
<option value="Canada">Canada</option>
<option value="Denmark">Denmark</option>
</select>
</td>
</tr>
</table>
</script>

```

Create a JSX file and paste the following content

#### JAVASCRIPT

```

var EditGrid = React.createClass({
  actionComplete: function(e) {
    $("#EmployeeID").ejNumericTextbox();
    $("#Freight").ejNumericTextbox();
    $("#ShipCity").ejDropDownList();
  },
  pageSettings = {pageSize:5},
  editSettings: { allowEditing: true, allowAdding: true, allowDeleting: true,
  editMode: "externalformtemplate", externalFormTemplateID: "#template" },
  toolbarItems: { showToolbar: true, toolbarItems: ["add", "edit", "delete",
  "update", "cancel"] },
  render: function () {
    return (
      //The datasource "window.gridData" is referred from
      'http://js.syncfusion.com/demos/web/scripts/jsondata.min.js'
      <EJ.Grid dataSource = {window.gridData} allowPaging = {true}
      editSettings={this.editSettings} toolbarSettings={this.toolbarItems}
      actionComplete={this.actionComplete}>
      <columns>
      <column field="OrderID" isPrimaryKey={true} />
      <column field="CustomerID" />
      <column field="ShipCity" />
      </columns>
      </EJ.Grid>
    );
  }
});

```

```
});
ReactDOM.render(<EditGrid />, document.getElementById('Grid'));
```

The following output is displayed as a result of the above code example.

OrderID	CustomerID	ShipCity
10248	VINET	Reims
10249	TOMSP	Münster
10250	HANAR	Rio de Janeiro
10251	VICTE	Lyon
10252	SUPRD	Charleroi

1 of 40 pages (200 items)

### Details of 10250

Order ID:  Customer ID:

Employee ID:  Ship City:

Before the template elements are converted to JS controls

OrderID	CustomerID	ShipCity
10248	VINET	Reims
10249	TOMSP	Münster
10250	HANAR	Rio de Janeiro
10251	VICTE	Lyon
10252	SUPRD	Charleroi

1 of 40 pages (200 items)

### Details of 10250

Order ID:  Customer ID:

Employee ID:  Ship City:

After the template elements are converted to JS controls using `actionComplete` event

#### *Batch / Excel-like*

Users can start editing by clicking a cell and typing data into it. Edited cell will be marked while navigating to next cell or any other row, so that you know which fields or cells has been edited. Set `editMode` as `batch` to enable batch editing.

**Note:** `getBatchChanges` method of grid holds the unsaved record changes.

Refer the KB [link](#) for "How to suppress grid confirmation messages" in batch mode.

The following code example describes the above behavior.

#### HTML

```
<div id="Grid"></div>
<script type="text/babel" src="app.jsx">
</script>
<div ng-controller="BatchCtrl">
```

Create a JSX file and paste the following content

#### JAVASCRIPT

```
var editSettings = { allowEditing: true, allowAdding: true, allowDeleting:
true, editMode:"batch" };
var toolbarItems = { showToolBar: true, toolbarItems: ["add", "edit",
"delete", "update", "cancel"] };
```

```

ReactDOM.render(
  //The datasource "window.gridData" is referred from
  'http://js.syncfusion.com/demos/web/scripts/jsondata.min.js'
  <EJ.Grid dataSource = {window.gridData} allowPaging = {true}
  editSettings={editSettings} toolbarSettings={toolbarItems}>
  <columns>
  <column field="OrderID" isPrimaryKey={true} />
  <column field="CustomerID" />
  <column field="Freight" editType="numericedit" />
  <column field="ShipCountry" editType="dropdownedit"/>
  <column field="OrderDate" editType="datepicker" format="{0:MM/dd/yyyy}"/>
  </columns>
  </EJ.Grid>,
  document.getElementById('Grid')
);

```

The following output is displayed as a result of the above code example.

OrderID	CustomerID	Freight	ShipCountry	OrderDate
10248	VINET	32.38	France	04/07/1996
10249	TOMSP	11.61	Germany	05/07/1996
10250	CHOPS	65.83	Brazil	08/07/1996
10251	CENTC	41.34	France	08/07/1996
10252	SUPRD	51.3	Belgium	09/07/1996
10253	HANAR	58.17	Belgium	10/07/1996
10254	CHOPS	22.98	Switzerland	11/07/1996
10255	RICSU	150	Switzerland	12/07/1996
10256	WELLI	13.97	Brazil	09/09/2015
10257	HILAA	81.91	Venezuela	24/07/1996
10258	ERNSH	140.51	Austria	17/07/1996
10259	CENTC	3.25	Mexico	18/07/1996

1 of 17 pages (200 items)

### Confirmation messages

To show the confirm dialog while saving or discarding the batch changes (discarding during the grid action like filtering, sorting and paging), set [showConfirmDialog](#) as `true`.

**Note:** [showConfirmDialog](#) property is only for batch editing mode.

The following code example describes the above behavior.

**HTML**

```
<div id="Grid"></div>
<script type="text/babel" src="app.jsx">
</script>
```

Create a JSX file and paste the following content

**JAVASCRIPT**

```
var editSettings = { allowEditing: true, allowAdding: true, allowDeleting:
true, editMode:"batch",showConfirmDialog:true };
var toolbarItems = { showToolbar: true, toolbarItems: ["add", "edit",
"delete", "update", "cancel"] };
ReactDOM.render(
//The datasource "window.gridData" is referred from
'http://js.syncfusion.com/demos/web/scripts/jsondata.min.js'
<EJ.Grid dataSource = {window.gridData} allowPaging = {true}
editSettings={editSettings} toolbarSettings={toolbarItems}>
<columns>
<column field="OrderID" isPrimaryKey={true} />
<column field="CustomerID" />
<column field="Freight" editType="numericedit" />
<column field="ShipCountry" editType="dropdownedit"/>
<column field="OrderDate" editType="datepicker" format="{0:MM/dd/yyyy}"/>
</columns>
</EJ.Grid>,
document.getElementById('Grid')
);
```

The following output is displayed as a result of the above code example.

OrderID	CustomerID	Freight	ShipCountry	OrderDate
10248	VINET	32.38	France	04/07/1996
10249	TOMSP	11.61	Germany	05/07/1996
10250	ERNSH	65.83	Argentina	08/07/1996
10251	CENTC	41.34	France	08/07/1996
10252	SUPRD	51.3	Belgium	17/07/1996
10253	HANAR			10/07/1996
10254	CHOPS		Switzerland	11/07/1996
10255	RICSU	148.33	Switzerland	12/07/1996
10256	RICSU	13.97	Brazil	15/07/1996
10257	HILAA	81.91	Venezuela	16/07/1996
10258	ERNSH	140.51	Austria	17/07/1996
10259	CENTC	3.25	Mexico	18/07/1996

Are you sure you want to save changes?

OK Cancel

1 of 17 pages (200 items)

To show delete confirm dialog while deleting a record, set [showDeleteConfirmDialog](#) as true.

**Note:** [showDeleteConfirmDialog](#) property is for all type of [editMode](#).

The following code example describes the above behavior.

#### HTML

```
<div id="Grid"></div>
<script type="text/babel" src="app.jsx">
</script>
```

Create a JSX file and paste the following content

#### JAVASCRIPT

```
var editSettings = { allowEditing: true, allowAdding: true, allowDeleting:
true, showDeleteConfirmDialog:true };
var toolbarItems = { showToolbar: true, toolbarItems: ["add", "edit",
"delete", "update", "cancel"] };
ReactDOM.render(
//The datasource "window.gridData" is referred from
'http://js.syncfusion.com/demos/web/scripts/jsondata.min.js'
<EJ.Grid dataSource = {window.gridData} allowPaging = {true}
editSettings={editSettings} toolbarSettings={toolbarItems}>
<columns>
<column field="OrderID" isPrimaryKey={true} />
```

```

<column field="CustomerID" />
<column field="Freight" editType="numericedit" />
<column field="ShipCountry" editType="dropdownedit"/>
<column field="OrderDate" editType="datepicker" format="{0:MM/dd/yyyy}"/>
</columns>
</EJ.Grid>,
document.getElementById('Grid')
);

```

The following output is displayed as a result of the above code example.

OrderID	CustomerID	Freight	ShipCountry	OrderDate
10248	VINET	32.38	France	04/07/1996
10249	TOMSP	11.61	Germany	05/07/1996
10250	HANAR	65.83	Brazil	08/07/1996
10251	VICTE	41.34	France	08/07/1996
10252	SUPRD	51.3	Belgium	09/07/1996
10253	HANAR			10/07/1996
10254	CHOPS		Switzerland	11/07/1996
10255	RICSU	148.33	Switzerland	12/07/1996
10256	WELLI	13.97	Brazil	15/07/1996
10257	HILAA	81.91	Venezuela	16/07/1996
10258	ERNSH	140.51	Austria	17/07/1996
10259	CENTC	3.25	Mexico	18/07/1996

Are you sure you want to Delete Record?

OK Cancel

1 2 3 4 5 6 7 8 ... 1 of 17 pages (200 items)

### Column Validation

We can validate the value of the added or edited record cell before saving.

The below validation script files are needed when editing is enabled with validation.

1. [jquery.validate.min.js](#)
2. [jquery.validate.unobtrusive.min.js](#)

### jQuery Validation

You can set validation rules using [validationRules](#) property of [columns](#). The following are jQuery validation methods.

### List of JQuery validation methods



Rules	Description
required	Requires an element.
remote	Requests a resource to check the element for validity.
minlength	Requires the element to be of given minimum length.
maxlength	Requires the element to be of given maximum length.
range	Requires the element to be in given value range.
min	The element requires a given minimum.
max	The element requires a given maximum.
range	Requires the element to be in a given value range.
email	The element requires a valid email.
url	The element requires a valid URL
date	Requires the element to be a date.
dateISO	The element requires an ISO date.
number	The element requires a decimal number.
digits	The element requires digits only.
creditcard	Requires the element to be a credit card number.
equalTo	Requires the element to be the same as another.

Grid supports all the standard validation methods of jQuery, please refer the jQuery validation documentation [link](#) for more information.

The following code example describes the above behavior.

#### HTML

```
<div id="Grid"></div>
<script type="text/babel" src="app.jsx">
</script>
```

Create a JSX file and paste the following content

#### JAVASCRIPT

```
var editSettings = { allowEditing: true, allowAdding: true, allowDeleting:
true, showDeleteConfirmDialog: true };
var toolbarItems = { showToolbar: true, toolbarItems: ["add", "edit",
"delete", "update", "cancel"] };
var requireValid = { required: true };
var lengthValid = { minlength: 3 };
var rangeValid = { range: [0, 1000] };
ReactDOM.render(
```

```
//The datasource "window.gridData" is referred from
'http://js.syncfusion.com/demos/web/scripts/jsondata.min.js'
<EJ.Grid dataSource = {window.gridData} allowPaging = {true}
editSettings={editSettings} toolbarSettings={toolbarItems}>
<columns>
<column field="OrderID" isPrimaryKey={true} validationRules={requireValid}
/>
<column field="CustomerID" validationRules={lengthValid}/>
<column field="ShipCity" />
<column field="Freight" editType="numericedit"
validationRules={rangeValid}/>
<column field="ShipCountry" />
</columns>
</EJ.Grid>,
document.getElementById('Grid')
);
```

The following output is displayed as a result of the above code example.

OrderID	CustomerID	ShipCity	Freight	ShipCountry
	VI		4,567	
OrderID is required.	Please enter at least 3 characters.	Reims	Please enter a value between 0 and 1000.	France
10249	TOMSP	Münster		Germany
10250	HANAR	Rio de Janeiro	65.83	Brazil
10251	VICTE	Lyon	41.34	France
10252	SUPRD	Charleroi	51.3	Belgium
10253	HANAR	Rio de Janeiro	58.17	Brazil
10254	CHOPS	Bern	22.98	Switzerland
10255	RICSU	Genève	148.33	Switzerland
10256	WELLI	Resende	13.97	Brazil
10257	HILAA	San Cristóbal	81.91	Venezuela
10258	ERNSH	Graz	140.51	Austria

1 of 17 pages (201 items)

### Persisting data in Server

Edited data can be persisted in database using RESTful web services.

All the CRUD operations in grid are done through DataManager. DataManager have an option to bind all the CRUD related data in server side. Please refer the [link](#) to know about the DataManager.

For you information ODataAdaptor persist data in server as per OData protocol.

In the below section, we have explained how to get the edited data details at the server side using URLAdaptor.

#### URL Adaptor

You can use the `UrlAdaptor` of `ejDataManager` when binding datasource from remote data. At initial load of Grid, using URL property of DataManager, data are fetched from remote data and bound to Grid. You can map CRUD operation in Grid to Server-Side Controller action using the properties `insertUrl`, `removeUrl`, `updateUrl`, `crudUrl` and `batchUrl`.

The following code example describes the above behavior.

#### HTML

```
<div id="Grid"></div>
<script type="text/babel" src="app.jsx">
</script>
```

Create a JSX file and paste the following content

#### JAVASCRIPT

```
var data = ej.DataManager({
  url: "/Home/DataSource",
  adaptor: "UrlAdaptor",
  updateUrl : "/Home/Update",
  insertUrl : "/Home/Insert",
  removeUrl : "/Home/Delete",
});
var editSettings = { allowEditing: true, allowAdding: true, allowDeleting:
true, showDeleteConfirmDialog:true};
var toolbarItems = { showToolbar: true, toolbarItems: ["add", "edit",
"delete", "update", "cancel"] };
ReactDOM.render(
<EJ.Grid dataSource = {data} allowPaging = {true}
editSettings={editSettings} toolbarSettings={toolbarItems}>
<columns>
<column field="OrderID" isPrimaryKey={true} />
<column field="CustomerID" />
<column field="EmployeeID" />
<column field="Freight" editType="numericedit" format="{0:C}" />
<column field="ShipName" />
<column field="ShipCountry" />
</columns>
</EJ.Grid>,
document.getElementById('Grid')
);
```

Also when you use `UrlAdaptor`, you need to return the data as `JSON` and the JSON object must contain a property as `result` with `dataSource` as its value and one more property `count` with the `dataSource` total records count as its value.

The following code example describes the above behavior.

#### C#

```

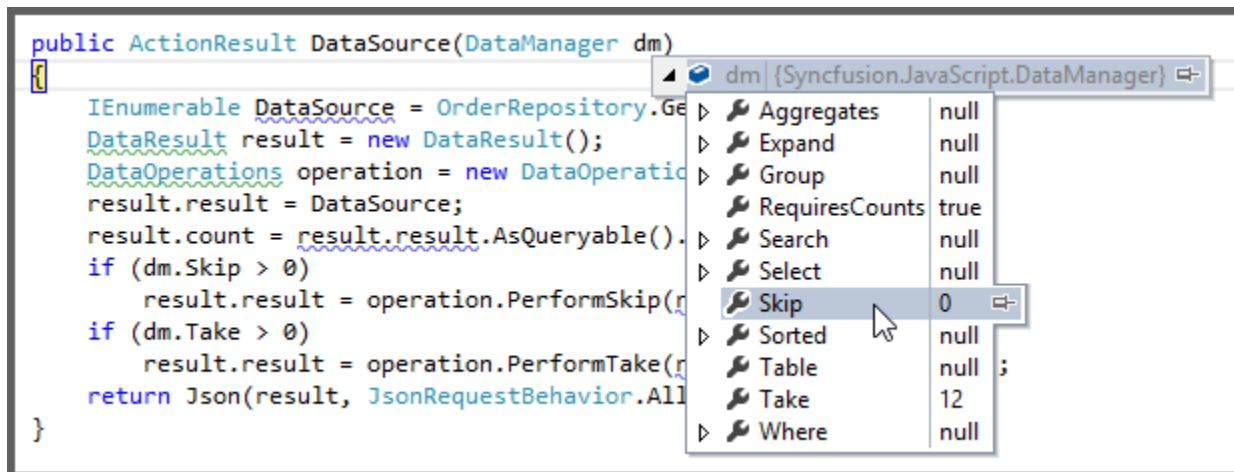
public ActionResult DataSource(DataManager data)
{
    IEnumerable DataSource = OrderRepository.GetAllRecords();
    DataResult result = new DataResult();
    DataOperations operation = new DataOperations();
    result.result = DataSource;
    result.count = result.result.AsQueryable().Count();
    if (data.Skip > 0)
        result.result = operation.PerformSkip(result.result, data.Skip);
    if (data.Take > 0)
        result.result = operation.PerformTake(result.result, data.Take);
    return Json(result, JsonRequestBehavior.AllowGet);
}

public class DataResult
{
    public IEnumerable result { get; set; }
    public int count { get; set; }
}

```

The grid actions (sorting, filtering, paging, searching, and aggregates) details are obtained in the 'DataManager' class. While initializing the grid, paging only enabled hence in the below screen shot paging details are bound to the DataManager class.

Please refer the below screen shot.



Also, using 'DataOperations' helper class you can perform grid action at server side. The in-built methods that we have provided in the DataOperations class are listed below.

1. PerformSorting
2. PerformFiltering
3. PerformSearching
4. PerformSkip
5. PerformTake
6. PerformWhereFilter
7. PerformSelect
8. Execute

*Accessing CRUD action request details in server side:*

The 'Server-Side' function must be declared with the following parameter name for each editing functionality.

Parameters Table

Action	Parameter Name	Example
Update,Insert	value	public ActionResult Update(EditableOrder value){ }
	public ActionResult Insert(EditableOrder value){ }	
Remove	key	public ActionResult Remove(int key){ }
Batch Add	added	public ActionResult BatchUpdate(string action, List <editableorder> added, List <editableorder> changed, List <editableorder> deleted, int? key){ }
Batch Update	changed	
Batch Delete	deleted	
Crud Update,Crud Insert	value, action	public ActionResult CrudUrl(EditableOrder value, string action){ }
Crud Remove	action, key, keyColumn	public ActionResult CrudUrl(string action, int? key, string keyColumn){ }
Crud Remove - Multi Delete	action, key, deleted	public ActionResult CrudUrl(string action, string key, List deleted){ }

*Insert Record:*

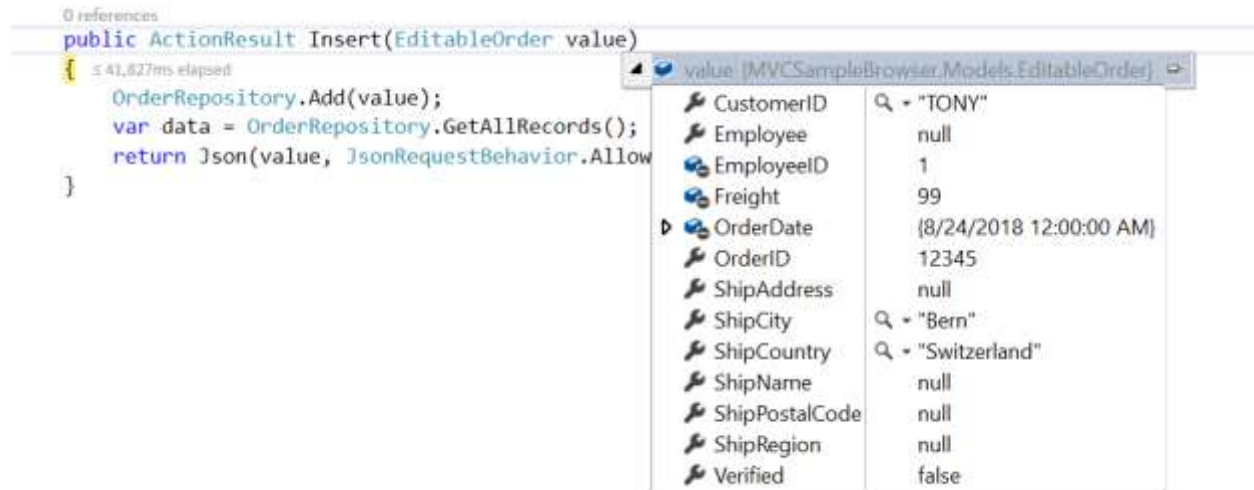
Using `insertUrl` property, you can specify the controller action mapping URL to perform insert operation at server side.

The following code example describes the above behavior.

**C#**

```
public ActionResult Insert(EditableOrder value)
{
    OrderRepository.Add(value);
    var data = OrderRepository.GetAllRecords();
    return Json(value, JsonRequestBehavior.AllowGet);
}
```

The newly added record details are bound to the 'value' parameter. Please refer the below image.



#### Update Record:

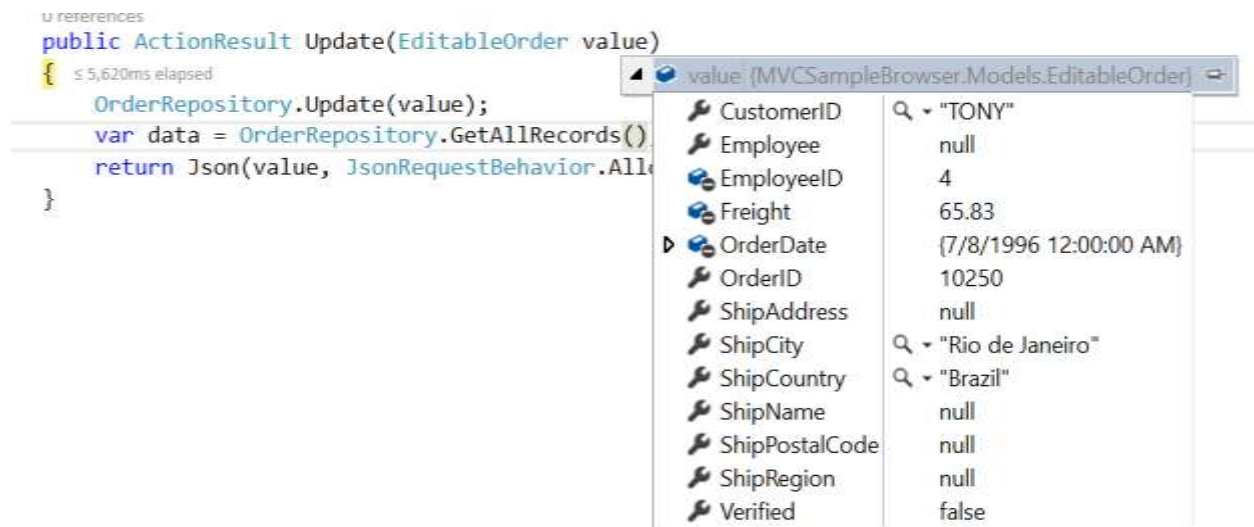
Using `updateUrl` property, you can specify the controller action mapping URL to perform save/update operation at server side.

The following code example describes the above behavior.

#### C#

```
public ActionResult Update(EditableOrder value)
{
    OrderRepository.Update(value);
    var data = OrderRepository.GetAllRecords();
    return Json(value, JsonRequestBehavior.AllowGet);
}
```

The updated record details are bound to the 'value' parameter. Please refer the below image.



*Delete Record:*

Using `removeUrl` property, you can specify the controller action mapping URL to perform delete operation at server side.

The following code example describes the above behavior.

**C#**

```
public ActionResult Remove(int key)
{
    OrderRepository.Delete(key);
    var data = OrderRepository.GetAllRecords();
    return Json(key, JsonRequestBehavior.AllowGet);
}
```

The deleted record primary key value is bound to the 'key' parameter. Please refer the below image.

0 references

```
public ActionResult Remove(int key)
{
    OrderRepository.Delete(key);
    var data = OrderRepository.GetAllRecords();
    return Json(key, JsonRequestBehavior.AllowGet);
}
```

≤ 9,693ms elapsed

key 10248

*CRUD URL:*

Instead of specifying separate controller action method for CRUD (insert, update and delete) operation, using `crudUrl` property you can specify the controller action mapping URL to perform all the CRUD operation at server side using single method.

The action parameter of `crudUrl` is used to get the corresponding CRUD action.

The following code example describes the above behavior.

**HTML**

```
<div id="Grid"></div>
<script type="text/babel" src="app.jsx">
</script>
```

Create a JSX file and paste the following content

**JAVASCRIPT**

```
var data = ej.DataManager({
    url: "/Home/DataSource",
    adaptor: "UrlAdaptor",
    crudUrl : "Home/CrudUpdate"
});
var editSettings = { allowEditing: true, allowAdding: true, allowDeleting: true, showDeleteConfirmDialog: true };
var toolbarItems = { showToolbar: true, toolbarItems: ["add", "edit", "delete", "update", "cancel"] };
ReactDOM.render(
```

```

<EJ.Grid dataSource = {data} allowPaging = {true}
editSettings={editSettings} toolbarSettings={toolbarItems}>
<columns>
<column field="OrderID" isPrimaryKey={true} />
<column field="CustomerID" />
<column field="EmployeeID" />
<column field="Freight" editType="numericedit" format="{0:C}" />
<column field="ShipName" />
<column field="ShipCountry" />
</columns>
</EJ.Grid>,
document.getElementById('Grid')
);

```

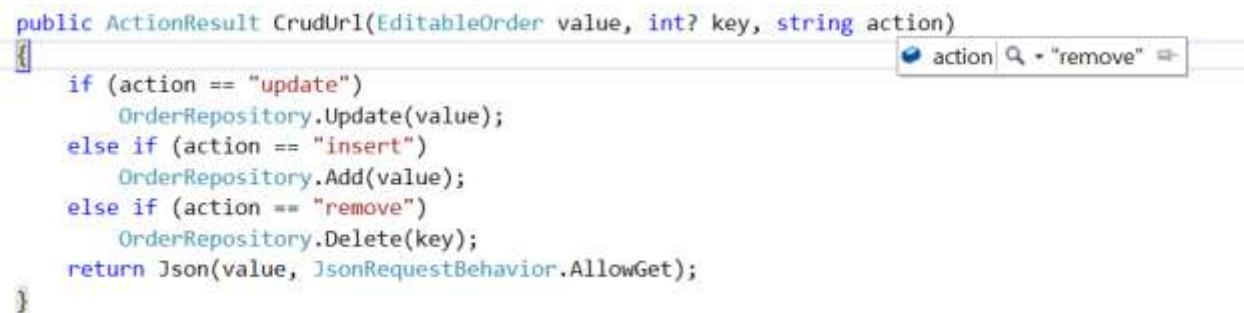
### C#

```

public ActionResult CrudUpdate(EditableOrder value, string action, int key)
{
    if (action == "update")
        OrderRepository.Update(value);
    else if (action == "insert")
        OrderRepository.Add(value);
    else if (action == "remove")
        OrderRepository.Delete(key);
    return Json(value, JsonRequestBehavior.AllowGet);
}

```

Please refer the below image to know about the action parameter



```

public ActionResult CrudUrl(EditableOrder value, int? key, string action)
{
    if (action == "update")
        OrderRepository.Update(value);
    else if (action == "insert")
        OrderRepository.Add(value);
    else if (action == "remove")
        OrderRepository.Delete(key);
    return Json(value, JsonRequestBehavior.AllowGet);
}

```

**Note:** If you specify `insertUrl` along with `CrudUrl` then while adding `insertUrl` only called.

#### Batch URL:

The `batchUrl` property supports only for batch editing mode. You can specify the controller action mapping URL to perform Batch operation at server side.

The following code example describes the above behavior.

### HTML

```

<div id="Grid"></div>
<script type="text/babel" src="app.jsx">
</script>

```



Create a JSX file and paste the following content

### JAVASCRIPT

```
var data = ej.DataManager({
  url: "/Home/DataSource",
  adaptor: "UrlAdaptor",
  batchUrl : "Home/BatchUpdate"
});
var editSettings = { allowEditing: true, allowAdding: true, allowDeleting:
true, showDeleteConfirmDialog: true };
var toolbarItems = { showToolbar: true, toolbarItems: ["add", "edit",
"delete", "update", "cancel"] };
ReactDOM.render(
<EJ.Grid dataSource = {data} allowPaging = {true}
editSettings={editSettings} toolbarSettings={toolbarItems}>
<columns>
<column field="OrderID" isPrimaryKey={true} />
<column field="CustomerID" />
<column field="EmployeeID" />
<column field="Freight" editType="numericedit" format="{0:C}" />
<column field="ShipName" />
<column field="ShipCountry" />
</columns>
</EJ.Grid>,
document.getElementById('Grid')
);
```

### C#

```
public ActionResult BatchUpdate(string action, List<EditableOrder> added,
List<EditableOrder> changed, List<EditableOrder> deleted, int? key)
{
  if (changed != null)
    OrderRepository.Update(changed);
  if (deleted != null)
    OrderRepository.Delete(deleted);
  if (added != null)
    OrderRepository.Add(added);
  var data = OrderRepository.GetComplexRecords();
  return Json(new { changed = changed, added = added, deleted = deleted },
    JsonRequestBehavior.AllowGet);
}
```

Please refer the below image for more information about batch parameters



### Adding New Row Position

To add new row in the top or bottom position of grid content, set [rowPosition](#) property of [editSettings](#) depending on the requirement.

The following code example describes the above behavior.

#### HTML

```
<div id="Grid"></div>
<script type="text/babel" src="app.jsx">
</script>
```

Create a JSX file and paste the following content

#### JAVASCRIPT

```
var editSettings = { allowEditing: true, allowAdding: true, allowDeleting:
true, rowPosition:"bottom" };
var toolbarItems = { showToolbar: true, toolbarItems: ["add", "edit",
"delete", "update", "cancel"] };
ReactDOM.render(
//The datasource "window.gridData" is referred from
'http://js.syncfusion.com/demos/web/scripts/jsondata.min.js'
<EJ.Grid dataSource = {window.gridData} allowPaging = {true}
editSettings={editSettings} toolbarSettings={toolbarItems}>
<columns>
<column field="OrderID" isPrimaryKey={true} />
<column field="CustomerID" />
<column field="ShipCity" />
<column field="Freight" editType="numericedit" />
<column field="ShipCountry" />
</columns>
</EJ.Grid>,
document.getElementById('Grid')
);
```

The following output is displayed as a result of the above code example.

OrderID	CustomerID	ShipCity	Freight	ShipCountry
10248	VINET	Reims	32.38	France
10249	TOMSP	Münster	11.61	Germany
10250	HANAR	Rio de Janeiro	65.83	Brazil
10251	VICTE	Lyon	41.34	France
10252	SUPRD	Charleroi	51.3	Belgium
10253	HANAR	Rio de Janeiro	58.17	Brazil
10254	CHOPS	Bern	22.98	Switzerland
10255	RICSU	Genève	148.33	Switzerland
10256	WELLI	Resende	13.97	Brazil
10257	HILAA	San Cristóbal	81.91	Venezuela
10258	ERNSH	Graz	140.51	Austria
<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text" value="Enter value"/>	<input type="text"/>

1 2 3 4 5 6 7 8 ... 17

1 of 17 pages (201 items)

Render with blank row for easy add new

The blank add new row is displayed in the grid content during grid initialization itself to add a new record easily. To enable show add new row by default, set [showAddNewRow](#) property of [editSettings](#) as `true`.

The blank add new row is displayed either in the top or bottom of the corresponding page, its position is based on the [rowPosition](#) property of [editSettings](#).

The following code example describes the above behavior.

### HTML

```
<div id="Grid"></div>
<script type="text/babel" src="app.jsx">
</script>
```

Create a JSX file and paste the following content

### JAVASCRIPT

```
var editSettings = { allowEditing: true, allowAdding: true, allowDeleting:
true, showAddNewRow:true};
var toolbarItems = { showToolbar: true, toolbarItems: ["add", "edit",
"delete", "update", "cancel"] };
ReactDOM.render(
```

```
//The datasource "window.gridData" is referred from
'http://js.syncfusion.com/demos/web/scripts/jsondata.min.js'
<EJ.Grid dataSource = {window.gridData} allowPaging = {true}
editSettings={editSettings} toolbarSettings={toolbarItems}>
<columns>
<column field="OrderID" isPrimaryKey={true} />
<column field="CustomerID" />
<column field="ShipCity" />
<column field="Freight" editType="numericedit" />
<column field="ShipCountry" />
</columns>
</EJ.Grid>,
document.getElementById('Grid')
);
```

The following output is displayed as a result of the above code example.

OrderID	CustomerID	ShipCity	Freight	ShipCountry
			Enter value	
10248	VINET	Reims	32.38	France
10249	TOMSP	Münster	11.61	Germany
10250	HANAR	Rio de Janeiro	65.83	Brazil
10251	VICTE	Lyon	41.34	France
10252	SUPRD	Charleroi	51.3	Belgium
10253	HANAR	Rio de Janeiro	58.17	Brazil
10254	CHOPS	Bern	22.98	Switzerland
10255	RICSU	Genève	148.33	Switzerland
10256	WELLI	Resende	13.97	Brazil
10257	HILAA	San Cristóbal	81.91	Venezuela
10258	ERNSH	Graz	140.51	Austria
10259	CENTC	México D.F.	3.25	Mexico

1 of 17 pages (200 items)

**Note:** 1. If it is remote, then the newly added record is placed based on the index from current view data.

2. If it is local, then the newly added record is added at the top of the page even if the added new [rowPosition](#) is mentioned as "bottom".

### Default column values on add new

While adding new record in grid, there is an option to set the default value for the columns. Using [defaultValue](#) property of [columns](#) you can set the default values for that particular column while editing or adding a new row.

The following code example describes the above behavior.

#### HTML

```
<div id="Grid"></div>
<script type="text/babel" src="app.jsx">
</script>
```

Create a JSX file and paste the following content

#### JAVASCRIPT

```
var editSettings = { allowEditing: true, allowAdding: true, allowDeleting:
true };
var toolbarItems = { showToolbar: true, toolbarItems: ["add", "edit",
"delete", "update", "cancel"] };
ReactDOM.render(
//The datasource "window.gridData" is referred from
'http://js.syncfusion.com/demos/web/scripts/jsondata.min.js'
<EJ.Grid dataSource = {window.gridData} allowPaging = {true}
editSettings={editSettings} toolbarSettings={toolbarItems}>
<columns>
<column field="OrderID" isPrimaryKey={true} />
<column field="CustomerID" />
<column field="ShipCity" defaultValue="Bern" />
<column field="Freight" editType="numericedit" defaultValue={45} />
<column field="ShipCountry" defaultValue="Brazil"/>
</columns>
</EJ.Grid>,
document.getElementById('Grid')
);
```

The following output is displayed as a result of the above code example.

OrderID	CustomerID	ShipCity	Freight	ShipCountry
		Bern	45	Brazil
10248	VINET	Reims	32.38	France
10249	TOMSP	Münster	11.61	Germany
10250	HANAR	Rio de Janeiro	65.83	Brazil
10251	VICTE	Lyon	41.34	France
10252	SUPRD	Charleroi	51.3	Belgium
10253	HANAR	Rio de Janeiro	58.17	Brazil
10254	CHOPS	Bern	22.98	Switzerland
10255	RICSU	Genève	148.33	Switzerland
10256	WELLI	Resende	13.97	Brazil
10257	HILAA	San Cristóbal	81.91	Venezuela
10258	ERNSH	Graz	140.51	Austria

1 2 3 4 5 6 7 8 ... 1 of 17 pages (201 items)

## Filtering

Filtering helps to view particular or related records from dataSource which meets a given filtering criteria. To enable filter, set `allowFiltering` as `true`.

The Grid supports three types of filter, they are

1. Filter bar
2. Menu
3. Excel

And also four types of filter menu is available in all filter types, they are

1. String
2. Numeric
3. Date
4. Boolean

The corresponding filter menu is opened based on the column type.

**Note:** 1. Need to specify the `type` of column, when first record data value is empty or null otherwise the filter menu is not opened.

2. The default filter type is Filter bar, when `allowFiltering` is enabled and `filtertype` is not set.

The following code example describes the above behavior.

### HTML

```
<div id="Grid"></div>
<script type="text/babel" src="app.jsx">
</script>
```

Create a JSX file and paste the following content

### JAVASCRIPT

```
ReactDOM.render(
  //The datasource "window.gridData" is referred from
  'http://js.syncfusion.com/demos/web/scripts/jsondata.min.js'
  <EJ.Grid dataSource = {window.gridData} allowPaging = {true}
  allowFiltering={true}>
    <columns>
      <column field="OrderID" />
      <column field="EmployeeID" />
      <column field="CustomerID" />
      <column field="ShipCountry" />
      <column field="Freight" />
    </columns>
  </EJ.Grid>,
  document.getElementById('Grid')
);
```

The following output is displayed as a result of the above code example.

OrderID	EmployeeID	CustomerID	ShipCountry	Freight
<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>
10248	5	VINET	France	32.38
10249	6	TOMSP	Germany	11.61
10250	4	HANAR	Brazil	65.83
10251	3	VICTE	France	41.34
10252	4	SUPRD	Belgium	51.3
10253	3	HANAR	Brazil	58.17
10254	5	CHOPS	Switzerland	22.98
10255	9	RICSU	Switzerland	148.33
10256	3	WELLI	Brazil	13.97
10257	4	HILAA	Venezuela	81.91
10258	1	ERNSH	Austria	140.51
10259	4	CENTC	Mexico	3.25

« ◀ 1 2 3 4 5 6 7 8 ... ▶ »
1 of 17 pages (200 items)

### Menu filter

You can enable menu filter by setting [filterType](#) of `[filterSettings]` as `menu`.

There is an option to show or hide the additional filter options in the menu by setting [showPredicate](#) of `[filterSettings]` as `true` or `false` respectively.

**Note:** For [filterType](#) property you can assign either `string` value ("menu") or `enum` value (ej.Grid.FilterType.Menu).

We can also filter a specified range of values by using the `between` operator for the column type `number`, `date` and `datetime`.

The following code example describes the above behavior.

### HTML

```
<div id="Grid"></div>
<script type="text/babel" src="app.jsx">
</script>
```

Create a JSX file and paste the following content

### JAVASCRIPT

```
var filterType = {filterType:"menu"};
ReactDOM.render(
```



```
//The datasource "window.gridData" is referred from
'http://js.syncfusion.com/demos/web/scripts/jsondata.min.js'
<EJ.Grid dataSource = {window.gridData} allowPaging = {true}
allowFiltering={true} filterSettings={filterType}>
<columns>
<column field="OrderID" />
<column field="EmployeeID" />
<column field="CustomerID" />
<column field="OrderDate" format="{0:MM/dd/yyyy}" />
<column field="Verified" />
</columns>
</EJ.Grid>,
document.getElementById('Grid')
);
```

The following output is displayed as a result of the above code example.

OrderID	EmployeeID	CustomerID	OrderDate	Verified
10248	5			<input checked="" type="checkbox"/>
10249	6			<input type="checkbox"/>
10250	4			<input checked="" type="checkbox"/>
10251	3			<input checked="" type="checkbox"/>
10252	4			<input checked="" type="checkbox"/>
10253	3			<input checked="" type="checkbox"/>
10254	5	CHOPS	11/07/1996	<input type="checkbox"/>
10255	9	RICSU	12/07/1996	<input checked="" type="checkbox"/>
10256	3	WELLI	15/07/1996	<input type="checkbox"/>
10257	4	HILAA	16/07/1996	<input checked="" type="checkbox"/>
10258	1	ERNSH	17/07/1996	<input checked="" type="checkbox"/>
10259	4	CENTC	18/07/1996	<input type="checkbox"/>

1 of 17 pages (200 items)

### Numeric Filter

OrderID	EmployeeID	CustomerID	OrderDate	Verified
10248	5	VINET		
10249	6	TOMSP		
10250	4	HANAR		
10251	3	VICTE		
10252	4	SUPRD		
10253	3	HANAR	10/07/1996	<input checked="" type="checkbox"/>
10254	5	CHOPS	11/07/1996	<input type="checkbox"/>
10255	9	RICSU	12/07/1996	<input checked="" type="checkbox"/>
10256	3	WELLI	15/07/1996	<input type="checkbox"/>
10257	4	HILAA	16/07/1996	<input checked="" type="checkbox"/>
10258	1	ERNSH	17/07/1996	<input checked="" type="checkbox"/>
10259	4	CENTC	18/07/1996	<input type="checkbox"/>

1 of 17 pages (200 items)

## String Filter

OrderID	EmployeeID	CustomerID	OrderDate	Verified
10248	5	VINET	04/07/1996	
10249	6	TOMSP	05/07/1996	
10250	4	HANAR	08/07/1996	
10251	3	VICTE	08/07/1996	
10252	4	SUPRD	09/07/1996	
10253	3	HANAR	10/07/1996	
10254	5	CHOPS	11/07/1996	<input type="checkbox"/>
10255	9	RICSU	12/07/1996	<input checked="" type="checkbox"/>
10256	3	WELLI	15/07/1996	<input type="checkbox"/>
10257	4	HILAA	16/07/1996	<input checked="" type="checkbox"/>
10258	1	ERNSH	17/07/1996	<input checked="" type="checkbox"/>
10259	4	CENTC	18/07/1996	<input type="checkbox"/>

1 of 17 pages (200 items)

## Date Filter

OrderID	EmployeeID	CustomerID	OrderDate	Verified
10248	5	VINET	04/07/1996	<input checked="" type="checkbox"/>
10249	6	TOMSP	05/07/1996	<input type="checkbox"/>
10250	4	HANAR	08/07/1996	<input checked="" type="checkbox"/>
10251	3	VICTE	08/07/1996	<input checked="" type="checkbox"/>
10252	4	SUPRD	09/07/1996	<input checked="" type="checkbox"/>
10253	3	HANAR	10/07/1996	<input checked="" type="checkbox"/>
10254	5	CHOPS	11/07/1996	<input type="checkbox"/>
10255	9	RICSU	12/07/1996	<input checked="" type="checkbox"/>
10256	3	WELLI	15/07/1996	<input type="checkbox"/>
10257	4	HILAA	16/07/1996	<input checked="" type="checkbox"/>
10258	1	ERNSH	17/07/1996	<input checked="" type="checkbox"/>
10259	4	CENTC	18/07/1996	<input type="checkbox"/>

Filter Value :

1 of 17 pages (200 items)

### Boolean Filter

#### Excel-like filter

You can enable excel menu by setting of `filterType` of `[filterSettings]` as `excel`. The excel menu contains an option such as Sorting, Clear filter, submenu for advanced filtering.

The following code example describes the above behavior.

#### HTML

```
<div id="Grid"></div>
<script type="text/babel" src="app.jsx">
</script>
```

Create a JSX file and paste the following content

#### JAVASCRIPT

```
var filterType = {filterType:"excel"};
ReactDOM.render(
  //The datasource "window.gridData" is referred from
  'http://js.syncfusion.com/demos/web/scripts/jsondata.min.js'
  <EJ.Grid dataSource = {window.gridData} allowPaging = {true}
  allowFiltering={true} filterSettings={filterType}>
    <columns>
      <column field="OrderID" />
      <column field="EmployeeID" />
      <column field="CustomerID" />
      <column field="ShipCountry"/>
      <column field="Freight" />
    </columns>
  </EJ.Grid>,
  document.getElementById('Grid')
);
```

The following output is displayed as a result of the above code example.

OrderID	EmployeeID	CustomerID	ShipCountry	Freight
10248	5			32.38
10249	6			11.61
10250	4			65.83
10251	3			41.34
10252	4			51.3
10253	3			58.17
10254	5			22.98
10255	9			148.33
10256	3			13.97
10257	4			81.91
10258	1	ERINSH	Austria	140.51
10259	4	CENTC	Mexico	3.25

Sort Smallest to Largest

Sort Largest to Smallest

Clear Filter

Number Filters

Search

☒ (Select All)
 ☒ 1
 ☒ 2
 ☒ 3
 ☒ 4
 ☒ 5

OKCancel

12345678...

1 of 17 pages (200 items)

#### Checkbox list generation:

By default, the checkbox list is generated from distinct values of the filter column from dataSource which gives an option to search and select the required items.

Also on checkbox list generation, if the number of distinct values are greater than 1000, then the excel filter will display only first 1000 values to ensure the best performance on rendering and searching. However this limit has been customized according to your requirement by setting [maxFilterChoices](#) of [filterSettings] with required limit in integer.

**Note:** 1. Using excel filter events you can change the dataSource of the checkbox list.

2. [ej.Query](#) of checkbox list can also be changed using excel filter events.

The following code example describes the above behavior.

#### HTML

```
<div id="Grid"></div>
<script type="text/babel" src="app.jsx">
</script>
```

Create a JSX file and paste the following content

#### JAVASCRIPT

```
var filterType = {filterType:"excel", maxFilterChoices : 4 };
ReactDOM.render(
```

```
//The datasource "window.gridData" is referred from
'http://js.syncfusion.com/demos/web/scripts/jsondata.min.js'
<EJ.Grid dataSource = {window.gridData} allowPaging = {true}
allowFiltering={true} filterSettings={filterType}>
<columns>
<column field="OrderID" />
<column field="EmployeeID" />
<column field="CustomerID" />
<column field="ShipCountry"/>
<column field="Freight" />
</columns>
</EJ.Grid>,
document.getElementById('Grid')
);
```

The following output is displayed as a result of the above code example.

OrderID	EmployeeID	CustomerID	ShipCountry	Freight
10248	5			32.38
10249	6			11.61
10250	4			65.83
10251	3			41.34
10252	4			51.3
10253	3			58.17
10254	5			22.98
10255	9			148.33
10256	3			13.97
10257	4			81.91
10258	1			140.51
10259	4	CENTC	Mexico	3.25

A Z Sort Smallest to Largest  
 Z A Sort Largest to Smallest  
 Clear Filter  
 Number Filters  
 Search  
 Not all items showing  
☒ (Select All)  
☒ 1  
☒ 2  
☒ 3  
☒ 4  
 OK Cancel

1 of 17 pages (200 items)

*Add current selection to filter checkbox:*

When filtering is done multiple times on the same column then the previously filtered values on the column will be cleared. So, to retain the old values Add current selection to filter checkbox can be used which is displayed when data is searched in the search bar.

The following image describes the above mentioned behavior.

Order ID	Customer ID	Employee ID
10278	BERGS	8
10280	BERGS	2
10355	AROUT	6
10365	ANTON	3
10383	AROUT	8
10384	BERGS	3
10444	BERGS	3
10445	BERGS	3

Clear Filter  
Text Filters  
BLONP  
☒ (Select All)  
☒ Add current selection to filter  
☒ BLONP  
OK Cancel

1 of 1 pages (8 items)

### Case Sensitivity

To perform filter operation with case sensitive in excel styled filter menu mode by setting [enableCaseSensitivity](#) as `true`.

The following code example describes the above behavior.

### HTML

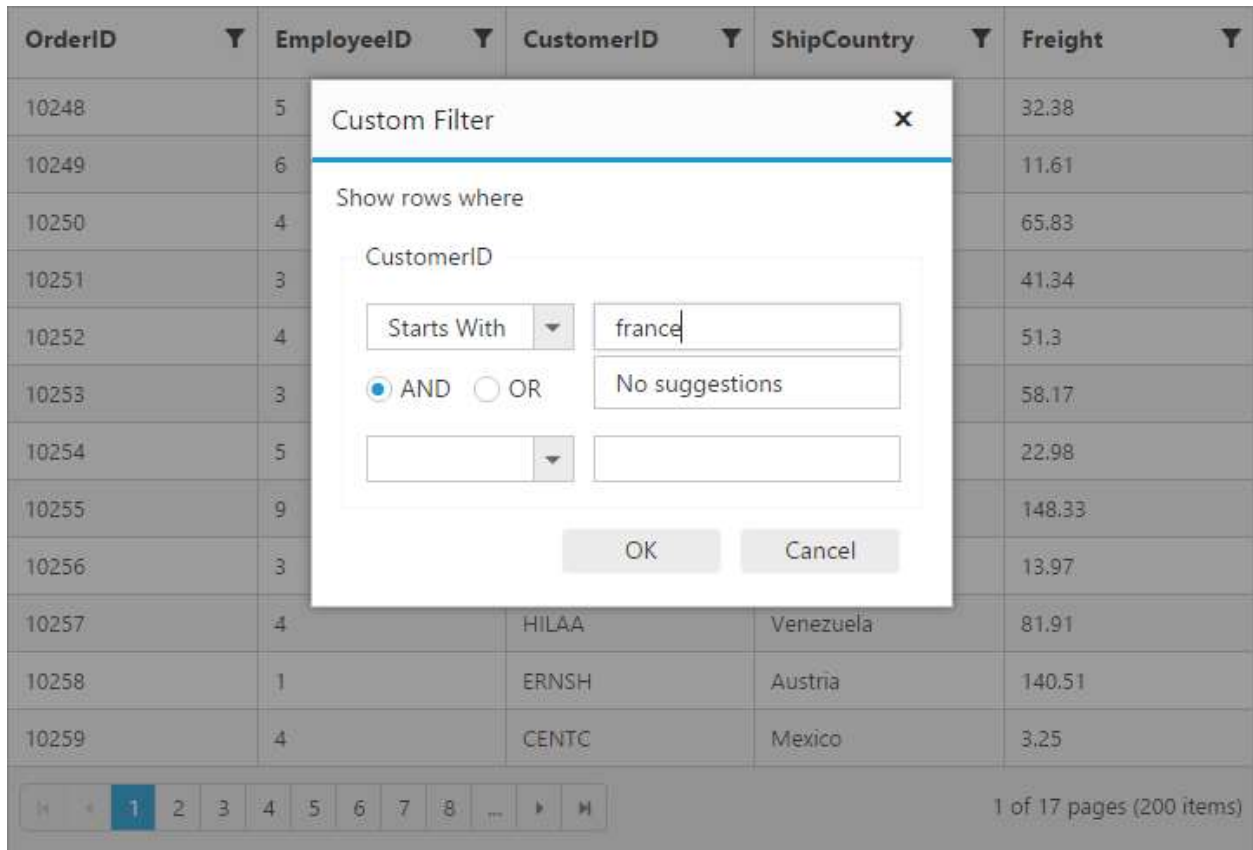
```
<div id="Grid"></div>
<script type="text/babel" src="app.jsx">
</script>
```

Create a JSX file and paste the following content

### JAVASCRIPT

```
var filterType = {filterType: "excel", enableCaseSensitivity: true };
ReactDOM.render(
  //The datasource "window.gridData" is referred from
  'http://js.syncfusion.com/demos/web/scripts/jsondata.min.js'
  <EJ.Grid dataSource = {window.gridData} allowPaging = {true}
  allowFiltering={true} filterSettings={filterType}>
    <columns>
      <column field="OrderID" />
      <column field="EmployeeID" />
      <column field="CustomerID" />
      <column field="ShipCountry"/>
      <column field="Freight" />
    </columns>
  </EJ.Grid>,
  document.getElementById('Grid')
);
```

The following output is displayed as a result of the above code example.



### Filter bar

[Filterbar](#) row is located next to column header of grid. You can filter the records with different expressions depending upon the column type. To show the filter bar, set the [filterType](#) as `filterbar`.

List of Filter bar Expressions:

You can enter the below filter expressions manually in the filter bar.

Expression	Example	Description	Column Type
=	= value	equal	Numeric
!=	!= value	notequal	
>	> value	greaterthan	
<	< value	lessthan	
>=	>= value	greaterthanorequal	>
<=	<= value	lessthanorequal	
N/A	N/A	Always <code>startswith</code> operator will be used for string filter	String
N/A	N/A	Always <code>equal</code> operator will be used for Date filter	Date
N/A	N/A	Always <code>equal</code> operator will be used for Boolean filter	Boolean

The following code example describes the above behavior.

### HTML

```
<div id="Grid"></div>
<script type="text/babel" src="app.jsx">
</script>
```

Create a JSX file and paste the following content

### JAVASCRIPT

```
var filterType = {filterType:"filterbar"};
ReactDOM.render(
  //The datasource "window.gridData" is referred from
  'http://js.syncfusion.com/demos/web/scripts/jsondata.min.js'
  <EJ.Grid dataSource = {window.gridData} allowPaging = {true}
  allowFiltering={true} filterSettings={filterType}>
    <columns>
      <column field="OrderID" />
      <column field="EmployeeID" />
      <column field="CustomerID" />
      <column field="ShipCountry"/>
      <column field="Freight" />
    </columns>
  </EJ.Grid>,
  document.getElementById('Grid')
);
```

The following output is displayed as a result of the above code example.



OrderID	EmployeeID	CustomerID	ShipCountry	Freight
	>5			
10249	6	TOMSP	Germany	11.61
10255	9	RICSU	Switzerland	148.33
10262	8	RATTC	USA	48.29
10263	9	ERNSH	Austria	146.06
10264	6	FOLKO	Sweden	3.67
10268	8	GROSR	Venezuela	66.29
10271	6	SPLIR	USA	4.54
10272	6	RATTC	USA	98.03
10274	6	VINET	France	6.01
10276	8	TORTU	Mexico	13.84
10278	8	BERGS	Sweden	92.69
10279	8	LEHMS	Germany	25.83

1

2

3

4

5

6

1 of 6 pages (66 items)

EmployeeID: >5

Filter bar modes:

This specifies the grid to start the filter action while typing in the filter bar or after pressing the enter key based on [filterBarMode](#). There are two types of [filterBarMode](#), they are

1. OnEnter
2. Immediate

**Note:** For [filterBarMode](#) property you can assign either `string` value (onenter) or `enum` value (ej.Grid.FilterBarMode.OnEnter).

Filter bar message:

The filter bar message is supported only for the [filterType](#) as filterbar. The filtered data with column name is displayed in the grid pager itself. By default [showFilterBarStatus](#) is true.

The following code example describes the above behavior.

#### HTML

```
<div id="Grid"></div>
<script type="text/babel" src="app.jsx">
</script>
```

Create a JSX file and paste the following content

### JAVASCRIPT

```
var filterType = {showFilterBarStatus: true };
ReactDOM.render(
  //The datasource "window.gridData" is referred from
  'http://js.syncfusion.com/demos/web/scripts/jsondata.min.js'
  <EJ.Grid dataSource = {window.gridData} allowPaging = {true}
  allowFiltering={true} filterSettings={filterType}>
    <columns>
      <column field="OrderID" />
      <column field="EmployeeID" />
      <column field="CustomerID" />
      <column field="ShipCountry"/>
      <column field="Freight" />
    </columns>
  </EJ.Grid>,
  document.getElementById('Grid')
);
```

The following output is displayed as a result of the above code example.

OrderID	EmployeeID	CustomerID	ShipCountry	Freight
	5			
10248	5	VINET	France	32.38
10254	5	CHOPS	Switzerland	22.98
10269	5	WHITC	USA	4.56
10297	5	BLONP	France	5.74
10320	5	WARTH	Finland	34.57
10333	5	WARTH	Finland	0.59
10358	5	LAMAI	France	19.64
10359	5	SEVES	UK	288.43
10372	5	QUEEN	Brazil	890.78
10378	5	FOLKO	Sweden	5.44
10397	5	PRINI	Portugal	60.26

⏪

⏩

1

⏪

⏩

1 of 1 pages (11 items)

EmployeeID: 5

### Filter Operators

The grid controls uses filter operators from [ej.DataManager](#), which are used at the time of filtering.

List of Column type and Filter operators

Column Type	Filter Operators
Number	ej.FilterOperators.greaterThan
	ej.FilterOperators.greaterThanOrEqualTo
	ej.FilterOperators.lessThan
	ej.FilterOperators.lessThanOrEqualTo
	ej.FilterOperators.equal
	ej.FilterOperators.notEqual
String	ej.FilterOperators.startsWith
	ej.FilterOperators.endsWith
	ej.FilterOperators.contains
	ej.FilterOperators.equal
	ej.FilterOperators.notEqual
Boolean	ej.FilterOperators.equal
	ej.FilterOperators.notEqual
Date	ej.FilterOperators.greaterThan
	ej.FilterOperators.greaterThanOrEqualTo
	ej.FilterOperators.lessThan
	ej.FilterOperators.lessThanOrEqualTo
	ej.FilterOperators.equal
	ej.FilterOperators.notEqual

### FilterBar Template

Usually enabling `allowFiltering`, will create default textbox in Grid FilterBar. So, Using `[filterBarTemplate]` property of `column` we can render any other controls like AutoComplete, DropDownList etc in filterbar to filter the grid data for the particular column.

It has three functions. They are

1. `create` - It is used to create the control at time of initialize.
2. `read` - It is used to read the Filter value selected.
3. `write` - It is used to render the control and assign the value selected for filtering.

The following code example describes the above behavior.

### HTML

```
<div id="Grid"></div>
<script type="text/babel" src="app.jsx">
</script>
```

Create a JSX file and paste the following content

### JAVASCRIPT

```
var filterFreight = {
  write: function (args) {
    args.element.ejNumericTextbox({ width: "100%", decimalPlaces: 2, focusOut:
    ej.proxy(args.column.filterBarTemplate.read, this, args) });
  },
  read: function (args) {
    this.filterColumn(args.column.field, "equal", args.element.val(), "and",
    true)
  }
};

var filtercustomer = {
  create: function (args) {
    return "<input>"
  },
  write: function (args) {
    var data = ej.DataManager(window.gridData).executeLocal(new
    ej.Query().select("CustomerID"));
    args.element.ejAutocomplete({ width: "100%", dataSource: data,
    enableDistinct: true, focusOut: ej.proxy(args.column.filterBarTemplate.read,
    this, args) });
  },
  read: function (args) {
    this.filterColumn(args.column.field, "equal", args.element.val(), "and",
    true)
  }
};

var filteremployee = {
  write: function (args) {
    var data = [{ text: "clear", value: "clear" }, { text: "1", value: 1 }, {
    text: "2", value: 2 }, { text: "3", value: 3 }, { text: "4", value: 4 },
    { text: "5", value: 5 }, { text: "6", value: 6 }, { text: "7", value: 7 }, {
    text: "8", value: 8 }, { text: "9", value: 9 }
    ]
    args.element.ejDropDownList({ width: "100%", dataSource: data, change:
    ej.proxy(args.column.filterBarTemplate.read, this, args) })
  },
  read: function (args) {
    if (args.element.val() == "clear") {
      this.clearFiltering(args.column.field);
      args.element.val("")
    }
    this.filterColumn(args.column.field, "equal", args.element.val(), "and",
    true)
  }
};
```

```

ReactDOM.render(
  //The datasource "window.gridData" is referred from
  'http://js.syncfusion.com/demos/web/scripts/jsondata.min.js'
  <EJ.Grid dataSource = {window.gridData} allowPaging = {true}
  allowFiltering={true}>
    <columns>
      <column field="OrderID" headerText="Order ID" textAlign="right" width={90}/>
      <column field="CustomerID" headerText="CustomerID" textAlign="left"
      width={90} filterBarTemplate={filtercustomer} />
      <column field="EmployeeID" headerText="EmployeeID" textAlign="left"
      width={90} filterBarTemplate={filteremployee} />
      <column field="Freight" headerText="Freight" textAlign="left"
      format="{0:C2}" width={90} filterBarTemplate={filterFreight} />
      <column field="ShipCountry" headerText="Ship Country" textAlign="left"
      width={90} />
      <column field="Verified" headerText="EmployeeID" width={90} />
    </columns>
  </EJ.Grid>,
  document.getElementById('Grid')
);

```

The following output is displayed as a result of the above code example.

Order ID	CustomerID	EmployeeID	Freight	Ship Country	Verified
		clear	58.17		<input type="checkbox"/>
10253	HANAR	3	58.17	Brazil	<input checked="" type="checkbox"/>

1 of 1 pages (1 items)

Freight: 58.17

After Filtering

### Selection

Selection provides an interactive support to highlight the row, cell or column that you select. Selection can be done through simple Mouse down or Keyboard interaction. To enable selection, set [allowSelection](#) as `true`.

### Types of Selection

There are two types of selections available in grid. They are

1. Single
2. Multiple

### Single Selection

Single selection is an interactive support to select a specific row, cell or column in grid by mouse or keyboard interactions. To enable single selection, set [selectionType](#) property as `single` and also set [allowSelection](#) property as `true`.

### Multiple Selections

Multiple selections is an interactive support to select a group of rows, cells or columns in grid by mouse or keyboard interactions. To enable multiple selections, set [selectionType](#) property as `multiple` and also set [allowSelection](#) property as `true`.

### Row Selection

Row selection is enabled by setting [selectionMode](#) property of [selectionSettings](#) as `row`. For random row selection, press **“Ctrl + mouse left”** click and for continuous row selection press **“Shift + mouse left”** click on the grid rows. To unselect selected rows, press **“Ctrl + mouse left”** click on selected row.

The following code example describes the above behavior.

### HTML

```
<div id="Grid"></div>
<script type="text/babel" src="app.jsx">
</script>
```

Create a JSX file and paste the following content

### JAVASCRIPT

```
var selectionMode = { selectionMode: ["row"] };
ReactDOM.render(
  //The datasource "window.gridData" is referred from
  'http://js.syncfusion.com/demos/web/scripts/jsondata.min.js'
  <EJ.Grid dataSource = {window.gridData} allowPaging = {true}
  allowSelection={true} selectionSettings={selectionMode}
  selectionType="multiple">
    <columns>
      <column field="OrderID" />
      <column field="EmployeeID" />
      <column field="ShipCity" />
      <column field="ShipCountry"/>
      <column field="Freight" />
    </columns>
  </EJ.Grid>,
  document.getElementById('Grid')
);
```

The following output is displayed as a result of the above code example

OrderID	EmployeeID	ShipCity	ShipCountry	Freight
10248	5	Reims	France	32.38
10249	6	Münster	Germany	11.61
10250	4	Rio de Janeiro	Brazil	65.83
10251	3	Lyon	France	41.34
10252	4	Charleroi	Belgium	51.3
10253	3	Rio de Janeiro	Brazil	58.17
10254	5	Bern	Switzerland	22.98
10255	9	Genève	Switzerland	148.33
10256	3	Resende	Brazil	13.97
10257	4	San Cristóbal	Venezuela	81.91
10258	1	Graz	Austria	140.51
10259	4	México D.F.	Mexico	3.25

1

2

3

4

5

6

7

8

...

1 of 17 pages (200 items)

### Cell Selection

Cell selection is enabled by setting `selectionMode` property of `selectionSettings` as `cell`. For random cell selection, press **“Ctrl + mouse left”** click and for continuous cell selection, press **“Shift + mouse left”** click on the grid cells. To unselect selected cells, press **“Ctrl + mouse left”** on selected cell click.

The following code example describes the above behavior.

### HTML

```
<div id="Grid"></div>
<script type="text/babel" src="app.jsx">
</script>
```

Create a JSX file and paste the following content

### JAVASCRIPT

```
var selectionMode = { selectionMode: ["cell"] };
ReactDOM.render(
  //The datasource "window.gridData" is referred from
  'http://js.syncfusion.com/demos/web/scripts/jsondata.min.js'
  <EJ.Grid dataSource = {window.gridData} allowPaging = {true}
  allowSelection={true} selectionSettings={selectionMode}
  selectionType="multiple">
    <columns>
    <column field="OrderID" />
    <column field="EmployeeID" />
    <column field="ShipCity" />
```

```
<column field="ShipCountry"/>
<column field="Freight" />
</columns>
</EJ.Grid>,
document.getElementById('Grid')
);
```

The following output is displayed as a result of the above code example

OrderID	EmployeeID	ShipCity	ShipCountry	Freight
10248	5	Reims	France	32.38
10249	6	Münster	Germany	11.61
10250	4	Rio de Janeiro	Brazil	65.83
10251	3	Lyon	France	41.34
10252	4	Charleroi	Belgium	51.3
10253	3	Rio de Janeiro	Brazil	58.17
10254	5	Bern	Switzerland	22.98
10255	9	Genève	Switzerland	148.33
10256	3	Resende	Brazil	13.97
10257	4	San Cristóbal	Venezuela	81.91
10258	1	Graz	Austria	140.51
10259	4	México D.F.	Mexico	3.25

1
2
3
4
5
6
7
8
...

1 of 17 pages (200 items)

### Column Selection

[Column selection](#) can be enabled by setting [selectionMode](#) property of [selectionSettings](#) as `column`. For random column selection, press **“Ctrl + mouse left click”** and for continuous column selection, press **“Shift + mouse left click”** on the top of the column header. To unselect selected columns, press **“Ctrl + mouse left click”** on top of the selected column header.

The following code example describes the above behavior.

### HTML

```
<div id="Grid"></div>
<script type="text/babel" src="app.jsx">
</script>
```

Create a JSX file and paste the following content

### JAVASCRIPT



```

var selectionMode = { selectionMode: ["column"] };
ReactDOM.render(
  //The datasource "window.gridData" is referred from
  'http://js.syncfusion.com/demos/web/scripts/jsondata.min.js'
  <EJ.Grid dataSource = {window.gridData} allowPaging = {true}
  allowSelection={true} selectionSettings={selectionMode}
  selectionType="multiple">
    <columns>
      <column field="OrderID" />
      <column field="EmployeeID" />
      <column field="ShipCity" />
      <column field="ShipCountry"/>
      <column field="Freight" />
    </columns>
  </EJ.Grid>,
  document.getElementById('Grid')
);

```

The following output is displayed as a result of the above code example

OrderID	EmployeeID	ShipCity	ShipCountry	Freight
10248	5	Reims	France	32.38
10249	6	Münster	Germany	11.61
10250	4	Rio de Janeiro	Brazil	65.83
10251	3	Lyon	France	41.34
10252	4	Charleroi	Belgium	51.3
10253	3	Rio de Janeiro	Brazil	58.17
10254	5	Bern	Switzerland	22.98
10255	9	Genève	Switzerland	148.33
10256	3	Resende	Brazil	13.97
10257	4	San Cristóbal	Venezuela	81.91
10258	1	Graz	Austria	140.51
10259	4	México D.F.	Mexico	3.25

1
2
3
4
5
6
7
8
...

1 of 17 pages (200 items)

### Touch options

While using grid in a [touch](#) device environment, there is an option for multi selection through single tap on the row and it will show a popup with multi-selection symbol. Tap the icon to enable multi selection in a single tap.

The following code example describes the above behavior.

**HTML**

```
<div id="Grid"></div>
<script type="text/babel" src="app.jsx">
</script>
```

Create a JSX file and paste the following content

**JAVASCRIPT**

```
ReactDOM.render (
  //The datasource "window.gridData" is referred from
  'http://js.syncfusion.com/demos/web/scripts/jsondata.min.js'
  <EJ.Grid dataSource = {window.gridData} allowPaging = {true}
  allowSelection={true} enableTouch={true} selectionType="multiple">
    <columns>
      <column field="OrderID" />
      <column field="EmployeeID" />
      <column field="ShipCity" />
      <column field="ShipCountry"/>
      <column field="Freight" />
    </columns>
  </EJ.Grid>,
  document.getElementById('Grid')
);
```

The following output is displayed as a result of the above code example.

OrderID	EmployeeID	ShipCity	ShipCountry	Freight
10248	5	Reims	France	32.38
10249	6	Münster	Germany	11.61
10250	4	Rio de Janeiro	Brazil	65.83
10251	3	Lyon	France	41.34
10252	4	Charleroi	Belgium	51.3
10253	3	Rio de Janeiro	Brazil	58.17
10254	5	Bern	Switzerland	22.98
10255	9	Genève	Switzerland	33
10256	3	Resende	Brazil	13.97
10257	4	San Cristóbal	Venezuela	81.91
10258	1	Graz	Austria	140.51
10259	4	México D.F.	Mexico	3.25

1
2
3
4
5
6
7
8
...

1 of 17 pages (200 items)

## Toggle Selection

The [Toggle] selection allows to perform selection and unselection of the particular row, cell or column. To enable toggle selection, set `enableToggle` property of `selectionSettings` as `true`. If you click on the selected row, cell or column then it will be unselected and vice versa.

**Note:** If multi selection is enabled, then in first click on any selected row (without pressing Ctrl key), it will clear multi selection and in second click on the same row, it will be unselected.

The following code example describes the above behavior.

### HTML

```
<div id="Grid"></div>
<script type="text/babel" src="app.jsx">
</script>
```

Create a JSX file and paste the following content

### JAVASCRIPT

```
var settings = {enableToggle: true };
ReactDOM.render(
  //The datasource "window.gridData" is referred from
  'http://js.syncfusion.com/demos/web/scripts/jsondata.min.js'
  <EJ.Grid dataSource = {window.gridData} allowPaging = {true}
  allowSelection={true} selectionSettings={settings}>
    <columns>
      <column field="OrderID" />
      <column field="EmployeeID" />
      <column field="ShipCity" />
      <column field="ShipCountry"/>
      <column field="Freight" />
    </columns>
  </EJ.Grid>,
  document.getElementById('Grid')
);
```

## GroupButton

### Overview

GroupButton is a UI component which provides the option to have the multiple buttons in succession. This component also has data source binding and more flexible options to provide the friendlier implementation and usage. Different modes with GroupButton gives the flexible and rich options to manage the actions based on the application needs.

### Key Features

- Trendy Look: Rich appearance with theme Support
- RTL: Supports for right to left alignment
- Different Modes of Button: Supports the rising of click event repeatedly
- DataSource: Supports data binding with JSON data and remote data.
- Easy Customization: The customization of Button control to any form is made simple

## Getting Started

You can create a ReactJS GroupButton component in your application with the help of the following steps. The basic rendering of ReactJS GroupButton is achieved with default functionality.

### Create a GroupButton Using JSX template

You can create a ReactJS application and add necessary scripts and styles with the help of the given link <https://help.syncfusion.com/reactjs/overview>

Define an HTML element for adding GroupButton in the application and refer the JSX file.

#### HTML

```
<body>
<div id="groupbutton"></div>
<script type="text/babel" src="GroupButton.jsx"> </script>
</body>
```

Create a JSX file for rendering GroupButton components using '<EJ.GroupButton>' tag. Add required properties in the with this tag element.

#### HTML

```
"use strict";
var data=[
{ text: 'Day', contentType: 'textonly' },
{ text: 'Week', contentType: 'textonly' },
{ text: 'Work Week', contentType: 'textonly' },
{ text: 'Month', contentType: 'textonly', selected: 'selected' },
{ text: 'Agenda', contentType: 'textonly' }
];
```

#### HTML

```
ReactDOM.render(
<EJ.GroupButton id="dtp" dataSource={data} >
</EJ.GroupButton>
,
document.getElementById('groupbutton')
);
```

This will render the GroupButton in the above HTML page in the div element with id sample.



### Create a GroupButton without using JSX template

GroupButton can be created without using JSX template. It can be done by using script section in HTML file.

The following script code will render the GroupButton component.

#### HTML

```
<div id="groupbutton">
<script>
```

```

var data = [
  { text: 'Day', contentType: 'textonly' },
  { text: 'Week', contentType: 'textonly' },
  { text: 'Work Week', contentType: 'textonly' },
  { text: 'Month', contentType: 'textonly', selected: 'selected' },
  { text: 'Agenda', contentType: 'textonly' }
];
ReactDOM.render(
  React.createElement(EJ.GroupButton, { id: 'groupbutton', dataSource: data,
    showRoundedCorner: true }
  ),
  document.getElementById('groupbutton')
);
</script>
</div>

```

### Configure Properties

you can make use of all available properties in GroupButton in ReactJS framework. Please refer below code to configure the all properties in this framework

#### JAVASCRIPT

```

"use strict";
var data=[
  { text: 'Day', contentType: 'textonly' },
  { text: 'Week', contentType: 'textonly' },
  { text: 'Work Week', contentType: 'textonly' },
  { text: 'Month', contentType: 'textonly', selected: 'selected' },
  { text: 'Agenda', contentType: 'textonly' }
];
ReactDOM.render(
  <EJ.GroupButton id="dtp" dataSource={data} showRoundedCorner={true}>
  </EJ.GroupButton>
  ,
  document.getElementById('groupbutton')
);

```

Run the above code to render the following output.



## Gantt

### Overview

The Essential React JS Gantt control is designed to visualize and edit the project schedule, and track the project progress. It helps to organize and schedule the projects and also you can update the project schedule through interactions like editing, dragging and resizing.

#### Key Features

- Sorting
- Editing

- Task relationship
- Stripline
- Baseline
- Timeline customization
- Work break down structure
- Taskbar customization
- Task label customization
- Localization
- Resources

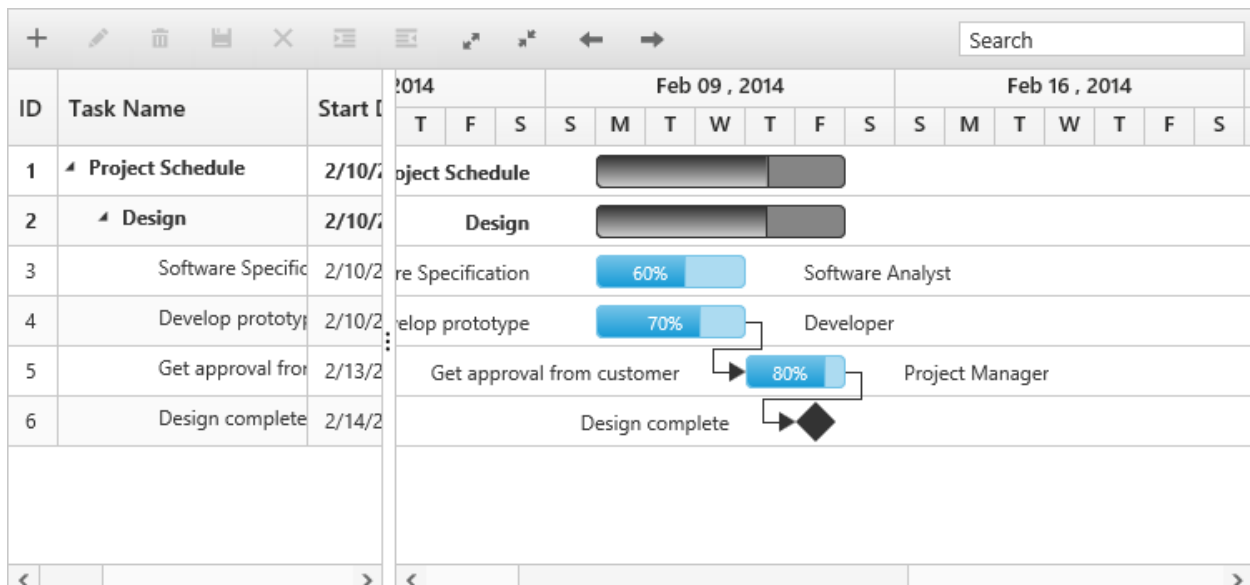
## Getting Started

This section explains briefly about how to create a Gantt chart in your application with ReactJS.

### Create your first Gantt in ReactJS

To get started Syncfusion ReactJS application refer [this](#) page for basic control integration and script references.

In this tutorial, you can learn how to create a simple Gantt chart, add tasks or subtasks, and set relationship between tasks during the design phase of a software project. The following screenshot displays the desired output after completing this tutorial,



The required ReactJS script dependencies as follows. And you can also refer [React](#) to know more about react js.

- react.min.js - <http://cdn.syncfusion.com/js/assets/external/react.min.js>
- react-dom.min.js - <http://cdn.syncfusion.com/js/assets/external/react-dom.min.js>
- browser.min.js - <http://cdnjs.cloudflare.com/ajax/libs/babel-core/5.8.34/browser.min.js>
- ej.web.react.min.js - <http://cdn.syncfusion.com/{{ site.releaseversion }}/js/common/ej.web.react.min.js>

### Adding script references

Create an HTML file and add the following template to the HTML file.

To get started, you can use the `ej.web.all.min.js` file that encapsulates all the `ej` controls and frameworks in one single file. So the complete boilerplate code is

### HTML

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title>Getting Started with Gantt Control for Aurelia</title>
<!-- style sheet for default theme(flat azure) -->
<link href="http://cdn.syncfusion.com/{{ site.releaseversion }}/js/web/flat-azure/ej.web.all.min.css" rel="stylesheet" />
<!--scripts-->
<script
src="http://cdn.syncfusion.com/js/assets/external/react.min.js"></script>
<script src="http://cdn.syncfusion.com/js/assets/external/react-dom.min.js"></script>
<script src="https://cdnjs.cloudflare.com/ajax/libs/babel-core/5.8.34/browser.min.js"></script>
<script src="http://cdn.syncfusion.com/js/assets/external/jquery-1.10.2.min.js"></script>
<script
src="http://cdn.syncfusion.com/js/assets/external/jsrender.min.js"></script>
<script
src="http://cdn.syncfusion.com/js/assets/external/jquery.globalize.min.js"></script>
<script
src="http://cdn.syncfusion.com/js/assets/external/jquery.easing.1.3.min.js"></script>
<script src="http://cdn.syncfusion.com/{{ site.releaseversion }}/js/web/ej.web.all.min.js "></script>
<script src="http://cdn.syncfusion.com/{{ site.releaseversion }}/js/common/ej.web.react.min.js"></script>
</head>
<body>
<!--Add Gantt control here-->
</body>
</html>
```

Initialize the Gantt with data source

Create a Gantt

Control can be initialized in two ways.

- Using jsx Template
- Without using jsx Template

Using jsx Template

By using the jsx template, we can create the html file and jsx file. The `.jsx` file can be convert to `.js` file and it can be referred in html page.

Please refer to the code of HTML file.

### HTML

```
<div id="Gantt-default"></div>
```

```
<script type="text/babel" src="app.jsx">
</script>
```

Create an app.JSX file and paste the following content

### JS

```
var projectData = [{
  TaskId: 1,
  TaskName: "Project Schedule",
  StartDate: "02/03/2014",
  EndDate: "03/07/2014",
  coll: "check",
  Children: [{
    TaskId: 2,
    TaskName: "Design",
    StartDate: "02/10/2014",
    EndDate: "02/14/2014",
    Children: [{
      TaskId: 3,
      TaskName: "Software Specification",
      StartDate: "02/10/2014",
      EndDate: "02/12/2014",
      Duration: 3,
      Progress: "60",
      resourceId: [2]
    }, {
      TaskId: 4,
      TaskName: "Develop prototype",
      StartDate: "02/10/2014",
      EndDate: "02/12/2014",
      Duration: 3,
      Progress: "100",
      resourceId: [3]
    }, {
      TaskId: 5,
      TaskName: "Get approval from customer",
      StartDate: "02/13/2014",
      EndDate: "02/14/2014",
      Duration: 2,
      Progress: "100",
      Predecessor: "4FS",
      resourceId: [1]
    }, {
      TaskId: 6,
      TaskName: "Design complete",
      StartDate: "02/14/2014",
      EndDate: "02/14/2014",
      Duration: 0,
      Predecessor: "5FS"
    }
  ]
}, ],
  ]];
ReactDOM.render(
  <EJ.Gantt dataSource = {projectData}
  childMapping = "Children"
```

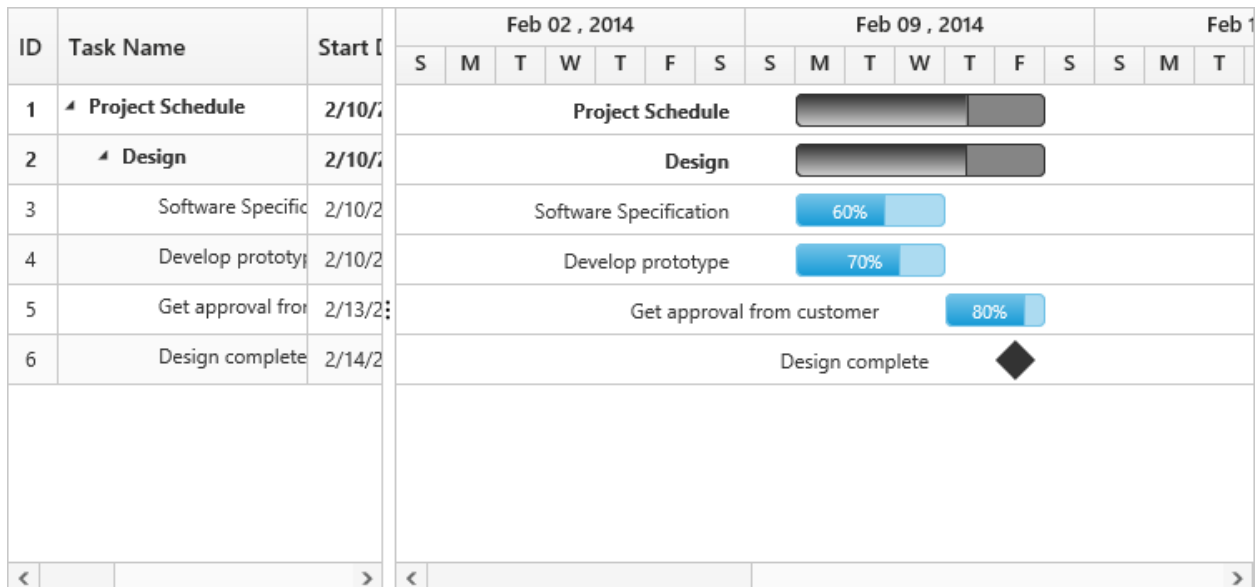


```

taskIdMapping = "TaskId"
taskNameMapping = "TaskName"
startDateMapping = "StartDate"
endDateMapping = "EndDate"
durationMapping = "Duration"
progressMapping = "Progress"
treeColumnIndex = {1}>
</EJ.Gantt>,
document.getElementById('Gantt-default')
);

```

A Gantt chart is created as shown in the following screen shot.



### Enable Toolbar

Gantt control contains toolbar options to edit, search, expand or collapse all records, indent, outdent, delete, and add a task. You can enable toolbar using the [toolbarSettings](#) property.

### JS

```

var toolbarsettings = {
  showToolbar: true,
  toolbarItems: [
    ej.Gantt.ToolbarItems.Add,
    ej.Gantt.ToolbarItems.Edit,
    ej.Gantt.ToolbarItems.Delete,
    ej.Gantt.ToolbarItems.Update,
    ej.Gantt.ToolbarItems.Cancel,
    ej.Gantt.ToolbarItems.Indent,
    ej.Gantt.ToolbarItems.Outdent,
    ej.Gantt.ToolbarItems.ExpandAll,
    ej.Gantt.ToolbarItems.CollapseAll
  ]
};
var editsettings = {
  allowEditing: true,
  allowAdding: true,

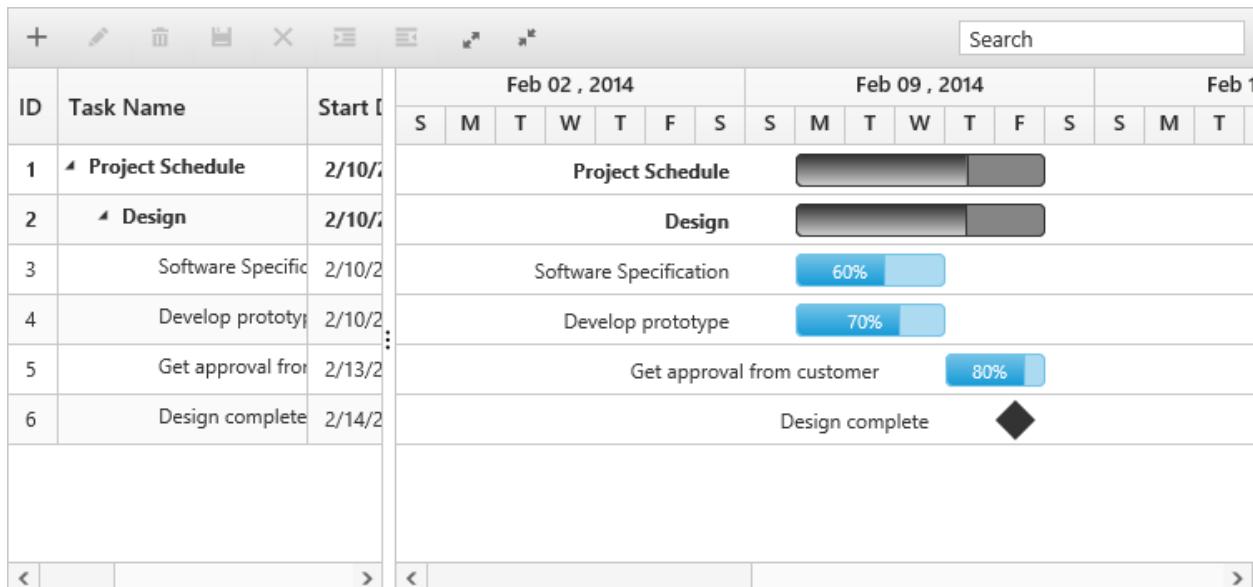
```

```

allowDeleting: true,
allowIndent: true,
editMode: 'cellEditing'
};
ReactDOM.render(
<EJ.Gantt toolbarSettings = {toolbarsettings}
editSettings = {editsettings}>
</EJ.Gantt>,
document.getElementById('Gantt-default')
);

```

The following screen shot displays a Tool bar in Gantt chart control:



**Note:** Add, edit, delete, indent and outdent options are enabled when enabling the allowEditing, allowAdding, allowDelete, allowIndent and allowOutdent properties in the edit Options.

### Enable Sorting

The Gantt control has sorting functionality to arrange the tasks in ascending or descending order based on a particular column.

### Multicolumn Sorting

Enable the multicolumn sorting in Gantt by setting [allowMultiSorting](#) as true. You can sort multiple columns in Gantt, by selecting the desired column header while holding the CTRL key.

### JAVASCRIPT

```

ReactDOM.render(
<EJ.Gantt dataSource = {projectData}
allowSorting = {true}
allowMultiSorting = {true}>
</EJ.Gantt>,
document.getElementById('Gantt-default')
);

```

### Enable Editing

You can enable editing using [editSettings](#) and [allowGanttChartEditing](#) options.

#### Cell Editing

Modify the task details through the grid cell editing by setting the [editMode](#) as [cellEditing](#).

#### Normal Editing

Modify the task details through the edit dialog by setting the [editMode](#) as [normal](#).

#### Taskbar Editing

Modify the task details through user interaction such as resizing and dragging the taskbar.

#### Predecessor Editing

Modify the predecessor details of a task using mouse interactions by setting [allowGanttChartEditing](#) as `true` and setting the value for `predecessorMapping` property.

### JAVASCRIPT

```
var editsettings = {
  allowEditing: true,
  allowAdding: true,
  allowDeleting: true,
  allowIndent: true,
  editMode: 'cellEditing'
};
ReactDOM.render(
  <EJ.Gantt dataSource = {projectData}
  allowGanttChartEditing = {true}
  predecessorsMapping = "Predecessor"
  editSettings = {editsettings}>
  </EJ.Gantt>,
  document.getElementById('Gantt-default')
);
```

The following screen shot displays a Gantt chart control with Enable Editing options.

Task Information
✕

General
Predecessors
Resources
Custom Fields

ID
3

Task Name

Start Date
2/3/2014

End Date
2/7/2014

Duration
5 days

Progress
100

Work
40

Task Type
Fixed Units

Effort Driven
No

Delete Task
Save
Cancel

**Note:** Both cellEditing and normal editing operations are performed through double-click or single click action that can be defined by `editSettings.beginEditAction` property.

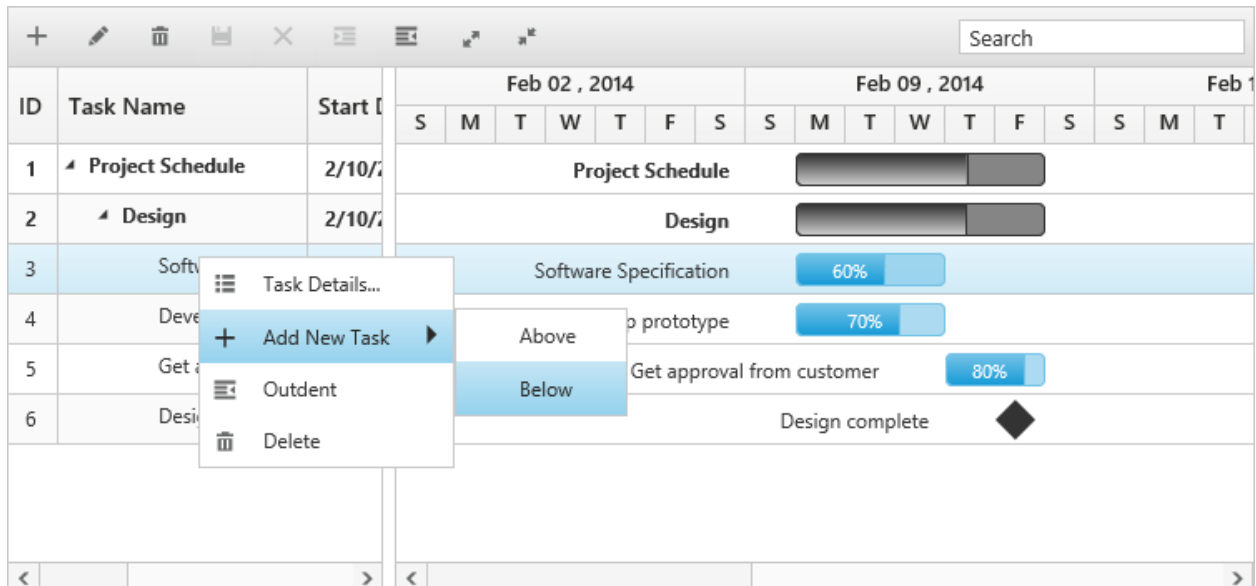
### Enable Context Menu

You can enable the context menu in Gantt, by setting the `enableContextMenu` as `true`.

### JAVASCRIPT

```
ReactDOM.render(
  <EJ.Gantt dataSource = {projectData}
  enableContextMenu = {true}>
</EJ.Gantt>,
  document.getElementById('Gantt-default')
);
```

The following screen shot displays Gantt chart in which Context menu option is enabled:



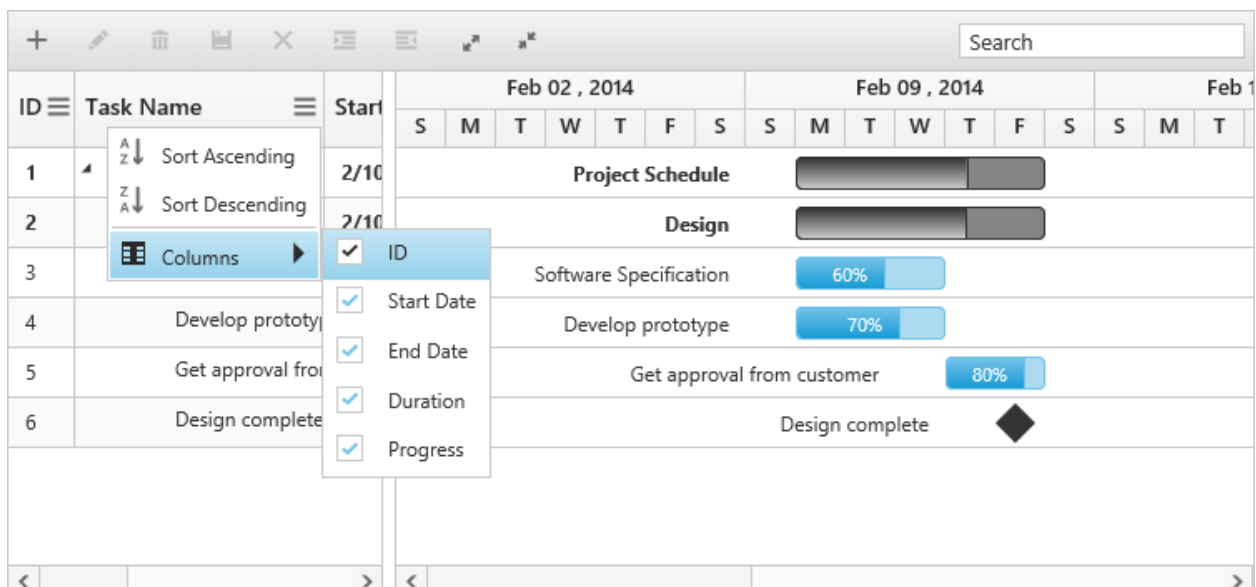
### Enable Column Menu

You can enable the column menu in Gantt, by setting the [showColumnChooser](#) as `true`.

### JAVASCRIPT

```
ReactDOM.render(
  <EJ.Gantt dataSource = {projectData}
  showColumnChooser = {true}>
</EJ.Gantt>,
  document.getElementById('Gantt-default')
);
```

The following screen shot displays Gantt chart in which column chooser option is enabled:



### Provide tasks relationship

In Gantt, you have the predecessor support to show the relationship between two different tasks.

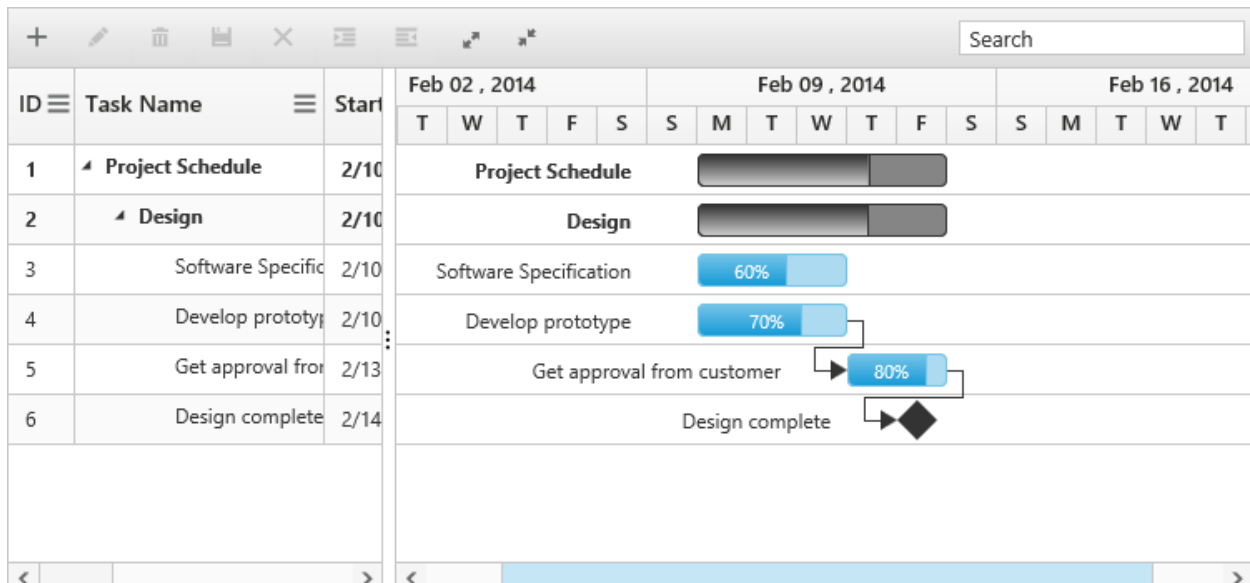
- **Start to Start (SS)** - You cannot start a task until the other task also starts.
- **Start to Finish (SF)** - You cannot finish a task until the other task finishes.
- **Finish to Start (FS)** - You cannot start a task until the other task completes.
- **Finish to Finish (FF)** - You cannot finish a task until the other task completes.

You can show the relationship in tasks, by using the [predecessorMapping](#), as shown in the following code example.

#### JAVASCRIPT

```
ReactDOM.render(
  <EJ.Gantt dataSource = {projectData}
    predecessorsMapping = "Predecessor" >
  </EJ.Gantt>,
  document.getElementById('Gantt-default')
);
```

The following screenshot displays the relationship between tasks.



#### Provide Resources

In Gantt control, you can display and assign the resource for each task. Create a collection of **JSON** object, which contains id and name of the resource and assign it to [resources](#) property. Then, specify the field name for id and name of the resource in the resource collection to [resourceIdMapping](#) and [resourceNameMapping](#) options. The name of the field, which contains the actual resources assigned for a particular task in the **dataSource** is specified using [resourceInfoMapping](#).

1. Create the resource collection to be displayed in ejGantt

#### JAVASCRIPT

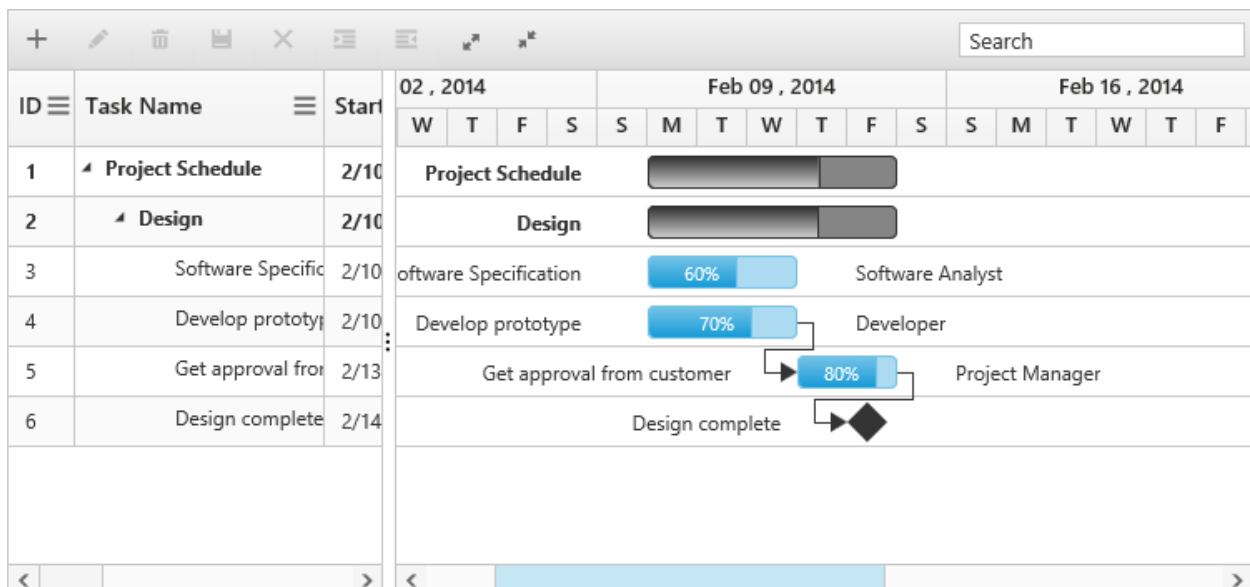
```
var projectResources = [{
  resourceId: 1,
  resourceName: "Project Manager"
}, {
  resourceId: 2,
```

```

resourceName: "Software Analyst"
}, {
resourceId: 3,
resourceName: "Developer"
}, {
resourceId: 4,
resourceName: "Testing Engineer"
}];
ReactDOM.render(
<EJ.Gantt resourceInfoMapping = "resourceId"
resourceNameMapping = "resourceName"
resourceIdMapping = "resourceId"
resources = {
projectResources
}>
</EJ.Gantt>,
document.getElementById('Gantt-default')
);

```

The following screenshot displays resource allocation for tasks in Gantt chart.



By following these steps, you have learned how to provide data source to Gantt chart, how to configure Gantt to set task relationships, assign resources for each task, and add toolbar with necessary buttons.

### Highlight Weekend

In Gantt, you can on or off weekends high lighting by setting the [highlightWeekends](#) as `true` or `false`.

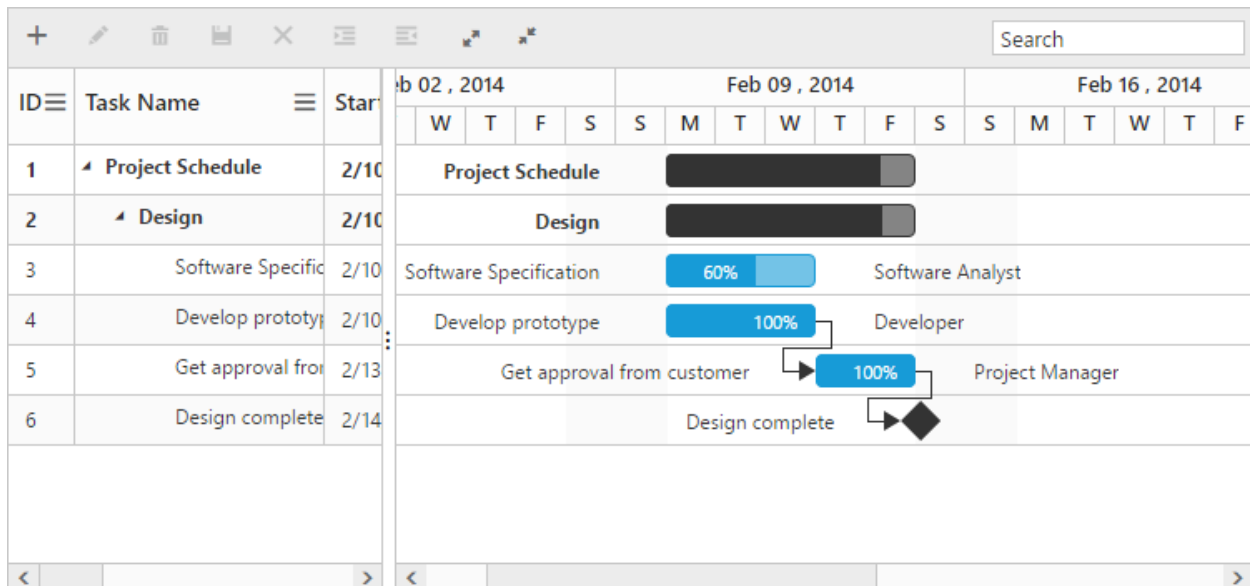
### JAVASCRIPT

```

ReactDOM.render(
<EJ.Gantt dataSource = {projectData}
highlightWeekends = {true}>
</EJ.Gantt>,
document.getElementById('Gantt-default')
);

```

The following screen shot displays Gantt chart in which highlight weekends is enabled:



## HeatMap

### Overview

**Essential HeatMap React JS** represents tabular data values as gradient colors instead of numbers. Low and high values are different colors with different gradients.

Product Name	Y2010	Y2011	Y2012	Y2013	Y2014	Y2015	Y2016	Y2017	Y2018
Veggie-spread	24	27	89	53	29	97	18	22	9
Tofuuaa	25	82	7	32	36	84	2	51	5
Alice Mutton	44	13	43	13	60	21	71	17	1
Konbu	12	45	66	7	99	73	49	14	7
Fløtemysost	66	1	74	42	8	1	40	6	3
Perth Pasties	90	50	89	36	62	97	91	96	2
Boston Crab Meat	92	39	17	70	17	52	88	24	6
Raclette Courdavault	70	65	99	58	96	8	8	63	1

Key features:

- 2 types of data source mapping (TableMapping, CellMapping)
- Color mapping
- Legend
- Virtualization

### Getting Started

This section helps to get started of the HeatMap component for ReactJS.



### Adding Script Reference

Create an **HTML** page and add the scripts references in the order mentioned in the following code example.

- [jQuery](#) 1.10.2 and later versions

The required ReactJS script dependencies as follows. And you can also refer [React](#) to know more about react js.

- react.min.js - <http://cdn.syncfusion.com/js/assets/external/react.min.js>
- react-dom.min.js - <http://cdn.syncfusion.com/js/assets/external/react-dom.min.js>
- ej.web.react.min.js - <http://cdn.syncfusion.com/{{ site.releaseversion }}/js/common/ej.web.react.min.js>

To get started, you can use the `ej.web.all.min.js` file that encapsulates all the `ej` controls and frameworks in one single file.

### HTML

```
<!DOCTYPE html>
<html>
<head>
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<meta name="description" content="Essential Studio for React JS">
<meta name="author" content="Syncfusion">
<title>Getting Started for Ribbon React JS</title>
<!-- Essential Studio for JavaScript theme reference -->
<link href="http://cdn.syncfusion.com/{{ site.releaseversion }}/js/web/flat-
azure/ej.web.all.min.css" rel="stylesheet" />
<!-- Essential Studio for JavaScript script references -->
<script src="http://cdn.syncfusion.com/js/assets/external/jquery-
3.0.0.min.js"></script>
<script
src="http://cdn.syncfusion.com/js/assets/external/react.min.js"></script>
<script src="http://cdn.syncfusion.com/js/assets/external/react-
dom.min.js"></script>
<script src="http://cdn.syncfusion.com/{{ site.releaseversion
}}/js/web/ej.web.all.min.js"></script>
<script src="http://cdn.syncfusion.com/{{ site.releaseversion
}}/js/common/ej.web.react.min.js"></script>
<!-- Add your custom scripts here -->
</head>
<body>
</body>
</html>
```

**Note:** 1. In production, we highly recommend you to use our [custom script generator](#) to create custom script file with required controls and its dependencies only. Also to reduce the file size further please use [GZip compression](#) in your server.

2. For themes, you can use the `ej.web.all.min.css` CDN link from the code snippet given. To add the themes in your application, please refer to [this link](#).

### Control Initialization

Control can be initialized in two ways.

- Using jsx Template
- Without using jsx Template

### Using jsx Template

By using the jsx template, we can create the html file and jsx file. The .jsx file can be convert to .js file and it can be referred in html page.

#### Initialize HeatMap

The HeatMap can be created from a HTML 'div' element. To create the HeatMap, you should call the 'ejHeatMap' jQuery plug-in function.

#### HTML

```
<div id="heatmap-default" style="height:99%;margin: 0 auto;"></div>
<script src="app/heatmap/default.js">
</script>
</div>
```

#### Prepare and Populate data

Populate product information in a collection called **ItemsSource**.

#### Map data into HeatMap

Now data is ready, next we need to configure data source and map rows and columns to visualize. For that, need to prepare **ItemsMapping** add it in resource and set items source and mapping.

Next we can configure color range for these values using color mapping and also configure items mapping based on items source.

#### JAVASCRIPT

```
var colorMappingCollection = [
{ value: 0, color: "#8ec8f8", label: { text: "0" } },
{ value: 100, color: "#0d47a1", label: { text: "100" } }
];
var columns = ["Veggie-spread", "Tofu", "Alice Mutton", "Konbu",
"Fløtemysost", "Perth Pasties", "Boston Crab Meat", "Raclette Courdavault"];
var itemSource = [];
for (var i = 0; i < columns.length; i++) {
for (var j = 0; j < 8; j++) {
var value = Math.floor((Math.random() * 100) + 1);
itemSource.push({ ProductName: columns[i], Year: "Y" + (2011 + j), Value:
value })
}
};
var itemsMapping = {
column: { "propertyName": "ProductName", "displayName": "Product Name" },
row: { "propertyName": "Year", "displayName": "Year", },
value: { "propertyName": "Value" },
headerMapping: { "propertyName": "Year", "displayName": "Year", columnStyle:
{ width: 105, textAlign: "right" } },
columnMapping: [
{ "propertyName": columns[0], "displayName": columns[0] },
```

```

{ "propertyName": columns[1], "displayName": columns[1] },
{ "propertyName": columns[2], "displayName": columns[2] },
{ "propertyName": columns[3], "displayName": columns[3] },
{ "propertyName": columns[4], "displayName": columns[4] },
{ "propertyName": columns[5], "displayName": columns[5] },
{ "propertyName": columns[6], "displayName": columns[6] },
{ "propertyName": columns[7], "displayName": columns[7] },
],
};
ReactDOM.render(
<div className="default"></div>
<EJ.HeatMap id="heatmap1" width="810px" height="50px" itemsSource={
itemSource} itemsMapping= {itemsMapping} isResponsive="true
colorMappingCollection="colorMappingCollection">
</EJ.HeatMap>,
document.getElementById('heatmap-default')
);

```

Product Name	Y2010	Y2011	Y2012	Y2013	Y2014	Y2015	Y2016	Y2017
Vegie-spread	64	30	3	95	98	54	56	66
Tofuua	56	42	28	36	100	61	31	39
Alice Mutton	21	5	59	5	37	86	77	55
Konbu	95	9	40	70	32	9	55	20
Fløtemysost	68	42	89	92	26	6	83	80
Perth Pasties	42	88	98	9	54	63	88	8
Boston Crab Meat	20	8	25	78	70	43	61	55
Raclette Courdavault	12	81	10	33	75	91	71	87

### Initialize Legend

A legend control is used to represent range value in a gradient, create a legend with the same color mapping as shown below.

### HTML

```

<div id="heatmap-default-legend" style="height:99%;margin-left:100px;"></div>
<script src="app/heatmap/default.js">
</script>

```

### JAVASCRIPT

```

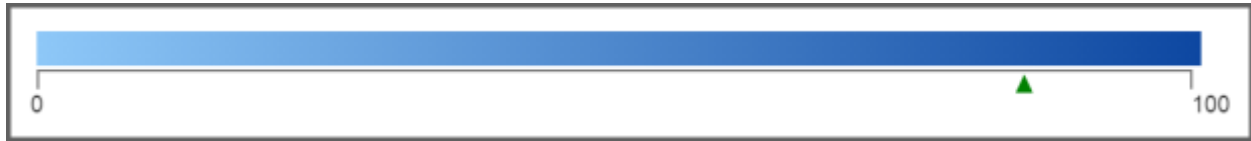
var colorMappingCollection = [
{ value: 0, color: "#8ec8f8", label: { text: "0" } },
{ value: 100, color: "#0d47a1", label: { text: "100" } }
];
ReactDOM.render(
<div>
<div className="default"></div>
<EJ.HeatMapLegend id="heatmap1_legend" width="75%" height="50px"
isResponsive="true colorMappingCollection="colorMappingCollection">
</EJ.HeatMapLegend>

```

```

</div>,
document.getElementById('heatmap-default-legend')
);

```



Without using jsx Template

#### Initialize HeatMap

The HeatMap can be created from a HTML 'div' element. To create the HeatMap, you should call the 'ejHeatMap' jQuery plug-in function.

#### HTML

```

<div id="heatmap-default" style="height:99%;margin: 0 auto;"></div>
</div>

```

#### JAVASCRIPT

```

var colorMappingCollection = [
{ value: 0, color: "#8ec8f8", label: { text: "0" } },
{ value: 100, color: "#0d47a1", label: { text: "100" } }
];
var columns = ["Veggie-spread", "Tofuuaa", "Alice Mutton", "Konbu",
"Fløtemysost", "Perth Pasties", "Boston Crab Meat", "Raclette Courdavault"];
var itemSource = [];
for (var i = 0; i < columns.length; i++) {
for (var j = 0; j < 8; j++) {
var value = Math.floor((Math.random() * 100) + 1);
itemSource.push({ ProductName: columns[i], Year: "Y" + (2011 + j), Value:
value });
}
};
var itemsMapping= {
column: { "propertyName": "ProductName", "displayName": "Product Name" },
row: { "propertyName": "Year", "displayName": "Year", },
value: { "propertyName": "Value" },
headerMapping: { "propertyName": "Year", "displayName": "Year", columnStyle:
{ width: 105, textAlign: "right" } },
columnMapping: [
{ "propertyName": columns[0], "displayName": columns[0] },
{ "propertyName": columns[1], "displayName": columns[1] },
{ "propertyName": columns[2], "displayName": columns[2] },
{ "propertyName": columns[3], "displayName": columns[3] },
{ "propertyName": columns[4], "displayName": columns[4] },
{ "propertyName": columns[5], "displayName": columns[5] },
{ "propertyName": columns[6], "displayName": columns[6] },
{ "propertyName": columns[7], "displayName": columns[7] },
],
};
React.createElement(EJ.HeatMap, {
id: "heatmap1",
colorMappingCollection: colorMappingCollection,

```

```
width: "810",
isResponsive: true,
itemsSource: itemSource,
itemsMapping: itemsMapping
}),
document.getElementById('heatmap-default')
```

Product Name	Y2010	Y2011	Y2012	Y2013	Y2014	Y2015	Y2016	Y2017
Vegie-spread	64	30	3	95	98	54	56	66
Tofuua	56	42	28	36	100	61	31	39
Alice Mutton	21	5	59	5	37	86	77	55
Konbu	95	9	40	70	32	9	55	20
Fløtemysost	68	42	89	92	26	6	83	80
Perth Pasties	42	88	98	9	54	63	88	8
Boston Crab Meat	20	8	25	78	70	43	61	55
Raclette Courdavault	12	81	10	33	75	91	71	87

### Initialize Legend

A legend control is used to represent range value in a gradient, create a legend with the same color mapping as shown below.

#### HTML

```
<div id="heatmap-default-legend" style="height:99%;margin-left:100px;"></div>
</script>
```

#### JAVASCRIPT

```
var colorMappingCollection = [
{ value: 0, color: "#8ec8f8", label: { text: "0" } },
{ value: 100, color: "#0d47a1", label: { text: "100" } }
];
React.createElement(EJ.HeatMapLegend, {
id: "heatmap1_legend",
colorMappingCollection: colorMappingCollection,
height: "50px",
width: "75%",
isResponsive: true
}),
document.getElementById('heatmap-default-legend')
```



## Kanban Board

### Overview

The Kanban control for ReactJS is an efficient way to visualize the workflow at each stage along its path to completion. The most important features available are Swim lane, filtering, and editing.

Some important features of the Kanban control are:

- \* *Data sources* - Bind the Kanban control with an array of JSON objects or [ej.DataManager](#) which support OData and remote web service binding.
- \* *Swim lane* – Supports Swim lane grouping.
- \* *Filters* – Supports filtering based on user Query.
- \* *Editing* - Offers card editing, inserting, and deleting using Dialog.
- \* *Card template* - Offers to render the card based on user template.

### Getting Started

#### Preparing HTML document

The Kanban control has the following list of external JavaScript dependencies.

- [jQuery 1.7.1](#) and later versions
- [jsRender](#) - to render the templates

The required ReactJS script dependencies as follows. And you can also refer [React](#) to know more about react js.

- react.min.js - <http://cdn.syncfusion.com/js/assets/external/react.min.js>
- react-dom.min.js - <http://cdn.syncfusion.com/js/assets/external/react-dom.min.js>
- ej.web.react.min.js - [http://cdn.syncfusion.com/{{ site.releaseversion }}](http://cdn.syncfusion.com/{{ site.releaseversion }}/js/common/ej.web.react.min.js)

Refer to the internal dependencies in the following table.

Files	Description/Usage
ej.core.min.js	It is referred always before using all the JS controls.
ej.data.min.js	Used to handle data operation and is used while binding data to the JS controls.
ej.touch.min.js	It is referred when using touch functionalities in Kanban.
ej.draggable.min.js	It is referred when using drag and drop functionalities in Kanban.
ej.kanban.min.js	The Kanban™s main file.
ej.globalize.min.js	It is referred when using localization in Kanban.
ej.scroller.min.js	It is referred when scrolling is used in the Kanban.
ej.waitingpopup.min.js	It is referred when waiting popup used.
ej.dropdownlist.min.js	These files are used while enable the Editing feature in the Kanban.
ej.dialog.min.js	

ej.button.min.js	
ej.datepicker.min.js	
ej.datetimestpicker.min.js	
ej.editor.min.js	
ej.toolbar.min.js	These files are used while enable the Filtering feature in the Kanban.
ej.menu.min.js	These files are used while enable the context menu feature in the Kanban.
ej.checkbox.min.js	
ej.rte.min.js	These files are used while using the cell edit type as RTE in the Kanban.

To get started, you can use the `ej.web.all.min.js` file that encapsulates all the `ej` controls and frameworks in one single file. So the complete boilerplate code is

### HTML

```
<!DOCTYPE html>
<html>
<head>
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<meta name="description" content="Essential Studio for JavaScript">
<meta name="author" content="Syncfusion">
<title>Getting Started for Kanban React JS</title>
<!-- Essential Studio for JavaScript theme reference -->
<link href="http://cdn.syncfusion.com/{{ site.releaseversion }}/js/web/flat-
azure/ej.web.all.min.css" rel="stylesheet" />
<!-- Essential Studio for JavaScript script references -->
<script src="http://cdn.syncfusion.com/js/assets/external/jquery-
3.0.0.min.js"></script>
<script
src="http://cdn.syncfusion.com/js/assets/external/jsrender.min.js"></script>
<script
src="http://cdn.syncfusion.com/js/assets/external/react.min.js"></script>
<script src="http://cdn.syncfusion.com/js/assets/external/react-
dom.min.js"></script>
<script src="http://cdn.syncfusion.com/{{ site.releaseversion
}}/js/web/ej.web.all.min.js"></script>
<script src="http://cdn.syncfusion.com/{{ site.releaseversion
}}/js/common/ej.web.react.min.js"></script>
<!-- Add your custom scripts here -->
</head>
<body>
</body>
</html>
```

**Note:** 1. In production, we highly recommend you to use our [custom script generator](#) to create custom script file with required controls and its dependencies only. Also to reduce the file size further please use [GZip compression](#) in your server.

2. For themes, you can use the `ej.web.all.min.css` CDN link from the code snippet given. To add the themes in your application, please refer to [this link](#).

### Create a Kanban

Control can be initialized in two ways.

- Using jsx Template
- Without using jsx Template

### Using jsx Template

By using the jsx template, we can create the html file and jsx file. The `.jsx` file can be convert to `.js` file and it can be referred in html page.

Please refer to the code of HTML file.

### HTML

```
<div id="kanbanBoard-default"></div>
<script src="app/kanbanBoard/default.js"></script>
```

Kanban control can be initialized with the following in HTML document.

### HTML

```
ReactDOM.render (
  <EJ.Kanban >
    <columns>
      <column headerText="Backlog" />
      <column headerText="In Progress" />
      <column headerText="Done" />
    </columns>
  </EJ.Kanban>,
  document.getElementById('kanbanBoard-default')
);
```



### Data Binding

Data binding in the Kanban is achieved by using the `ej.DataManager` that supports both RESTful JSON data services binding and local JSON array binding. To set the data source to Kanban, the `dataSource` property is assigned with the instance of the `ej.DataManger`.

For demonstration purpose, [Northwind OData service](#) is used in this tutorial. Refer to the following code example.

### HTML

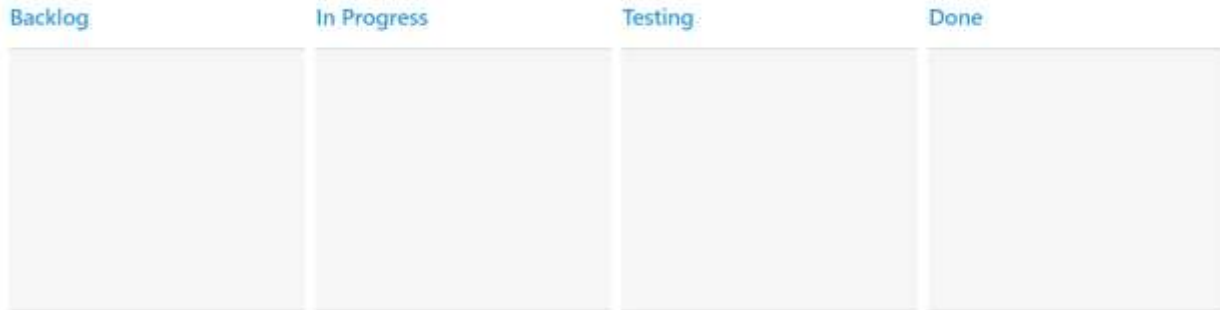
```
var dataManager = new
ej.DataManager("http://mvc.syncfusion.com/Services/Northwnd.svc/Tasks");
ReactDOM.render (
  <EJ.Kanban dataSource = {dataManager} >
    <columns>
```



```

<column headerText="Backlog" key="Open" />
<column headerText="In Progress" key="InProgress" />
<column headerText="Done" key="Close"/>
</columns>
</EJ.Kanban>,
document.getElementById('kanbanBoard-default')
);

```



**Note:** ODataAdaptor is the default adaptor used within DataManager. While binding to other web services, proper [data adaptor](#) needs to be set for adaptor option of DataManager.

#### Mapping Values

In order to display cards in Kanban control, you need to map the database fields to Kanban cards and columns. The required mapping field are listed as follows

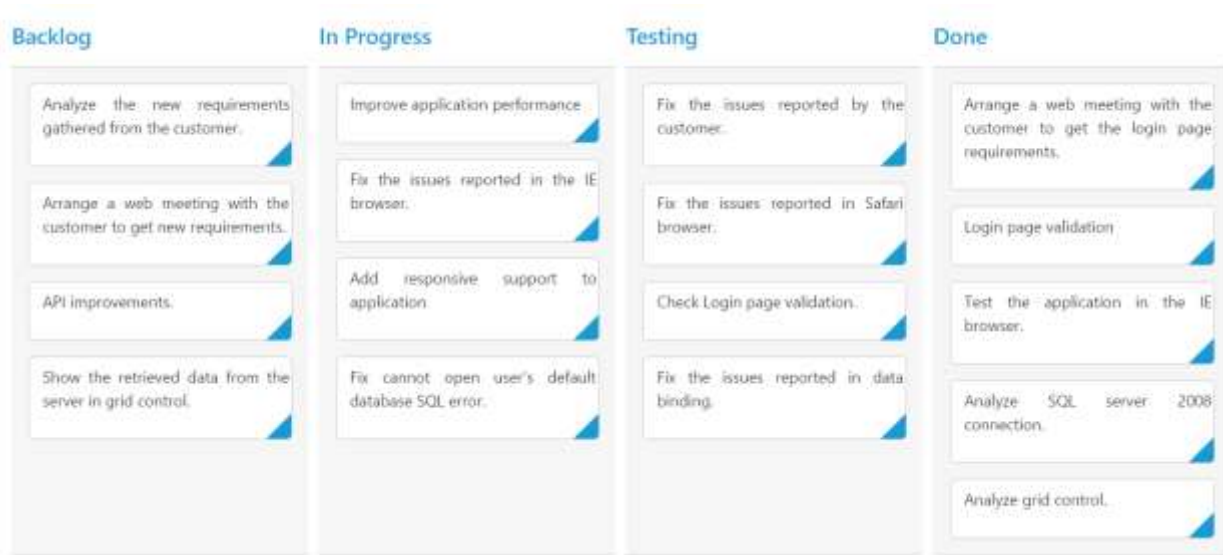
- **keyField** - Map the column name to use as **key** values to columns.
- **columns** - Map the corresponding **key** values of **keyField** column to each columns
- **fields.content** - Map the column name to use as content to cards.
- **fields.primaryKey** - Map the column name to use as primary Key.

#### HTML

```

var dataManager = new
ej.DataManager("http://mvc.syncfusion.com/Services/Northwnd.svc/Tasks");
ReactDOM.render(
<EJ.Kanban dataSource = {dataManager} keyField = "Status" allowTitle={true}
fields-content= "Summary" fields-primaryKey = "Id" allowSearching={true}
fields-imageUrl="ImgUrl" allowSelection={false} >
<columns>
<column headerText="Backlog" key="Open" />
<column headerText="In Progress" key="InProgress" />
<column headerText="Done" key="Close"/>
</columns>
</EJ.Kanban>,
document.getElementById('kanbanBoard-default')
);

```



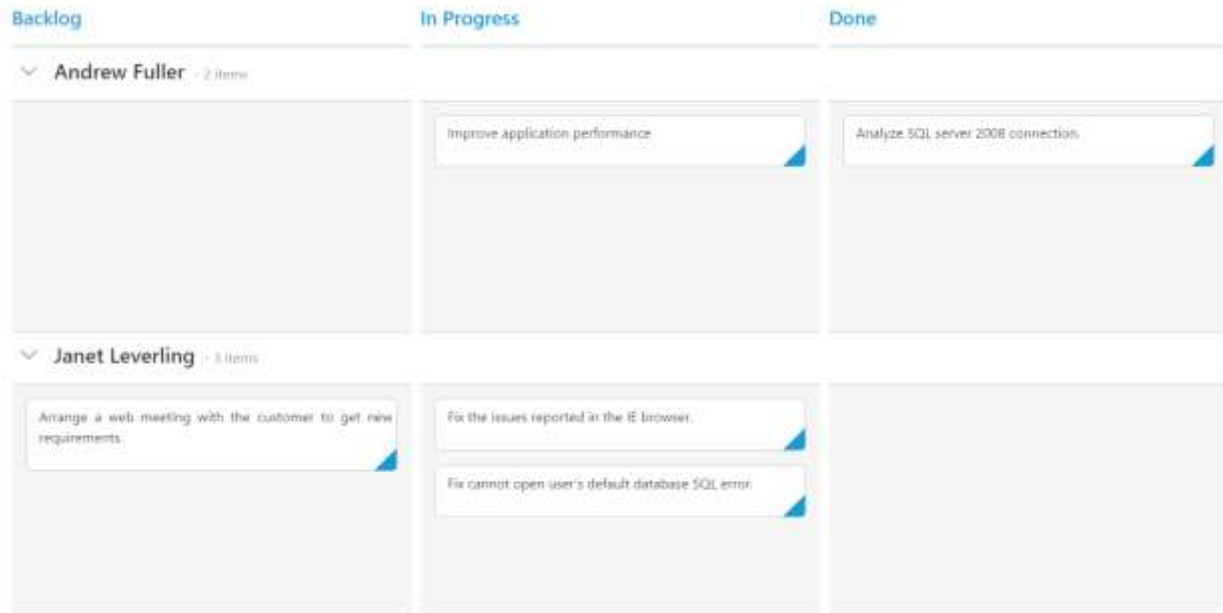
**Note:** `fields.primaryKey` field is mandatory for “Drag and Drop”, “Selection” and “Editing” Features.

#### Enable Swimlane

Swimlane can be enabled by mapping the `fields.swimlaneKey` to appropriate column name in `dataSource`. This enables the grouping of the cards based on the mapped column values.

#### HTML

```
var dataManager = new
ej.DataManager("http://mvc.syncfusion.com/Services/Northwnd.svc/Tasks");
ReactDOM.render(
<EJ.Kanban dataSource = {dataManager} keyField = "Status" allowTitle={true}
fields-content= "Summary" fields-primaryKey = "Id" fields-swimlaneKey =
"Assignee" allowSearching={true} fields-imageUrl="ImgUrl"
allowSelection={false} >
<columns>
<column headerText="Backlog" key="Open" />
<column headerText="In Progress" key="InProgress" />
<column headerText="Done" key="Close"/>
</columns>
</EJ.Kanban>,
document.getElementById('kanbanBoard-default')
);
```

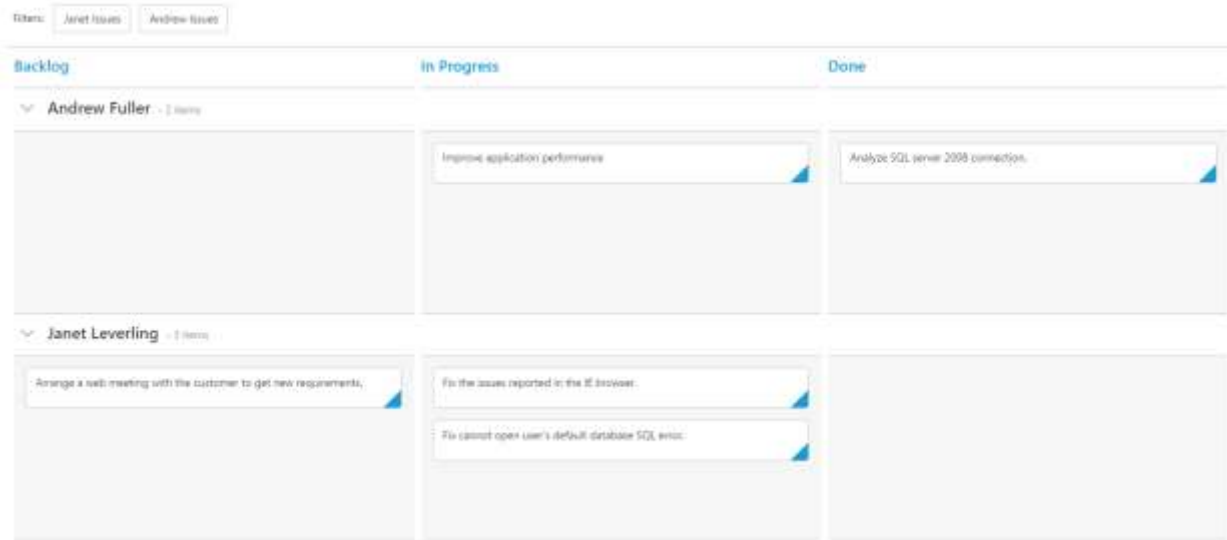


### Adding Filters

Filters allows to filter the collection of cards from `dataSource` which meets the predefined `query` in the filters collection. To enable filtering, define `filterSettings` collection with display `text` and [ej.Query](#).

### HTML

```
var dataManager = new
ej.DataManager("http://mvc.syncfusion.com/Services/Northwnd.svc/Tasks");
var filter = [
{ text: "Janet Issues", query: new ej.Query().where("Assignee", "equal",
"Janet Leverling"), description: "Displays issues which matches the assignee
as 'Janet Leverling'" },
{ text: "Andrew Issues", query: new ej.Query().where("Assignee", "equal",
"Andrew Fuller"), description: "Displays issues which matches the assignee
as 'Andrew Fuller'" }
];
ReactDOM.render(
<EJ.Kanban dataSource = {dataManager} keyField = "Status" allowTitle={true}
fields-content= "Summary" fields-primaryKey = "Id" fields-swimlaneKey =
"Assignee" allowSearching={true} fields-imageUrl="ImgUrl"
allowSelection={false} filterSettings={filter}>
<columns>
<column headerText="Backlog" key="Open" />
<column headerText="In Progress" key="InProgress" />
<column headerText="Done" key="Close"/>
</columns>
</EJ.Kanban>,
document.getElementById('kanbanBoard-default')
);
```



### Without using jsx Template

The Kanban can be created from a HTML **DIV** element with the HTML **id** attribute set to it. Refer to the following code example.

#### HTML

```
<body>
<div id="kanbanboard-default"></div>
</body>
```

Initialize the Kanban control by adding the following script code to the body section of the HTML document.

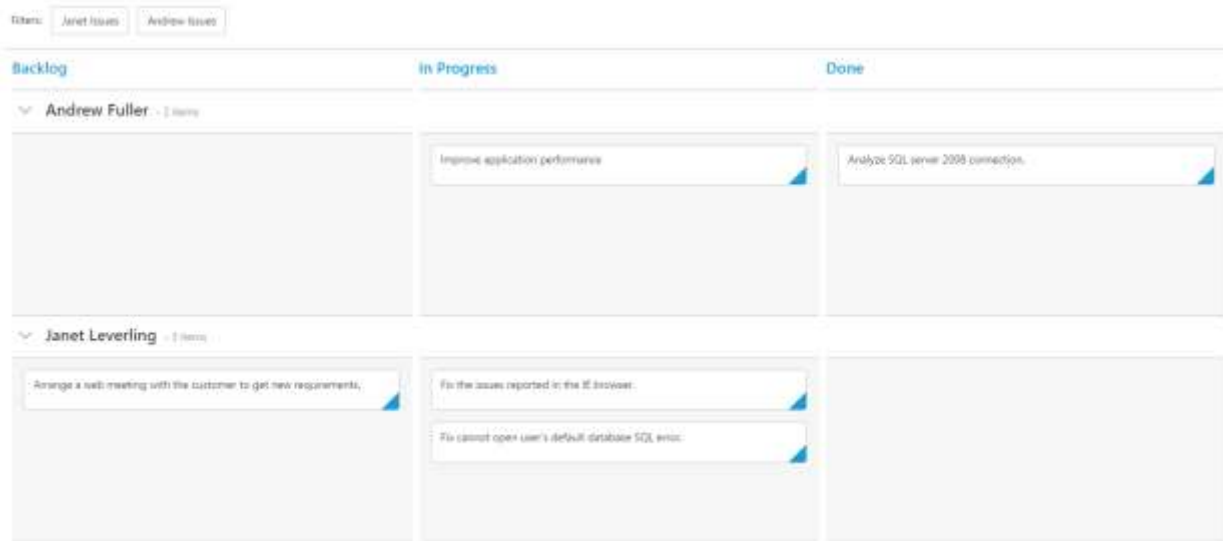
#### HTML

```
<div id="kanbanboard-default"></div>
<script type="text/javascript">
var dataManager = new
ej.DataManager("http://mvc.syncfusion.com/Services/Northwnd.svc/Tasks");
var filter = [
{ text: "Janet Issues", query: new ej.Query().where("Assignee", "equal",
"Janet Leverling"), description: "Displays issues which matches the assignee
as 'Janet Leverling' " },
{ text: "Open Issues", query: new ej.Query().where("Status", "equal",
"Open"), description: "Displays issues which matches the status as 'Open' " }
];
ReactDOM.render(
React.createElement(EJ.Kanban,
{
dataSource: data,
keyField: "Status",
"fields-content": "Summary",
"fields-primaryKey": "Id",
"fields-swimlaneKey": "Assignee",
filterSettings: filter
},
React.createElement("columns", null,
```

```

React.createElement("column", { headerText: "Backlog", key: "Open" })),
React.createElement("column", { headerText: "In Progress", key: "InProgress"
}),
React.createElement("column", { headerText: "Done", key: "Close" })
),
),
document.getElementById('kanbanBoard-default')
);
</script>

```



## Columns

Column fields are present in the [dataSource](#) schema and it is rendering cards based its mapping column values.

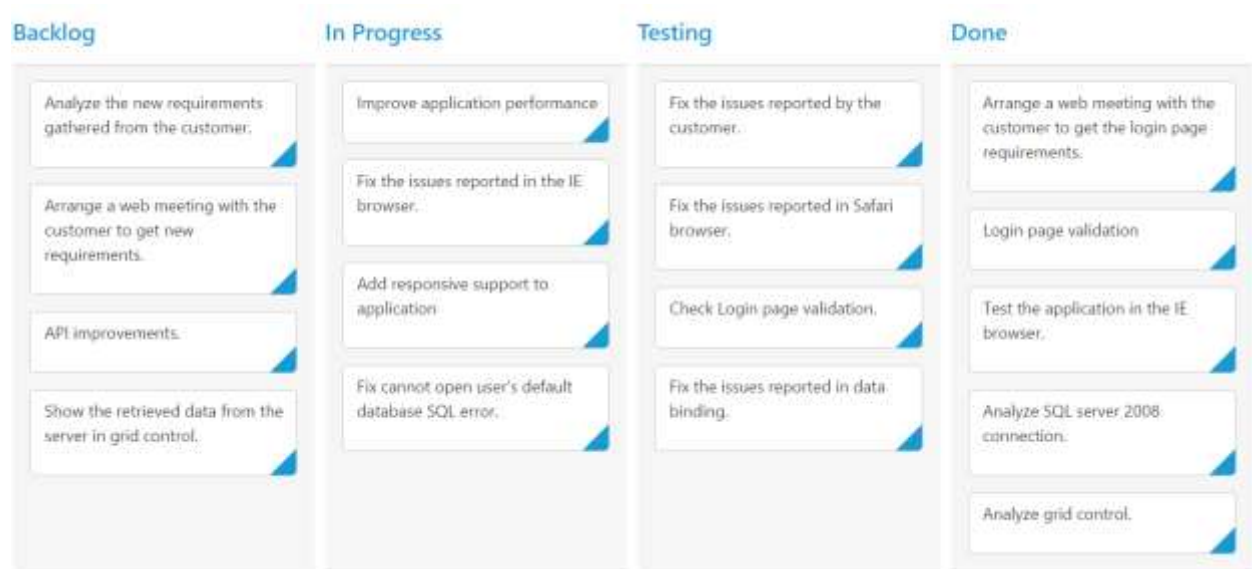
## Key Mapping

To render Kanban with simple cards, you need to map the `dataSource` fields to Kanban cards and [columns](#). The required mapping field are listed as follows

Mapping Fields	Description
<a href="#">keyField</a>	Map the column name to use as <a href="#">key</a> values to columns.
<a href="#">column-key</a>	Map the corresponding <code>key</code> values of <code>keyField</code> column to each columns.
<a href="#">column-headerText</a>	It represents the title for particular column
<a href="#">fields-content</a>	Map the column name to use as content to cards.

**Note:** 1. If the column with `keyField` is not in the `dataSource` and `key` values specified will not available in column values, then the cards will not be rendered.

## 2. If the



`fields-content` is not in the `dataSource`, then empty cards will be rendered.

The following code example describes the above behavior.

**HTML**

```
var data =
ej.DataManager(window.kanbanData).executeLocal(ej.Query().take(10));
ReactDOM.render(
<EJ.Kanban dataSource={data} keyField="Status" fields-content="Summary"
fields-primaryKey="Id">
<columns>
<column headerText="Backlog" key="Open"></column>
<column headerText="In Progress" key="InProgress"></column>
<column headerText="Testing" key="Testing"></column>
<column headerText="Done" key="Close"></column>
</columns>
</EJ.Kanban>,
document.getElementById('kanbanboard-default')
);
```

The following output is displayed as a result of the above code example.



**Multiple Key Mapping**

You can map more than one `datasource` fields as `key` values to show different key cards into single column. For e.g , you can map "Validate,In progress" keys under "In progress" column.

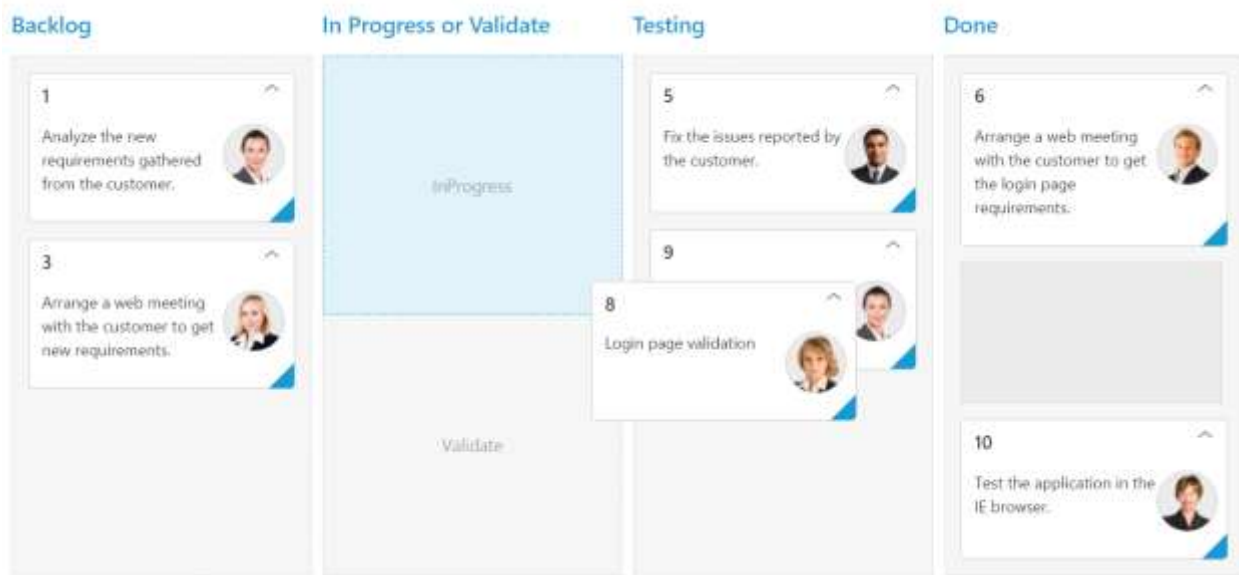
The following code example and screenshot which describes the above behavior.

**HTML**

```
var data =
ej.DataManager(window.kanbanData).executeLocal(ej.Query().take(20));
ReactDOM.render(
```

```
<EJ.Kanban dataSource={data} keyField="Status" fields-content="Summary"
fields-primaryKey="Id" allowTitle={true}>
<columns>
<column headerText="Backlog" key="Open"></column>
<column headerText="In Progress or Validate"
key="InProgress,Validate"></column>
<column headerText="Testing" key="Testing"></column>
<column headerText="Done" key="Close"></column>
</columns>
</EJ.Kanban>,
document.getElementById('kanbanboard-default')
);
```

The following output is displayed as a result of the above code example.



## Headers

### Header Template

The template design that applies on for the column header. To render template, set [headerTemplate](#) property of the [column](#).

You can use JsRender syntax in the template. For more information about JsRender syntax, please refer the [link](#).

The following code example describes the above behavior.

### HTML

```
<div id="kanbanboard-default"></div>
<script src="app/kanbanboard/default.js"></script>
<script id="column1" type="text/x-jsrender">
<span class="e-backlog e-icon"></span> Backlog
</script>
<div id="column4">
<span class="e-done e-icon"></span> Done
</div>
<style type="text/css">
```

```

.e-backlog,.e-done {
font-size: 16px;
padding-right: 5px;
display: inline-block;
}
.e-backlog:before {
content: "\e807";
}
.e-done:before {
content: "\e80a";
}
</style>

```

Kanban control can be initialized with the following in HTML document.

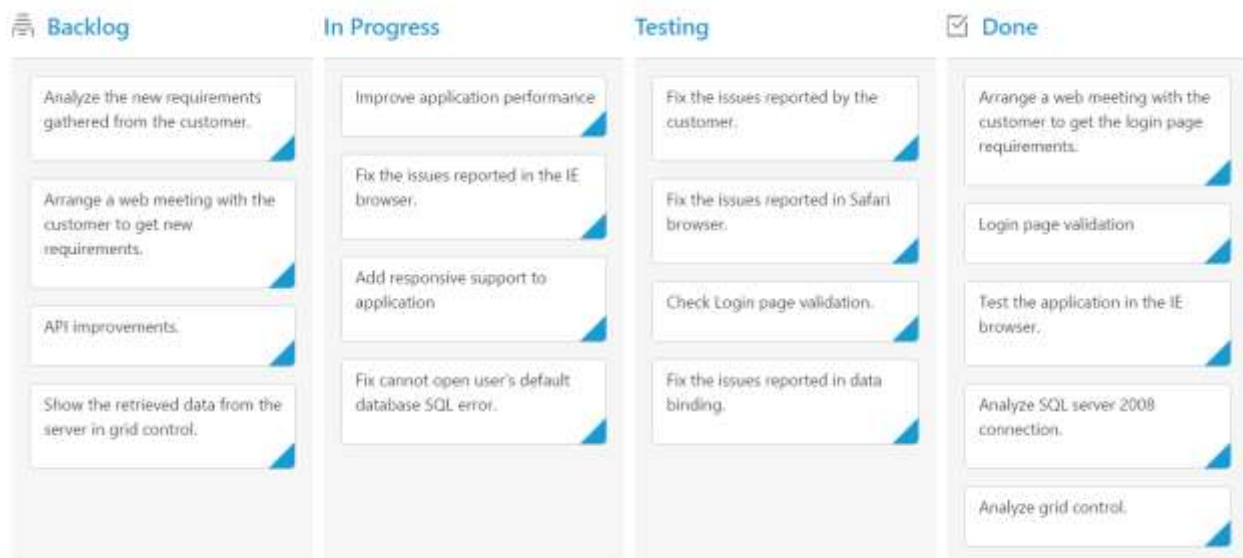
### HTML

```

var data =
ej.DataManager(window.kanbanData).executeLocal(ej.Query().take(20));
ReactDOM.render(
<EJ.Kanban dataSource={data} keyField="Status" fields-content="Summary"
fields-primaryKey="Id">
<columns>
<column headerText="Backlog" key="Open" headerTemplate="#column1"></column>
<column headerText="In Progress" key="InProgress"></column>
<column headerText="Testing" key="Testing"></column>
<column headerText="Done" key="Close" headerTemplate="#column4"></column>
</columns>
</EJ.Kanban>,
document.getElementById('kanbanboard-default')
);

```

The following output is displayed as a result of the above code example.





### Width

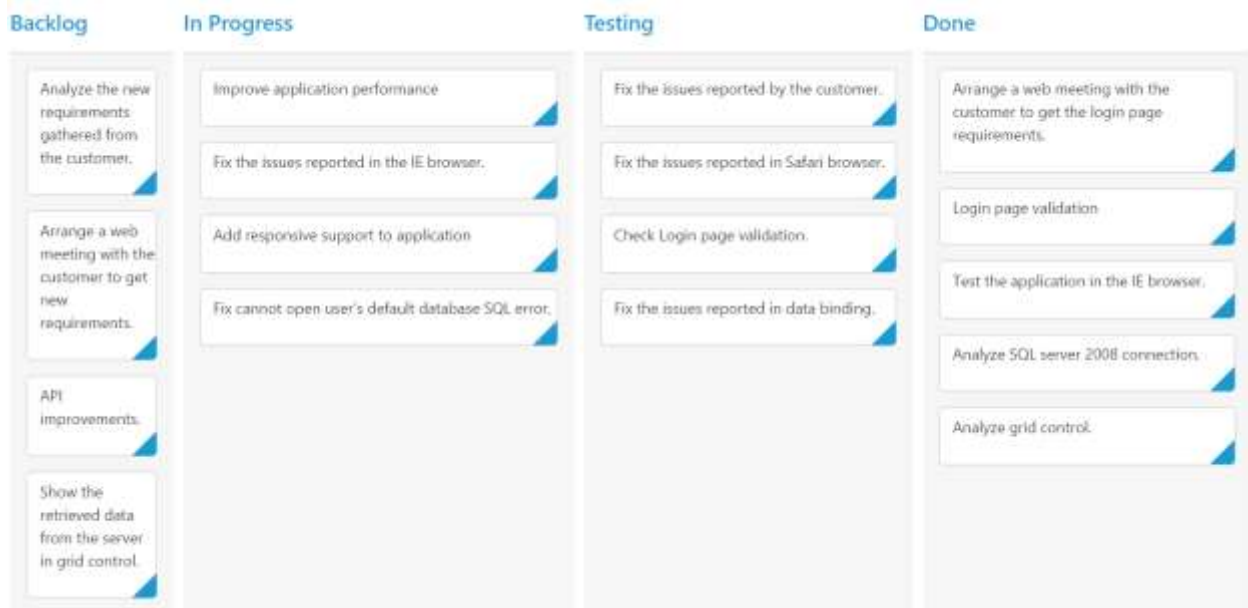
You can specify the width for particular column by setting [width](#) property of [column](#) as in pixel (ex: 100) or in percentage (ex: 40%).

The following code example describes the above behavior.

### HTML

```
var data =
ej.DataManager(window.kanbanData).executeLocal(ej.Query().take(20));
ReactDOM.render(
<EJ.Kanban dataSource={data} keyField="Status" fields-content="Summary"
fields-primaryKey="Id">
<columns>
<column headerText="Backlog" key="Open" width="5%"></column>
<column headerText="In Progress" key="InProgress" width="12%"></column>
<column headerText="Testing" key="Testing" width="100"></column>
<column headerText="Done" key="Close" width="100"></column>
</columns>
</EJ.Kanban>,
document.getElementById('kanbanboard-default')
);
```

The following output is displayed as a result of the above code example.



### Visibility

You can hide particular column in Kanban by setting [visible](#) property of it as false.

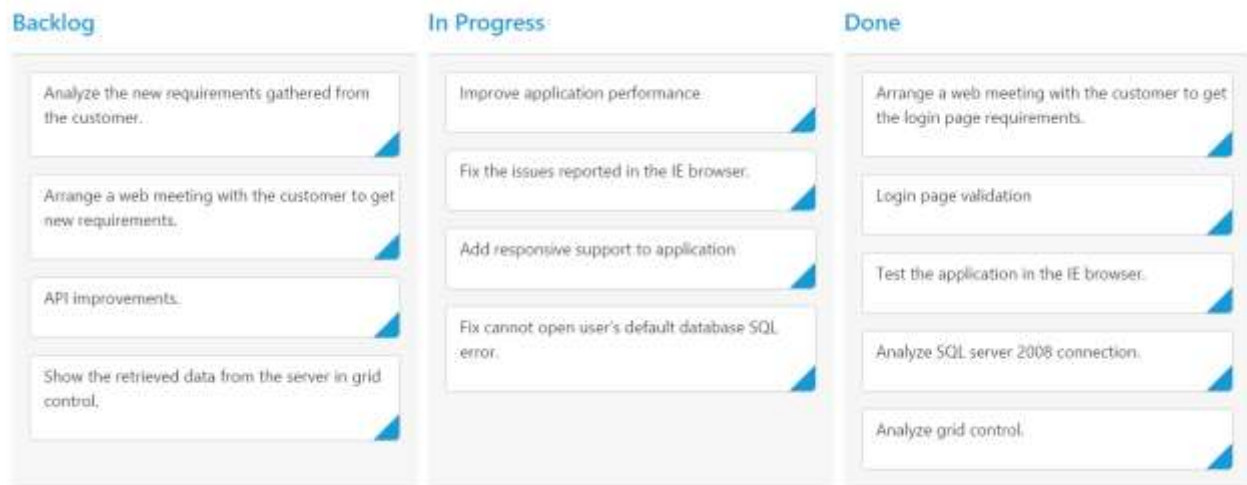
The following code example describes the above behavior.

### HTML

```
var data =
ej.DataManager(window.kanbanData).executeLocal(ej.Query().take(20));
ReactDOM.render(
```

```
<EJ.Kanban dataSource={data} keyField="Status" fields-content="Summary"
fields-primaryKey="Id">
<columns>
<column headerText="Backlog" key="Open"></column>
<column headerText="In Progress" key="InProgress"></column>
<column headerText="Testing" key="Testing" visible={false}></column>
<column headerText="Done" key="Close"></column>
</columns>
</EJ.Kanban>,
document.getElementById('kanbanboard-default')
);
```

The following output is displayed as a result of the above code example.



### Toggle

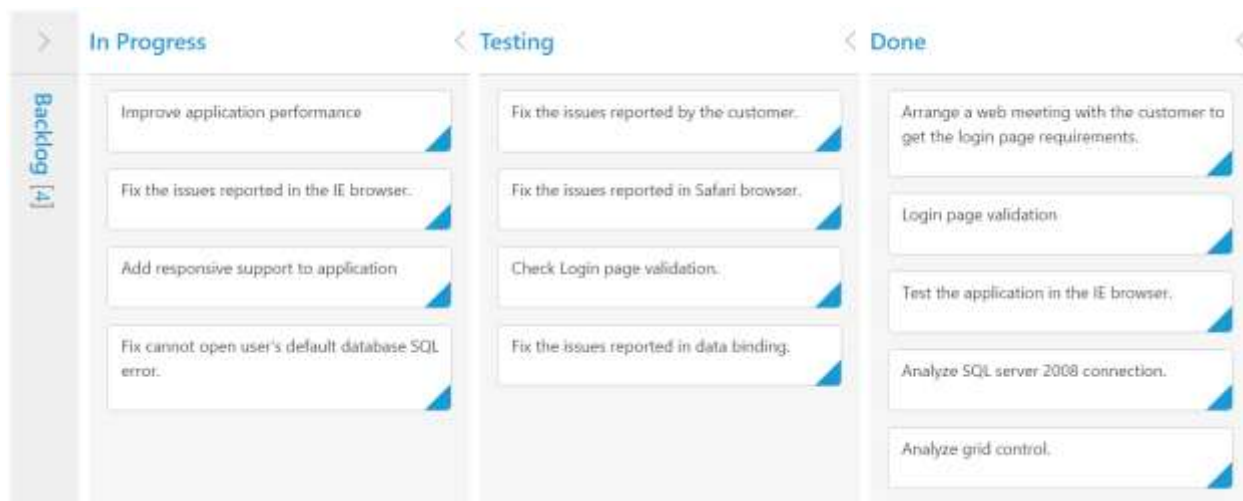
You can set particular column collapsed state in Kanban by setting [isCollapsed](#) property of it as true. You need to set [allowToggleColumn](#) as true to use “Expand/Collapse” Column.

The following code example describes the above behavior.

### HTML

```
var data =
ej.DataManager(window.kanbanData).executeLocal(ej.Query().take(20));
ReactDOM.render(
<EJ.Kanban dataSource={data} keyField="Status" fields-content="Summary"
fields-primaryKey="Id" allowToggleColumn={true}>
<columns>
<column headerText="Backlog" key="Open" isCollapsed={true}></column>
<column headerText="In Progress" key="InProgress"></column>
<column headerText="Testing" key="Testing"></column>
<column headerText="Done" key="Close"></column>
</columns>
</EJ.Kanban>,
document.getElementById('kanbanboard-default')
);
```

The following output is displayed as a result of the above code example.



### Allow Dragging

You can enable and disable drag behavior to the cards in the Kanban columns using the `allowDrag` property and the default value is `true`.

The following code example describes the above behavior.

### HTML

```
var data =
ej.DataManager(window.kanbanData).executeLocal(ej.Query().take(20));
ReactDOM.render(
<EJ.Kanban dataSource={data} keyField="Status" fields-content="Summary"
fields-primaryKey="Id" fields-priority="RankId">
<columns>
<column headerText="Backlog" key="Open" allowDrag={false}></column>
<column headerText="In Progress" key="InProgress"></column>
<column headerText="Done" key="Close"></column>
</columns>
</EJ.Kanban>,
document.getElementById('kanbanboard-default')
);
```

The following output is displayed as a result of the above code example.



### Allow Dropping

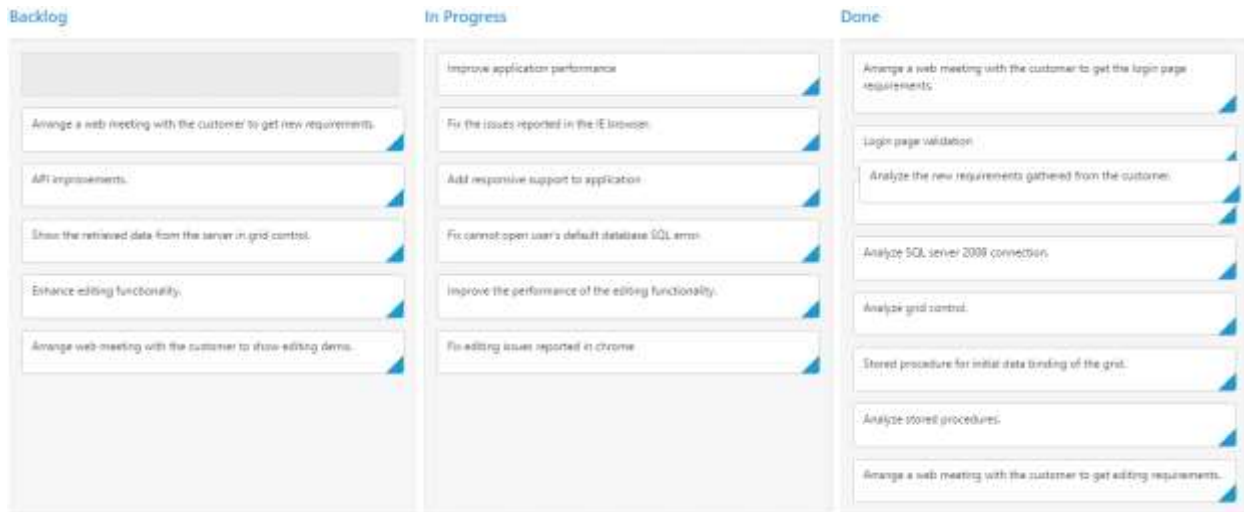
You can enable and disable drop behavior to the cards in the Kanban columns using the `allowDrop` property and the default value is `true`.

The following code example describes the above behavior.

### HTML

```
var data =
ej.DataManager(window.kanbanData).executeLocal(ej.Query().take(20));
ReactDOM.render(
<EJ.Kanban dataSource={data} keyField="Status" fields-content="Summary"
fields-primaryKey="Id" fields-priority="RankId">
<columns>
<column headerText="Backlog" key="Open"></column>
<column headerText="In Progress" key="InProgress"></column>
<column headerText="Done" key="Close" allowDrop={false}></column>
</columns>
</EJ.Kanban>,
document.getElementById('kanbanboard-default')
);
```

The following output is displayed as a result of the above code example.



### Items Count

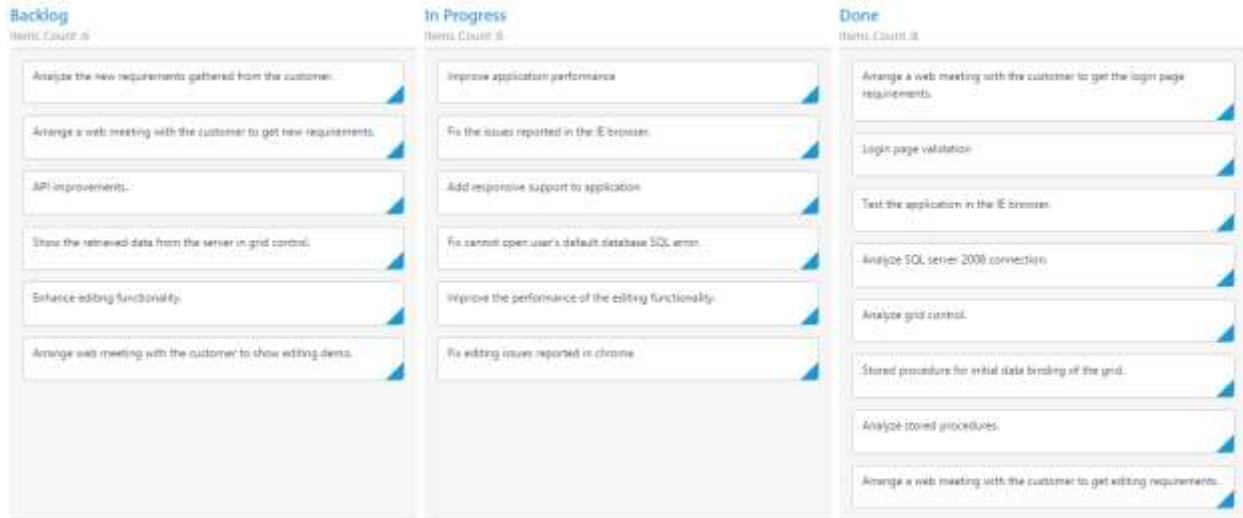
You can show total cards count in each column's header using the property `enableTotalCount` and the default value is `false`.

The following code example describes the above behavior.

### HTML

```
var data =
ej.DataManager(window.kanbanData).executeLocal(ej.Query().take(20));
ReactDOM.render(
<EJ.Kanban dataSource={data} keyField="Status" fields-content="Summary"
fields-primaryKey="Id" enableTotalCount={true}>
<columns>
<column headerText="Backlog" key="Open"></column>
<column headerText="In Progress" key="InProgress"></column>
<column headerText="Done" key="Close"></column>
</columns>
</EJ.Kanban>,
document.getElementById('kanbanboard-default')
);
```

The following output is displayed as a result of the above code example.



### Customize Items Count Text

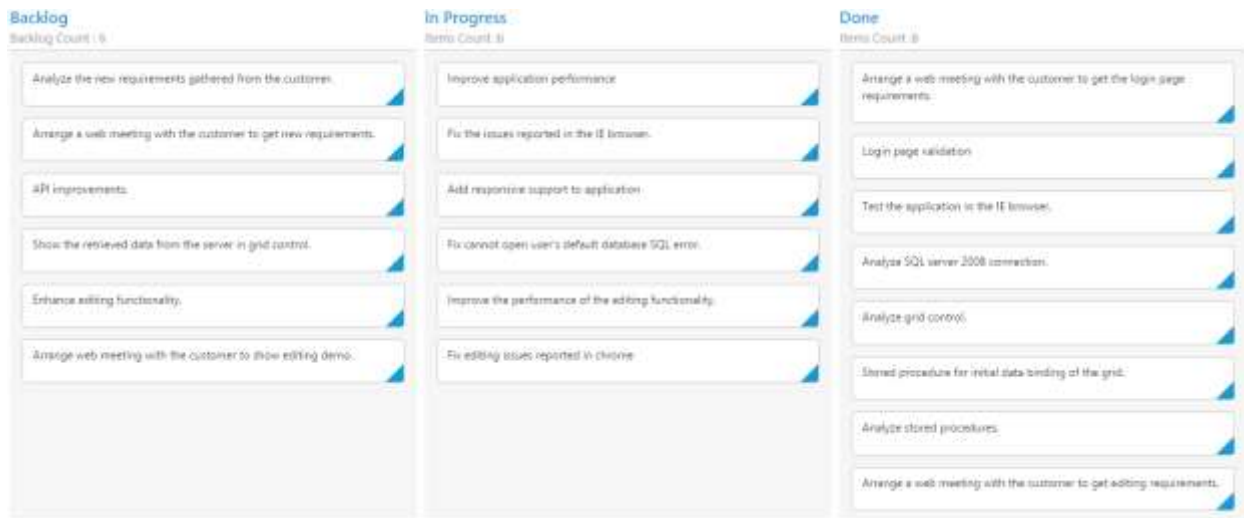
You can customize the Items count text using the property `totalCount-text`.

The following code example describes the above behavior.

### HTML

```
var data =
ej.DataManager(window.kanbanData).executeLocal(ej.Query().take(20));
ReactDOM.render(
<EJ.Kanban dataSource={data} keyField="Status" fields-content="Summary"
fields-primaryKey="Id" enableTotalCount={true}>
<columns>
<column headerText="Backlog" key="Open" totalCount-text="Backlog
Count"></column>
<column headerText="In Progress" key="InProgress"></column>
<column headerText="Done" key="Close"></column>
</columns>
</EJ.Kanban>,
document.getElementById('kanbanboard-default')
);
```

The following output is displayed as a result of the above code example.



## Workflows

Workflows can be defined to set the flow of card moving between the Kanban column statuses and it is applicable to drag and drop and context menu features.

You can set [workflows](#) as array of Objects which consists of [key](#) and [allowedTransitions](#) properties. The [allowedTransitions](#) accepts more than one transition of the specific column key mentioned in [key](#) property.

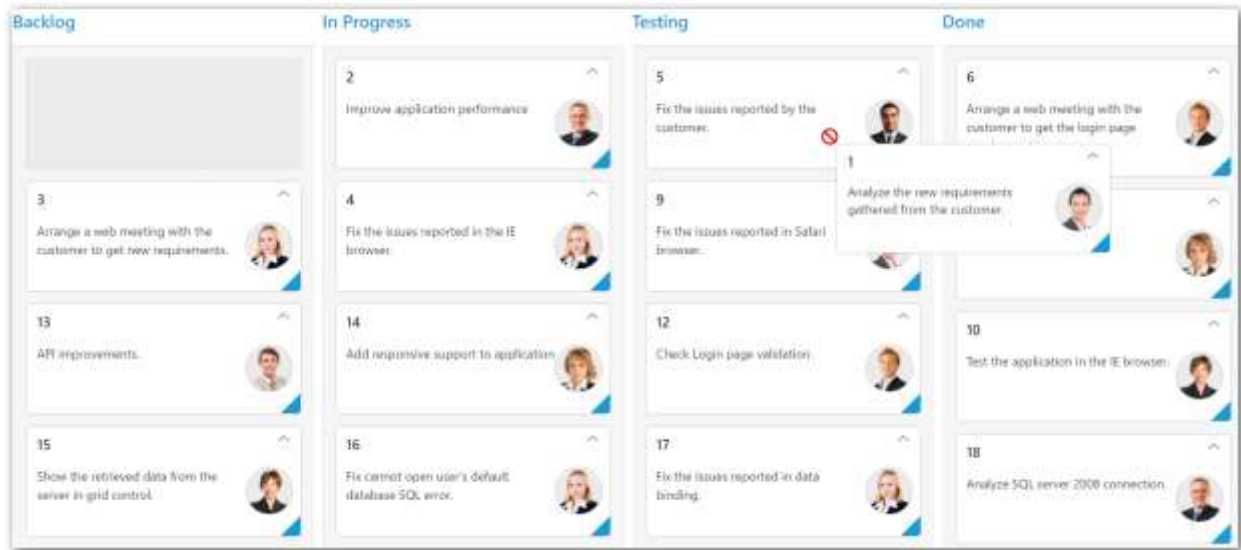
If a card is to be dragged to not allowed transition columns, then not supported warning symbol will be displayed for denoting the error.

The following code example describes the above Workflow functionality.

## HTML

```
var data =
ej.DataManager(window.kanbanData).executeLocal(ej.Query().take(20));
var workflow = [
{ key: "Open", allowedTransitions: "InProgress" },
{ key: "InProgress", allowedTransitions: "Testing, Close" }
];
ReactDOM.render(
<EJ.Kanban dataSource={data} keyField="Status" fields-content="Summary"
fields-primaryKey="Id" workflows={workflow} fields-imageUrl="ImgUrl"
allowTitle={true}>
<columns>
<column headerText="Backlog" key="Open"></column>
<column headerText="In Progress" key="InProgress"></column>
<column headerText="Testing" key="Testing"></column>
<column headerText="Done" key="Close"></column>
</columns>
</EJ.Kanban>,
document.getElementById('kanbanboard-default')
);
```

The following output is displayed as a result of the above code example.



## Swim lanes

Swim lanes are a horizontal categorization of issues in the Kanban control which brings transparency to the workflow. This can be enabled by mapping the [fields-swimlaneKey](#) to appropriate column name in the [dataSource](#).

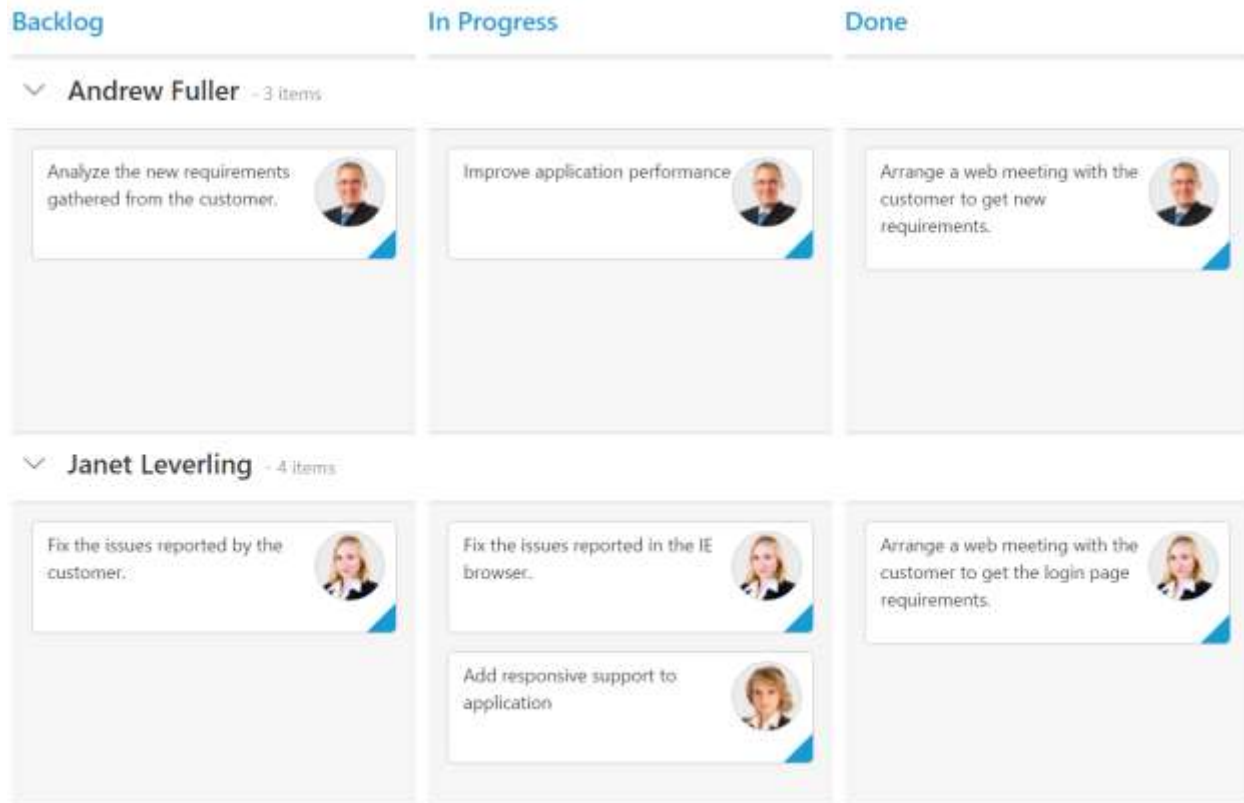
The following code example describes the above behavior.

### HTML

```
var data =
ej.DataManager(window.kanbanData).executeLocal(ej.Query().take(20));
ReactDOM.render(
<EJ.Kanban dataSource={data} keyField="Status" fields-content="Summary"
fields-primaryKey="Id" fields-swimlaneKey="Assignee" fields-
imageUrl="ImgUrl">
<columns>
<column headerText="Backlog" key="Open"></column>
<column headerText="In Progress" key="InProgress"></column>
<column headerText="Done" key="Close"></column>
</columns>
</EJ.Kanban>,
document.getElementById('kanbanBoard-default')
);
```

The following output is displayed as a result of the above code example.





### Drag And Drop between swim lanes

You can set '[allowDragAndDrop](#)' property of '[swimlaneSettings](#)' as true to enable Drag and Drop between the swim lanes.

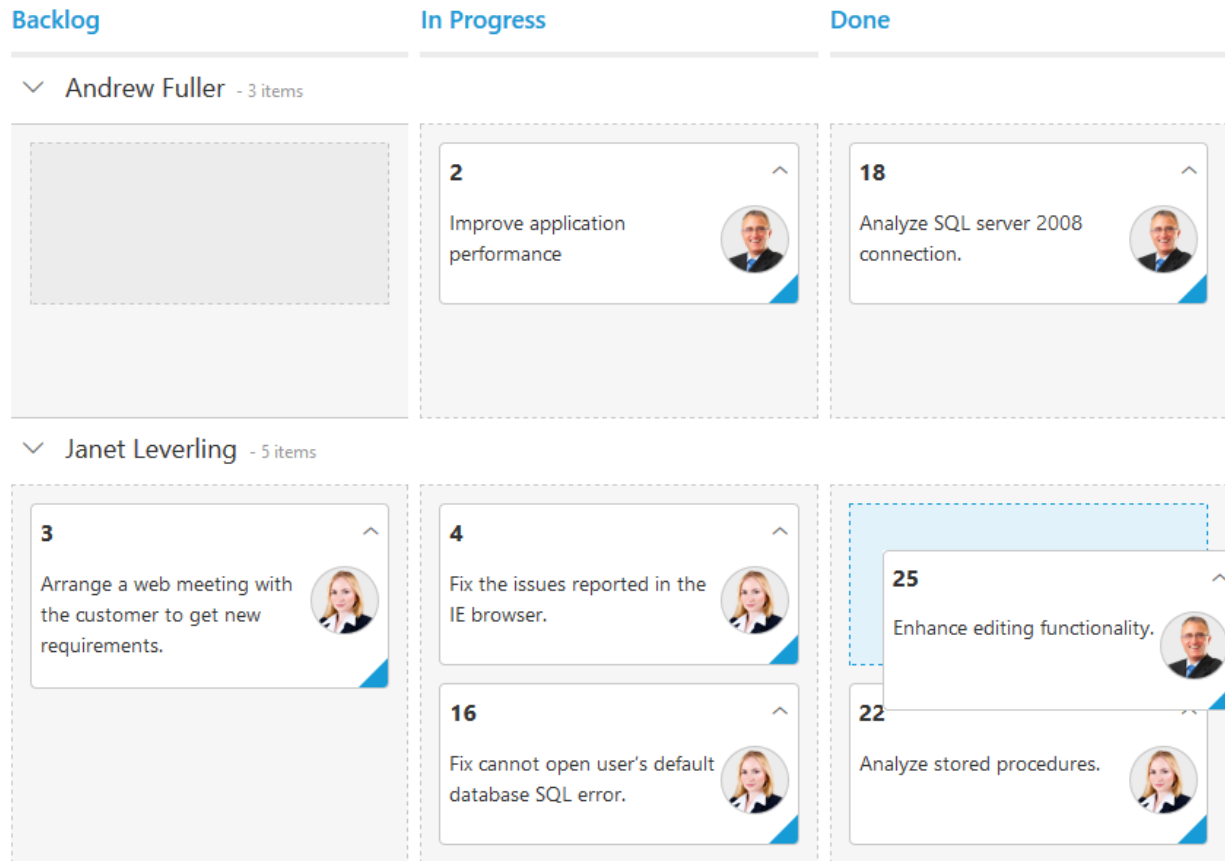
If a card is to be dragged in the same swim lane, only a droppable target cell is added to the dotted line border. If a card is dragged from one swim lane to another, all the Kanban cells will be added to the dotted line borders, except the dragged card cell.

The following code example describes the above behavior.

### HTML

```
var data =
ej.DataManager(window.kanbanData).executeLocal(ej.Query().take(20));
ReactDOM.render(
<EJ.Kanban dataSource={data} keyField="Status" fields-content="Summary"
fields-primaryKey="Id" fields-swimlaneKey="Assignee" fields-
imageUrl="ImgUrl" swimlaneSettings-allowDragAndDrop={true}>
<columns>
<column headerText="Backlog" key="Open"></column>
<column headerText="In Progress" key="InProgress"></column>
<column headerText="Done" key="Close"></column>
</columns>
</EJ.Kanban>,
document.getElementById('kanbanBoard-default')
);
```

The following output is displayed as a result of the above code example.



### Unassigned swim lane group

Unassigned swim lane feature provides option to group some common swim lane key values as separate swim lane group. You can enable and disable this behavior using the property [swimlaneSettings-unassignedGroup-enable](#).

User can use default common key values or user defined key values.

- Using default values
- Using user defined values

**Note:** By default, given common keys are grouped under the swim lane name “Unassigned”, user can customize the name using localization.

### Using default values

By default, the swim lane keys of card which is having null, undefined, empty string ("") values will be grouped as unassigned category when [swimlaneSettings-unassignedGroup-enable](#) property is set as true.

Default values in the [swimlaneSettings-unassignedGroup-keys](#) collection are null, undefined, empty string ("").

The following code example describes the above behavior.

### HTML

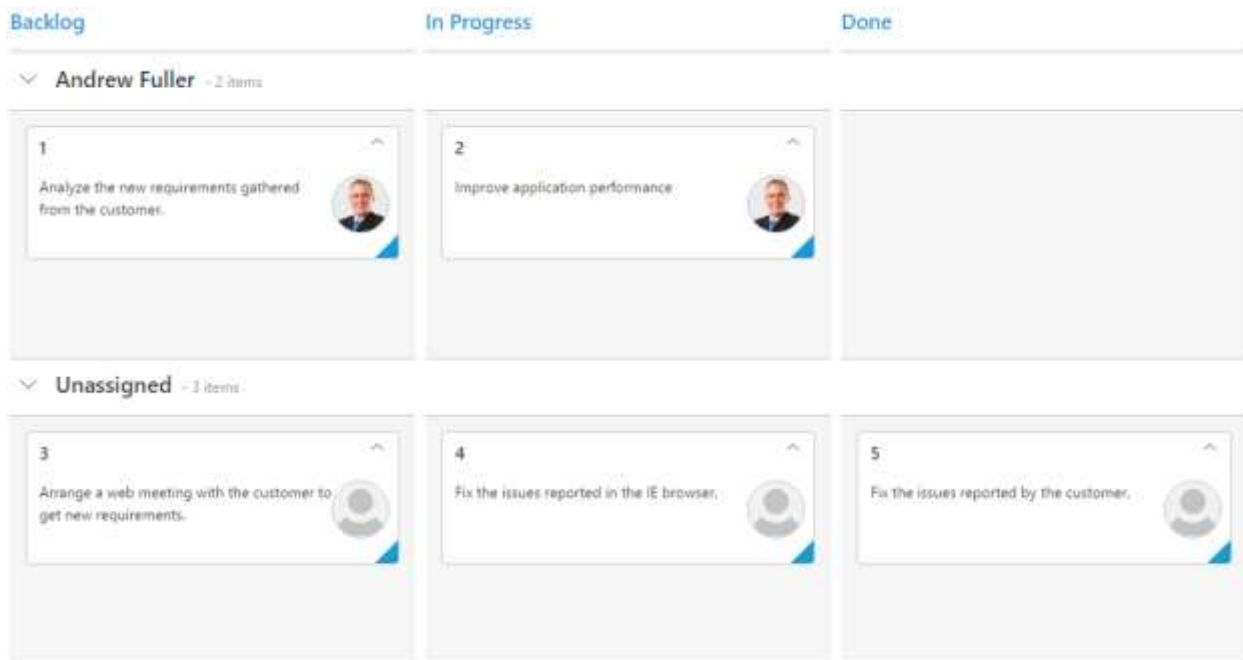
```
window.kanbanData = [{ Id: 1, Status: "Open", Summary: "Analyze the new requirements gathered from the customer.", Type: "Story", Priority: "Low",
```

```

Tags: "Analyze, Customer", Estimate: 3.5, Assignee: "Andrew Fuller", ImgUrl:
"../content/images/kanban/2.png", RankId: 1 }, { Id: 2, Status:
"InProgress", Summary: "Improve application performance", Type:
"Improvement", Priority: "Normal", Tags: "Improvement", Estimate: 6,
Assignee: "Andrew Fuller", ImgUrl: "../content/images/kanban/2.png", RankId:
1 }, { Id: 3, Status: "Open", Summary: "Arrange a web meeting with the
customer to get new requirements.", Type: "Others", Priority: "Critical",
Tags: "Meeting", Estimate: 5.5, Assignee: undefined, RankId: 2 }, { Id: 4,
Status: "InProgress", Summary: "Fix the issues reported in the IE browser.",
Type: "Bug", Priority: "Release Breaker", Tags: "IE", Estimate: 2.5,
Assignee: null, RankId: 2 }, { Id: 5, Status: "Close", Summary: "Fix the
issues reported by the customer.", Type: "Bug", Priority: "Low", Tags:
"Customer", Estimate: "3.5", Assignee: "", RankId: 1 }]];
var data =
ej.DataManager(window.kanbanData).executeLocal(ej.Query().take(20));
ReactDOM.render(
<EJ.Kanban dataSource={data} keyField="Status" fields-content="Summary"
fields-primaryKey="Id" fields-swimlaneKey="Assignee" fields-
imageUrl="ImgUrl">
<columns>
<column headerText="Backlog" key="Open"></column>
<column headerText="In Progress" key="InProgress"></column>
<column headerText="Done" key="Close"></column>
</columns>
</EJ.Kanban>,
document.getElementById('kanbanBoard-default')
);

```

The output of the above code example.



### Drag and Drop

By default [allowDragAndDrop](#) is true. Cards can be transited from one column to another column, by dragging and dropping. And it has drop position indicator which enables easier positioning of cards

**Note:** It is not possible transit cards to other swim lanes through Drag and Drop.

### Prioritization of cards

Prioritizing cards is easy with drag-and-drop re-ordering that helps you to categorize most important work at the top. Cards can be categorized with priority by mapping specific database field into [fields-priority](#) property.

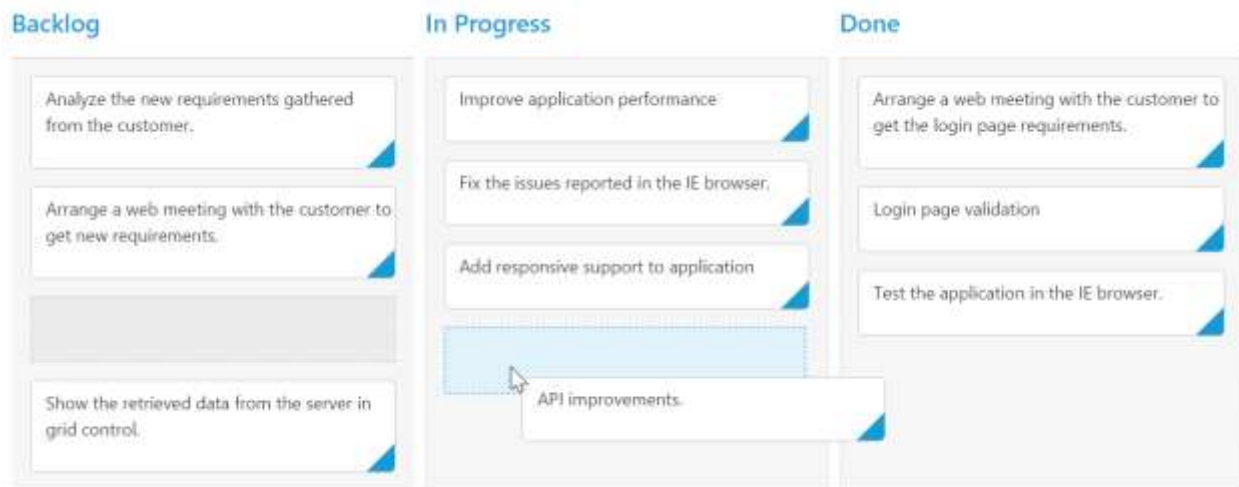
**RankId** defined in the **dataSource** which is consist priority of cards. The **RankId** will be changed while card ordering changes through **Drag and Drop** and **Editing**.

The following code example describes the above behavior.

### HTML

```
var data =
ej.DataManager(window.kanbanData).executeLocal(ej.Query().take(20));
ReactDOM.render(
<EJ.Kanban dataSource={data} keyField="Status" fields-content="Summary"
fields-primaryKey="Id" fields-priority="RankId">
<columns>
<column headerText="Backlog" key="Open"></column>
<column headerText="In Progress" key="InProgress"></column>
<column headerText="Done" key="Close"></column>
</columns>
</EJ.Kanban>,
document.getElementById('kanbanboard-default')
);
```

The following output is displayed as a result of the above code example.



### Cards

#### Customization

Cards can be customized with appropriate mapping fields from the database. The customizable mapping properties are listed as follows

Mapping Fields	Description
<a href="#">fields-content</a>	Map the column name to use as content to cards.

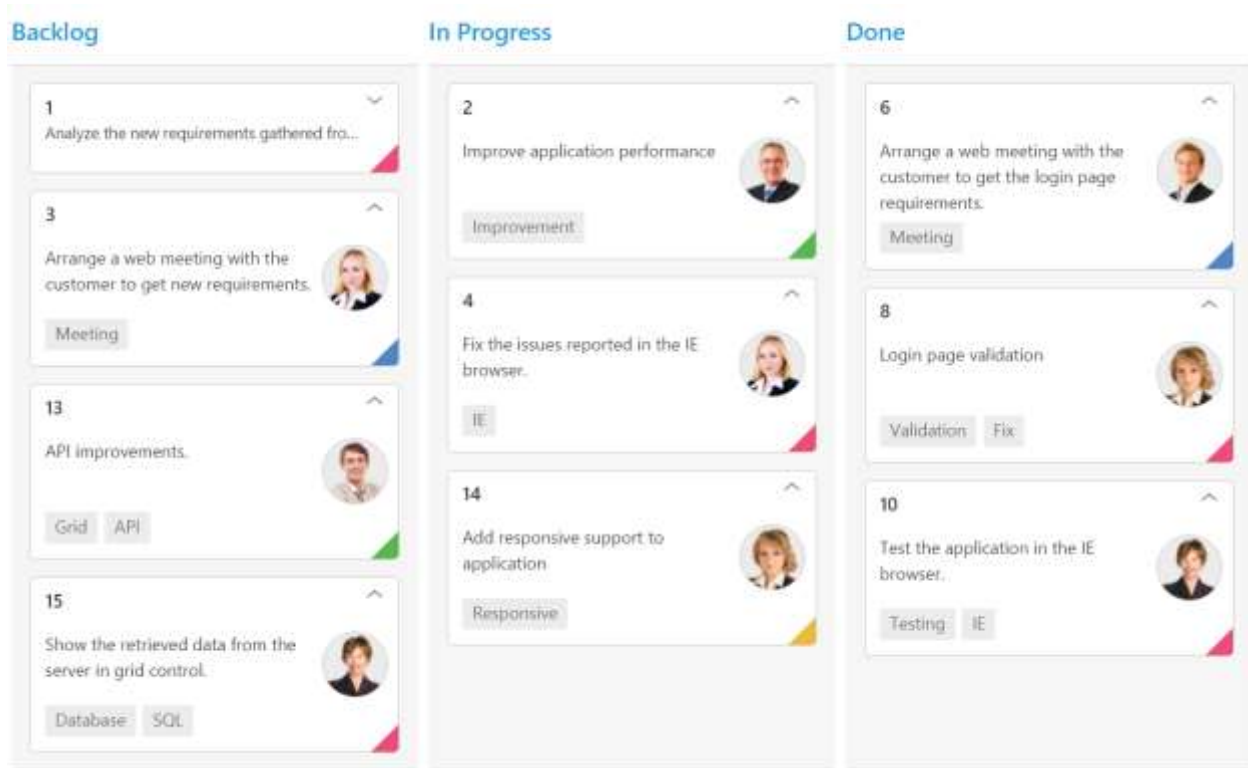
<a href="#">fields-tag</a>	Map the column name to use as tag. Multiple tags can be given with comma separated. E.g. "API", "SQL, Database".
<a href="#">fields-color</a>	Map the column name to use as colors to highlight cards left border.
<a href="#">cardSettings-colorMapping</a>	Map the colors to use with column values which is mapped with <code>fields-color</code> .
<a href="#">fields-imageUrl</a>	Map the column name to use as image to cards.
<a href="#">fields-primaryKey</a>	Map the column name to use as primary key to cards.
<a href="#">fields-priority</a>	Map the column name to use as priority to cards.
<a href="#">fields-title</a>	Map the column name to use as title to cards. Default title is <code>primaryKey</code> .
<a href="#">allowTitle</a>	Set as true to enable title for card.

The following code example describes the above behavior.

#### HTML

```
var data =
ej.DataManager(window.kanbanData).executeLocal(ej.Query().take(20));
var colormap = {
"#cb2027": "Bug, Story",
"#67ab47": "Improvement",
"#fbae19": "Epic",
"#6a5da8": "Others"
};
ReactDOM.render(
<EJ.Kanban dataSource={data} keyField="Status" fields-content="Summary"
fields-primaryKey="Id" fields-priority="RankId" fields-tag="Tags" fields-
color="Type" fields-imageUrl="ImgUrl" cardSettings-colorMapping={colormap}
allowTitle={true}>
<columns>
<column headerText="Backlog" key="Open"></column>
<column headerText="In Progress" key="InProgress"></column>
<column headerText="Done" key="Close"></column>
</columns>
</EJ.Kanban>,
document.getElementById('kanbanboard-default')
);
```

The following output is displayed as a result of the above code example.



### Template

Templates are used to create custom card layout as per the user convenient. HTML templates can be specified in the [template](#) property of the [cardSettings](#) as an ID of the template's HTML element.

You can use JsRender syntax in the template. For more information about JsRender syntax, please refer this [link](#).

The following code example describes the above behavior.

### HTML

```
<div id="kanbanboard-default"></div>
<script src="app/kanbanboard/default.js"></script>
<script id="cardtemplate" type="text/x-jsrender">
<table class="e-templatetable">
<colgroup>
<col width="10%">
<col width="90%">
</colgroup>
<tbody>
<tr>
<td class="photo">

</td>
<td class="details">
<table>
<colgroup>
<col width="10%">
<col width="90%">
</colgroup>
<tbody>
```

```

<tr>
<td class="CardHeader"> Assignee: </td>
<td>{{:Assignee}}</td>
</tr>
<tr>
<td class="CardHeader"> Summary: </td>
<td>{{:Summary}}</td>
</tr>
<tr>
<td class="CardHeader"> Type: </td>
<td>{{:Type}}</td>
</tr>
</tbody>
</table>
</td>
</tr>
</tbody>
</table>
</script>
<style type="text/css">
.e-templatetable {
width: 100%;
}
.details >table {
margin-left:2px;
border-collapse: separate;
border-spacing: 2px;
width: 100%;
}
.details td {
vertical-align: top;
}
.details {
padding: 8px 8px 10px 0;
}
.photo {
padding: 8px 6px 10px 6px;
text-align: center;
}
.CardHeader {
font-weight: bolder;
padding-right: 10px;
}
</style>

```

## HTML

```

var data =
ej.DataManager(window.kanbanData).executeLocal(ej.Query().take(20));
var colormap = {
"#cb2027": "Bug, Story",
"#67ab47": "Improvement",
"#fbae19": "Epic",
"#6a5da8": "Others"
};
ReactDOM.render(

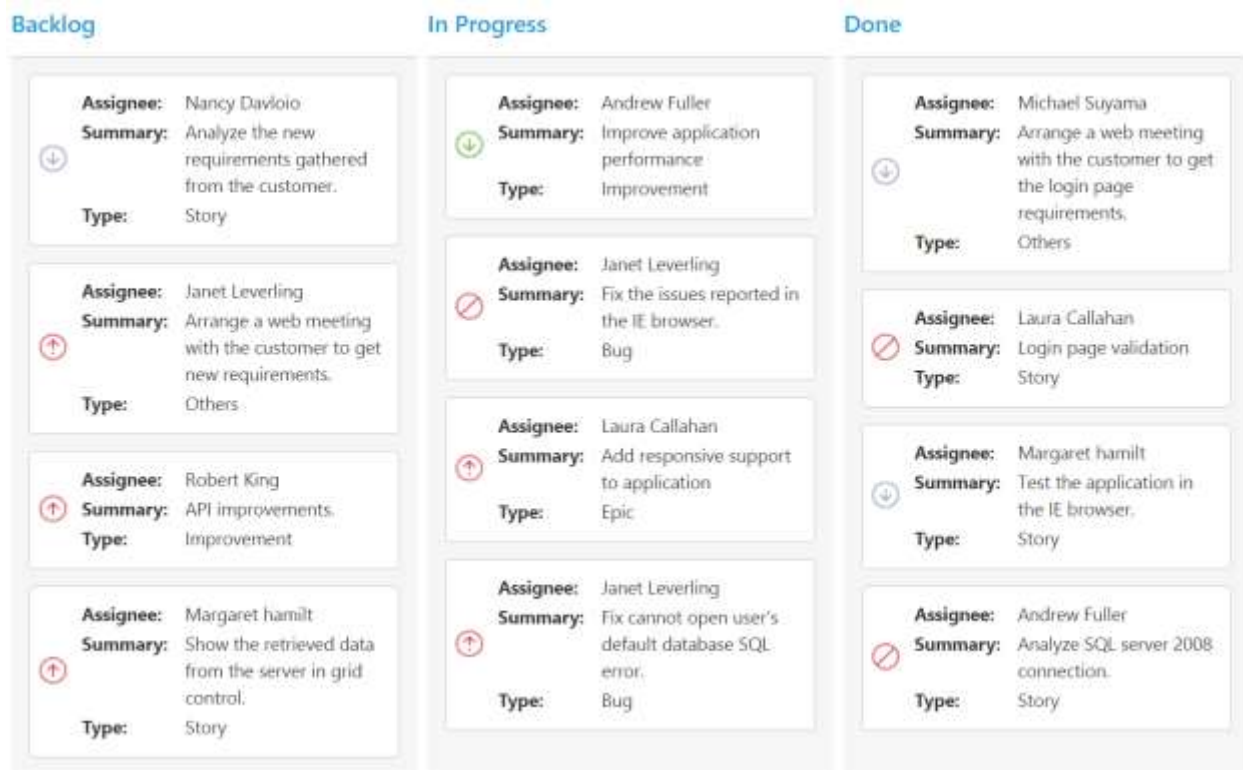
```

```

<EJ.Kanban dataSource={data} keyField="Status" fields-primaryKey="Id"
fields-color="Type" cardSettings-template="#cardtemplate" cardSettings-
colorMapping={colormap} allowTitle={true}>
<columns>
<column headerText="Backlog" key="Open"></column>
<column headerText="In Progress" key="InProgress"></column>
<column headerText="Done" key="Close"></column>
</columns>
</EJ.Kanban>,
document.getElementById('kanbanboard-default')
);

```

The following output is displayed as a result of the above code example.



### Tooltip

You can enable HTML tooltip for Kanban card elements by setting [enable](#) property as true in [tooltipSettings](#).

The following code example describes the above behavior.

### HTML

```

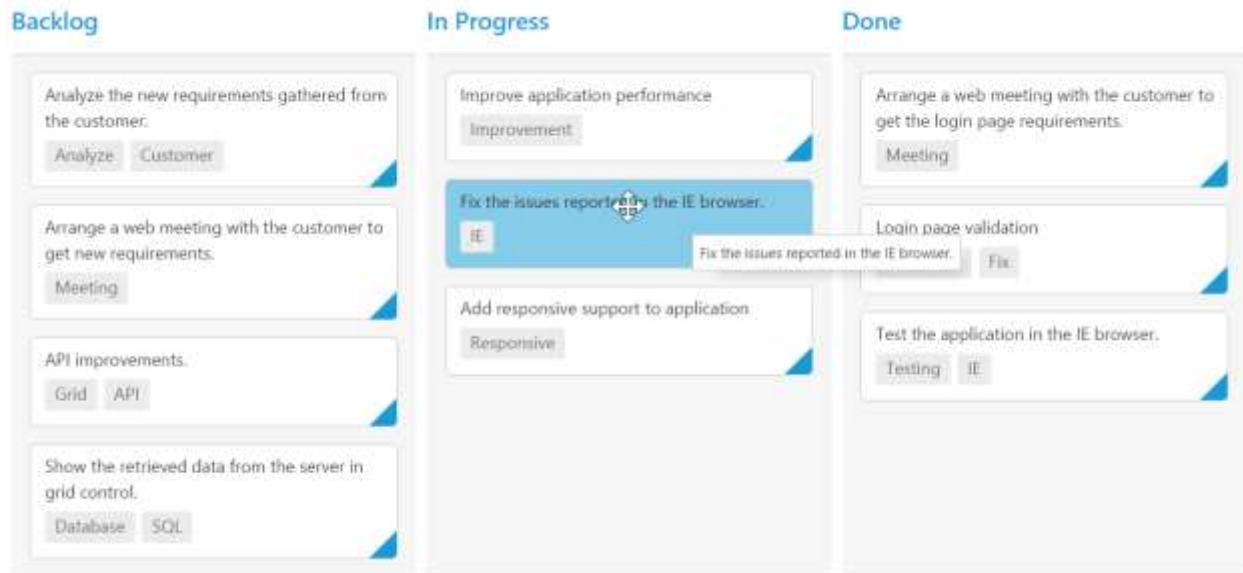
ReactDOM.render (
<EJ.Kanban dataSource={data} keyField="Status" fields-content="Summary"
fields-primaryKey="Id" fields-tag="Tags" tooltipSettings-enable={true}>
<columns>
<column headerText="Backlog" key="Open"></column>
<column headerText="In Progress" key="InProgress"></column>
<column headerText="Done" key="Close"></column>
</columns>

```



```
</EJ.Kanban>,
document.getElementById('kanbanboard-default')
);
```

The following output is displayed as a result of the above code example.



### Template

By making use of template feature with tooltip, all the field names that are mapped from the **dataSource** can be accessed to define the [tooltipSettings-template](#) tooltip for card. The [tooltipSettings-enable](#) must be enabled first.

The following code example describes the tooltip template.

### HTML

```
<script id="tooltipTemp" type="text/x-jsrender">
<div class='e-kanbantooltiptemplate'>
<table>
<tr>
<td class="photo">

</td>
<td class="details">
<table>
<colgroup>
<col width="30%">
<col width="70%">
</colgroup>
<tbody>
<tr>
<td class="CardHeader">Assignee:</td>
<td>{ :Assignee }</td>
</tr>
<tr>
<td class="CardHeader">Type:</td>
<td>{ :Type }</td>
```

```

</tr>
<tr>
<td class="CardHeader">Estimate:</td>
<td>{{:Estimate}}</td>
</tr>
<tr>
<td class="CardHeader">Summary:</td>
<td>{{:Summary}}</td>
</tr>
</tbody>
</table>
</td>
</tr>
</table>
</div>
</script>
<style>
.details >table {
width: 100%;
margin-left:2px;
border-collapse: separate;
border-spacing: 1px;
}
.e-kanbantooltiptemplate {
width: 250px;
padding: 3px;
}
.e-kanbantooltiptemplate > table {
width: 100%;
}
.e-kanbantooltiptemplate td {
vertical-align: top;
}
td.details {
padding-left: 10px;
}
.CardHeader {
font-weight: bolder;
}
</style>

```

## HTML

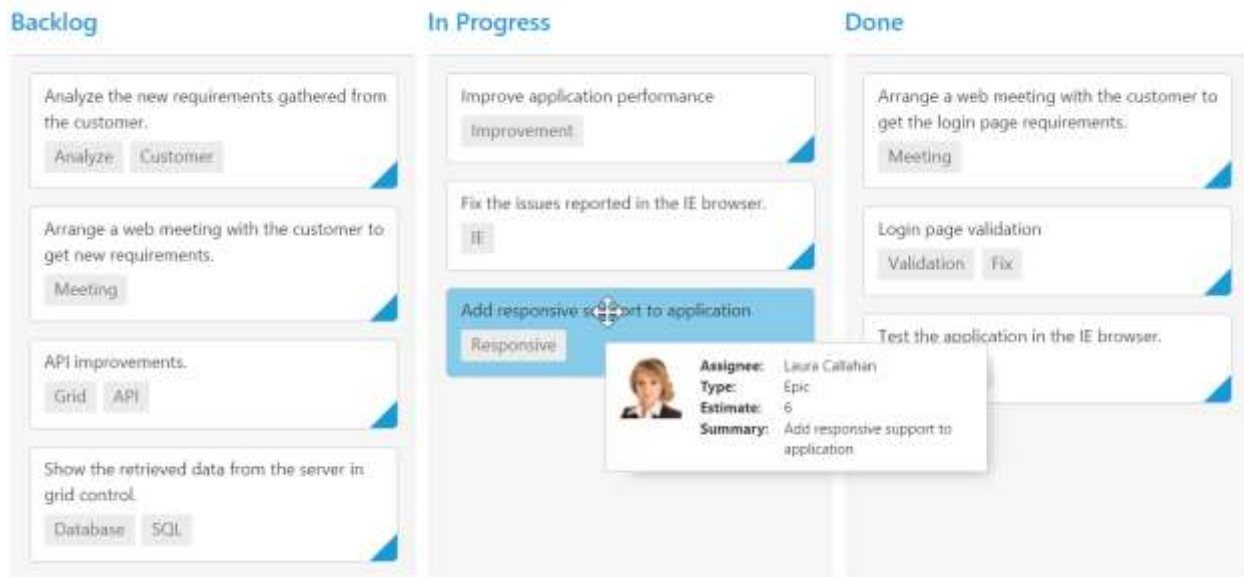
```

var data =
ej.DataManager(window.kanbanData).executeLocal(ej.Query().take(20));
ReactDOM.render(
<EJ.Kanban dataSource={data} keyField="Status" fields-content="Summary"
fields-primaryKey="Id" fields-tag="Tags" tooltipSettings-
template="#tooltipTemp" tooltipSettings-enable={true}>
<columns>
<column headerText="Backlog" key="Open"></column>
<column headerText="In Progress" key="InProgress"></column>
<column headerText="Done" key="Close"></column>
</columns>
</EJ.Kanban>,
document.getElementById('kanbanboard-default')

```

```
);
```

The following output is displayed as a result of the above code example.



## Editing

The Kanban control has support for dynamic insertion, updating and deletion of cards.

Set [editSettings-allowEditing](#) and [editSettings-allowAdding](#) property as true to enable editing/inserting respectively. The primary key for the data source should be defined in [fields-primaryKey](#), for editing to work properly.

You can start the edit action by double clicking the particular card. Similarly, you can add new card to Kanban either by double clicking the particular cell or on an external button which is bound to call [addCard](#) method of Kanban.

Deletion of the card is possible by using [deleteCard](#) by passing primary key as attribute.

**Note:** In Kanban, the **primary key** column will be automatically set to **read only** while editing the card which is to avoid duplicate entry in the cards.

## Configuring Edit Items

You need to configure the list of data source fields that are allowable in editing state using [editSettings-editItems](#) property. The [field](#) property of [editSettings-editItems](#) needs to be mapped with data source fields.

You can map the data source field as title to edit form using [title](#) property of [fields](#). By default, it's mapped with [fields-primaryKey](#).

The following code example describes the above behavior.

## HTML

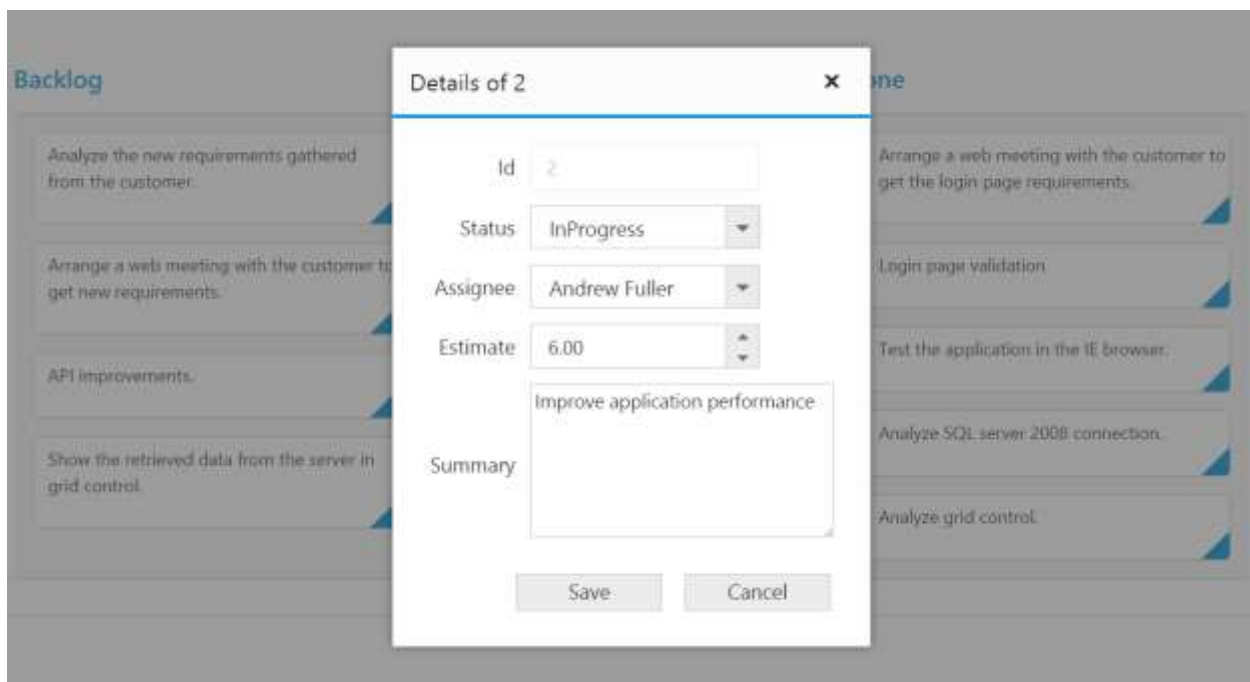
```
var data =
ej.DataManager(window.kanbanData).executeLocal(ej.Query().take(20));
var editItems = [
{ field: "Id" },
```

```

{ field: "Status", editType: ej.Kanban.EditingType.Dropdown },
{ field: "Assignee", editType: ej.Kanban.EditingType.Dropdown },
{ field: "Estimate", editType: ej.Kanban.EditingType.Numeric, editParams: {
decimalPlaces: 2 } },
{ field: "Summary", editType: ej.Kanban.EditingType.TextArea, editParams:
{height:100,width:200}}
];
ReactDOM.render(
<EJ.Kanban dataSource={data} keyField="Status" fields-content="Summary"
fields-primaryKey="Id" editSettings-editItems={editItems} editSettings-
allowEditing={true} editSettings-allowAdding={true}>
<columns>
<column headerText="Backlog" key="Open"></column>
<column headerText="In Progress" key="InProgress"></column>
<column headerText="Done" key="Close"></column>
</columns>
</EJ.Kanban>,
document.getElementById('kanbanboard-default')
);

```

The following output is displayed as a result of the above code example.



## Edit modes

### Dialog

Set [editSettings-editMode](#) as `dialog` to edit data using a dialog box, which displays the fields associated with the data card being edited. Default value is `dialog`.

**Note:** For [editSettings-editMode](#) property you can assign either `string` value ("dialog") or `enum` value (`ej.Kanban.EditMode.Dialog`).

The following code example describes the above behavior.

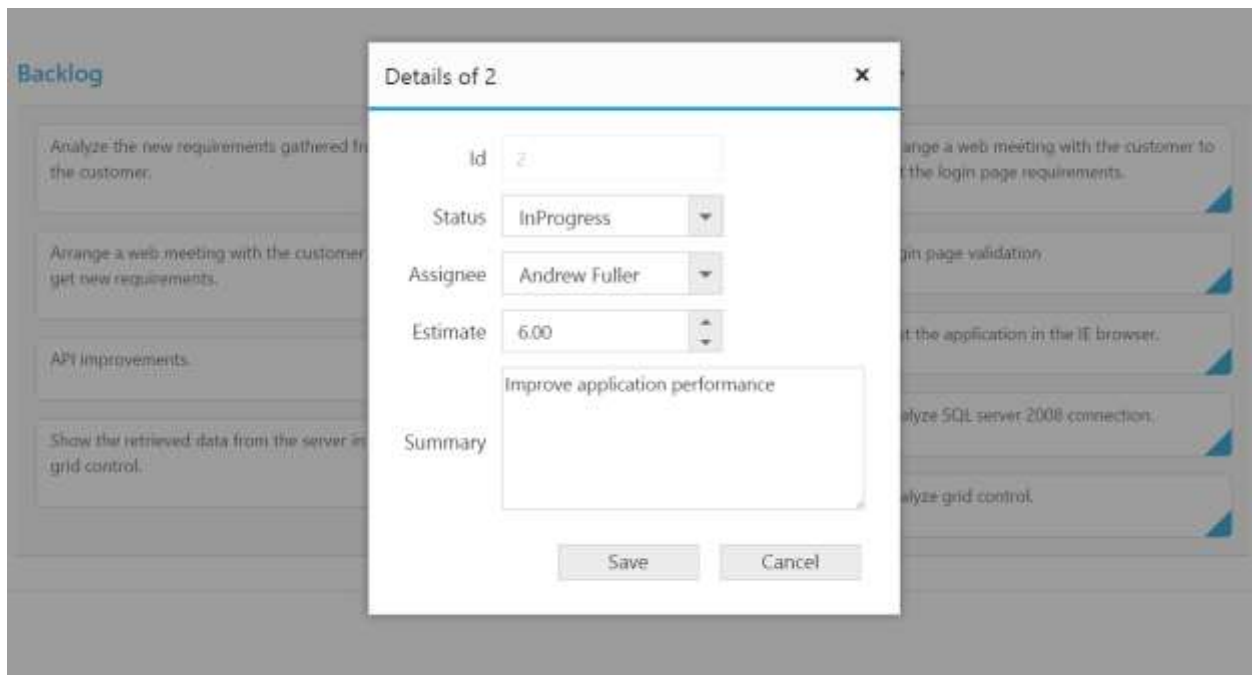
**HTML**

```

var data =
ej.DataManager(window.kanbanData).executeLocal(ej.Query().take(20));
var editItems = [
{ field: "Id" },
{ field: "Status", editType: ej.Kanban.EditingType.Dropdown },
{ field: "Assignee", editType: ej.Kanban.EditingType.Dropdown },
{ field: "Estimate", editType: ej.Kanban.EditingType.Numeric, editParams: {
decimalPlaces: 2 } },
{ field: "Summary", editType: ej.Kanban.EditingType.TextArea }
];
ReactDOM.render(
<EJ.Kanban dataSource={data} keyField="Status" fields-content="Summary"
fields-primaryKey="Id" editSettings-editItems={editItems} editSettings-
allowEditing={true} editSettings-allowAdding={true}>
<columns>
<column headerText="Backlog" key="Open"></column>
<column headerText="In Progress" key="InProgress"></column>
<column headerText="Done" key="Close"></column>
</columns>
</EJ.Kanban>,
document.getElementById('kanbanboard-default')
);

```

The following output is displayed as a result of the above code example.

*Dialog Template Form*

You can edit any of the fields pertaining to a single card of data and apply it to a template so that the same format is applied to all the other cards that you may edit later.

Using this template support, you can edit the fields that are not bound to [editSettings-editItems](#).

To edit the cards using Dialog template form, set [editSettings-editMode](#) as `dialogtemplate` and specify the template id to [dialogTemplate](#) property of [editSettings](#).

- Note:**
1. `value` attribute is used to bind the corresponding field value while editing.
  2. `name` attribute is used to get the changed field values while save the edited card.
  3. For [editSettings-editMode](#) property you can assign either `string` value ("dialogtemplate").

The following code example describes the above behavior.

### HTML

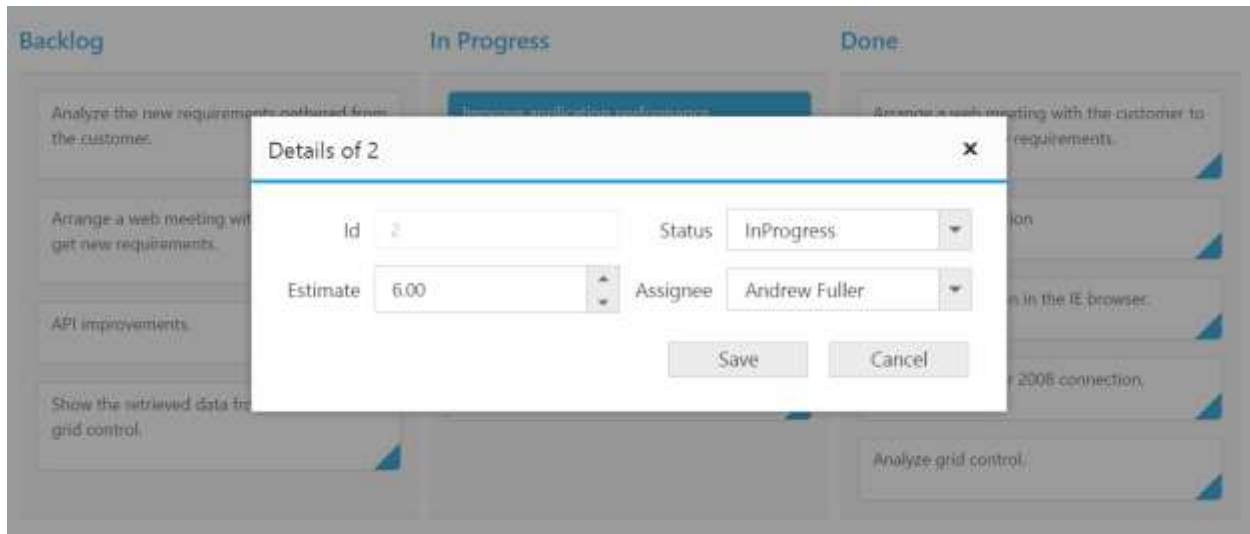
```
<div id="kanbanboard-default"></div>
<script src="app/kanbanboard/default.js"></script>
<script id="template" type="text/template">
<table cellpadding="10">
<tr>
<td style="text-align: right;">Id
</td>
<td style="text-align: left">
<input id="Id" name="Id" value="{: Id}" class="e-field e-einputtext valid
e-disable" style="text-align: right; width: 175px; height: 28px"
disabled="disabled" />
</td>
<td style="text-align: right;">Status
</td>
<td style="text-align: left">
<select id="Status" name="Status">
<option value="Close">Close</option>
<option value="InProgress">InProgress</option>
<option value="Open">Open</option>
</select>
</td>
</tr>
<tr>
<td style="text-align: right;">Estimate
</td>
<td style="text-align: left">
<input type="text" id="Estimate" name="Estimate" value="{: Estimate}" />
</td>
<td style="text-align: right;">Assignee
</td>
<td style="text-align: left">
<select id="Assignee" name="Assignee">
<option value="Nancy Davloio">Nancy Davloio</option>
<option value="Andrew Fuller">Andrew Fuller</option>
<option value="Janet Leverling">Janet Leverling</option>
<option value="Margaret hamilt">Margaret hamilt</option>
<option value="Steven walker">Steven walker</option>
<option value="Michael Suyama">Michael Suyama</option>
<option value="Robert King">Robert King</option>
<option value="Laura Callahan">Laura Callahan</option>
</select>
</td>
</tr>
</table>
</script>
```

While using template, you can change the elements that are defined in the `template`, to appropriate Syncfusion JS controls based on the column type. This can be achieved by using [actionComplete](#) event of Kanban. Please refer to following code snippets.

### HTML

```
var data =
ej.DataManager(window.kanbanData).executeLocal(ej.Query().take(20));
function complete(args) {
if ((args.requestType == "beginedit" || args.requestType == "add") &&
args.model.editSettings.editMode == "dialogtemplate") {
$("#Estimate").ejNumericTextbox({ value: parseFloat($("#Estimate").val()),
width: "175px", height: "34px", decimalPlaces: 2 });
$("#Assignee").ejDropDownList({ width: '175px' });
$("#Status").ejDropDownList({ width: '175px' });
$("#Priority").ejDropDownList({ width: '175px' });
if (args.requestType == "beginedit" || args.requestType == "add") {
$("#Assignee").ejDropDownList("setSelectedValue", args.data['Assignee']);
$("#Priority").ejDropDownList("setSelectedValue", args.data['Priority']);
$("#Status").ejDropDownList("setSelectedValue", args.data['Status']);
}
}
}
ReactDOM.render(
<EJ.Kanban dataSource={data} keyField="Status" fields-content="Summary"
fields-primaryKey="Id" editSettings-allowEditing={true} editSettings-
allowAdding={true} editSettings-editMode="dialogtemplate" editSettings-
dialogTemplate="#template" actionComplete={complete}>
<columns>
<column headerText="Backlog" key="Open"></column>
<column headerText="In Progress" key="InProgress"></column>
<column headerText="Done" key="Close"></column>
</columns>
</EJ.Kanban>,
document.getElementById('kanbanboard-default')
);
```

The following output is displayed as a result of the above code example.



### External Form

Set the [editSettings-editMode](#) as externalform to open the edit form in outside kanban content.

The following code example describes the above behavior.

### HTML

```
var data =
ej.DataManager(window.kanbanData).executeLocal(ej.Query().take(20));
var editItems = [
{ field: "Id", editType: ej.Kanban.EditingType.String, validationRules: {
required: true, number: true }},
{ field: "Status", editType: ej.Kanban.EditingType.Dropdown },
{ field: "Assignee", editType: ej.Kanban.EditingType.Dropdown },
{ field: "Estimate", editType: ej.Kanban.EditingType.Numeric, editParams: {
decimalPlaces: 2 }, validationRules: { range: [0, 1000] }},
{ field: "Summary", editType:
ej.Kanban.EditingType.TextArea, validationRules: { required: true }},
];
var scrollSetting={
width: 700, height: 500
};
ReactDOM.render(
<EJ.Kanban dataSource={data} keyField="Status" fields-content="Summary"
allowScrolling={true} fields-primaryKey="Id" allowTitle={true} editSettings-
allowEditing={true} editSettings-allowAdding={true} editSettings-
editItems={editItems} editSettings-editMode="externalform"
scrollSettings={scrollSetting}>
<columns>
<column headerText="Backlog" key="Open"></column>
<column headerText="In Progress" key="InProgress"></column>
<column headerText="Done" key="Close"></column>
</columns>
</EJ.Kanban>,
document.getElementById('kanbanboard-default')
);
```

The following output is displayed as a result of the above code example.



The screenshot displays a Kanban board interface. It features two columns: 'Backlog' on the left and 'In Progress' on the right. The 'Backlog' column contains three items: item 1 (highlighted in blue with the summary 'Analyze the new requirements gathered from the customer.'), item 3 ('Arrange a web meeting with the customer to get new requirements.'), and item 13 ('API improvements.'). The 'In Progress' column contains three items: item 2 ('Improve application performance'), item 4 ('Fix the issues reported in the IE browser.'), and item 14 ('Add responsive support to application'). Below the board, a 'Details of 1' modal is open, showing the details for item 1. The modal includes a form with the following fields: 'Id' (value: 1), 'Status' (dropdown menu showing 'Open'), and 'Summary' (text area with the value 'Analyze the new requirements gathered from the customer.'). At the bottom right of the modal are 'Save' and 'Cancel' buttons.

Form Position:

Form Position can be customized by setting the [formPosition](#) property of [editSettings](#) as "right" or "bottom".

The following code example describes the above behavior.

### HTML

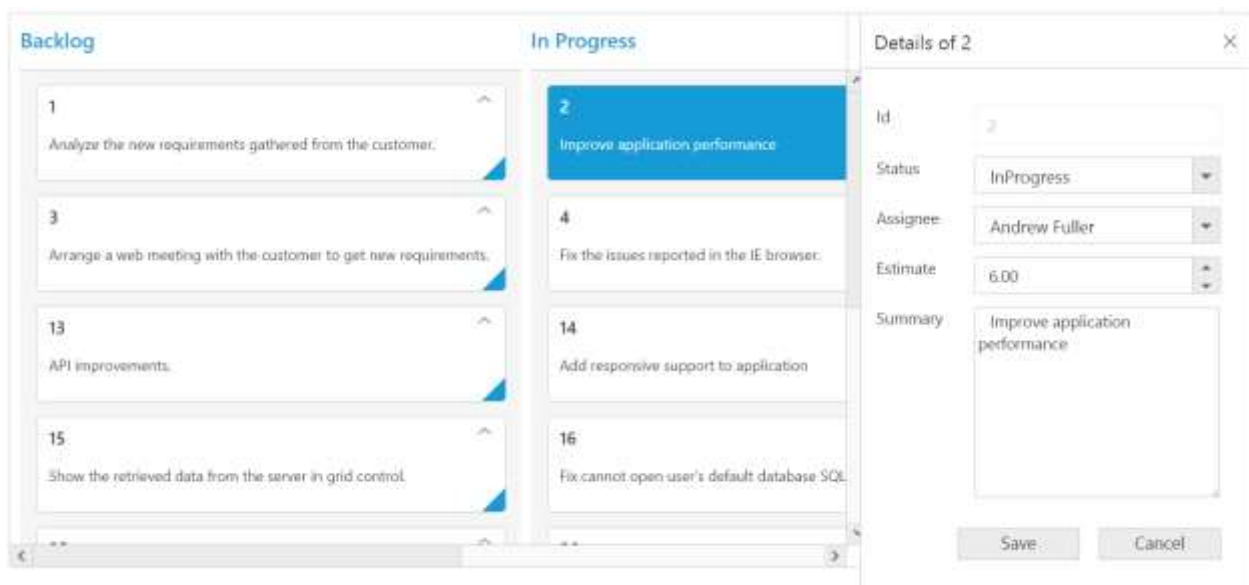
```
var data =
ej.DataManager(window.kanbanData).executeLocal(ej.Query().take(20));
var editItems = [
{ field: "Id", editType: ej.Kanban.EditingType.String, validationRules: {
required: true, number: true }},
{ field: "Status", editType: ej.Kanban.EditingType.Dropdown },
{ field: "Assignee", editType: ej.Kanban.EditingType.Dropdown },
{ field: "Estimate", editType: ej.Kanban.EditingType.Numeric, editParams: {
decimalPlaces: 2 }, validationRules: { range: [0, 1000] }},
{ field: "Summary", editType:
ej.Kanban.EditingType.TextArea, validationRules: { required: true }},
];
var scrollSetting={
width: 700, height: 250
};
ReactDOM.render(
```

```

<EJ.Kanban dataSource={data} keyField="Status" fields-content="Summary"
allowScrolling={true} fields-primaryKey="Id" allowTitle={true} editSettings-
allowEditing={true} editSettings-allowAdding={true} editSettings-
editItems={editItems} editSettings-editMode="externalform"
scrollSettings={scrollSetting} editSettings-formPosition="right">
  <columns>
    <column headerText="Backlog" key="Open"></column>
    <column headerText="In Progress" key="InProgress"></column>
    <column headerText="Done" key="Close"></column>
  </columns>
</EJ.Kanban>,
document.getElementById('kanbanboard-default')
);

```

The following output is displayed as a result of the above code example.



### External Template Form

You can edit any of the fields pertaining to a single card of data and apply it to a template so that the same format is applied to all the other cards that you may edit later.

Using this template support, you can edit the fields that are not bound to Kanban Edit Items.

To edit the cards using External template form, set [editMode](#) as `externalformtemplate` and specify the template id to [externalFormTemplate](#) property of [editSettings](#).

While using template, you can change the elements that are defined in the template, to appropriate Syncfusion JS controls based on the column type. This can be achieved by using [actionComplete](#) event of Kanban.

- Note:**
1. `value` attribute is used to bind the corresponding field value while editing.
  2. `name` attribute is used to get the changed field values while save the edited card.
  3. For [editMode](#) property you can assign either `string` value ("externalformtemplate").

The following code example describes the above behavior.

**HTML**

```

var data =
ej.DataManager(window.kanbanData).executeLocal(ej.Query().take(20));
var scrollSetting={
width: 700, height: 450
};
function complete(args) {
if ((args.requestType == "beginedit" || args.requestType == "add") &&
args.model.editSettings.editMode == "externalformtemplate") {
$("#Assignee").ejDropDownList({ width: '175px' });
$("#Status").ejDropDownList({ width: '175px' });
if(args.requestType == "beginedit" || args.requestType == "add" ){
$("#Assignee").ejDropDownList("setSelectedValue", args.data['Assignee']);
$("#Status").ejDropDownList("setSelectedValue", args.data['Status']);
}
}
};
ReactDOM.render(
<EJ.Kanban dataSource={data} keyField="Status" fields-content="Summary"
allowScrolling={true} fields-primaryKey="Id" allowTitle={true} editSettings-
allowEditing={true} editSettings-allowAdding={true} editSettings-
editMode="externalformtemplate" editSettings-
externalFormTemplate="#template" scrollSettings={scrollSetting}
actionComplete={complete}>
<columns>
<column headerText="Backlog" key="Open"></column>
<column headerText="In Progress" key="InProgress"></column>
<column headerText="Done" key="Close"></column>
</columns>
</EJ.Kanban>,
document.getElementById('kanbanboard-default')
);

```

**HTML**

```

<script id="template" type="text/template">
<table cellpadding="10">
<tr>
<td style="text-align:left;">Id
</td>
<td style="text-align: left">
<input id="Id" name="Id" value="{:{ Id} }" class="e-field e-ejinputtext valid
e-disable" style="text-align: right; width: 175px; height: 28px"
disabled="disabled" />
</td>
</tr>
<tr>
<td style="text-align: left;">Status
</td>
<td style="text-align: left">
<select id="Status" name="Status">
<option value="Close">Close</option>
<option value="InProgress">InProgress</option>
<option value="Open">Open</option>
<option value="Testing">Testing</option>

```

```

<option value="Validate">Validate</option>
</select>
</td>
</tr>
<tr>
<td style="text-align: left;">Assignee
</td>
<td style="text-align: left">
<select id="Assignee" name="Assignee">
<option value="Nancy Davloio">Nancy Davloio</option>
<option value="Andrew Fuller">Andrew Fuller</option>
<option value="Janet Leverling">Janet Leverling</option>
<option value="Margaret hamilt">Margaret hamilt</option>
<option value="Steven walker">Steven walker</option>
<option value="Michael Suyama">Michael Suyama</option>
<option value="Robert King">Robert King</option>
<option value="Laura Callahan">Laura Callahan</option>
</select>
</td>
</tr>
<tr>
<td style="text-align: left;">Priority
</td>
<td style="text-align: left">
<input id="Priority" name="Priority" value="{{: Priority}}" class="e-field
e-ejinputtext valid" style="width: 175px; height: 28px" />
</td>
</tr>
<tr>
<td style="text-align: left;">Summary
</td>
<td style="text-align: left">
<textarea id="Summary" name="Summary" class="e-ejinputtext" value="{{:
Summary}}" style="width: 270px; height: 95px">{{: Summary}}</textarea>
</td>
</tr>
</table>
</script>

```

The following output is displayed as a result of the above code example.

The screenshot displays a Kanban Board interface with two columns: 'Backlog' and 'In Progress'. The 'Backlog' column contains three items: item 1 (blue header, 'Analyze the new requirements gathered from the customer.'), item 3 ('Arrange a web meeting with the customer to get new requirements.'), and item 13 ('API improvements.'). The 'In Progress' column contains three items: item 2 ('Improve application performance'), item 4 ('Fix the issues reported in the IE browser.'), and item 14 ('Add responsive support to application'). Below the columns, a 'Details of 1' dialog is open, showing the following fields: 'Id' (1), 'Status' (Open), and 'Assignee' (Nancy Davloio). The dialog has 'Save' and 'Cancel' buttons at the bottom right.

### Cell edit type and its params

The edit type of bound column can be customized using [editType](#) property of [editItems](#). The following Essential JavaScript controls are supported built-in by [editType](#). And also you can define the model for all the edit types controls while editing through [editParams](#) property of [editItems](#).

The following table describes [editType](#) and their corresponding [editParams](#) of the specific data type of the column.

EditType	EditParams	Description	Example
Numeric	<a href="#">ejTextBoxes</a>	control for integers, double, and decimal data™s	editParams: { decimalPlaces: 2 }
String	HTML Textbox	HTML Textbox	-
DatePicker	<a href="#">ejDatePicker</a>	control for date data	editParams: { buttonText : "Now" }
DateTimePicker	<a href="#">ejDateTimePicker</a>	control for date data-time data	editParams: { enabled: true }
DropDown	<a href="#">ejDropDownList</a>	control for list of data	editParams: { allowGrouping: true }

RTE	<a href="#">ejRTE</a>	control for customizing text in RTE format	editParams: { allowResize: true }
TextArea	HTML TextArea	Control for multi-line plain-text editing	editParams:{height:100,width:200}

**Note:** 1. If [editType](#) is not set, then by default it will display HTML textbox while editing a card.

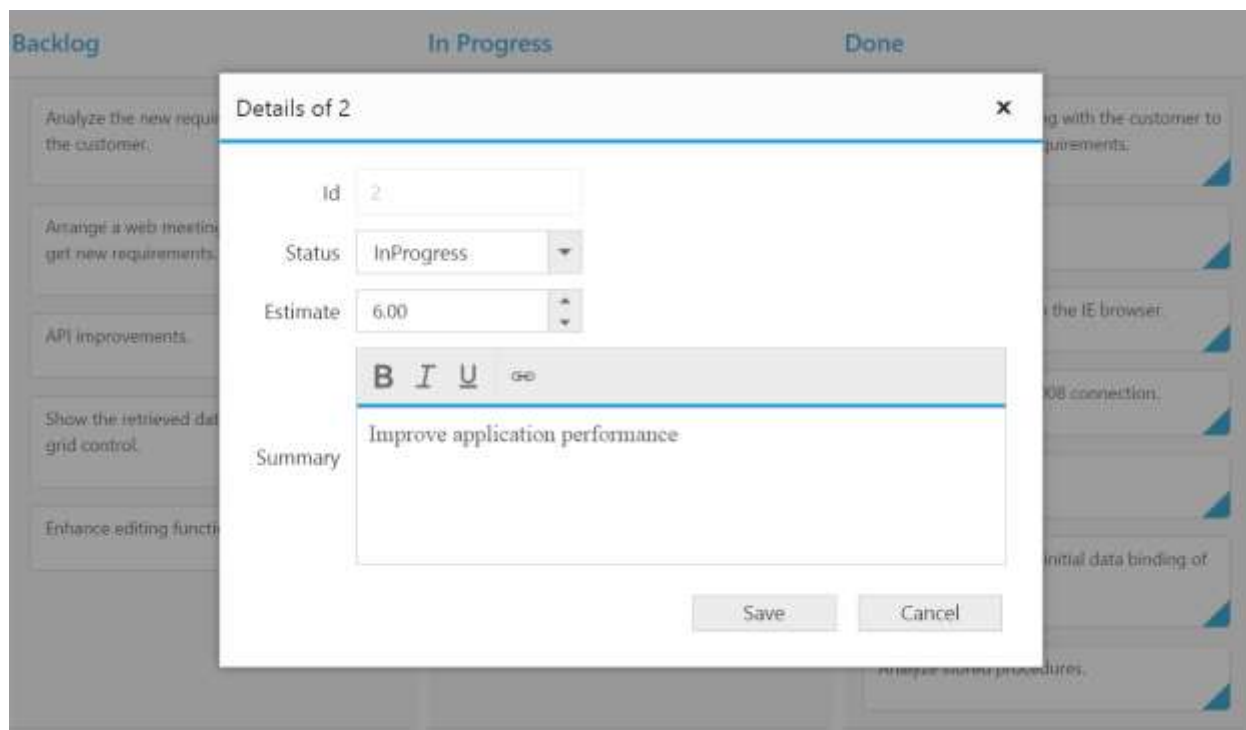
2. For [editType](#) property you can assign either string value ("numericedit").

The following code example describes the above behavior.

### HTML

```
var data =
ej.DataManager(window.kanbanData).executeLocal(ej.Query().take(20));
var editItems = [
{ field: "Id" },
{ field: "Status", editType: ej.Kanban.EditingType.Dropdown },
{ field: "Estimate", editType: ej.Kanban.EditingType.Numeric, editParams: {
decimalPlaces: 2 } },
{ field: "Summary", editType: ej.Kanban.EditingType.RTE, editParams: {
height:150,minHeight: 100 } }
];
ReactDOM.render(
<EJ.Kanban dataSource={data} keyField="Status" fields-content="Summary"
fields-primaryKey="Id" editSettings-editItems={editItems} editSettings-
allowEditing={true} editSettings-allowAdding={true}>
<columns>
<column headerText="Backlog" key="Open"></column>
<column headerText="In Progress" key="InProgress"></column>
<column headerText="Done" key="Close"></column>
</columns>
</EJ.Kanban>,
document.getElementById('kanbanboard-default')
);
```

The following output is displayed as a result of the above code example.



### Column Validation

We can validate the value of the added or edited card cell before saving.

The below validation script files are needed when editing is enabled with validation.

1. jquery.validate.min.js 2. jquery.validate.unobtrusive.min.js

**Note:** If you enabled the unobtrusive option, then need to refer the jquery.validate.unobtrusive.min.js file in your application along with the other script.

### jQuery Validation

You can set validation rules using [validationRules](#) property of [columns](#). The following are jQuery validation methods.

#### List of jQuery validation methods

Rules	Description
required	Requires an element.
remote	Requests a resource to check the element for validity.
minlength	Requires the element to be of given minimum length.
maxlength	Requires the element to be of given maximum length.
rangelength	Requires the element to be in given value range.
min	The element requires a given minimum.
max	The element requires a given maximum.
range	Requires the element to be in a given value range.

email	The element requires a valid email.
url	The element requires a valid url.
date	Requires the element to be a date.
dateISO	The element requires an ISO date.
number	The element requires a decimal number.
digits	The element requires digits only.
creditcard	Requires the element to be a credit card number.
equalTo	Requires the element to be the same as another.

Kanban supports all the standard validation methods of jQuery, please refer the jQuery validation documentation [link](#) for more information.

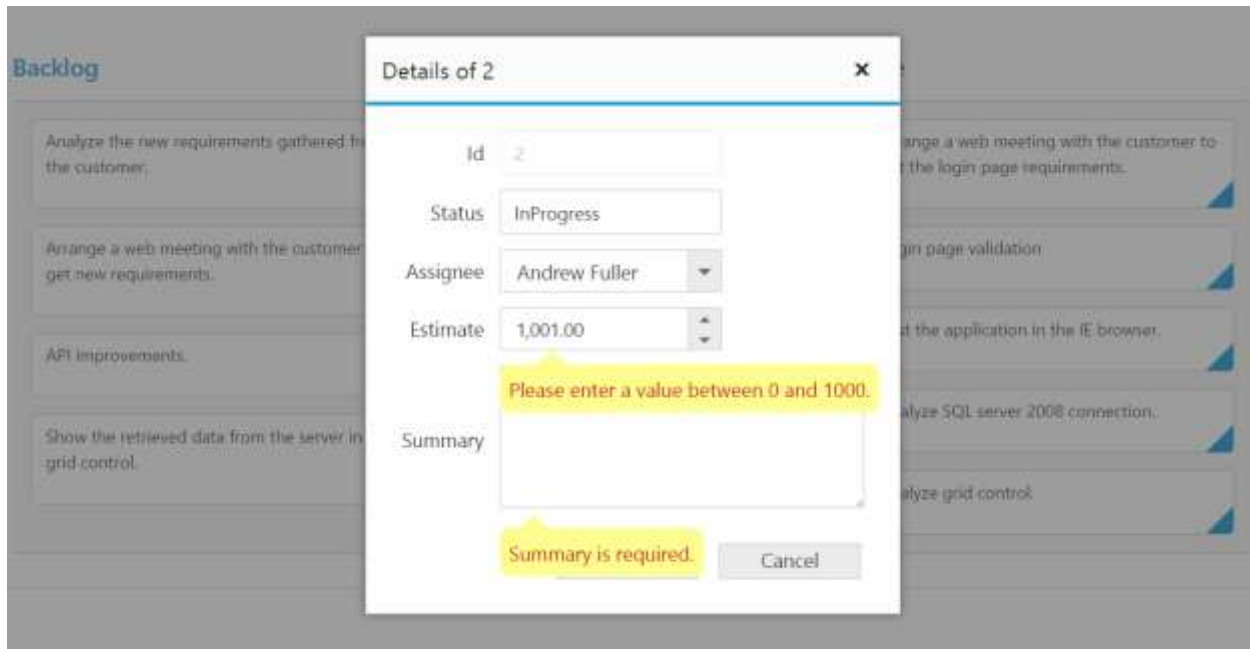
The following code example describes the above behavior.

#### HTML

```
var data =
ej.DataManager(window.kanbanData).executeLocal(ej.Query().take(20));
var editItems = [
{ field: "Id" },
{ field: "Status", editType: ej.Kanban.EditingType.String },
{ field: "Assignee", editType: ej.Kanban.EditingType.Dropdown },
{ field: "Estimate", editType: ej.Kanban.EditingType.Numeric, editParams: {
decimalPlaces: 2 }, validationRules: { range: [0, 1000] } },
{ field: "Summary", editType: ej.Kanban.EditingType.TextArea,
validationRules: { required: true } }
];
ReactDOM.render(
<EJ.Kanban dataSource={data} keyField="Status" fields-content="Summary"
fields-primaryKey="Id" editSettings-editItems={editItems} editSettings-
allowEditing={true} editSettings-allowAdding={true}>
<columns>
<column headerText="Backlog" key="Open"></column>
<column headerText="In Progress" key="InProgress"></column>
<column headerText="Done" key="Close"></column>
</columns>
</EJ.Kanban>,
document.getElementById('kanbanboard-default')
);
```

The following output is displayed as a result of the above code example.





## Filtering

Filtering allows to filter the collection of cards from `dataSource` which meets the predefined `query` in the quick filters collection. To enable filtering, define `filterSettings` collection with display `text` and `ej.Query`.

You can also define display tip to describe filter definition to user using property `description`. If the `description` property is not defined, given `text` will act as display tip.

We can also customize filter option through external button or `customToolbarItems` by using `filterCards()` method.

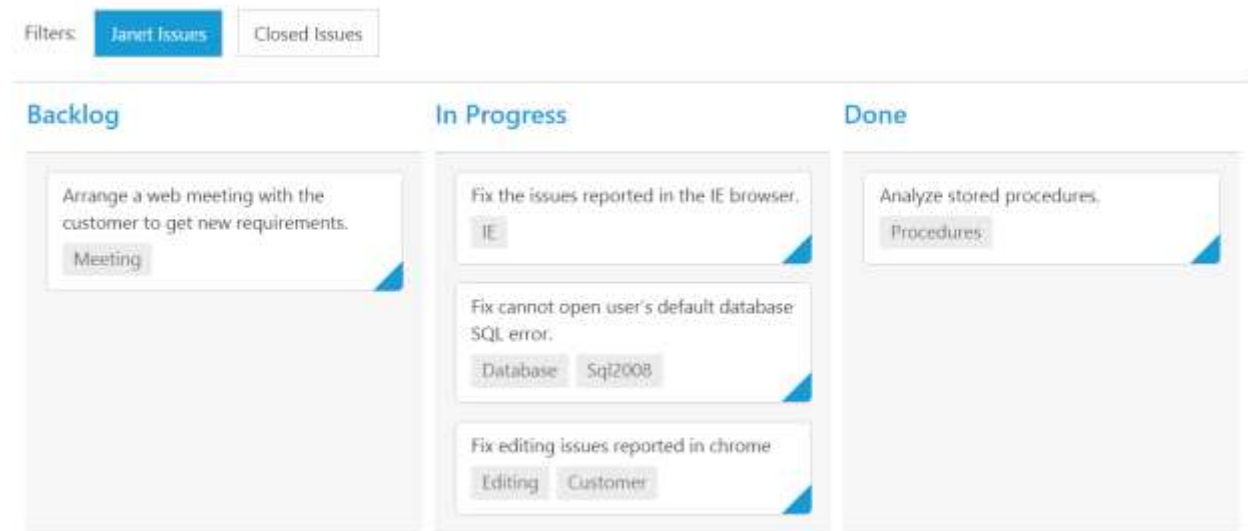
The following code example describes the above behavior.

## HTML

```
var data =
ej.DataManager(window.kanbanData).executeLocal(ej.Query().take(20));
var filter = [
{ text: "Janet Issues", query: new ej.Query().where("Assignee", "equal",
"Janet Leverling"), description: "Displays issues which matches the assignee
as 'Janet Leverling' },
{ text: "Andrew Issues", query: new ej.Query().where("Assignee", "equal",
"Andrew Fuller"), description: "Displays issues which matches the assignee
as 'Andrew Fuller' }
];
ReactDOM.render(
<EJ.Kanban dataSource={data} keyField="Status" fields-content="Summary"
fields-primaryKey="Id" fields-tag="Tags" filterSettings={filter}>
<columns>
<column headerText="Backlog" key="Open"></column>
<column headerText="In Progress" key="InProgress"></column>
<column headerText="Done" key="Close"></column>
</columns>
</EJ.Kanban>,
document.getElementById('kanbanboard-default')
```

```
);
```

The following output is displayed as a result of the above code example.



### Scrolling

Scrolling can be enabled by setting [allowScrolling](#) as true. The height and width can be set to Kanban by using the properties [scrollSettings-height](#) and [scrollSettings-width](#) respectively.

**Note:** The height and width can be set in percentage and pixel. The default value for [height](#) and [width](#) in [scrollSettings](#) is 0 and auto respectively.

Set width and height in pixel

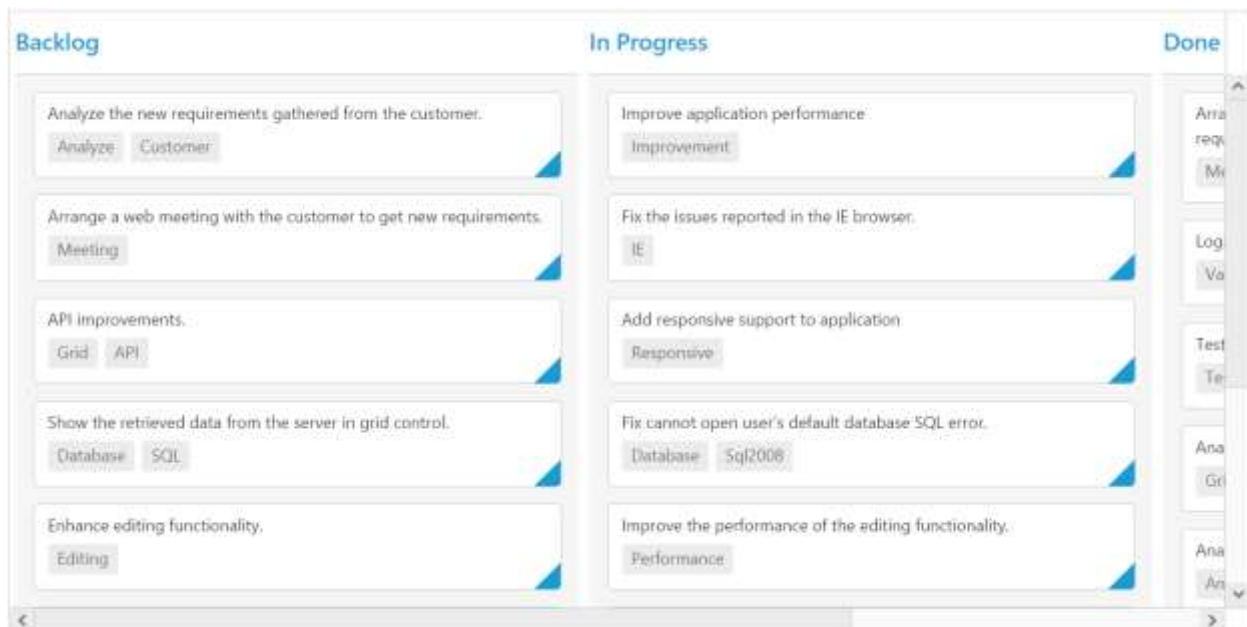
To specify the [scrollSettings-width](#) and [scrollSettings-height](#) in pixel, by set the pixel value as integer.

The following code example describes the above behavior.

### HTML

```
var data =
ej.DataManager(window.kanbanData).executeLocal(ej.Query().take(20));
var scrollSetting={
width: 900,
height: 450
};
ReactDOM.render(
<EJ.Kanban dataSource={data} keyField="Status" fields-content="Summary"
fields-primaryKey="Id" fields-tag="Tags" allowScrolling={true}
scrollSettings={scrollSetting}>
<columns>
<column headerText="Backlog" key="Open"></column>
<column headerText="In Progress" key="InProgress"></column>
<column headerText="Done" key="Close"></column>
</columns>
</EJ.Kanban>,
document.getElementById('kanbanboard-default')
);
```

The following output is displayed as a result of the above code example.



Set height and width in percentage

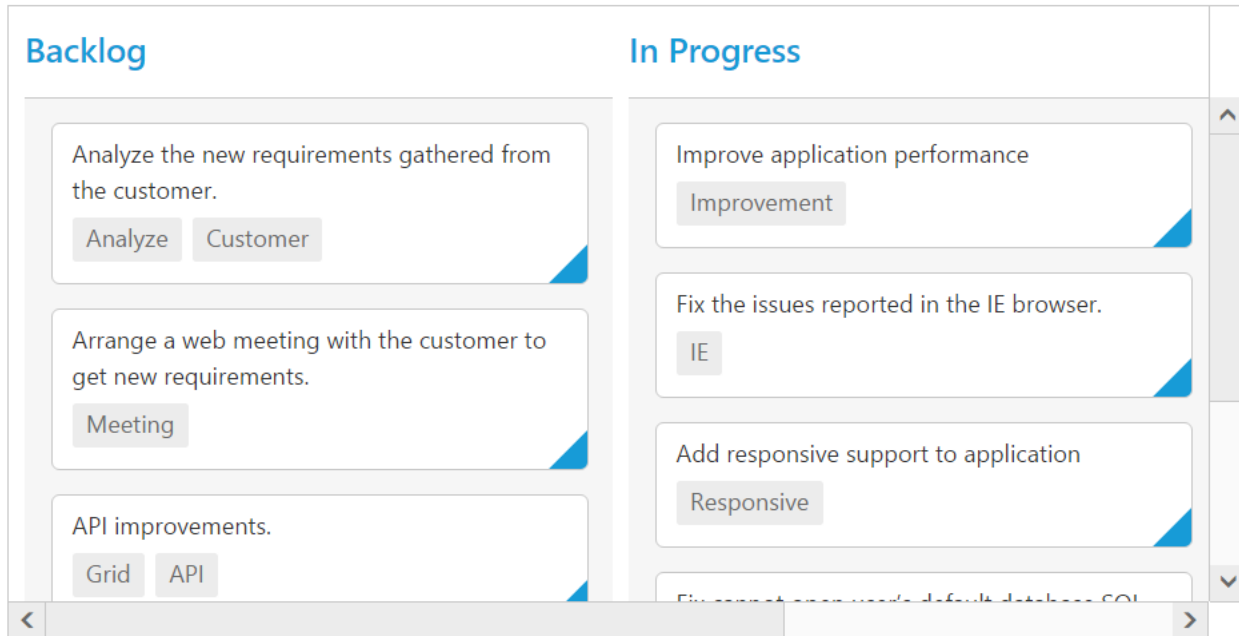
To specify the [scrollSettings-width](#) and [scrollSettings-height](#) in percentage, by set the percentage value as string.

The following code example describes the above behavior.

### HTML

```
var data =
ej.DataManager(window.kanbanData).executeLocal(ej.Query().take(20));
var scrollSetting={
width: "70%", height: "70%"
};
ReactDOM.render(
<EJ.Kanban dataSource={data} keyField="Status" fields-content="Summary"
fields-primaryKey="Id" fields-tag="Tags" allowScrolling={true}
scrollSettings={scrollSetting}>
<columns>
<column headerText="Backlog" key="Open"></column>
<column headerText="In Progress" key="InProgress"></column>
<column headerText="Done" key="Close"></column>
</columns>
</EJ.Kanban>,
document.getElementById('kanbanboard-default')
);
```

The following output is displayed as a result of the above code example.



Set width as auto

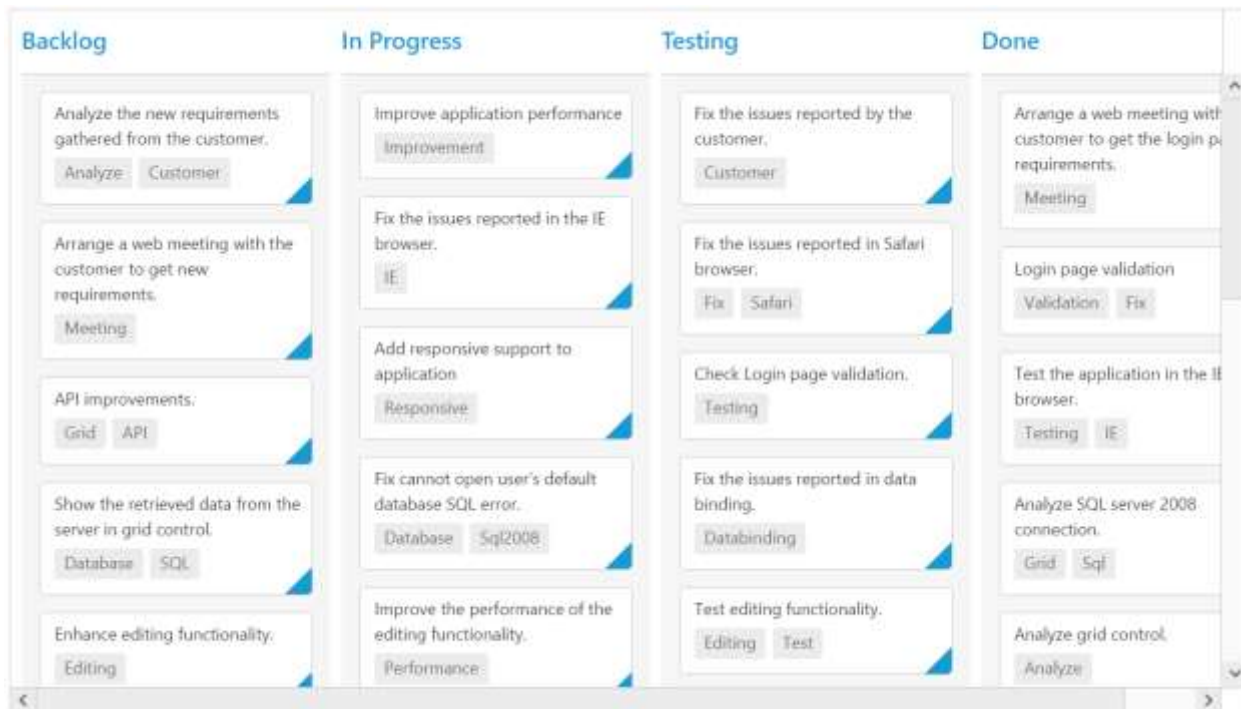
Specify [width](#) property of [scrollSettings](#) as auto, then the scrollbar is rendered only when the Kanban width exceeds the browser window width.

The following code example describes the above behavior.

#### HTML

```
var data =
ej.DataManager(window.kanbanData).executeLocal(ej.Query().take(20));
var scrollSetting={
width: "auto", height: 300
};
ReactDOM.render(
<EJ.Kanban dataSource={data} keyField="Status" fields-content="Summary"
fields-primaryKey="Id" fields-tag="Tags" allowScrolling={true}
scrollSettings={scrollSetting}>
<columns>
<column headerText="Backlog" key="Open"></column>
<column headerText="In Progress" key="InProgress"></column>
<column headerText="Done" key="Close"></column>
</columns>
</EJ.Kanban>,
document.getElementById('kanbanboard-default')
);
```

The following output is displayed as a result of the above code example.



### Enabling freeze swim lane

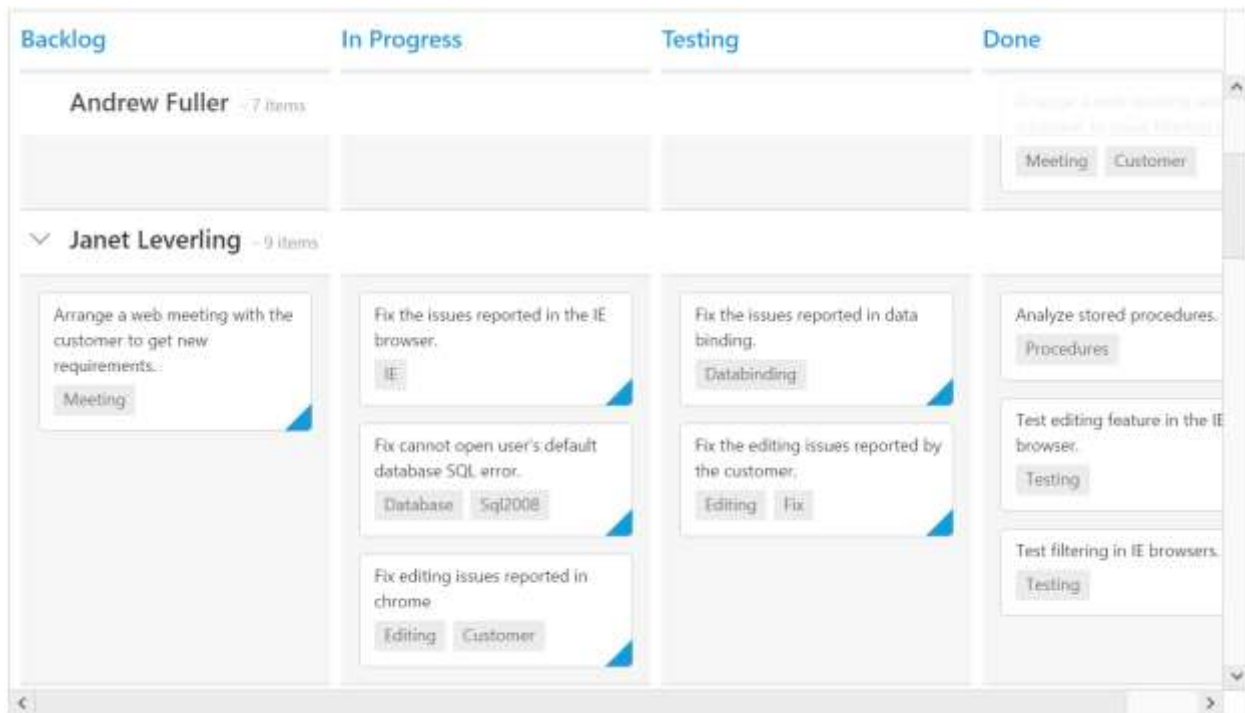
Set [scrollSettings-allowFreezeSwimlane](#) as true. This enables scrolling with freezing of swim lane until you scroll to the next Swim lane, which helps user to aware of current swim lane target.

The following code example describes the above behavior.

### HTML

```
var data =
ej.DataManager(window.kanbanData).executeLocal(ej.Query().take(20));
var scrollSetting= { width: "auto", height: 500,allowFreezeSwimlane: true };
ReactDOM.render(
<EJ.Kanban dataSource={data} keyField="Status" fields-content="Summary"
fields-primaryKey="Id" fields-swimlaneKey="Assignee" fields-tag="Tags"
allowScrolling={true} scrollSettings={scrollSetting}>
<columns>
<column headerText="Backlog" key="Open"></column>
<column headerText="In Progress" key="InProgress"></column>
<column headerText="Testing" key="Testing"></column>
<column headerText="Done" key="Close"></column>
</columns>
</EJ.Kanban>,
document.getElementById('kanbanboard-default')
);
```

The following output is displayed as a result of the above code example.



**Note:** `allowFreezeSwimlane` is applicable when swim lane grouping enabled by setting `swimlaneKey`.

### Selection and Hovering

Selection provides an interactive support to highlight the card that you select. Selection can be done through simple Mouse down or Keyboard interaction. To enable selection, set `allowSelection` as true.

You can see the mouse hovering effect on the corresponding cards using `allowHover` property. By default selection and hovering is `true`.

### Types of Selection

Two types of selections available in Kanban are,

1. Single
2. Multiple

#### Single Selection

To enable single selection by setting `selectionType` property as single.

#### Multiple Selection

Multiple selections is an interactive support to select a group of cards in Kanban by mouse or keyboard interactions. To enable multiple selections by set `selectionType` property as `multiple`.

You can select multiple random cards below key press.

Keys	Description
Ctrl + mouse left	To select multiple random cards.
Shift + mouse left	To continuous cards select.

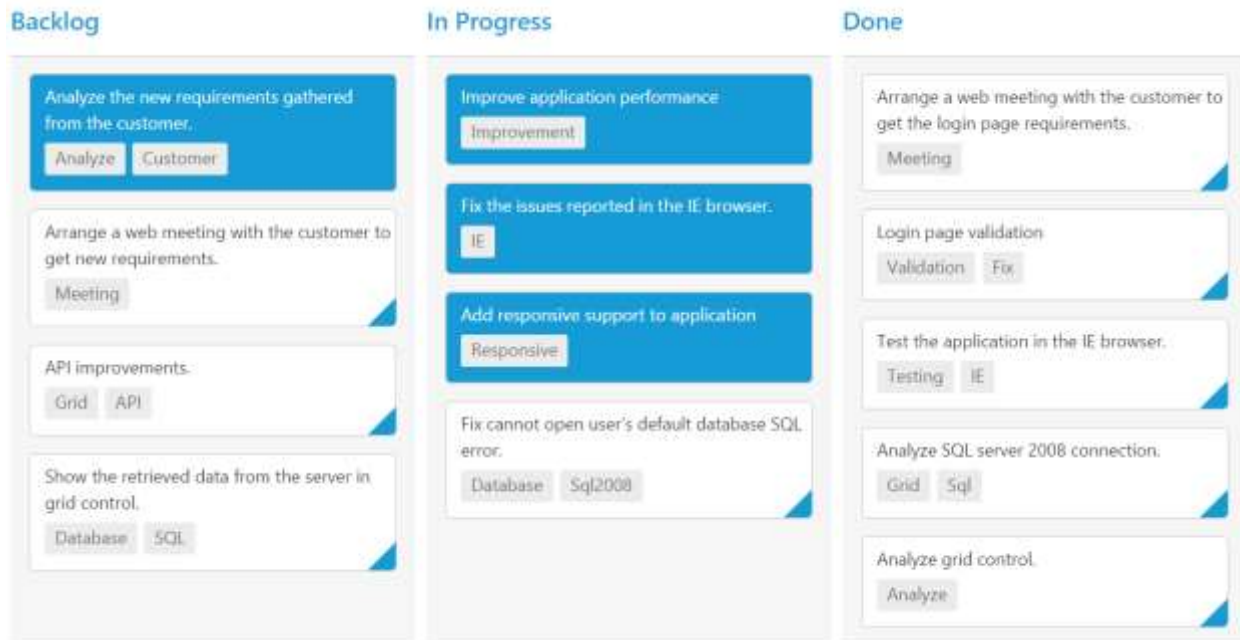
To unselect selected cards, by press “Shift + mouse left” click on selected row.

The following code example describes the above behavior.

### HTML

```
ReactDOM.render(
  <EJ.Kanban dataSource={data} keyField="Status" fields-content="Summary"
  fields-primaryKey="Id" fields-tag="Tags" selectionType="multiple">
    <columns>
      <column headerText="Backlog" key="Open"></column>
      <column headerText="In Progress" key="InProgress"></column>
      <column headerText="Done" key="Close"></column>
    </columns>
  </EJ.Kanban>,
  document.getElementById('kanbanboard-default')
);
```

The following output is displayed as a result of the above code example.

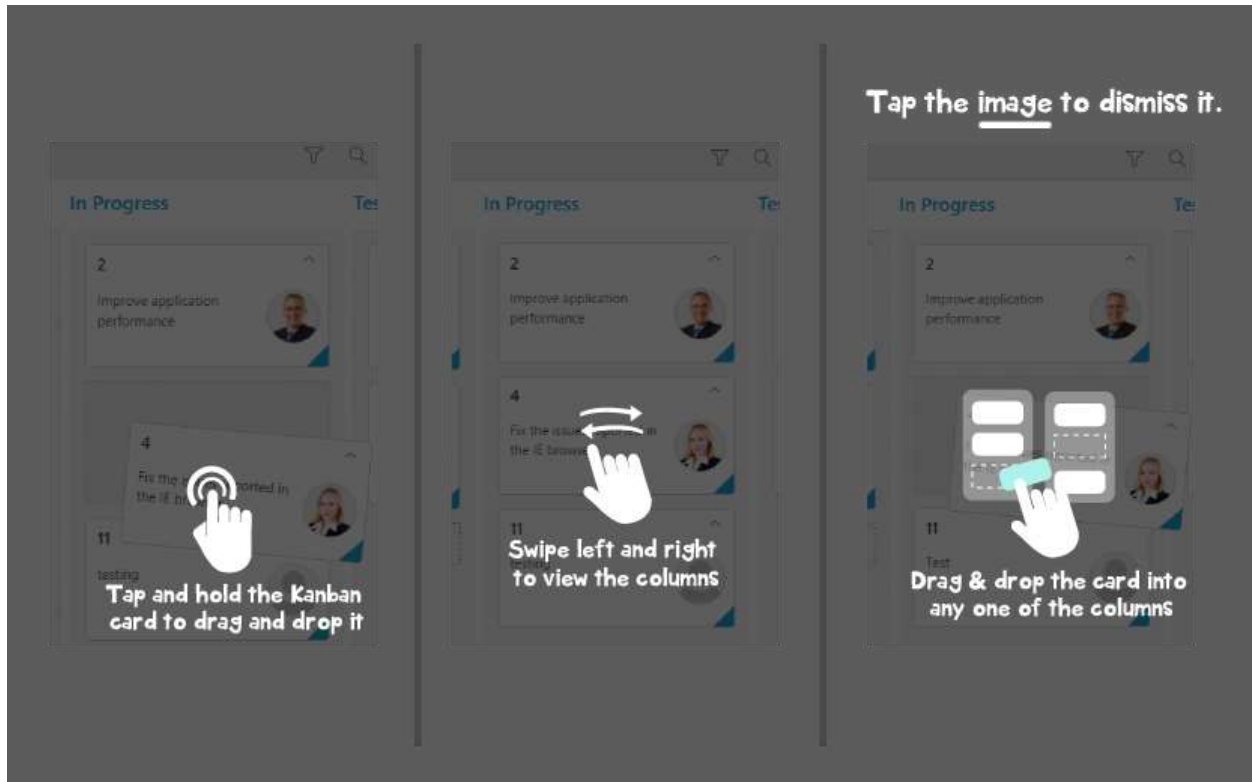


### Responsive

The Kanban control has support for responsive behavior based on client browser's width and height. To enable responsive, [isResponsive](#) property should be true. There are two modes of responsive layout is available in Kanban based on client width. They are.

- Mobile(<480px)
- Desktop(>480px)

You can check the image representation of touch actions from the below image.



### Mobile Layout

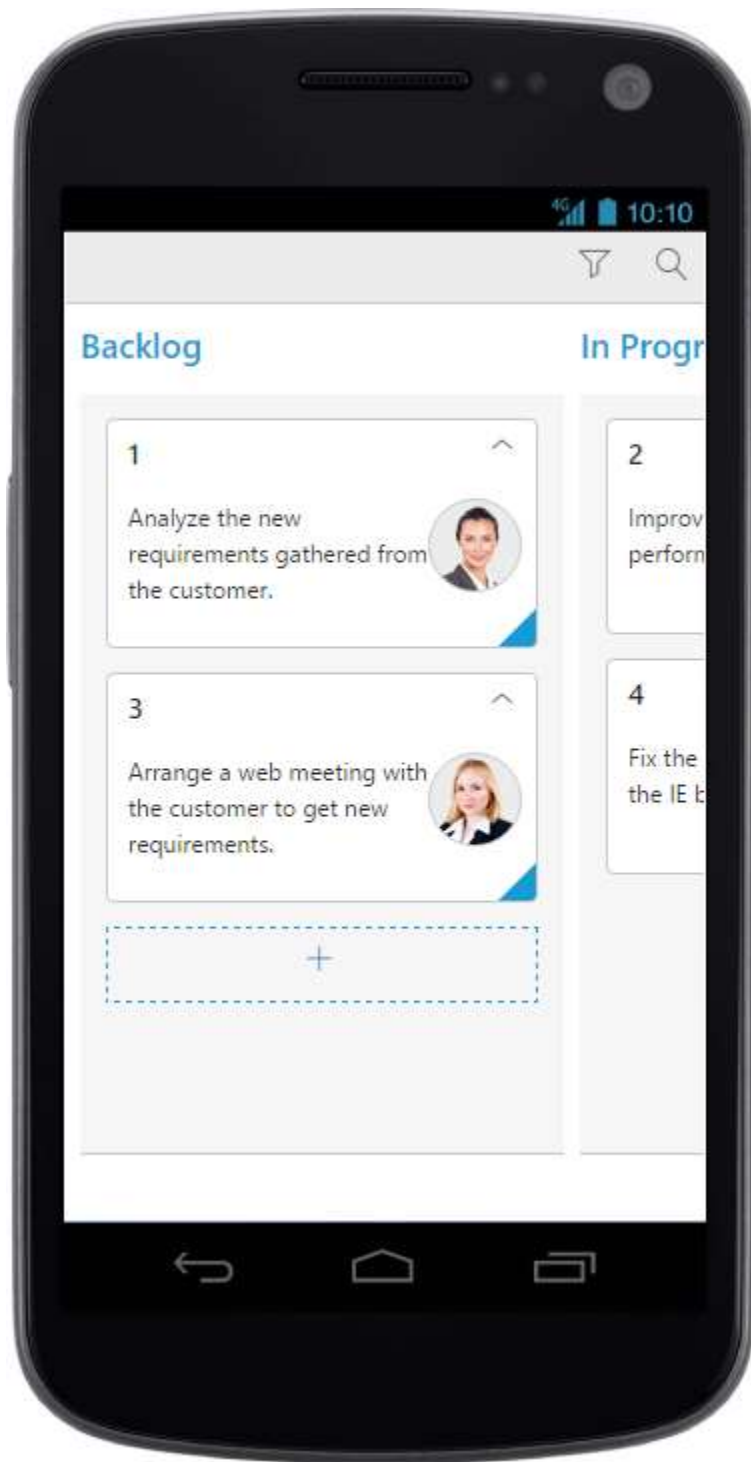
If client width is less than 480px, the Kanban will render in mobile mode. In which, you can see that kanban user interface is customized and redesigned for best view in small screens. To enable responsive, [isResponsive](#) property should be true.

### HTML

```
var data =
ej.DataManager(window.kanbanData).executeLocal(ej.Query().take(20));
var editItems = [
{ field: "Id", editType: ej.Kanban.EditingType.String, validationRules: {
required: true, number: true } },
{ field: "Status", editType: ej.Kanban.EditingType.Dropdown },
{ field: "Assignee", editType: ej.Kanban.EditingType.Dropdown },
{ field: "Estimate", editType: ej.Kanban.EditingType.Numeric, editParams: {
decimalPlaces: 2 }, validationRules: { range: [0, 1000] } },
{ field: "Summary", editType: ej.Kanban.EditingType.TextArea,
validationRules: { required: true } }
];
var filter = [
{ text: "Janet Issues", query: new ej.Query().where("Assignee", "equal",
"Janet Leverling"), description: "Displays issues which matches the assignee
as 'Janet Leverling'" },
{ text: "Andrew Issues", query: new ej.Query().where("Assignee", "equal",
"Andrew Fuller"), description: "Displays issues which matches the assignee
as 'Andrew Fuller'" }
];
ReactDOM.render(
<EJ.Kanban dataSource={data} keyField="Status" isResponsive={true} fields-
content="Summary" fields-primaryKey="Id" fields-imageUrl="ImageUrl"
```



```
editSettings-editItems={editItems} editSettings-allowEditing={true}
editSettings-allowAdding={true} allowSearching={true}
filterSettings={filter}>
<columns>
<column headerText="Backlog" key="Open"></column>
<column headerText="In Progress" key="InProgress"></column>
<column headerText="Testing" key="Testing"></column>
<column headerText="Done" key="Close"></column>
</columns>
</EJ.Kanban>,
document.getElementById('kanbanboard-default')
);
```



**Warning:** IE8 and IE9 does not support responsive kanban. `ej.responsive.css` should be referred to display Responsive Kanban.

Details of 1

Id

1

Status

Open

Assignee

Nancy Davloio

Estimate

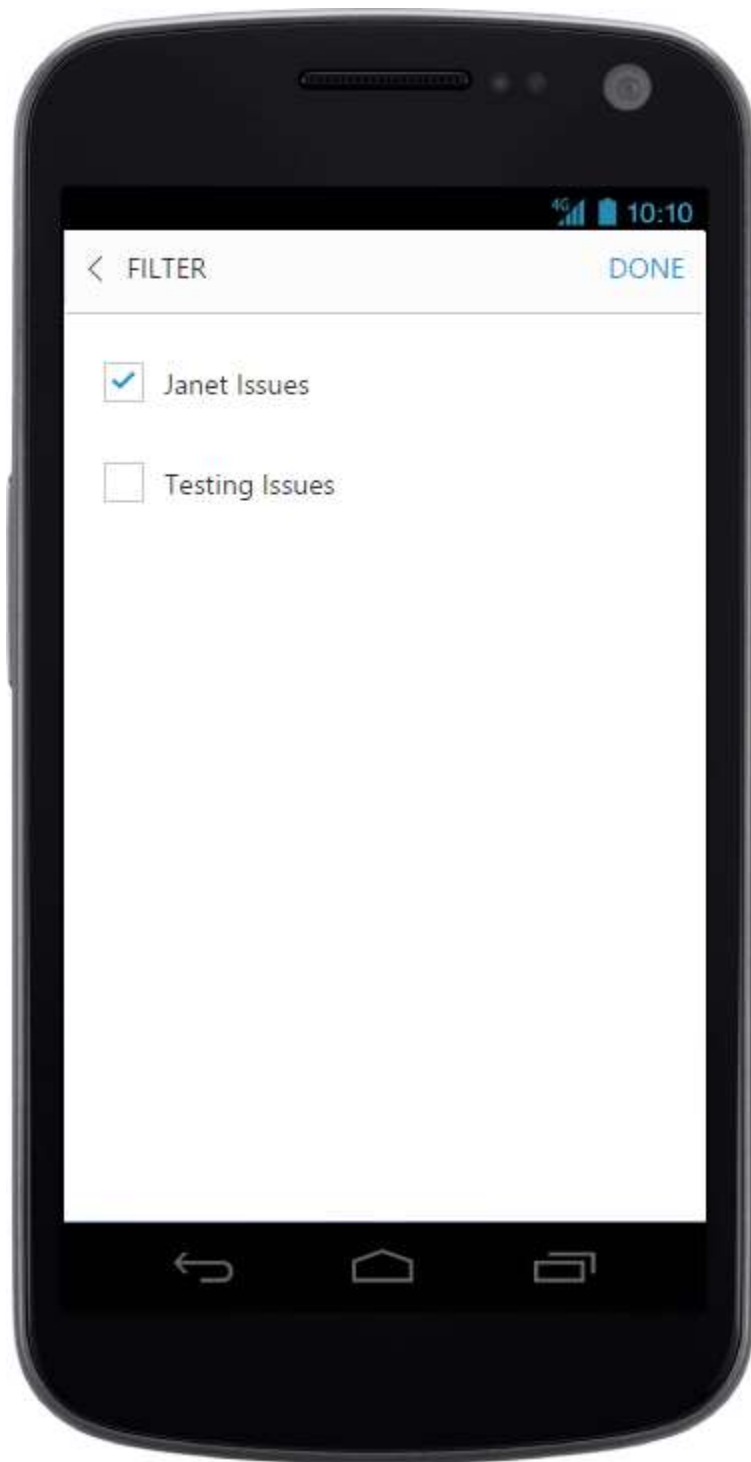
3.50

Summary

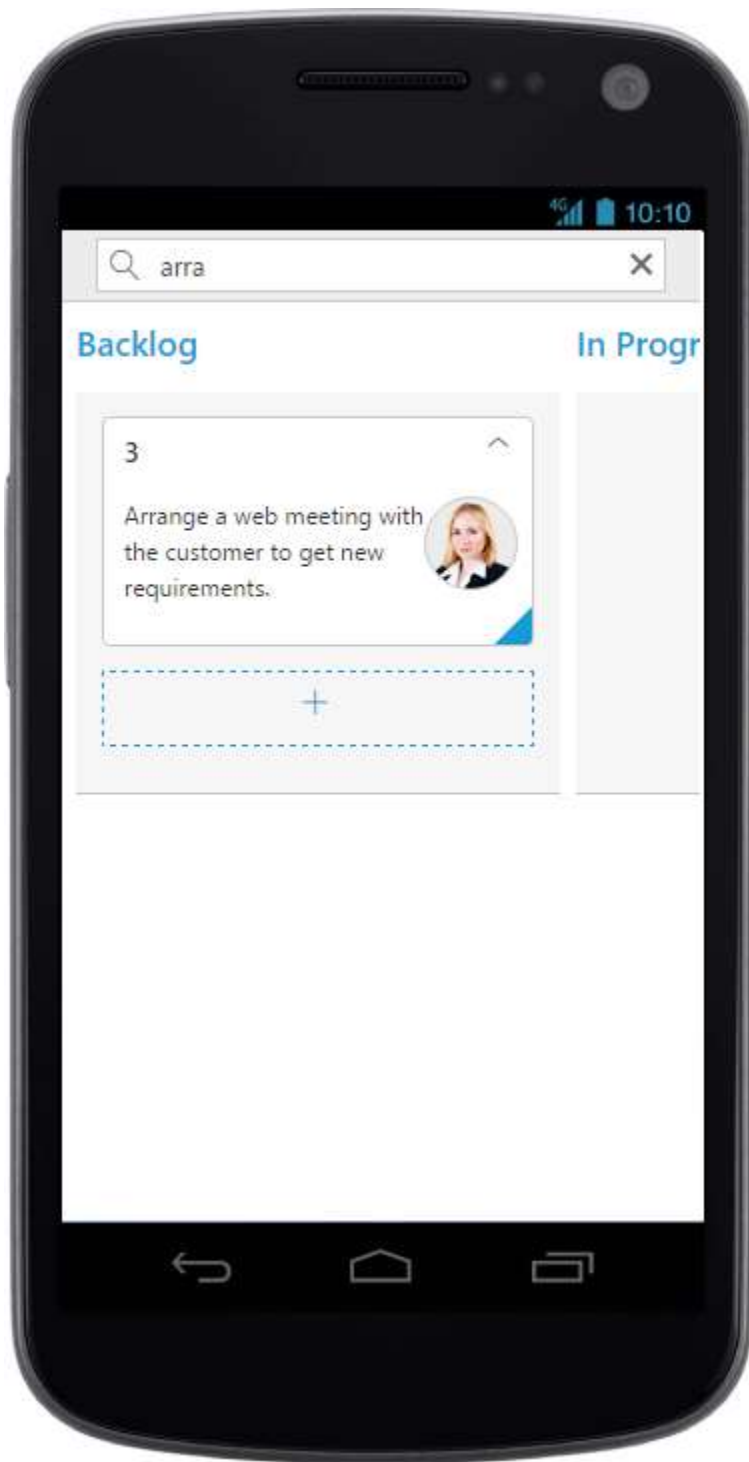
Analyze the new requirements gathered from the customer.

Save

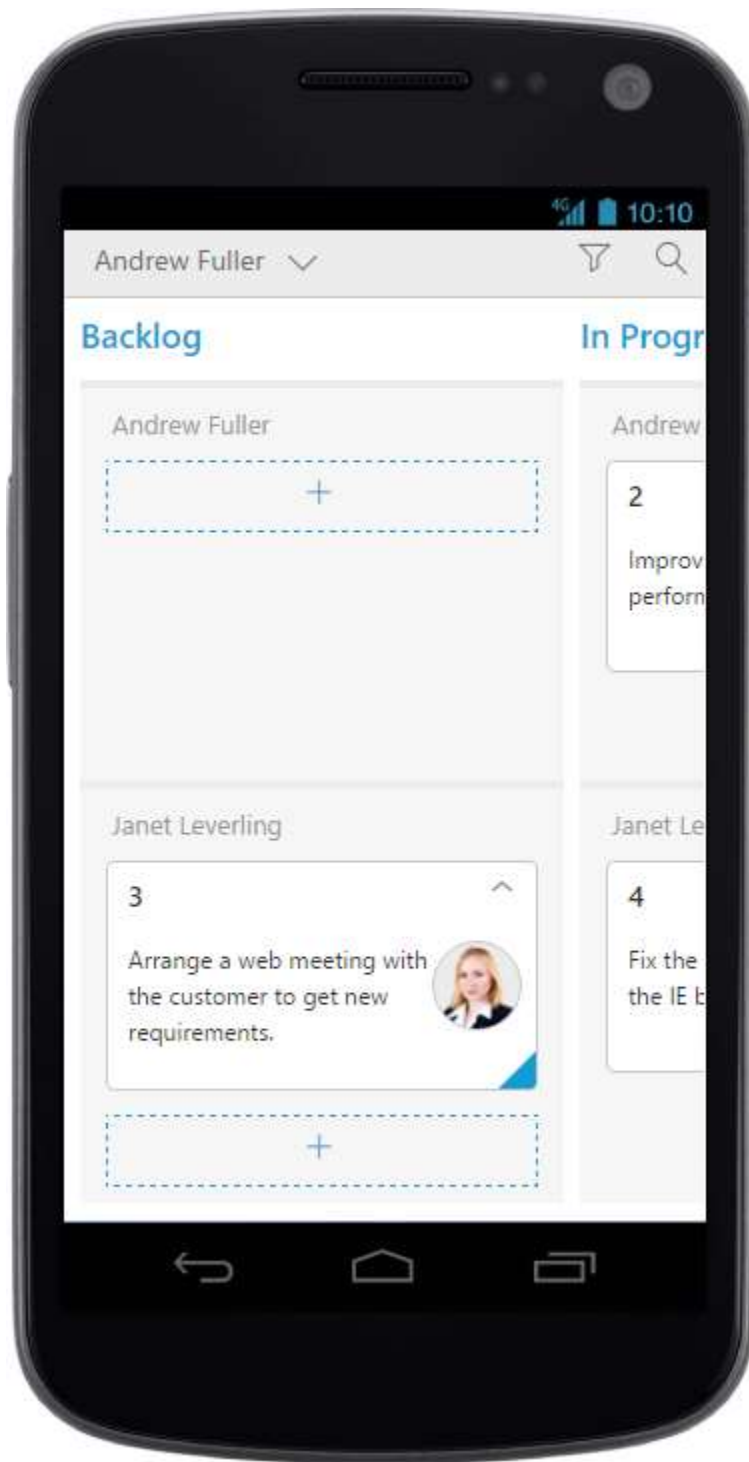
CRUD in mobile layout

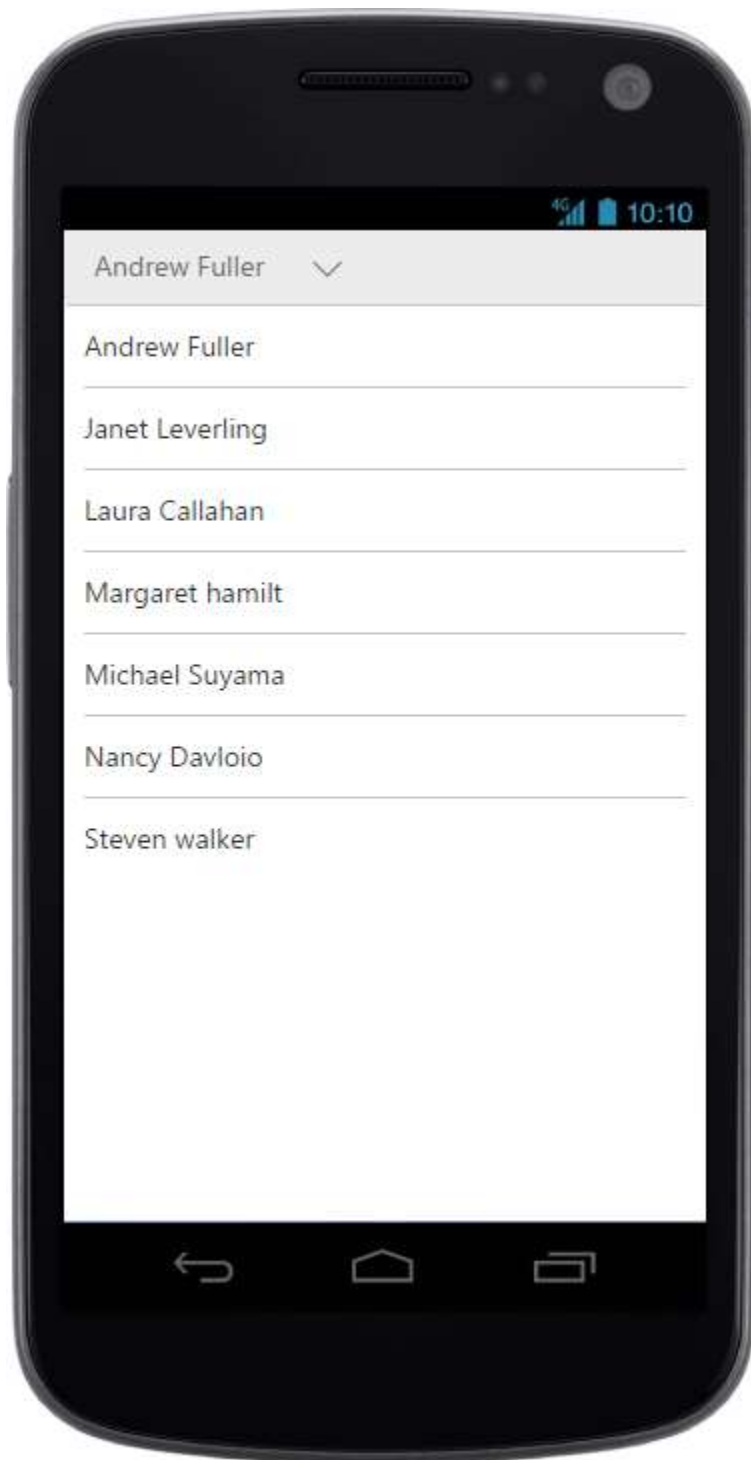


Filtering in mobile layout



Searching in mobile layout





Kanban with Swim-lane

#### Width

By default, the Kanban is adaptable to its parent container. It can adjust its width of columns based on parent container width. You can also assign width of [columns](#) in percentage.

The following code example describes the above behavior.

**HTML**

```

var data =
ej.DataManager(window.kanbanData).executeLocal(ej.Query().take(20));
ReactDOM.render(
<EJ.Kanban dataSource={data} keyField="Status" isResponsive={true} fields-
content="Summary" fields-primaryKey="Id" fields-tag="Tags">
<columns>
<column headerText="Backlog" key="Open" width="10%"></column>
<column headerText="In Progress" key="InProgress" width="10%"></column>
<column headerText="Done" key="Close" width="10%"></column>
</columns>
</EJ.Kanban>,
document.getElementById('kanbanboard-default')
);

```

**Note:** `allowScrolling` should be false while defining width in percentage.

**Min Width**

Min Width is used to maintain minimum width for the Kanban. If the Kanban width is less than [minWidth](#) then the scrollbar will be displayed to maintain minimum width.

The following code example describes the above behavior.

**HTML**

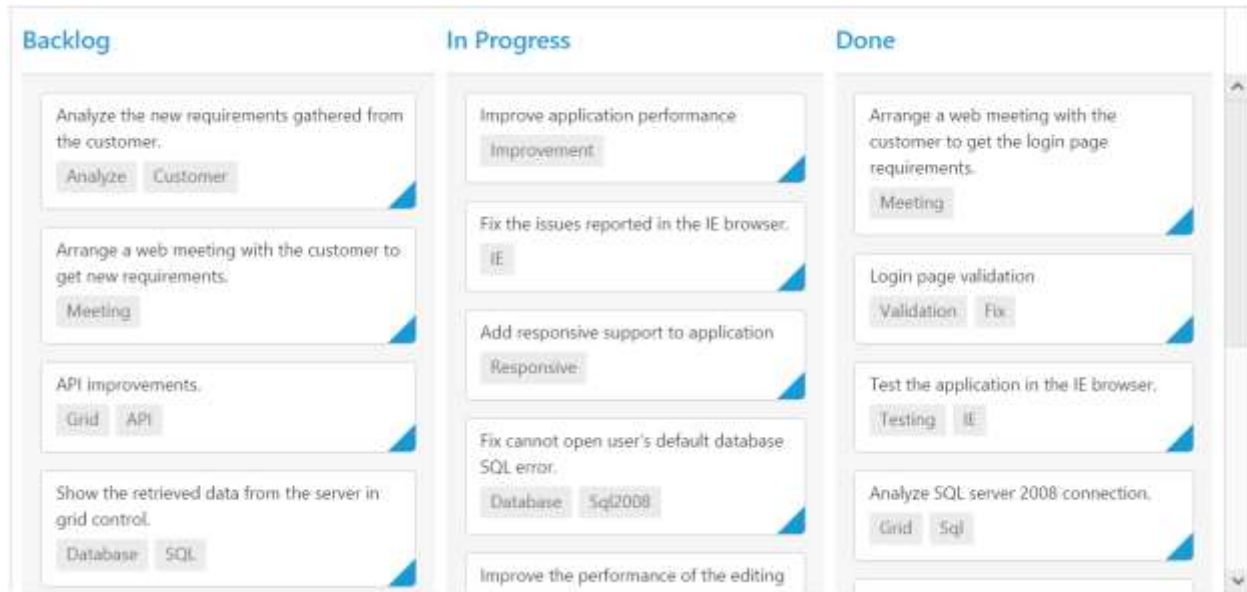
```

var data =
ej.DataManager(window.kanbanData).executeLocal(ej.Query().take(20));
ReactDOM.render(
<EJ.Kanban dataSource={data} keyField="Status" minWidth="700"
isResponsive={true} fields-content="Summary" fields-primaryKey="Id" fields-
tag="Tags">
<columns>
<column headerText="Backlog" key="Open" width="120"></column>
<column headerText="In Progress" key="InProgress" width="110"></column>
<column headerText="Done" key="Close" width="110"></column>
</columns>
</EJ.Kanban>,
document.getElementById('kanbanboard-default')
);

```

The following output is displayed as a result of the above code example.





## Stacked Headers

The stacked headers helps you to group the logical columns in Kanban. It can be shown by setting `showStackedHeader` as true and by defining [stackedHeaderRows](#).

### Adding Stacked header columns

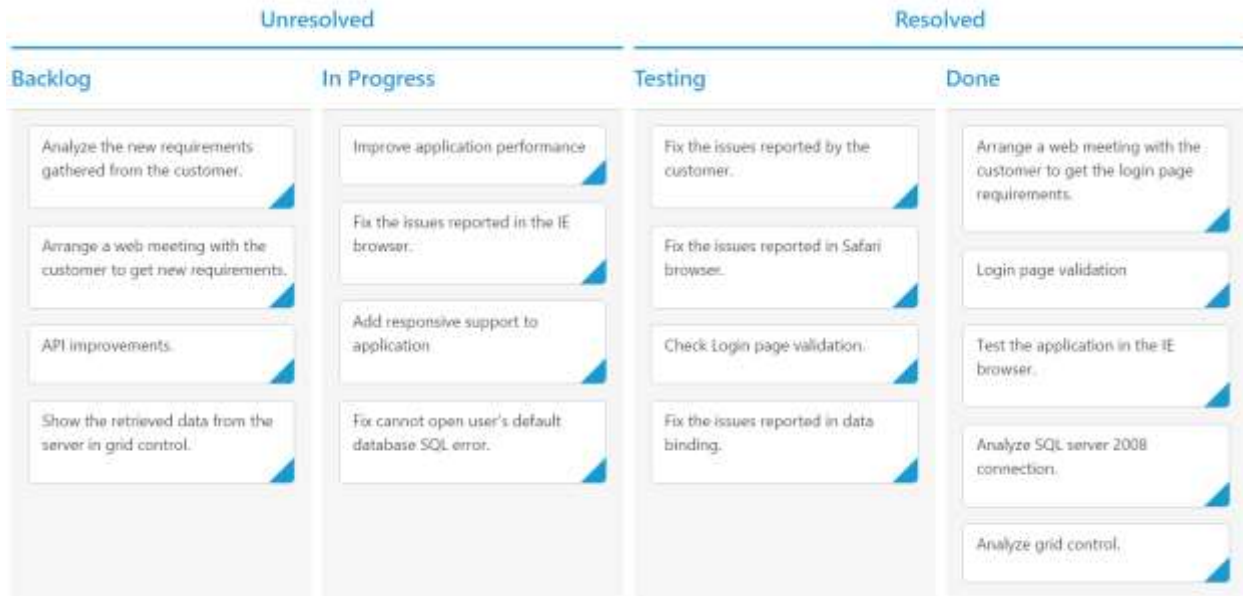
To stack columns in stacked header, you need to define [column](#) property in [stackedHeaderColumns](#) with field names of visible columns.

The following code example describes the above behavior.

### HTML

```
var data =
ej.DataManager(window.kanbanData).executeLocal(ej.Query().take(20));
var stackedHeaderRow = [{
  stackedHeaderColumns: [{
    headerText: "Unresolved",
    column: "Backlog, In Progress"
  }, {
    headerText: "Resolved",
    column: "Testing, Done"
  }]
}];
ReactDOM.render(
<EJ.Kanban dataSource={data} keyField="Status" fields-content="Summary"
fields-primaryKey="Id" stackedHeaderRows={stackedHeaderRow}>
<columns>
<column headerText="Backlog" key="Open"></column>
<column headerText="In Progress" key="InProgress"></column>
<column headerText="Testing" key="Testing"></column>
<column headerText="Done" key="Close"></column>
</columns>
</EJ.Kanban>,
document.getElementById('kanbanboard-default')
);
```

The following output is displayed as a result of the above code example.



## Context Menu

Context menu is used to improve user action with Kanban using popup menu. It can be shown by defining [contextMenuSettings-enable](#) as true. Context menu has option to add default items in [contextMenuSettings-menuItems](#) and customized items in [contextMenuSettings-customMenuItems](#).

### Default Context Menu items

Please find the below table for default context menu items and its actions.

Section	Context menu items	Action
Header	Hide Column	Hide the current column
	Visible Columns	Show the column if already hidden
Content	Add Card	Start Add new card
Card	Edit Card	Start Edit in current card
	Delete Card	Delete the current card
	Top of Row	Move the card to Top of Row
	Bottom of Row	Move the card to Bottom of Row
	Move Up	Move the card in Up direction
	Move Down	Move the card in Down direction
	Move Left	Move the card in Left direction
	Move Right	Move the card in Right direction
	Move to Swimlane	Move the card to Swim lane which is chosen from given list

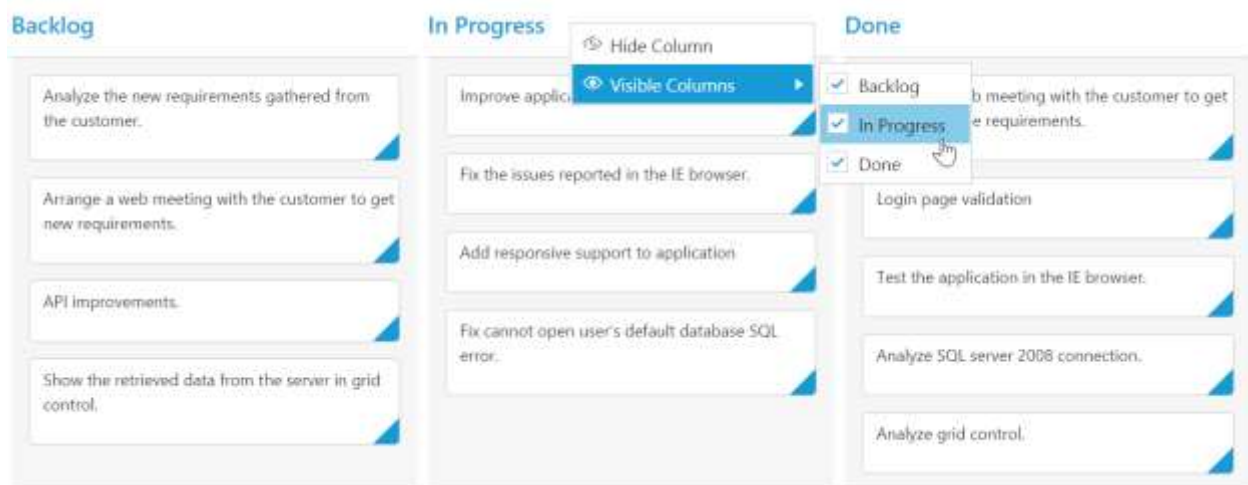
	Print Card	Print the specific card
--	------------	-------------------------

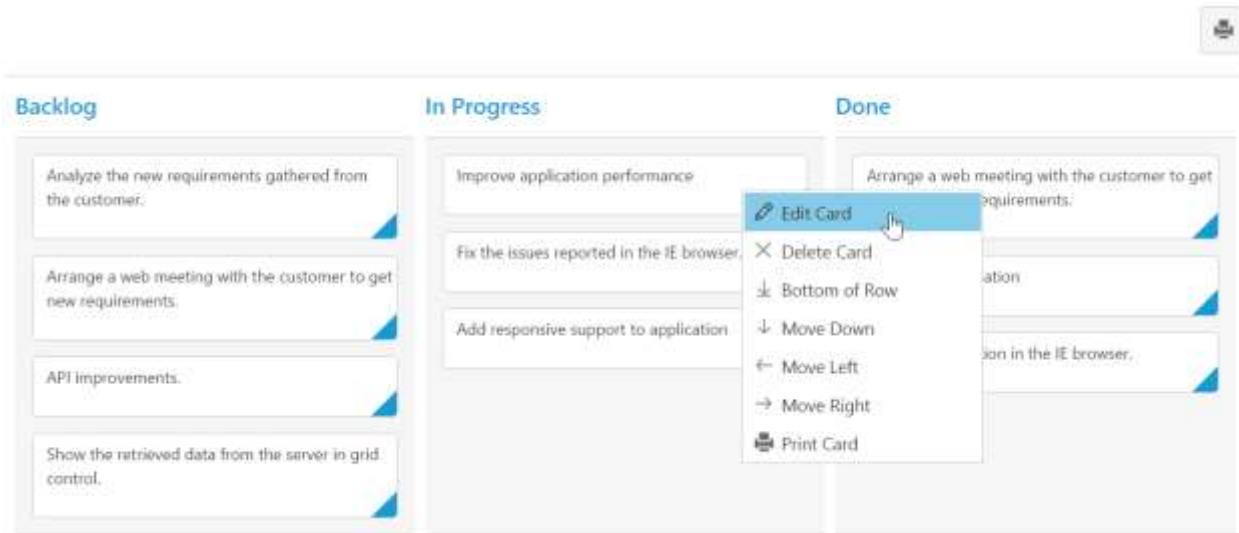
The following code example describes the above behavior.

### HTML

```
var data =
ej.DataManager(window.kanbanData).executeLocal(ej.Query().take(20));
var editItems = [
{ field: "Id", editType: ej.Kanban.EditingType.String, validationRules: {
required: true, number: true }},
{ field: "Status", editType: ej.Kanban.EditingType.Dropdown },
{ field: "Assignee", editType: ej.Kanban.EditingType.Dropdown },
{ field: "Estimate", editType: ej.Kanban.EditingType.Numeric, editParams: {
decimalPlaces: 2 }, validationRules: { range: [0, 1000] }},
{ field: "Summary", editType:
ej.Kanban.EditingType.TextArea, validationRules: { required: true}}
];
ReactDOM.render(
<EJ.Kanban dataSource={data} keyField="Status" fields-content="Summary"
fields-primaryKey="Id" fields-priority="RankId" editSettings-
editItems={editItems} editSettings-allowEditing={true} editSettings-
allowAdding={true} contextMenuSettings-enable={true}>
<columns>
<column headerText="Backlog" key="Open"></column>
<column headerText="In Progress" key="InProgress"></column>
<column headerText="Testing" key="Testing"></column>
<column headerText="Done" key="Close"></column>
</columns>
</EJ.Kanban>,
document.getElementById('kanbanBoard-default')
);
```

The following output is displayed as a result of the above code example.





### Custom Context Menu

Custom context menu is used to create your own menu item and its action. To add customized context menu items, you need to use [contextMenuSettings-customMenuItems](#) property and to bind required actions for this, use [contextClick](#) event.

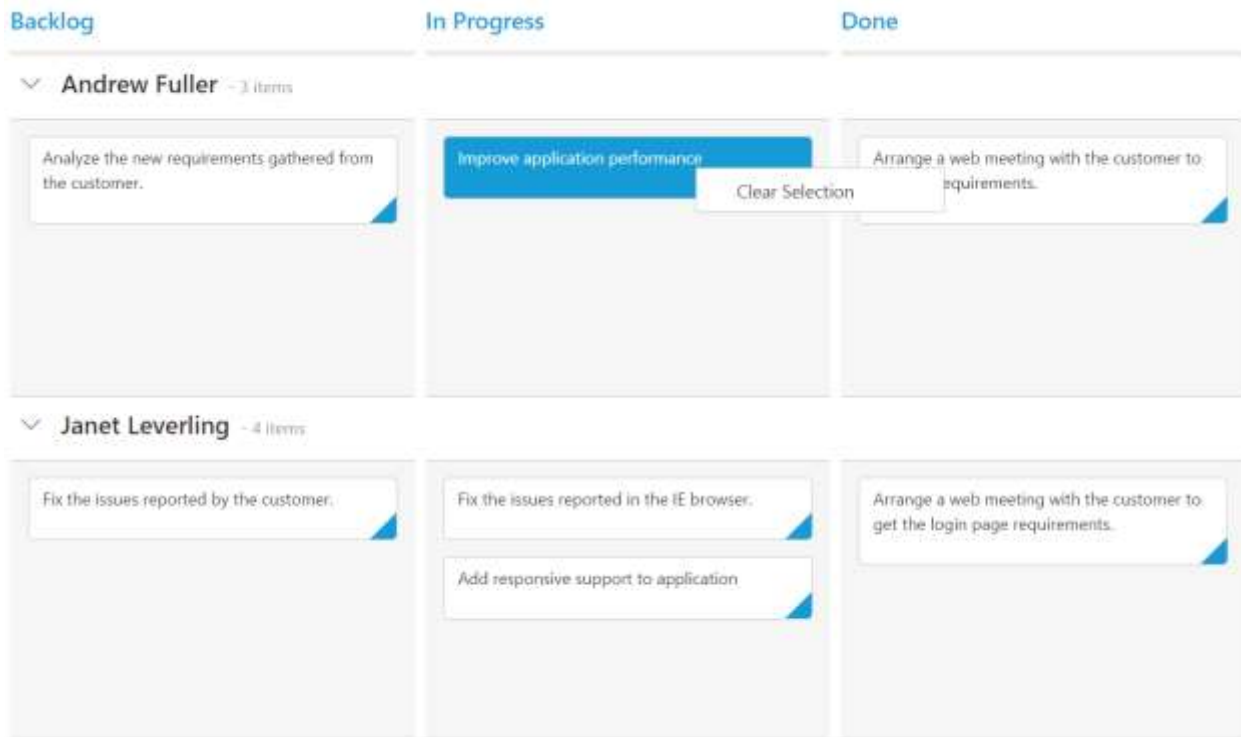
The following code example describes the above behavior.

### HTML

```
var data =
ej.DataManager(window.kanbanData).executeLocal(ej.Query().take(20));
var editItems = [
{ field: "Id"},
{ field: "Status", editType: ej.Kanban.EditingType.Dropdown },
{ field: "Assignee", editType: ej.Kanban.EditingType.Dropdown },
{ field: "Estimate", editType: ej.Kanban.EditingType.Numeric },
{ field: "Summary", editType: ej.Kanban.EditingType.TextArea }
];
var customMenuItem = [{ text: "Clear Selection" }];
var menuItem = [];
function contextClick(args) {
if (args.text == "Clear Selection")
this.KanbanSelection.clear();
}
ReactDOM.render(
<EJ.Kanban dataSource={data} keyField="Status" fields-content="Summary"
fields-primaryKey="Id" fields-priority="RankId" fields-
swimlaneKey="Assignee" editSettings-editItems={editItems} editSettings-
allowEditing={true} editSettings-allowAdding={true} contextMenuSettings-
enable={true} contextMenuSettings-customMenuItems={customMenuItem}
contextMenuSettings-menuItems={menuItem} contextClick={contextClick}>
<columns>
<column headerText="Backlog" key="Open"></column>
<column headerText="In Progress" key="InProgress"></column>
<column headerText="Done" key="Close"></column>
</columns>
</EJ.Kanban>,
document.getElementById('kanbanBoard-default')
```

```
);
```

The following output is displayed as a result of the above code example.



### Sub Context Menu

Sub context menu is used to add customized sub menu to the custom context menu item. To add a sub context menu, you need to use [contextMenuSettings-subMenu](#) property and to bind required actions for this, use [contextClick](#) event.

The following code example describes the above behavior.

### HTML

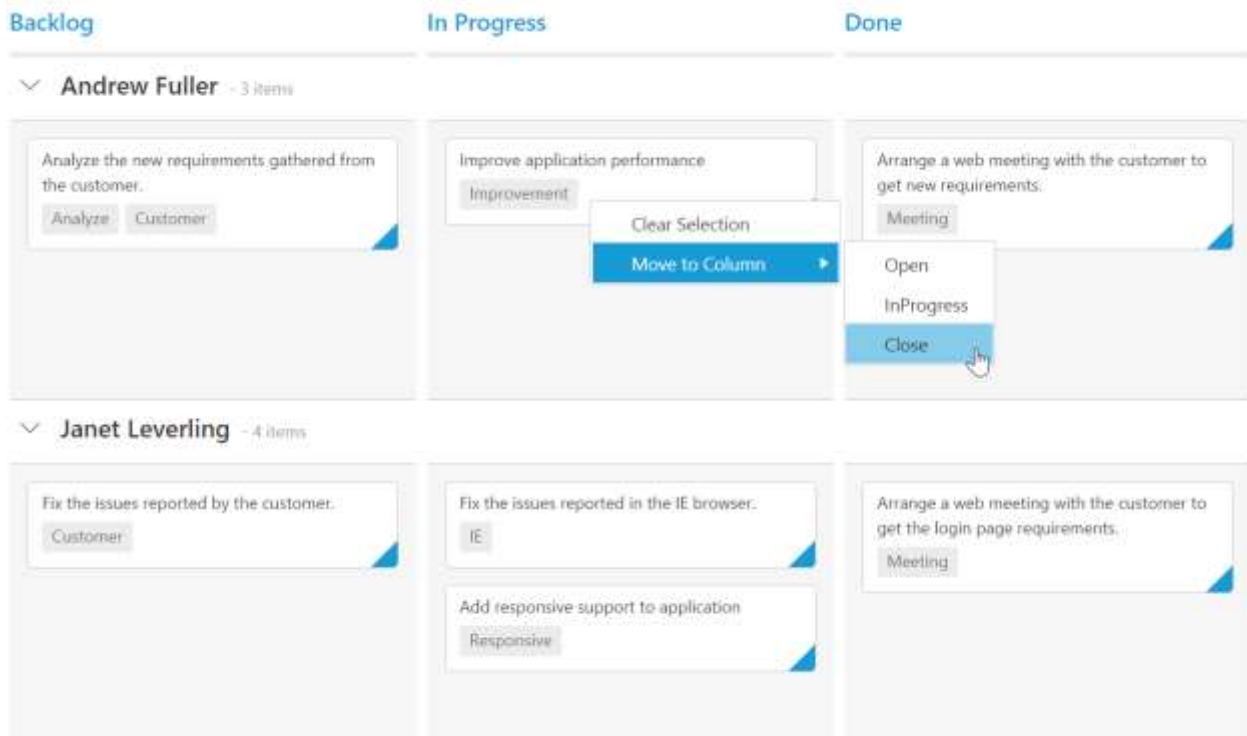
```
var data =
ej.DataManager(window.kanbanData).executeLocal(ej.Query().take(20));
var editItems = [
{ field: "Id"},
{ field: "Status", editType: ej.Kanban.EditingType.Dropdown },
{ field: "Assignee", editType: ej.Kanban.EditingType.Dropdown },
{ field: "Estimate", editType: ej.Kanban.EditingType.Numeric },
{ field: "Summary", editType: ej.Kanban.EditingType.TextArea }
];
var customMenuItem = [{ text: "Clear Selection" }, { text: "Move to Column",
template: "#submenu"}];
var menuItem = [];
function contextClick(args) {
if (args.text == "Clear Selection")
this.KanbanSelection.clear();
else if (args.text != "Move to Column")
this.updateCard(args.data.id, args.data);
}
```

```
ReactDOM.render(
  <EJ.Kanban dataSource={data} keyField="Status" fields-content="Summary"
  fields-primaryKey="Id" fields-priority="RankId" fields-
  swimlaneKey="Assignee" editSettings-editItems={editItems} editSettings-
  allowEditing={true} editSettings-allowAdding={true} contextMenuSettings-
  enable={true} contextMenuSettings-customMenuItems={customMenuItem}
  contextMenuSettings-menuItems={menuItem} contextClick={contextClick}>
    <columns>
      <column headerText="Backlog" key="Open"></column>
      <column headerText="In Progress" key="InProgress"></column>
      <column headerText="Done" key="Close"></column>
    </columns>
  </EJ.Kanban>,
  document.getElementById('kanbanBoard-default')
);
```

## HTML

```
<div id="kanbanboard-default"></div>
<script src="app/kanbanboard/default.js"></script>
<ul id="submenu">
  <li><a>Open</a></li>
  <li><a>InProgress</a></li>
  <li><a>Close</a></li>
</ul>
```

The following output is displayed as a result of the above code example.



## Print

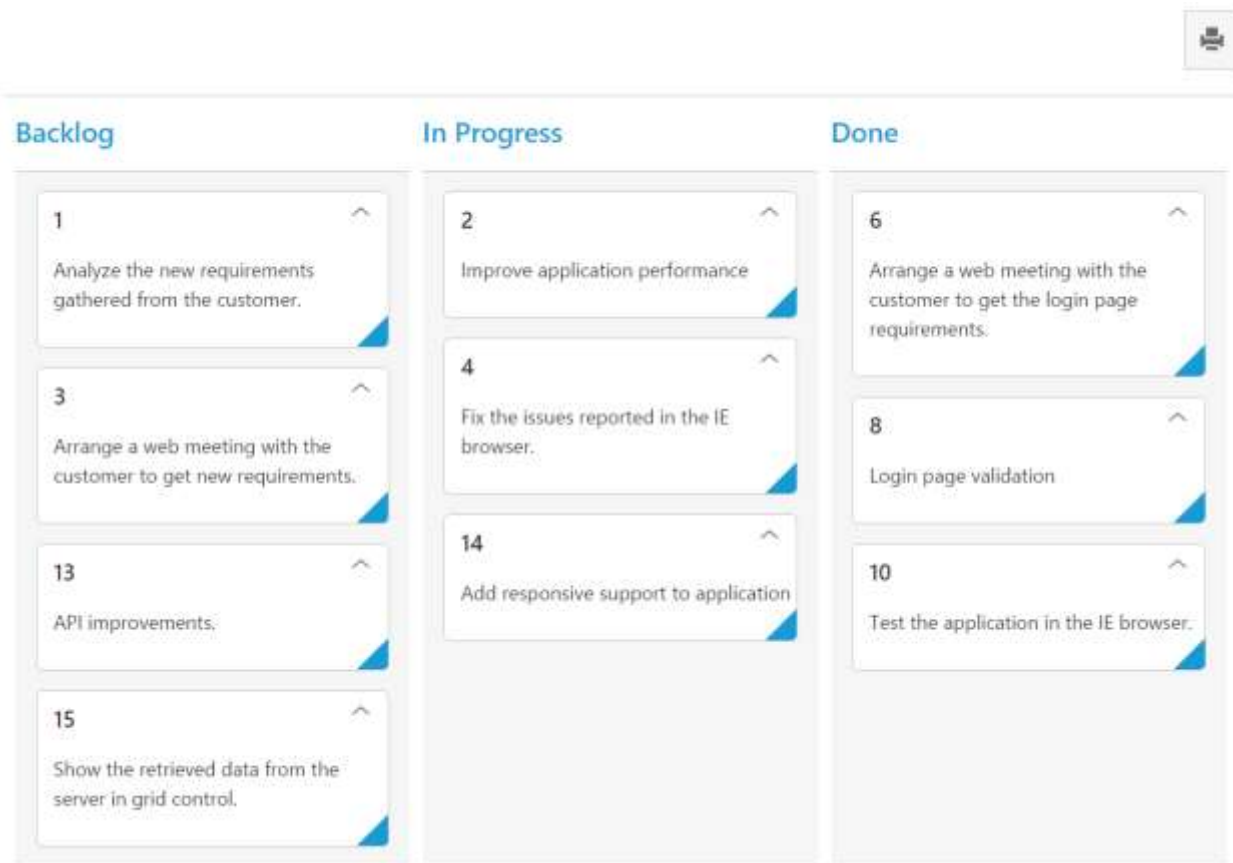
Set '[allowPrinting](#)' as true, to enable Print icon in the Kanban toolbar. You can use '[print\(\)](#)' method from Kanban instance to print the Kanban.

The following code example describes the above behavior.

## HTML

```
var data =
ej.DataManager(window.kanbanData).executeLocal(ej.Query().take(20));
ReactDOM.render(
<EJ.Kanban dataSource={data} keyField="Status" fields-content="Summary"
fields-primaryKey="Id" allowPrinting={true}>
<columns>
<column headerText="Backlog" key="Open"></column>
<column headerText="In Progress" key="InProgress"></column>
<column headerText="Done" key="Close"></column>
</columns>
</EJ.Kanban>,
document.getElementById('kanbanboard-default')
);
```

The following output is displayed as a result of the above code example.



## Localization

### Localization

All text in Kanban can be localized using `ej.Kanban.Locale` object. Please find the table with list of properties and its value in locale object.

Locale key words	Text
EmptyCard	No cards to display
SaveButton	Save
CancelButton	Cancel
EditFormTitle	Details of
AddFormTitle	Add New Card
SwimlaneCaptionFormat	"- {{:count}}{{if count == 1 }} item {{else}} items {{/if}}"
FilterSettings	Filters:
Min	Min
Max	Max
FilterOfText	Of
Cards	Cards
ItemsCount	Items Count :
Unassigned	Unassigned
AddCard	Add Card
EditCard	Edit Card
DeleteCard	Delete Card
TopofRow	Top of Row
BottomofRow	Bottom of Row
MoveUp	Move Up
MoveDown	Move Down
MoveLeft	Move Left
MoveRight	Move Right
MovetoSwimlane	Move to Swimlane
HideColumn	Hide Column
VisibleColumns	Visible Columns



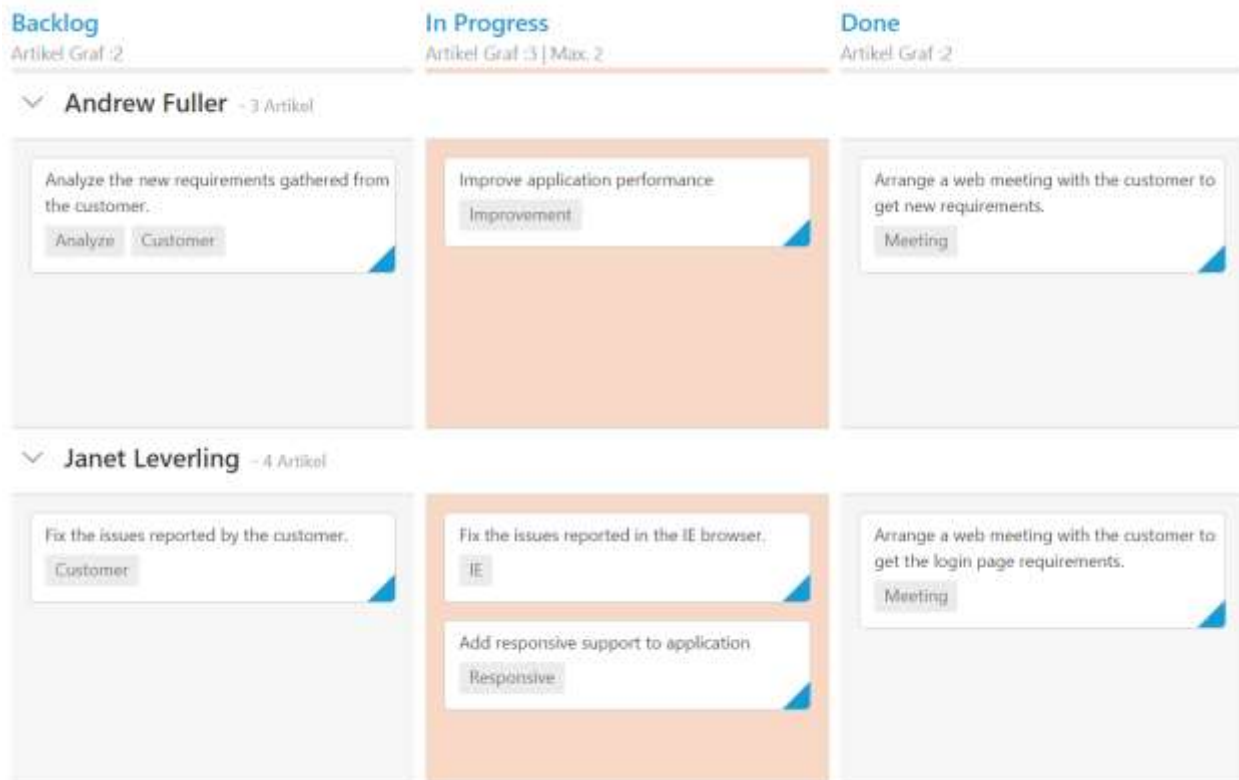
PrintCard	Print Card
Search	Search

The following code example describes the above behavior.

### HTML

```
var data =
ej.DataManager(window.kanbanData).executeLocal(ej.Query().take(20));
ej.Kanban.Locale["de-DE"] = {
EmptyCard: "Keine Karten angezeigt werden",
SaveButton: "Speichern",
CancelButton: "stornieren",
EditFormTitle: "Details von ",
AddFormTitle: "Neue Karte hinzufügen",
SwimlaneCaptionFormat: "- 8 Artikel Artikel ",
FilterSettings: "Filter:",
FilterOfText: "Von",
Max: "Max.",
Min: "Min.",
Cards: "Karten",
ItemsCount: "Artikel Graf :",
Unassigned: "Nicht zugewiesen",
};
ReactDOM.render(
<EJ.Kanban dataSource={data} locale="de-DE" keyField="Status" fields-
content="Summary" fields-primaryKey="Id" fields-tag="Tags" fields-
swimlaneKey="Assignee" enableTotalCount={true}>
<columns>
<column headerText="Backlog" key="Open"></column>
<column headerText="In Progress" key="InProgress" constraints-
max="2"></column>
<column headerText="Done" key="Close"></column>
</columns>
</EJ.Kanban>,
document.getElementById('kanbanboard-default')
);
```

The following output is displayed as a result of the above code example.



## Styling

### List of classes and its purposes

To modify Kanban appearance, you need to override default CSS of Kanban. Please find the list of CSS classes and its corresponding section in Kanban. Also you have an option to create your own custom theme for all JavaScript controls using our [Theme Studio](#)

Section	CSS class	Purpose of CSS class
Root	e-kanban	This classes are in this root element (div) of Kanban control.
	e-js	
Header	e-kanbantoolbar	This class is added at div element of Kanban toolbar.
	e-kanbanheader	This is class is added in the root element of header element. In this class, You can override thin line between header and content of Kanban.
	e-table	This class is added at 'table' of Kanban header. This CSS class makes table width as 100 %.
	e-columnheader	This class is added at 'tr' of Kanban header.
	e-headercell	This class is added in 'th' element of Kanban header. You can override background color of header and border color
	e-headercelldiv	This class is add in div which present 'th' element in header. You recommend you to use e-headercelldiv to override skeleton of header.

Body	e-kanbancontent	This class is added at root of body content. This is to override background color of body.
	e-table	This class is added to table of content. This CSS class makes table width as 100 %.
	e-swimlanerow	This class is added to all swim lane â€™s in Kanban.
	e-columnrow	This class is added to all next row of swimlane â€™s in Kanban
	e-rowcell	This class is added to all cells in Kanban. This is to override cells appearance and styling.
	e-cardselection	This class is added to card div element of Kanban. This is override selection.
	e-hover	This class adds to card of Kanban while hover cards.

## Web Accessibility

### Keyboard Navigation

Supported Keyboard Interactions keys with its description are tabulated as follows

Interaction Keys	Description
Alt + j	Focus the Kanban
Insert	Insert card in Kanban
Delete	Delete card in Kanban
F2	Edit card in Kanban
Enter	Save edited or added
Esc	Cancel add or edit state
Home	Go to first card
End	Go to last card
Up arrow	Move to up card selection
Down arrow	Move to down card selection
Right arrow	Move to right card selection
Left arrow	Move to left card selection
Ctrl + UpArrow	Collapse All swim lane groups
Ctrl + DownArrow	Expand All swim lane groups
Alt + UpArrow	Collapse selected swim lane
Alt + DownArrow	Expand selected swim lane

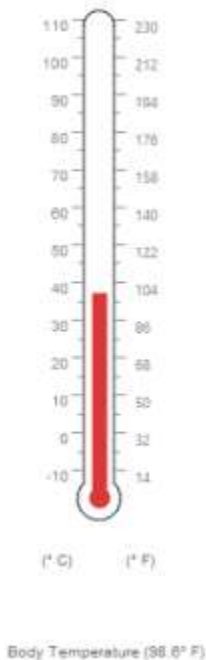
Alt + LeftArrow	Collapse selected column
Alt + RightArrow	Expand selected column
Shift + UpArrow	Multi Selection by Up Arrow
Shift + DownArrow	Multi Selection by Down Arrow
Shift + LeftArrow	Multi Selection by Left Arrow
Shift + RightArrow	Multi Selection by Right Arrow

## LinearGauge

### Getting Started

This section briefly explains on how to create a Linear Gauge control for your application.

- You can provide data for a Linear Gauge and display them in a required way. You can also customize the default Linear Gauge appearance to meet your requirements.
- In this example, you will learn how to create a Linear Gauge and how to design a thermometer, which can be used to check the body temperature of a person.



### Adding Script Reference

Create an **HTML** page and add the scripts references in the order mentioned in the following code example.

- [jQuery](#) 1.10.2 and later versions

The required ReactJS script dependencies as follows. And you can also refer [React](#) to know more about ReactJS.

- react.min.js - <http://cdn.syncfusion.com/js/assets/external/react.min.js>
- react-dom.min.js - <http://cdn.syncfusion.com/js/assets/external/react-dom.min.js>
- ej.web.react.min.js - <http://cdn.syncfusion.com/{{ site.releaseversion }}/js/common/ej.web.react.min.js>

To get started, you can use the `ej.web.all.min.js` file that encapsulates all the `ej` controls and frameworks in one single file.

## HTML

```
<!DOCTYPE html>
<html>
<head>
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<meta name="description" content="Essential Studio for React JS">
<meta name="author" content="Syncfusion">
<title>Getting Started for Ribbon React JS</title>
<!-- Essential Studio for JavaScript theme reference -->
<link href="http://cdn.syncfusion.com/{{ site.releaseversion }}/js/web/flat-azure/ej.web.all.min.css" rel="stylesheet" />
<!-- Essential Studio for JavaScript script references -->
<script src="http://cdn.syncfusion.com/js/assets/external/jquery-3.0.0.min.js"></script>
<script
src="http://cdn.syncfusion.com/js/assets/external/react.min.js"></script>
<script src="http://cdn.syncfusion.com/js/assets/external/react-dom.min.js"></script>
<script src="http://cdn.syncfusion.com/{{ site.releaseversion }}/js/web/ej.web.all.min.js"></script>
<script src="http://cdn.syncfusion.com/{{ site.releaseversion }}/js/common/ej.web.react.min.js"></script>
<!-- Add your custom scripts here -->
</head>
<body>
</body>
</html>
```

**Note:** 1. In production, we highly recommend you to use our [custom script generator](#) to create custom script file with required controls and its dependencies only. Also to reduce the file size further please use [GZip compression](#) in your server.

2. For themes, you can use the `ej.web.all.min.css` CDN link from the code snippet given. To add the themes in your application, please refer to [this link](#).

## Control Initialization

Control can be initialized in two ways.

- Using jsx Template
- Without using jsx Template

## Using jsx Template

By using the jsx template, we can create the HTML file and jsx file. The .jsx file can be convert to .js file and it can be referred in HTML page.

### Create a Linear Gauge

You can easily create the Linear Gauge widget by using the following steps.

1.Create a

tag.

#### HTML

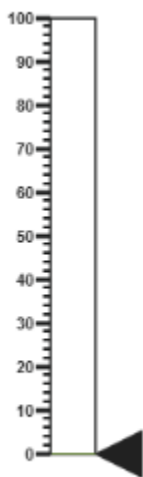
```
<!DOCTYPE html>
<html>
<body>
<div id="linearGauge-default" ></div>
<script src="app/lineargauge/default.js"></script>
</body>
</html>
```

2.Initialize the LinearGauge by using the EJ.LinearGauge tag

#### JAVASCRIPT

```
"use strict";
ReactDOM.render(
  <div className="default">
    <EJ.LinearGauge id="lineargauge1"></EJ.LinearGauge>,
  </div>,
  document.getElementById('linearGauge-default')
);
```

Run the above code example to get a default Linear Gauge with default values as follows.



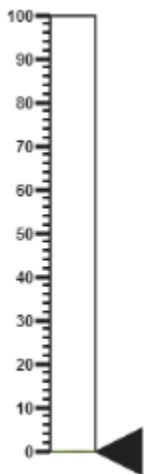
### Set Height and Width values

Basic attributes of each canvas elements are height and width. You can set the height and width of the gauge using the following code example. It sets the height and width of the canvas image where the thermometer is to be rendered.

#### JAVASCRIPT

```
<script type="text/babel">
<!DOCTYPE html>
<html>
<body>
<script type="text/babel">
ReactDOM.render(
<div className="default">
<EJ.LinearGauge id="lineargauge1" height={550}
width={500}></EJ.LinearGauge>,
</div>,
document.getElementById('linearGauge-default')
);
</script>
</body>
</html>
```

Run the above code example and you will get the following gauge as similar to default. Here height and width of the canvas are set for given values.



### Set Animation option and Label Color

- You can draw the Thermometer with some Label color to display the measurement value. For example give the labelColor as “#8c8c8c”.
- Set the EnableAnimation property as false to avoid animation on the pointers.

#### JAVASCRIPT

```
<script type="text/babel">
<!DOCTYPE html>
<html>
```

```

<body>
<script type="text/babel">
ReactDOM.render(
<div className="default">
<EJ.LinearGauge id="lineargauge1" height={550} width={500}
labelColor="#8c8c8c" enableAnimation={false}></EJ.LinearGauge>,
</div>,
document.getElementById('linearGauge-default')
);
</script>
</body>
</html>

```

Run the above code example and you will get the following gauge as the output.



### Provide Scale Values

- The scale must have the appearance of a thermometer. By giving ScaleType as Thermometer, you can render a thermometer design.
- Minimum temperature can go up to -10 and maximum temperature can rise up to 110, so you can give minimum scale value as -10 and maximum value as 110.
- Set the location values such as vertical and horizontal position of the thermometer and give the thermometer height as Length.
- You can give the Minor Interval value as 5 to get the exact temperature on the patient.

### JAVASCRIPT

```

<script type="text/babel">
var scale = [
{
type: "thermometer",
backgroundColor: "transparent",
minimum: -10,
maximum: 110,
minorIntervalValue: 5,
width: 20,

```



```

position: { x: 50, y: 18 },
length: 355,
border: { width: 0.5 }
};
<!DOCTYPE html>
<html>
<body>
<script type="text/babel">
ReactDOM.render(
<div className="default">
<EJ.LinearGauge id="lineargauge1" height={550} width={500}
labelColor="#8c8c8c" enableAnimation={false}
scales={scale}></EJ.LinearGauge>,
</div>,
document.getElementById('linearGauge-default')
);
</script>
</body>
</html>

```

Run the above code example and you will get the following gauge as the output.



### Add Pointers

In Linear Gauge the two types of pointers available are: Marker pointer and Bar pointer.

- Marker pointer is displayed as a pointer device that shows the actual values. But for your thermometer there is no need for the marker pointer. So you can hide the marker pointer by giving opacity as 0.
- Bar pointer acts as the mercury metal that shows the exact temperature of the patient. Set some of the basic properties of the Bar pointer such as Width, BarPointerDistanceFromScale, BarPointerValue and BarPointerBackgroundColor.

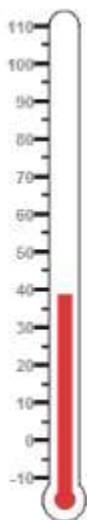
**JAVASCRIPT**

```

<script type="text/babel">
var scale = [
{ //Add the pointers customization code here
markerPointers: [{ opacity: 0 }],
barPointers: [{
width: 10,
distanceFromScale: -0.5,
value: 37,
backgroundColor: "#DB3738"
}],
}
];
<!DOCTYPE html>
<html>
<body>
<script type="text/babel">
ReactDOM.render(
<div className="default">
<EJ.LinearGauge id="lineargauge1" height={550} width={500}
labelColor="#8c8c8c" enableAnimation={false}
scales={scale}></EJ.LinearGauge>,
</div>,
document.getElementById('linearGauge-default')
);
</script>
</body>
</html>

```

On executing the above code sample renders a Linear Gauge with bar marker as follows.



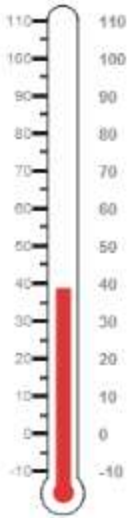
## Add Label Customization

- For thermometer, you can display the label value on two sides, to get temperature in different scales. For that you can add two label values in an array.
- To display the value around the scales, labels are used. You can customize the label placement, font (including its style and family) and its distance from scale.

### JAVASCRIPT

```
<script type="text/babel">
var scale = [{
  labels: [{
    placement: "near",
    font: {
      size: "10px", fontFamily: "Segoe UI",
      fontStyle: "Normal"
    }
  },
  {
    placement: "far",
    distanceFromScale: { x: 10 }
  }
  ]
};
<!DOCTYPE html>
<html>
<body>
<script type="text/babel">
ReactDOM.render(
<div className="default">
<EJ.LinearGauge id="lineargauge1" height={550} width={500}
labelColor="#8c8c8c" enableAnimation={false}
scales={scale}></EJ.LinearGauge>,
</div>,
document.getElementById('linearGauge-default')
);
</script>
</body>
</html>
```

On executing the above code sample renders a customized Linear Gauge as follows.



### Add Tick Details

- Tick style has two values called major interval and minor interval. You can set major ticks width and height greater than Minor ticks. And you can give TickColor, for better visibility in light backgrounds.
- Here four tick details are used for both sides having minor and major ticks. To display the tick value add the following code example.

### JAVASCRIPT

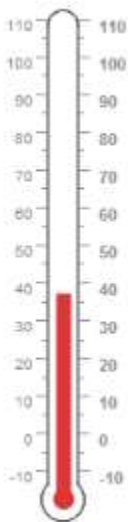
```
<script type="text/babel">
var scale = [{
  ticks: [{
    type: "majorInterval",
    height: 8,
    width: 1,
    color: "#8c8c8c",
    distanceFromScale: { y: -4 }
  }, {
    type: "minorInterval",
    height: 4,
    width: 1,
    color: "#8c8c8c",
    distanceFromScale: { y: -4 }
  }, {
    type: "majorInterval",
    placement: "far",
    height: 8,
    width: 1,
    color: "#8c8c8c",
    distanceFromScale: { y: -4 }
  }, {
    type: "minorInterval",
    placement: "far",
    height: 4,
    width: 1,
```

```

color: "#8c8c8c",
distanceFromScale: { y: -4 }
}],
}
];
<!DOCTYPE html>
<html>
<body>
<script type="text/babel">
ReactDOM.render(
<div className="default">
<EJ.LinearGauge id="lineargauge1" height={550} width={500}
labelColor="#8c8c8c" enableAnimation={false}
scales={scale}></EJ.LinearGauge>,
</div>,
document.getElementById('linearGauge-default')
);
</script>
</body>
</html>

```

On executing the above code sample renders a Linear Gauge with custom labels as follows.



### Add Custom Label Details

- Custom labels are used to specify the texts in the gauge.
- It can be customized through various properties.
- In order to show the custom labels, change the showIndicators property to true.
- Here you can use custom text to display three range descriptions.

### JAVASCRIPT

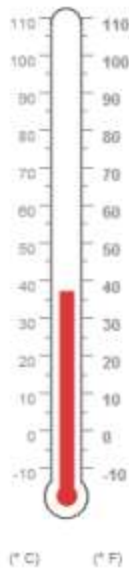
```

<script type="text/babel">
var scale = [{
showCustomLabels: true,

```

```
customLabels: [{
  value: "(° C)",
  position: { x: 44, y: 78 },
  font: { size: "13px", fontFamily: "Segoe UI", fontStyle: "bold" }, color:
"#666666"
}, {
  value: "(° F)",
  position: { x: 56, y: 78 },
  font: { size: "13px", fontFamily: "Segoe UI", fontStyle: "bold" }, color:
"#666666"
},
{
  position: { x: 51, y: 90 },
  font: { size: "13px", fontFamily: "Segoe UI", fontStyle: "bold" },
  color: "#666666"
}]
}
];
<!DOCTYPE html>
<html>
<body>
<script type="text/babel">
ReactDOM.render(
<div className="default">
<EJ.LinearGauge id="lineargauge1" height={550} width={500}
labelColor="#8c8c8c" enableAnimation={false}
scales={scale}></EJ.LinearGauge>,
</div>,
document.getElementById('linearGauge-default')
);
</script>
</body>
</html>
```

Run the above code example to get the following gauge as output.



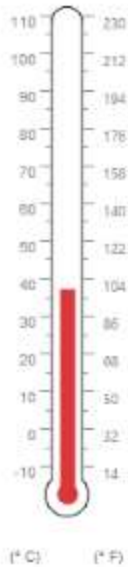
### Change Scale from Degree to Fahrenheit

Add the function that converts the temperature in degree to Fahrenheit in the label, with an index value of 1.

#### JAVASCRIPT

```
<script type="text/babel">
function label(args)
{
  if (args.label.index == 1) {
    args.style.textValue = (args.label.value * (9 / 5)) + 32;
    args.style.font = { size: "10px", fontFamily: "Segoe UI", fontStyle:
      "normal" };
  }
};
<!DOCTYPE html>
<html>
<body>
<script type="text/babel">
ReactDOM.render(
  <div className="default">
    <EJ.LinearGauge id="lineargauge1" height={550} width={500}
    labelColor="#8c8c8c" enableAnimation={false}
    drawLabels={label}></EJ.LinearGauge>,
    </div>,
    document.getElementById('linearGauge-default')
  );
</script>
</body>
</html>
```

Run the above code example and you will get the following gauge as output.



### Add Custom Label for Current Value

Add the function that displays the current temperature value in the custom label.

#### JAVASCRIPT

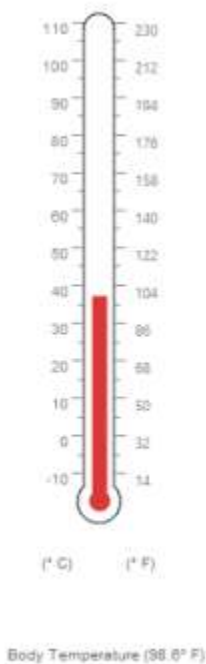
```
<script type="text/babel">
function customLabel(args) {
  if (args.customLabelIndex == 2) {
    var temp = args.scaleElement.barPointers[0].value;
    var faValue = (temp * (9 / 5)) + 32;
    if (temp == -10) {
      args.style.textValue = "Very Cold Weather" + " (" + faValue.toFixed(1) + "° F)";
    }
    else if ((temp > -10 && temp < 0) || (temp > 0 && temp < 15)) {
      args.style.textValue = "Cool Weather" + " (" + faValue.toFixed(1) + "° F)";
    }
    else if (temp == 0) {
      args.style.textValue = "Freezing point of Water" + " (" + faValue.toFixed(1) + "° F)";
    }
    else if (temp >= 15 && temp < 30) {
      args.style.textValue = "Room Temperature" + " (" + faValue.toFixed(1) + "° F)";
    }
    else if (temp == 30) {
      args.style.textValue = "Beach Weather" + " (" + faValue.toFixed(1) + "° F)";
    }
    else if (temp == 37) {
      args.style.textValue = "Body Temperature" + " (" + faValue.toFixed(1) + "° F)";
    }
    else if (temp == 40) {
      args.style.textValue = "Hot Bath Temperature" + " (" + faValue.toFixed(1) + "° F)";
    }
  }
}
```



```

else if (temp > 40 && temp < 100) {
args.style.textValue = "Very Hot Temperature" + " (" + faValue.toFixed(1) +
"° F) ";
}
else if (temp == 100) {
args.style.textValue = "Boiling point of Water" + " (" + faValue.toFixed(1)
+ "° F) ";
}
}
}
<!DOCTYPE html>
<html>
<body>
<script type="text/babel">
ReactDOM.render(
<div className="default">
<EJ.LinearGauge id="lineargauge1" height={550} width={500}
labelColor="#8c8c8c" enableAnimation={false} drawLabels={label}
drawCustomLabels={customLabel}></EJ.LinearGauge>,
</div>,
document.getElementById('linearGauge-default')
);
</script>
</body>
</html>

```



### Without using jsx Template

The Linear Gauge can be created from a HTML **DIV** element with the HTML **id** attribute set to it. Refer to the following code example.

#### HTML

```
<div id="linearGauge-default"></div>
```

### JAVASCRIPT

```
<script type="text/babel">
var scale = [
{
  type: "thermometer",
  backgroundColor: "transparent",
  minimum: -10,
  maximum: 110,
  minorIntervalValue: 5,
  width: 20,
  position: { x: 50, y: 18 },
  length: 355,
  border: { width: 0.5 }
}
];
ReactDOM.render(
  React.createElement(EJ.LinearGauge, {id: "default",
    labelColor: "#8c8c8c",
    scales: scale,
    width: 500,
    height: 550,
  }
),
  document.getElementById('linearGauge-default')
);
</script>
```

Run the above code example and you will see the following output.



## Basic Settings

### Adding Dimension

- The basic customization for any control is setting the dimension. Here dimension refers the two major attributes, height and width. The height and width assigned in the control will render the canvas element in the given size.
- The value attribute is used to set all pointer value in the Linear Gauge control. The attributes, minimum and maximum value are used to set the minimum value and maximum value for all the scales exist in the Linear Gauge control.

### HTML

```
<div id="LinearGauge1"></div>
```

### JAVASCRIPT

```
"use strict";
var scales = [{
  border: { color: "transparent", width: 0 },
  showMarkerPointers: false, showBarPointers: true,
  // Adding bar pointer collection
  barPointers: [{ width: 5, backgroundColor: "Grey" }],
  // Adding ticks collection
  ticks: [{
    type: "majorinterval", width: 2,
    color: "#8c8c8c", distanceFromScale: { x: 7, y: 0 }
  },
  {
    type: "minorinterval", width: 1, height: 6,
    color: "#8c8c8c", distanceFromScale: { x: 7, y: 0 }
  }
  ]
  }];
ReactDOM.render(
  <EJ.LinearGauge id="lineargauge1" scales = {scales} height= {500} width=
  {300} minimum= {10} maximum ={110} value={78}></EJ.LinearGauge>,
  document.getElementById('LinearGauge1')
);
```

Execute the above code to render the following output.



### Adding frame

- Frame is the element that decides the appearance of the **Linear Gauge**. You can customize it by using the object called **frame**. It contains frame inner width, frame outer width and frame background image URL properties.
- The **innerWidth** of the frame defines the distance between the canvas element and the frame and the **outerWidth** refers to distance from the frame. **backgroundUrl** is used to set the background image for the frame.

### HTML

```
<div id="LinearGauge1"></div>
```

### JAVASCRIPT

```
"use strict";
var scales = [{
  backgroundColor: "transparent",
  border: { color: "transparent", width: 0 },
  showMarkerPointers: false, showBarPointers: true,
  // Adding bar pointer collection
  barPointers: [{ width: 5, backgroundColor: "Grey" }],
  // Adding ticks collection
  ticks: [{
    type: "majorinterval", width: 2,
    color: "#8c8c8c", distanceFromScale: { x: 7, y: 0 }
  },
  {
    type: "minorinterval", width: 1, height: 6,
    color: "#8c8c8c", distanceFromScale: { x: 7, y: 0 }
  }]
}];
var frame = {
  // For setting frame inner width
  innerWidth: 8,
  // For setting frame outer width
  outerWidth: 10,
  // For setting back ground Image URL
  backgroundImageUrl: "Gauge_linear_light.png"
};
ReactDOM.render(
  <EJ.LinearGauge id="lineargauge"
  labelColor= "#8c8c8c"
  load="loadGaugeTheme" frame = {frame} scales = {scales} value={80}>
  </EJ.LinearGauge>,
  document.getElementById('LinearGauge1')
);
```

Execute the above code to render the following output.



### Appearance

- The attribute orientation is used to render the Linear Gauges either in horizontal position or vertical position. You can set the background color for the Linear Gauge for better appearance using the backgroundColor property. borderColor specifies the border color of the Linear Gauge. You can also add gradient effects to Linear Gauge with the help of pointerGradient1 and pointerGradient2 attribute.
- Theme is the basic property of any control. It is used to set the theme for Linear Gauge. There are two types of themes used for Linear Gauge such as
  - flatlight
  - flatdark

**HTML**

```
<div id="LinearGauge1"></div>
```

**JAVASCRIPT**

```
"use strict";
var scales = [{
  backgroundColor: "transparent",
  border: { color: "transparent", width: 0 },
  showMarkerPointers: false, showBarPointers: true,
  // Adding bar pointer collection
  barPointers: [{ width: 5, backgroundColor: "Grey" }],
  // Adding ticks collection
  ticks: [{
    type: "majorinterval", width: 2,
    color: "#8c8c8c", distanceFromScale: { x: 7, y: 0 }
  },
  {
    type: "minorinterval", width: 1, height: 6,
    color: "#8c8c8c", distanceFromScale: { x: 7, y: 0 }
  }
  ]];
var frame = {
  // For setting frame inner width
  innerWidth: 8,
  // For setting frame outer width
  outerWidth: 10,
  // For setting back ground Image URL
  backgroundImageUrl: "Gauge_linear_light1.png"
};
ReactDOM.render(
  <EJ.LinearGauge id="lineargauge"
  labelColor= "#8c8c8c"
  load="loadGaugeTheme" frame = {frame} orientation = "Horizontal" scales =
  {scales} value={80}>
  </EJ.LinearGauge>,
  document.getElementById('LinearGauge1')
);
```

Execute the above code to render the following output.



**Responsive**

- For any display devices, the control is to be rendered based on the space in that device. The control must be responsive. For this purpose resizing property is present in **Linear Gauge** control.
- The **Linear Gauge** renders with the specified value. When the browser changes its size, the canvas element checks the dimension with its parent element and if there are any changes in parent dimension, gauge control also changes the dimension based on its parent changes. You can enable this feature using **isResponsive** property.

**HTML**

```
<div id="LinearGauge1"></div>
```

**JAVASCRIPT**

```
"use strict";
var scales = [{
  backgroundColor: "transparent",
  border: { color: "transparent", width: 0 },
  showMarkerPointers: false, showBarPointers: true,
  // Adding bar pointer collection
  barPointers: [{ width: 5, backgroundColor: "Grey" }],
  // Adding ticks collection
  ticks: [{
    type: "majorinterval", width: 2,
    color: "#8c8c8c", distanceFromScale: { x: 7, y: 0 }
  },
  {
    type: "minorinterval", width: 1, height: 6,
    color: "#8c8c8c", distanceFromScale: { x: 7, y: 0 }
  }
  ]];
var frame = {
  // For setting frame inner width
  innerWidth: 8,
  // For setting frame outer width
  outerWidth: 10,
  // For setting back ground Image URL
  backgroundImageUrl: "Gauge_linear_light1.png"
};
ReactDOM.render(
  <EJ.LinearGauge id="lineargauge"
  labelColor= "#8c8c8c"
  load="loadGaugeTheme" frame = {frame} orientation = "Horizontal"
  isResponsive = {true} scales = {scales} value={80}>
  </EJ.LinearGauge>,
  document.getElementById('LinearGauge1')
);
```

Execute the above code to render the following output.



Responsiveness of the linear gauge is controlled by using `enableResize` property.

**JAVASCRIPT**

```
ReactDOM.render(
  <EJ.LinearGauge id="lineargauge"
  width={100} height={100} enableResize={true}>
  </EJ.LinearGauge>,
  document.getElementById('LinearGauge1')
);
```

### Localization

**LinearGauge** supports localization for its axis labels and tooltip. To render the gauge with specific culture you have to refer the corresponding globalize culture script and need to specify the culture name in `locale` property of gauge.

**Enable Group Separator** is used to Convert the date object to string while using the locale settings, you can set `enableGroupSeparator` property as **true**.

### JAVASCRIPT

```
ReactDOM.render(  
  <EJ.LinearGauge id="lineargauge"  
    locale="en-fr" enableGroupSeparator={true}>  
  </EJ.LinearGauge>,  
  document.getElementById('LinearGauge1')  
) ;
```

### Interaction and Animation

- **Linear Gauge** control contains **Interaction** feature. You can use this interaction feature to change the pointer values manually either by clicking or dragging the pointer over the **Gauge**. It dynamically changes the value of pointer when dragged. To Enable/Disable the user interaction you can use the **readOnly** Boolean property. The user interaction option is enabled when you set **readOnly** property as false. By default it holds the true value.
- **Linear Gauge** contains another attractive concept called **Animation**. The animation option enables the movement of the pointer from the minimum value to the current value. You can use animation option to change the pointer value dynamically. You can enable/ disable it using **enableAnimation** property. To enable animation set **enableAnimation** to "true".
- By default it holds the true value. You can control the speed of the pointer during animating using **animationSpeed**. It is a numerical value that holds the time in milliseconds. That is when setting value is 1000, it is considered as 1 second.

### HTML

```
<div id="LinearGauge1"></div>
```

### JAVASCRIPT

```
"use strict";  
var scales = [{  
  backgroundColor: "transparent",  
  border: { color: "transparent", width: 0 },  
  showMarkerPointers: false, showBarPointers: true,  
  // Adding bar pointer collection  
  barPointers: [{ width: 5, backgroundColor: "Grey" }],  
  // Adding ticks collection  
  ticks: [{  
    type: "majorinterval", width: 2,  
    color: "#8c8c8c", distanceFromScale: { x: 7, y: 0 }  
  },  
  {  
    type: "minorinterval", width: 1, height: 6,
```

```

color: "#8c8c8c", distanceFromScale: { x: 7, y: 0 }
}]
}];
var frame = {
// For setting frame inner width
innerWidth: 8,
// For setting frame outer width
outerWidth: 10,
// For setting back ground Image URL
backgroundImageUrl: "Gauge_linear_light1.png"
};
ReactDOM.render (
<EJ.LinearGauge id="lineargauge"
labelColor= "#8c8c8c"
load="loadGaugeTheme" frame = {frame} orientation = "Horizontal"
isResponsive = {true} scales = {scales} value={80}>
</EJ.LinearGauge>,
document.getElementById('LinearGauge1')
);

```

Execute the above code to render the following output.



#### Enable Marker Pointer Animation

Specifies the animate state for marker pointer, you can set `enableMarkerPointer` property as `true`

#### JAVASCRIPT

```

ReactDOM.render (
<EJ.LinearGauge id="lineargauge"
enableMarkerPointer={true}>
</EJ.LinearGauge>,
document.getElementById('LinearGauge1')
);

```

## Scales

**Scales** are the basic functional block of the **Linear Gauge**. You can improve the appearance of scales by customizing it. The functional blocks of **Linear Gauge** are

- Marker Pointers
- Bar Pointer
- Labels
- Custom Labels
- Indicators
- Ticks
- Ranges

#### Adding scale collection

**Scale** is the basic element of **Linear Gauge**. Scale collection is directly added to the gauge object. Refer the following code example to add scale collection in **Gauge** control.

#### HTML



```
<div id="LinearGauge1"></div>
```

## JAVASCRIPT

```
"use strict";
var scales = [{
  width: 8,
  position: { x: 20, y: 50 },
  backgroundColor: "Grey",
  border: { color: "Grey", width: 1 },
  showMarkerPointers: true, showBarPointers: false,
  // For Adding label collection
  labels: [{ distanceFromScale: { x: 50, y: 0 } }],
  // For Adding marker pointer collection
  markerPointers: [{
    type: "pentagon", placement: "near",
    length: 10, width: 20, distanceFromScale: 20, backgroundColor: "#FE8282"
  }],
  // For Adding tick collection
  ticks: [{
    type: "majorinterval", width: 2,
    color: "#8c8c8c", distanceFromScale: { x: 30, y: 0 }
  },
  {
    type: "minorinterval", width: 1, height: 6,
    color: "#8c8c8c", distanceFromScale: { x: 30, y: 0 }
  }
  ]];
var frame = {
  // For setting back ground Image URL
  backgroundImageUrl: "Gauge_linear_light.png"
};
ReactDOM.render(
  <EJ.LinearGauge id="lineargauge"
  labelColor= "#8c8c8c" frame = {frame} enableAnimation = {false} scales =
  {scales} >
  </EJ.LinearGauge>,
  document.getElementById('LinearGauge1')
);
```

Execute the above code to render the following output.



## Scale Customization

### Colors and Border

- The **Scale** border is modified with **border** object. It has two border property, **color** and **width** are used to customize the border color of the scale and border width of the scale. Setting the background color improves the look and feel of the **Linear Gauge**. You can customize the background color of the scale using **backgroundColor**.
- Scales are used to enable or disable various properties such as **showRanges**, **showIndicators**, **showCustomLabels**, **showLabels**, **showTicks**, **showBarPointers** and **showMarkerPointers**.

Enable/disable is done by setting the property into two states either “**true**” or “**false**”. You can adjust the Opacity of the scale with **opacity** property.

### HTML

```
<div id="LinearGauge1"></div>
```

### JAVASCRIPT

```
"use strict";
var scales = [{
  width: 8,
  position: { x: 20, y: 50 },
  backgroundColor: "#FE8282",
  border: { color: "Red", width: 1 },
  //For Adding opacity
  opacity: 0.5,
  //For Adding Shadow offset
  shadowOffset: 10,
  type: "roundedrectangle",
  showMarkerPointers: true,
  showBarPointers: false,
  //For Adding label collection
  labels: [{ distanceFromScale: { x: 50, y: 0 } }],
  //For Adding marker pointer collection
  markerPointers: [{
    type: "pentagon", placement: "near", length: 10,
    width: 20, distanceFromScale: 20, backgroundColor: "#C9E1E5"
  }],
  //For Adding ticks collection
  ticks: [{
    type: "majorinterval", width: 2,
    color: "#8c8c8c", distanceFromScale: { x: 30, y: 0 }
  },
  {
    type: "minorinterval", width: 1, height: 6,
    color: "#8c8c8c", distanceFromScale: { x: 30, y: 0 }
  }
  ]];
var frame = {
  // For setting back ground Image URL
  backgroundImageUrl: "Gauge_linear_light.png"
};
ReactDOM.render(
  <EJ.LinearGauge id="lineargauge"
  labelColor= "#8c8c8c" frame = {frame} enableAnimation = {false} scales =
  {scales}>
  </EJ.LinearGauge>,
  document.getElementById('LinearGauge1')
);
```

Execute the above code to render the following output.



## Appearance

- You can improve the appearance of **Linear Gauge** using various properties. You can set the interval values for the scale with **major interval value** and **minor interval value** properties and maximum and minimum value by **minimum** and **maximum** property. The **width** property is used to set the scale bar width.
- You can also adjust the Opacity of the scale with **opacity** property. The value for opacity lies between 0 and 1. **Linear Gauge** contains two scale directions, clockwise and counter clockwise. It can be set with **direction** property.

## HTML

```
<div id="LinearGauge1"></div>
```

## JAVASCRIPT

```
"use strict";
var scales = [{
  //For Adding scale width
  width: 18,
  //For Adding scale minimum value
  minimum: 10,
  //For Adding scale maximum value
  maximum: 210,
  //For Adding scale minor interval value
  minorIntervalValue: 25,
  //For Adding scale major interval value
  majorIntervalValue: 50,
  //For Adding scale direction
  direction: "counterclockwise",
  position: { x: 20, y: 50 },
  backgroundColor: "Grey",
  border: { color: "Grey", width: 1 },
  showMarkerPointers: true, showBarPointers: false,
  //Adding label collection
  labels: [{ distanceFromScale: { x: 50, y: 0 } }],
  //Adding marker pointer collection
  markerPointers: [{
    type: "pentagon", placement: "near",
    length: 10, width: 20, distanceFromScale: 20,
    backgroundColor: "#FE8282"
  }],
  //Adding tick collection
  ticks: [{
    type: "majorinterval", width: 2,
    color: "#8c8c8c", distanceFromScale: { x: 30, y: 0 }
  },
  {
    type: "minorinterval", width: 1, height: 6,
    color: "#8c8c8c", distanceFromScale: { x: 30, y: 0 }
  }
  ]
  };
var frame = {
  // For setting back ground Image URL
```

```

backgroundImageUrl: "Gauge_linear_light.png"
};
ReactDOM.render(
  <EJ.LinearGauge id="lineargauge"
  labelColor= "#8c8c8c"
  frame = {frame} enableAnimation = {false} scales = {scales} >
  </EJ.LinearGauge>,
  document.getElementById('lineargauge-default')
);

```

Execute the above code to render the following output.



### Scale Types

Scale Type is an element which decides the appearance of the gauge. **Linear Gauge** contains three scale types such as,

- Rectangle
- Rounded Rectangle
- Thermometer

### Rectangle

For rectangular scale type, the scale renders with rectangular structure. Refer the following code example.

#### HTML

```
<div id="LinearGauge1"></div>
```

#### JAVASCRIPT

```

"use strict";
var scales = [{
  width: 18, length: 300,
  position: { x: 54, y: 50 },
  //For Adding Scale type as rectangle
  type: "rectangle",
  backgroundColor: "#C0B08E",
  border: { color: "#C0B08E", width: 1 },
  showMarkerPointers: false, showBarPointers: false,
  //For Adding tick collection
  ticks: [{
    type: "majorinterval", width: 2,
    color: "#206BA4", distanceFromScale: { x: -27, y: 0 },
    placement: "far"
  },
  {
    type: "minorinterval", width: 1, height: 6,
    color: "#206BA4", distanceFromScale: { x: -27, y: 0 },
    placement: "far"
  }
  ]
  }];
var frame = {

```

```
// For setting back ground Image URL
backgroundImageUrl: "Gauge_linear_light.png"
};
ReactDOM.render(
<EJ.LinearGauge id="lineargauge"
labelColor= "#8c8c8c" frame = {frame} enableAnimation = {false} scales =
{scales} > </EJ.LinearGauge>,
document.getElementById('LinearGauge1')
);
```

Execute the above code to render the following output.



### Rounded Rectangle

For rounded rectangular scale type, the scale renders as rectangular structure but with constant radius rounded corner. Refer the following code example.

#### HTML

```
<div id="LinearGauge1"></div>
```

#### JAVASCRIPT

```
"use strict";
var scales = [{
width: 8,
direction: ej.datavisualization.LinearGauge.Directions.Clockwise,
position: { x: 60, y: 50 },
//Adding scale type as rounded rectangle
type: "roundedrectangle",
backgroundColor: "#206BA4",
border: { color: "#206BA4", width: 1 },
//Adding label collection
labels: [{ distanceFromScale: {x:-20,y:0}}],
//Adding tick collection
ticks: [{
type: "majorinterval", width: 2,
color: "#206BA4", distanceFromScale: { x: -27, y: 0 },
placement: "far"
},
{
type: "minorinterval", width: 1, height: 6,
color: "#206BA4", distanceFromScale: { x: -27, y: 0 },
placement: "far"
}]
}];
var frame = {
// For setting back ground Image URL
backgroundImageUrl: "Gauge_linear_light.png"
};
ReactDOM.render(
<EJ.LinearGauge id="lineargauge"
labelColor= "#8c8c8c" frame = {frame} enableAnimation = {false} scales =
{scales} >
</EJ.LinearGauge>,
```

```
document.getElementById('LinearGauge1')
);
```

Execute the above code to render the following output.



### Thermometer

For thermometer scale type, the scale renders as thermometer structure with rounded bottom. Refer the following code example.

### HTML

```
<div id="LinearGauge1"></div>
```

### JAVASCRIPT

```
var scales = [{
  width: 18, length: 300,
  position: { x: 54, y: 50 },
  //Adding scale type as thermometer
  type: "thermometer",
  backgroundColor: "#C0B08E",
  border: { color: "#C0B08E", width: 1 },
  showMarkerPointers: false, showBarPointers: false,
  //Adding tick collection
  ticks: [{
    type: "majorinterval", width: 2,
    color: "#206BA4", distanceFromScale: { x: -27, y: 0 },
    placement: "far"
  },
  {
    type: "minorinterval", width: 1, height: 6,
    color: "#206BA4", distanceFromScale: { x: -27, y: 0 },
    placement: "far"
  }
  ]
  }];
var frame = {
  // For setting back ground Image URL
  backgroundImageUrl: "Gauge_linear_light.png"
};
ReactDOM.render(
  <EJ.LinearGauge id="lineargauge"
  labelColor= "#8c8c8c" frame = {frame} enableAnimation = {false} scales =
  {scales} >
  </EJ.LinearGauge>,
  document.getElementById('LinearGauge1')
);
```

Execute the above code to render the following output.



### Adding multiple scales

You can set multiple scales for a single **Linear Gauge** control by using an array of scale objects. Each scale object is independent of each other. Refer the following code example to add multiple scale collection.

#### HTML

```
<div id="LinearGauge1"></div>
```

#### JAVASCRIPT

```
use strict";
var scales = [ //Adding Scale 1
{
width: 8,
position: { x: 15, y: 50 },
backgroundColor: "Grey",
border: { color: "Grey", width: 1 },
showMarkerPointers: true, showBarPointers: false,
//Adding label collection
labels: [{ distanceFromScale: { x: 50, y: 0 } }],
//Adding marker pointer collection
markerPointers: [{
type: "pentagon", placement: "near",
length: 10, width: 20, distanceFromScale: 20,
backgroundColor: "#FE8282"
}],
//Adding tick collection
ticks: [{
type: "majorinterval", width: 2,
color: "#8c8c8c", distanceFromScale: { x: 30, y: 0 }
},
{
type: "minorinterval", width: 1, height: 6,
color: "#8c8c8c", distanceFromScale: { x: 30, y: 0 }
}]
},
//Adding Scale 2
{
width: 8, direction: ej.datavisualization.LinearGauge.Directions.Clockwise,
position: { x: 90, y: 50 }, type: "roundedrectangle",
backgroundColor: "#206BA4",
border: { color: "#206BA4", width: 1 },
showMarkerPointers: false, showBarPointers: false, showLabels: false,
ticks: [{
type: "majorinterval", width: 2,
color: "#206BA4", distanceFromScale: { x: -27, y: 0 }, placement: "far"
},
{
type: "minorinterval", width: 1, height: 6,
color: "#206BA4", distanceFromScale: { x: -27, y: 0 },
placement: "far"
}]
},
//Adding Scale 3
{
```

```
width: 18, length: 300,
position: { x: 54, y: 50 }, type: "thermometer",
backgroundColor: "#C0B08E",
border: { color: "#C0B08E", width: 1 },
showMarkerPointers: false, showBarPointers: false,
showLabels: false, showTicks: false,
}];
var frame = {
// For setting back ground Image URL
backgroundImageUrl: "Gauge_linear_light.png"
};
ReactDOM.render(
<EJ.LinearGauge id="lineargauge"
labelColor= "#8c8c8c" frame = {frame} enableAnimation = {false} scales =
{scales} >
</EJ.LinearGauge>,
document.getElementById('LinearGauge1')
);
```

Execute the above code to render the following output.



## Marker Pointers

**Marker Pointer** value points out the actual value set in the **Linear Gauge**. You can set values for various pointer attributes such as value, type, length, width, border and color in pointer collection. You can also customize the pointers to improve the appearance of gauge.

### Adding marker pointer collection

You can add **Marker Pointer** collection directly to the scale object. To add pointer collection in a gauge control refer the following code example.

### HTML

```
<div id="LinearGauge1"></div>
```

### JAVASCRIPT

```
"use strict";
var scales = [{
border: { color: "transparent", width: 0 },
showBarPointers: true,
//Adding bar pointer collection
barPointers: [{ width: 5, backgroundColor: "Grey" }],
//Adding marker pointer collection
markerPointers: [{
width: 10,
length: 10,
backgroundColor: "Grey",
distanceFromScale: -12
}],
//Adding tick collection
ticks: [{
type: "majorinterval", width: 2,
color: "#8c8c8c", distanceFromScale: { x: 7, y: 0 }
```



```

},
{
  type: "minorinterval", width: 1, height: 6,
  color: "#8c8c8c", distanceFromScale: { x: 7, y: 0 }
}]
}];
var frame = {
  backgroundImageUrl: "Gauge_linear_light.png"
};
ReactDOM.render(
  <EJ.LinearGauge id="lineargauge1" frame = {frame} scales = {scales}
  value={80}></EJ.LinearGauge>,
  document.getElementById('LinearGauge1')
);

```

Execute the above code to render the following output.

![[/js/LinearGauge/Marker-Pointersimages/Marker-Pointersimg1.png]

#### Add marker pointer value

The **value** property is the important element in the marker pointer collection which indicates the gauge value. Real purpose of the **Linear Gauge** is based on the pointer value. You can set the pointer value either directly during rendering the control or it can be achieved by public method.

#### HTML

```
<div id="LinearGauge1"></div>
```

#### JAVASCRIPT

```

"use strict";
var scales = [
  {
    backgroundColor: "#AEC75F",
    direction: ej.datavisualization.LinearGauge.Directions.Clockwise,
    type: "roundedrectangle",
    border: { color: "#AEC75F", width: 30 },
    //Adding marker pointer collection
    markerPointers: [{
      width: 30, length: 30, backgroundColor: "#FE5C09",
      distanceFromScale: 20, placement: "near",
      value: 67.5
    }],
    //Adding label collection
    labels: [{ angle: 90, distanceFromScale: { x: 0, y: 100 } }],
    //Adding tick collection
    ticks: [{
      type: "majorinterval", width: 2,
      color: "#8c8c8c", distanceFromScale: { x: 45, y: -1 }
    },
    {
      type: "minorinterval", width: 1, height: 6,
      color: "#8c8c8c", distanceFromScale: { x: 45, y: -1 }
    }
  ]
}];
var frame = {
  // For setting back ground Image URL

```

```

backgroundImageUrl: "Gauge_linear_light1.png"
};
ReactDOM.render(
  <EJ.LinearGauge id="lineargauge"
  labelColor= "#8c8c8c"
  frame = {frame} enableAnimation = {false} scales = {scales}
  width = {600}
  height = {150}
  orientation = "Horizontal"
  labelColor = "Black"
  enableResize = {true}>
</EJ.LinearGauge>,
document.getElementById('LinearGauge1')
);

```

Execute the above code to render the following output.

![[/js/LinearGauge/Marker-Pointersimages/Marker-Pointersimg2.png]

## Pointer Styles

### Appearance

- Based on the value, the **pointer** points out the label value. You can set the pointer length and width using **length** and **width** property respectively. You can also adjust the opacity of the pointer using the **opacity** property which holds the value between 0 and 1. You can add the gradient effects to the pointer using **gradient** object.
- The marker pointer border is modified with the **border** object. It contains two border property namely **color** and **width** which are used to customize the border color of the scale and border width of the marker pointer. The background color can be customized with attribute **backgroundColor**.

## HTML

```
<div id="LinearGauge1"></div>
```

## JAVASCRIPT

```

"use strict";
var scales = [
{
  backgroundColor: "#AEC75F",
  direction: ej.datavisualization.LinearGauge.Directions.Clockwise,
  type: "roundedrectangle",
  border: { color: "#AEC75F", width: 30 },
  //Adding marker pointer collection
  markerPointers: [{
    //Adding width
    width: 30,
    //Adding height
    length: 30,
    //Adding opacity
    opacity: 0.4,
    //Adding background
    backgroundColor: "#FCDD34",

```

```

distanceFromScale: 20,
placement: "near",
value: 67.5
}],
//Adding label collection
labels: [{ angle: 90, distanceFromScale: { x: 0, y: 100 } }],
//Adding ticks collection
ticks: [{
type: "majorinterval", width: 2,
color: "#8c8c8c", distanceFromScale: { x: 45, y: -1 }
},
{
type: "minorinterval", width: 1, height: 6,
color: "#8c8c8c", distanceFromScale: { x: 45, y: -1 }
}]
}];
var frame = {
// For setting back ground Image URL
backgroundImageUrl: "Gauge_linear_light1.png"
};
ReactDOM.render(
<EJ.LinearGauge id="lineargauge"
labelColor= "#8c8c8c"
frame = {frame} enableAnimation = {false} scales = {scales}
width = {600}
height = {150}
orientation = "Horizontal"
labelColor = "Black"
enableResize = {true}>
</EJ.LinearGauge>,
document.getElementById('LinearGauge1')
);

```

Execute the above code to render the following output.

![[/js/LinearGauge/Marker-Pointersimages/Marker-Pointersimg3.png]

### Positioning the pointer

- You can position the Pointer with two properties, **distanceFromScale** and **placement**. The **distanceFromScale** property defines the distance between the scale and pointer.
- The **Placement** property is used to locate the pointer with respect to scale either inside or outside the scale or along the scale. It is an enumerable data type.

### HTML

```
<div id="LinearGauge1"></div>
```

### JAVASCRIPT

```

"use strict";
var scales = [
{
backgroundColor: "#AEC75F",
direction: ej.datavisualization.LinearGauge.Directions.Clockwise,

```

```

type: "roundedrectangle",
border: { color: "#AEC75F", width: 30 },
//For Adding marker pointer collection
markerPointers: [{
width: 30, height: 8, opacity: 0.4, backgroundColor: "#01A357",
distanceFromScale: 60,
placement: "near",
value: 55.5
}],
//For Adding label collection
labels: [{ angle: 90, distanceFromScale: { x: 0, y: 100 } }],
//For Adding tick collection
ticks: [{
type: "majorinterval", width: 2,
color: "#8c8c8c", distanceFromScale: { x: 45, y: -1 }
},
{
type: "minorinterval", width: 1, height: 6,
color: "#8c8c8c", distanceFromScale: { x: 45, y: -1 }
}]
}];
var frame = {
// For setting back ground Image URL
backgroundImageUrl: "Gauge_linear_light1.png"
};
ReactDOM.render(
<EJ.LinearGauge id="lineargauge"
labelColor= "#8c8c8c"
load="loadGaugeTheme" frame = {frame} enableAnimation = {false} scales =
{scales}
width = {600}
height = {150}
orientation = "Horizontal"
labelColor = "Black"
enableResize = {true}> </EJ.LinearGauge>,
document.getElementById('LinearGauge1')
);

```

Execute the above code to render the following output.



### Types

It is possible to change the dimension of the marker pointer. Dimensions available for marker pointer are,

- Rectangle
- Triangle
- Ellipse
- Diamond
- Pentagon
- Circle
- Slider
- Pointer

- Wedge
- Trapezoid
- Rounded Rectangle

### Multiple Marker Pointers

**Linear Gauge** can contain multiple pointers on it. You can use any combination and any number of pointers in a gauge. That is, a gauge can contain any number of marker pointer and any number of bar pointers. Refer the following code example containing multiple marker pointers.

#### HTML

```
<div id="LinearGauge1"></div>
```

#### JAVASCRIPT

```
"use strict";
var scales = [
{
  backgroundColor: "#AEC75F", showCustomLabels: true,
  direction: ej.datavisualization.LinearGauge.Directions.Clockwise,
  type: "roundedrectangle",
  border: { color: "#AEC75F", width: 30 },
  markerPointers: [
    // Adding marker pointer 1
    {
      width: 30, length: 30, backgroundColor: "#01A357",
      distanceFromScale: 60, placement: "near", value: 32.2
    },
    // Adding marker pointer 2
    {
      width: 10, length: 30, backgroundColor: "#90DAFB",
      distanceFromScale: 60, placement: "near", value: 23.7, type: "circle"
    },
    // Adding marker pointer 3
    {
      width: 3, length: 30, backgroundColor: "#90DAFB",
      distanceFromScale: 60, placement: "near", value: 23.7, type: "star"
    }
  ],
  // Adding label collection
  labels: [{ angle: 90, distanceFromScale: { x: 0, y: 100 } }],
  // Adding tick collection
  ticks: [{
    type: "majorinterval", width: 2,
    color: "#8c8c8c", distanceFromScale: { x: 45, y: -1 }
  },
  {
    type: "minorinterval", width: 1, height: 6,
    color: "#8c8c8c", distanceFromScale: { x: 45, y: -1 }
  }
  ],
  // Adding custom label collection
  customLabels: [{
    value: "Weather Condition in California", position: {
      x: 50, y: 20
    }
  },
  ]
}]
```

```

    }];
    var frame = {
    // For setting back ground Image URL
    backgroundImageUrl: "Gauge_linear_light1.png"
    };
    ReactDOM.render(
    <EJ.LinearGauge id="lineargauge"
    labelColor= "#8c8c8c" frame = {frame} enableAnimation = {false} scales =
    {scales}
    width = {600}
    height = {150}
    orientation = "Horizontal"
    labelColor = "Black"
    enableResize = {true}>
    </EJ.LinearGauge>,
    document.getElementById('LinearGauge1')
    );

```

Execute the above code to render the following output.



### Bar Pointers

**Bar Pointer** value points out the actual value set in the **Linear Gauge** as marker pointer. You can set the values of the various bar pointer attributes such as value, width, border and color in bar pointer collection. You can also customize the pointers to improve the appearance of gauge.

### Adding bar pointer collection

You can add Bar Pointer collection directly to the scale object. Refer the following code example.

### HTML

```
<div id="LinearGauge1"></div>
```

### JAVASCRIPT

```

"use strict";
var scales = [
{
    backgroundColor: "transparent",
    border: { color: "transparent", width: 0 },
    showMarkerPointers: false, showBarPointers: true,
    //For Adding bar pointer collection
    barPointers: [{ width: 5, backgroundColor: "Grey"}],
    //For Adding tick collection
    ticks: [{ type: "majorinterval", width: 2,
    color: "#8c8c8c", distanceFromScale: { x: 7, y: 0 } },
    { type: "minorinterval", width: 1,height:6,
    color: "#8c8c8c", distanceFromScale: { x: 7, y: 0 } }]
    }];
var frame = {
    innerWidth: 8,
    outerWidth: 10,
    // For setting back ground Image URL
    backgroundImageUrl: "Gauge_linear_light.png"
};

```

```
ReactDOM.render(
  <EJ.LinearGauge id="lineargauge"
    labelColor= "#8c8c8c"
    load="loadGaugeTheme" frame = {frame} value = {78} scales = {scales} >
  </EJ.LinearGauge>,
  document.getElementById('lineargauge-default')
);
```

Execute the above code to render the following output.



### Adding bar pointer value

Bar pointer value is also important element in the **Linear Gauge** as it indicates the gauge value. Real purpose of the **Linear Gauge** is based on the pointer value. You can set the bar pointer value either directly during rendering the control or it can be achieved by public method.

### HTML

```
<div id="LinearGauge1"></div>
```

### JAVASCRIPT

```
"use strict";
var scales = [
  {
    backgroundColor: "transparent",
    border: { color: "transparent", width: 0 },
    showMarkerPointers: false, showBarPointers: true,
    //For Adding bar pointer collection
    barPointers: [{
      width: 5,
      backgroundColor: "Grey",
      value: 91
    }],
    //For Adding tick collection
    ticks: [{
      type: "majorinterval", width: 2,
      color: "#8c8c8c", distanceFromScale: { x: 7, y: 0 }
    },
    {
      type: "minorinterval", width: 1, height: 6,
      color: "#8c8c8c", distanceFromScale: { x: 7, y: 0 }
    }
  ]
}];
var frame = {
  innerWidth: 8,
  outerWidth: 10,
  // For setting back ground Image URL
  backgroundImageUrl: "Gauge_linear_light.png"
};
ReactDOM.render(
  <EJ.LinearGauge id="lineargauge"
    labelColor= "#8c8c8c"
    load="loadGaugeTheme" frame = {frame} scales = {scales} >
  </EJ.LinearGauge>,
```

```
document.getElementById('lineargauge-default')
);
```

Execute the above code to render the following output.



## Pointer Styles

### Appearance

- Based on the value, the bar pointer points out the label value. You can set the bar pointer width using **width** property and you can also adjust the opacity of the pointer using **opacity** property that holds the value between 0 and 1. You can add the gradient effects to the pointer using **gradient** object.
- The marker pointer border is modified with the object **border**. It has two border property, **color** and **width** which are used to customize the border color of the scale and border width of the marker pointer. The background color can be customized with attribute **backgroundColor**.

## HTML

```
<div id="LinearGauge1"></div>
```

## JAVASCRIPT

```
"use strict";
var scales = [
{
  backgroundColor: "transparent",
  border: { color: "transparent", width: 0 },
  showMarkerPointers: false, showBarPointers: true,
  // Adding bar pointer collection
  barPointers: [{
    width: 10,
    backgroundColor: "Red",
    border: { color: "#860201", width: 2 },
    opacity: 0.7,
    value: 91 }],
  // Adding tick collection
  ticks: [{
    type: "majorinterval", width: 2,
    color: "#8c8c8c", distanceFromScale: { x: 7, y: 0 }
  },
  {
    type: "minorinterval", width: 1, height: 6,
    color: "#8c8c8c", distanceFromScale: { x: 7, y: 0 }
  }
  ]];
var frame = {
  innerWidth: 8,
  outerWidth: 10,
  // For setting back ground Image URL
  backgroundImageUrl: "Gauge_linear_light.png"
};
ReactDOM.render(
```



```
<EJ.LinearGauge id="lineargauge"
labelColor= "#8c8c8c"
load="loadGaugeTheme" frame = {frame} scales = {scales} >
</EJ.LinearGauge>,
document.getElementById('LinearGauge1')
);
```

Execute the above code to render the following output.

![[/js/LinearGauge/Bar-Pointersimages/Bar-Pointersimg3.png]

### Positioning the pointer

- Bar pointer can be positioned with two properties such as **distanceFromScale** and **placement**. The **distanceFromScale** property defines the distance between the scale and pointer element.
- The **placement** property is used to locate the pointer with respect to scale either inside or outside the scale or along the scale. It is an enumerable data type.

### HTML

```
<div id="LinearGauge1"></div>
```

### JAVASCRIPT

```
"use strict";
var scales = [
{
  backgroundColor: "transparent",
  border: { color: "transparent", width: 0 },
  showMarkerPointers: false, showBarPointers: true,
  //Adding bar pointer collection
  barPointers: [{
    width: 10,
    backgroundColor: "#8BABFF",
    value: 91,
    placement: "near",
    distanceFromScale: 20
  }],
  //Adding tick collection
  ticks: [{
    type: "majorinterval", width: 2,
    color: "#8c8c8c", distanceFromScale: { x: 7, y: 0 }
  },
  {
    type: "minorinterval", width: 1, height: 6,
    color: "#8c8c8c", distanceFromScale: { x: 7, y: 0 }
  }
  ]];
var frame = {
  innerWidth: 8,
  outerWidth: 10,
  // For setting back ground Image URL
  backgroundImageUrl: "Gauge_linear_light.png"
};
ReactDOM.render(
```

```
<EJ.LinearGauge id="lineargauge"
labelColor= "#8c8c8c"
load="loadGaugeTheme" frame = {frame} scales = {scales} >
</EJ.LinearGauge>,
document.getElementById('LinearGauge1')
);
```

Execute the above code to render the following output.

![[/js/LinearGauge/Bar-Pointersimages/Bar-Pointersimg4.png]

### Multiple Bar Pointers

**Linear Gauge** can contain multiple bar pointers on it. You can use any combination and any number of pointers in a gauge. That is, a Gauge can contain any number of marker pointer and any number of bar pointers. Refer the following code example containing multiple bar pointers.

### HTML

```
<div id="LinearGauge1"></div>
```

### JAVASCRIPT

```
"use strict";
var scales = [
{
backgroundColor: "transparent",
border: { color: "transparent", width: 0 },
showMarkerPointers: false, showBarPointers: true, showCustomLabels: true,
//Adding bar pointer collection
barPointers: [
//Adding bar pointer 1
{
width: 10, backgroundColor: "#8BABFF",
value: 91, placement: "near", distanceFromScale: 60
},
//Adding bar pointer 2
{
width: 10, backgroundColor: "#FDB761", value: 51,
placement: "near", distanceFromScale: 20
},
//Adding bar pointer 3
{
width: 10, backgroundColor: "Red", value: 88,
placement: "near", distanceFromScale: 100
}
],
//Adding tick collection
ticks: [{
type: "majorinterval", width: 2,
color: "#8c8c8c", distanceFromScale: { x: 7, y: 0 }
},
{
type: "minorinterval", width: 1, height: 6,
color: "#8c8c8c", distanceFromScale: { x: 7, y: 0 }
}],
//Adding custom label collection
```

```

customLabels: [
  {
    value: "Mathematics Mark Comparision",
    position: { x: 55, y: 97 }
  },
  { value: "Halfyearly", position: { x: 72, y: 87 }, textAngle: 90 },
  { value: "Quaterly", position: { x: 56, y: 87 }, textAngle: 90 },
  { value: "Annual", position: { x: 87, y: 87 }, textAngle: 90 }]
];
var frame = {
  innerWidth: 8,
  outerWidth: 10,
  // For setting back ground Image URL
  backgroundImageUrl: "Gauge_linear_dark1.png"
};
ReactDOM.render(
  <EJ.LinearGauge id="lineargauge"
    labelColor= "#8c8c8c"
    load="loadGaugeTheme" frame = {frame} scales = {scales} width = {300}
    height = {500} labelColor = "Grey" enableAnimation = {false} >
  </EJ.LinearGauge>,
  document.getElementById('LinearGauge1')
);

```

Execute the above code to render the following output.

![[/js/LinearGauge/Bar-Pointersimages/Bar-Pointersimg5.png]

## Labels

Labels are units that are used to display the values in the scales. You can customize Labels with the properties like angle, color, font, opacity, etc.

## Label Customization

### Appearance

- The attribute **angle** is used to display the labels in the specified angles and **color** attribute is used to display the labels in specified color. You can adjust the opacity of the label with the property **opacity** and the values of it lies between 0 and 1. The **includeFirstValue** is a special property by enabling this property, the first value of the label is not rendered.
- Font option is also available on the labels. The basic three properties of fonts such as size, family and style can be achieved by **size**, **fontStyle** and **fontFamily**. Labels are two types such as major and minor. Major type labels are for major interval values and minor type labels are for minor interval values.

## HTML

```
<div id="LinearGauge1"></div>
```

## JAVASCRIPT

```

"use strict";
var scales = [
{

```

```

backgroundColor: "transparent",
border: { color: "transparent", width: 0 },
showBarPointers: true, showCustomLabels: true, showMarkerPointers: false,
//Adding bar pointer collection
barPointers: [{ width: 10 }],
//Adding label collection
labels: [{
font: { size: "12px", fontFamily: "Arial", fontStyle: "Bold" },
textColor: "Red",
angle: 10,
opacity: 0.5,
includeFirstValue: false
}],
//Adding ticks collection
ticks: [{
type: "majorinterval", width: 2,
color: "#8c8c8c", distanceFromScale: { x: 7, y: 0 }
},
{
type: "minorinterval", width: 1, height: 6,
color: "#8c8c8c", distanceFromScale: { x: 7, y: 0 }
}]
}];
var frame = {
innerWidth: 8,
outerWidth: 10,
// For setting back ground Image URL
backgroundImageUrl: "Gauge_linear_light.png"
};
ReactDOM.render(
<EJ.LinearGauge id="lineargauge"
labelColor= "#8c8c8c"
load="loadGaugeTheme" frame = {frame} scales = {scales} enableAnimation =
{false} value = {40}>
</EJ.LinearGauge>,
document.getElementById('LinearGauge1')
);

```

Execute the above code to render the following output.



### Unit text and Positioning

- The **unitText** property is used to add some text along with the labels. For example, in speedometer, you need to mention the units in kph. You can also add the unit text in front of the labels. To achieve this use the enumerable property **unitTextPosition**.
- Labels can be positioned with the help of two properties such as **distanceFromScale** and **placement**. **distanceFromScale** property defines the distance between the scale and labels. **Placement** property is used to locate the labels with respect to scale either inside the scale or outside the scale or along the scale. It is an enumerable data type.

### HTML

```
<div id="LinearGauge1"></div>
```

**JAVASCRIPT**

```

"use strict";
var scales = [
{
width: 5, majorIntervalValue: 25, minorIntervalValue: 5,
backgroundColor: "White", showCustomLabels: true,
showMarkerPointers: false, showBarPointers: true,
direction: ej.datavisualization.LinearGauge.Directions.Clockwise,
type: "roundedrectangle",
border: { color: "#AEC75F", width: 2 },
// Adding bar pointer collection
barPointers: [{ width: 4, backgroundColor: "Red" }],
// Adding label collection
labels: [{
angle: 90,
distanceFromScale: { x: 0, y: 60 },
unitText: "%"
}],
// Adding tick collection
ticks: [{
type: "majorinterval", width: 2,
color: "#8c8c8c", distanceFromScale: { x: 0, y: 25 }
},
{
type: "minorinterval", width: 1, height: 6,
color: "#8c8c8c", distanceFromScale: { x: 0, y: 25 }
}],
customLabels: [{
value: "Download in Progress", position: { x: 50, y: 20 },
}]
}];
var frame = {
// For setting back ground Image URL
backgroundImageUrl: "Gauge_linear_light1.png"
};
ReactDOM.render(
<EJ.LinearGauge id="lineargauge"
labelColor= "#8c8c8c"
load="loadGaugeTheme" frame = {frame} orientation = "Horizontal" scales =
{scales} width = {600} height = {250} value = {31} labelColor = "Black"
enableResize = {true} enableAnimation = {false}>
</EJ.LinearGauge>,
document.getElementById('LinearGauge1')
);

```

Execute the above code to render the following output.



**Ticks**

Ticks are used to mark some values on the scale. Based on the tick's value you can set the labels on the required position.

### Adding tick collection

Tick collection can be directly added to the scale object. Refer the following code example to add tick collection in a **Linear Gauge** control.

#### HTML

```
<div id="LinearGauge1"></div>
```

#### JAVASCRIPT

```
"use strict";
var scales = [
{
width: 5, majorIntervalValue: 25, minorIntervalValue: 5,
backgroundColor: "White", showCustomLabels: true,
showMarkerPointers: false, showBarPointers: true,
direction: ej.datavisualization.LinearGauge.Directions.Clockwise,
type: "roundedrectangle",
border: { color: "#AEC75F", width: 2 },
// Adding bar pointer collection
barPointers: [{ width: 4, backgroundColor: "Red" }],
// Adding label collection
labels: [{
angle: 90,
distanceFromScale: { x: 0, y: 60 },
unitText: "%"
}],
// Adding tick collection
ticks: [{
type: "majorinterval", width: 2,
color: "#8c8c8c", distanceFromScale: { x: 0, y: 25 }
},
{
type: "minorinterval", width: 1, height: 6,
color: "#8c8c8c", distanceFromScale: { x: 0, y: 25 }
}],
customLabels: [{
value: "Download in Progress", position: { x: 50, y: 20 },
}]
}];
var frame = {
// For setting back ground Image URL
backgroundImageUrl: "Gauge_linear_light1.png"
};
ReactDOM.render(
<EJ.LinearGauge id="lineargauge"
labelColor= "#8c8c8c" frame = {frame} orientation = "Horizontal" scales =
{scales} width = {600} height = {250} value = {31} labelColor = "Black"
enableResize = {true} enableAnimation = {false}>
</EJ.LinearGauge>,
document.getElementById('LinearGauge1')
);
```

Execute the above code to render the following output.



## Tick Customization

### Appearance

- Height and width of the ticks can be applied by using the properties **height** and **width**. You can customize ticks with the properties like angle, color, etc. **angle** attribute is used to display the labels in the specified angles and **color** attribute is used to display the labels in specified color.
- Ticks are two types such as major and minor. The opacity of the labels can be adjusted with the property **opacity**. The opacity values lies between 0 and 1.

### HTML

```
<div id="LinearGauge1"></div>
```

### JAVASCRIPT

```
"use strict";
var scales = [
{
width:5,
backgroundColor: "transparent", type: "roundedrectangle",
border: { color: "Grey", width: 1 },showBarPointers: true,
//Adding label collection
labels: [{ distanceFromScale: { x: -25, y: 0 } }],
//Adding marker pointer collection
markerPointers:[{width:10,length:10, value:60}],
//Adding bar pointer collection
barPointers: [{ width: 5, backgroundColor: "#95C7E0" }],
//Adding ticks collection
ticks: [{
type: "majorinterval",
width: 2,
height:14,
angle:10,
color: "Black",
distanceFromScale: { x: -10, y: 0 },position:"far"
},
{
type: "minorinterval",
width: 1,
height: 10,
opacity:0.5,
color: "Black",
distanceFromScale: { x: -10, y: 0 }, position: "far"
}]
}];
var frame = {
innerWidth: 8,
outerWidth: 10,
// For setting back ground Image URL
backgroundImageUrl: "Gauge_linear_light.png"
};
ReactDOM.render(
<EJ.LinearGauge id="lineargauge"
labelColor= "#8c8c8c" frame = {frame} scales = {scales} enableAnimation =
{false} value = {78}>
```

```
</EJ.LinearGauge>,
document.getElementById('LinearGauge1')
);
```

Execute the above code to render the following output.

![[/js/LinearGauge/Ticksimages/Ticksimg2.png)

### Types

Ticks are two types such as **majorInterval** and **minorInterval**. Major type ticks are for major interval values and minor type ticks are for minor interval values.

### HTML

```
<div id="LinearGauge1"></div>
```

### JAVASCRIPT

```
"use strict";
var scales = [
{
width: 5,
backgroundColor: "transparent", type: "roundedrectangle",
border: { color: "Grey", width: 1 }, showBarPointers: true,
//Adding label collection
labels: [{ distanceFromScale: { x: -25, y: 0 } }],
//Adding marker pointer collection
markerPointers: [{ width: 10, length: 10, value: 60 }],
//Adding bar pointer collection
barPointers: [{ width: 5, backgroundColor: "#95C7E0" }],
//Adding tick collection
ticks: [{
type: "majorinterval", width: 2, height: 14,
color: "Black", position: "far"
},
{
type: "minorinterval",
}],
}];
var frame = {
innerWidth: 8,
outerWidth: 10,
// For setting back ground Image URL
backgroundImageUrl: "Gauge_linear_light.png"
};
ReactDOM.render(
<EJ.LinearGauge id="lineargauge"
labelColor= "#8c8c8c" frame = {frame} scales = {scales} enableAnimation =
{false} value = {78}>
</EJ.LinearGauge>,
document.getElementById('LinearGauge1')
);
```

Execute the above code to render the following output.



![[/js/LinearGauge/Ticksimages/Ticksimg3.png]

### Positioning the ticks

- You can position ticks with the help of two properties such as **distanceFromScale** and **placement**. The property **distanceFromScale** defines the distance between the scale and ticks.
- Placement** property is used to locate the ticks with respect to scale either inside the scale or outside the scale or along the scale. It is an enumerable data type.

### HTML

```
<div id="LinearGauge1"></div>
```

### JAVASCRIPT

```
"use strict";
var scales = [
{
width:5,
backgroundColor: "transparent", type: "roundedrectangle",
border:{ color: "Grey", width: 1 },showBarPointers: true,
//Adding label collection
labels: [{ distanceFromScale: { x: -25, y: 0 } }],
//Adding marker pointer collection
markerPointers:[{width:10,length:10, value:60}],
//Adding bar pointer collection
barPointers: [{ width: 5, backgroundColor: "#95C7E0" }],
ticks: [{
type: "majorinterval",
width: 2,
height:14,
color: "Red",
distanceFromScale: { x: -10, y: 0 },position:"far"
},
{
type: "minorinterval",
width: 1,
height: 10,
opacity:0.5,
color: "Black",
distanceFromScale: { x: -10, y: 0 }, position: "far"
}]
}];
var frame = {
innerWidth: 8,
outerWidth: 10,
// For setting back ground Image URL
backgroundImageUrl: "Gauge_linear_light.png"
};
ReactDOM.render(
<EJ.LinearGauge id="lineargauge"
labelColor= "#8c8c8c"
frame = {frame} scales = {scales} enableAnimation = {false} value = {78}>
</EJ.LinearGauge>,
document.getElementById('LinearGauge1')
```

```
);
```

Execute the above code to render the following output.



## Ranges

Ranges are used to specify or group the scale values. You can describe the values in the pointers using ranges.

### Adding range collection

Range collection can be directly added to the scale object. Refer the following code example to add range collection in a **Linear Gauge** control.

### HTML

```
<div id="LinearGauge1"></div>
```

### JAVASCRIPT

```
"use strict";
var scales= [{
width: 0,
backgroundColor: "#AEC75F",
direction: ej.datavisualization.LinearGauge.Directions.Clockwise,
border: { width: 0, color: "transparent" }, minimum: -20, maximum: 60,
showBarPointers: false, showRanges: true,
//Adding marker pointers collection
markerPointers: [{
width: 3, length: 30, backgroundColor: "#FE5C09", type: "star",
distanceFromScale: 20, placement: "near",
value: 55
}],
//Adding label collection
labels: [{ angle: 90, distanceFromScale: { x: 0, y: 50 } }],
//Adding tick collection
ticks: [{
type: "majorinterval", width: 2,
color: "#8c8c8c", distanceFromScale: { x: 25, y: -1 }
},
{
type: "minorinterval", width: 1, height: 6,
color: "#8c8c8c", distanceFromScale: { x: 25, y: -1 }
}],
//Adding range collection
ranges: [{
startValue: -20, endValue: 60,
startWidth: 0, endWidth: 20, backgroundColor: "#FEBE48",
placement: "near", distanceFromScale: 20
}]
}];
ReactDOM.render(
<EJ.LinearGauge id="lineargauge"
enableAnimation={false} width={600} height={150} orientation="Horizontal"
labelColor="Black" enableResize= {true}
scales={scales}
```

```
>
</EJ.LinearGauge>,
document.getElementById('LinearGauge1')
);
```

Execute the above code to render the following output.

![[/js/LinearGauge/Rangesimages/Rangesimg1.png)

## Range Customization

### Appearance

The major attributes for ranges are **startValue** and **endValue**. The **startValue** defines the start position of the range and **endValue** defines the end position of the range. The **startWidth** and **endWidth** are used to specify the range width at the starting and ending position of the ranges.

### HTML

```
<div id="LinearGauge1"></div>
```

### JAVASCRIPT

```
"use strict";
var scales= [{
position: { x: 50, y: 50 },
width: 4, backgroundColor: "#10ADF5", border: {
color:
"transparent", width: 0
}, showRanges: true, showScaleBar: true,
showMarkerPointers: false, length: 310,
//Adding label collection
labels: [{
font: {
size: "11px", fontFamily: "Segoe UI", fontStyle:
"bold"
}, distanceFromScale: { x: -12 }
}],
//Adding ticks collection
ticks: [{ type: "majorinterval", width: 1, color: "#8c8c8c" }],
//Adding ranges collection
ranges: [
{
endValue: 50, // For setting range end value
startValue: 0, //For setting range start value
startWidth: 8,
endWidth: 8,
backgroundColor: "#F6B53F",
distanceFromScale: 5
},
{
endValue: 100,
startValue: 70,
startWidth: 8,
endWidth: 8,
distanceFromScale: 5,
backgroundColor: "#E94649" //For setting range background color
```

```

    ]]
  }];
  ReactDOM.render(
    <EJ.LinearGauge id="lineargauge"
    labelColor= "#8c8c8c" width={500}
    scales={scales}
    >
    </EJ.LinearGauge>,
    document.getElementById('LinearGauge1')
  );

```

Execute the above code to render the following output.



### Colors and Border

- You can customize the ranges to improve the appearance of the **Gauge**. The range border is modified with the object called **border**. It has two border property such as **color** and **width** which are used to customize the border color of the ranges and border width of the ranges.
- You can set the background color to improve the look and feel of the **Linear Gauge**. For customizing the background color of the ranges, **backgroundColor** is used. You can add the gradient effects to the ranges by using **gradient** object.

### HTML

```
<div id="LinearGauge1"></div>
```

### JAVASCRIPT

```

"use strict";
var scales=[{
  position: { x: 50, y: 50 },
  //Adding label collection
  labels: [{
    font: {
      size: "11px", fontFamily: "Segoe UI", fontStyle:
      "bold"
    }, distanceFromScale: { x: -12 }
  }],
  //Adding ticks collection
  ticks: [{ type: "majorinterval", width: 1, color: "#8c8c8c" }],
  width: 4, backgroundColor: "transparent", border: {
    color:
    "transparent", width: 0
  }, showRanges: true, showScaleBar: true,
  showMarkerPointers: false, length: 310,
  //Adding ranges collection
  ranges: [{
    endValue: 50, // For setting range end value
    startValue: 0, // For setting range start value
    backgroundColor: "#F6B53F",
    border: { color: "black" },
    startWidth: 3,

```

```

endWidth: 18,
distanceFromScale: 10
}, {
endValue: 100,
startValue: 70,
backgroundColor: "#E94649",
border: { color: "black" },
startWidth: 18,
endWidth: 3,
distanceFromScale: 10
}]
}];
ReactDOM.render(
<EJ.LinearGauge id="lineargauge"
labelColor= "#8c8c8c" width={500}
scales={scales}
>
</EJ.LinearGauge>,
document.getElementById('LinearGauge1')
);

```

Execute the above code to render the following output.



### Positioning the ranges

- You can position ranges using two properties such as **distanceFromScale** and **placement**. The **distanceFromScale** property defines the distance between the scale and range.
- Placement** property is used to locate the pointer with respect to scale either inside the scale or outside the scale or along the scale. It is an enumerable data type.

### HTML

```
<div id="LinearGauge1"></div>
```

### JAVASCRIPT

```

"use strict";
var scales=[{
position: { x: 50, y: 50 },
//Adding label collection
labels: [{
font: {
size: "11px", fontFamily: "Segoe UI", fontStyle:
"bold"
}, distanceFromScale: { x: -12 }
}],
//Adding ticks collection
ticks: [{ type: "majorinterval", width: 1, color: "#8c8c8c" }],
width: 4, backgroundColor: "transparent", border: {
color:
"transparent", width: 0
}, showRanges: true, showScaleBar: true,

```

```

showMarkerPointers: false, length: 310,
//Adding ranges collection
ranges: [{
  endValue: 50, // For setting range end value
  startValue: 0, // For setting range start value
  backgroundColor: "#F6B53F",
  border: { color: "black" },
  startWidth: 3,
  endWidth: 18,
  distanceFromScale: -30,
  placement: "near"
}, {
  endValue: 100,
  startValue: 70,
  backgroundColor: "#E94649",
  border: { color: "black" },
  startWidth: 18,
  endWidth: 3,
  distanceFromScale: -30,
  placement: "near"
}]
}];
ReactDOM.render(
  <EJ.LinearGauge id="lineargauge"
  labelColor= "#8c8c8c" width={500}
  scales={scales}
  >
  </EJ.LinearGauge>,
  document.getElementById('LinearGauge1')
);

```

Execute the above code to render the following output.



### Multiple Ranges

You can set multiple ranges by adding an array of range objects. Refer the following code example for multiple range functionality.

#### HTML

```
<div id="LinearGauge1"></div>
```

#### JAVASCRIPT

```

"use strict";
//Adding scale collection
var scales= [{
  width: 0,
  backgroundColor: "#AEC75F",
  direction: ej.datavisualization.LinearGauge.Directions.Clockwise,
  border: { width: 0, color: "transparent" }, minimum: -20, maximum: 60,
  showBarPointers: false, showRanges: true,
  //Adding marker pointer collection
  markerPointers: [{

```

```

width: 3, length: 30, backgroundColor: "#FE5C09", type: "star",
distanceFromScale: 20, placement: "near",
value: 55
}],
//Adding label collection
labels: [{ angle: 90, distanceFromScale: { x: 0, y: 50 } }],
//Adding tick collection
ticks: [{
type: "majorinterval", width: 2,
color: "#8c8c8c", distanceFromScale: { x: 20, y: -1 }
},
{
type: "minorinterval", width: 1, height: 6,
color: "#8c8c8c", distanceFromScale: { x: 20, y: -1 }
}],
//Adding ranges collection
ranges: [
//Adding range 1
{
placement: "near",
distanceFromScale: 20,
startValue: -20, endValue: 0, startWidth: 5, endWidth: 10,
backgroundColor: "#2788B1", border: { color: "#2788B1" }
},
//Adding range 2
{
placement: "near",
distanceFromScale: 20,
startValue: 0, endValue: 20, startWidth: 10, endWidth: 15,
backgroundColor: "#A5BA28", border: { color: "#A5BA28" }
},
//Adding range 3
{
placement: "near",
distanceFromScale: 20,
startValue: 20, endValue: 40, startWidth: 15, endWidth: 20,
backgroundColor: "#FEBE48", border: { color: "#FEBE48" }
},
//Adding range 4
{
placement: "near",
distanceFromScale: 20,
startValue: 40, endValue: 60, startWidth: 20, endWidth: 25,
backgroundColor: "Red", border: { color: "Red" }
}}
];
ReactDOM.render(
<EJ.LinearGauge id="lineargauge"
enableAnimation={false} width={600} height={150} orientation=
"Horizontal" labelColor= "#8c8c8c"
scales={scales} labelColor= "Black" enableResize= {true}>
</EJ.LinearGauge>,
document.getElementById('lineargauge-default')
);

```

Execute the above code to render the following output.



## Custom labels

Custom labels are the text that can paste in any location of the **Linear Gauge**. It is used to define the purpose of the gauge.

### Adding Custom label collection

Custom labels collection can be directly added to the scale object. Refer the following code to add custom labels collection in a **Linear Gauge** control.

#### HTML

```
<div id="LinearGauge1"></div>
```

#### JAVASCRIPT

```
"use strict";
var frame= {
  innerWidth: 8,
  outerWidth: 10,
  backgroundImageUrl: "../images/gauge/Gauge_linear_light.png"
}
var scales=[{
  backgroundColor: "transparent",
  border: { color: "transparent", width: 0 },
  showMarkerPointers: false, showBarPointers: true,
  showCustomLabels: true,
  //Adding bar pointer collection
  barPointers: [{
    width: 10, backgroundColor: "#8BABFF",
    value: 91, placement: "near", distanceFromScale: 30
  }],
  //Adding tick collection
  ticks: [{
    type: "majorinterval", width: 2,
    color: "#8c8c8c", distanceFromScale: { x: 7, y: 0 }
  },
  {
    type: "minorinterval", width: 1, height: 6,
    color: "#8c8c8c", distanceFromScale: { x: 7, y: 0 }
  }],
  //Adding custom label collection
  customLabels: [{
    value: "Mathematics Mark", position: { x: 55, y: 97 }
  }]
}]
ReactDOM.render(
  <EJ.LinearGauge id="lineargauge"
  enableAnimation={false} width={500} height= {200} orientation="Horizontal"
  labelColor="Grey"
  //Adding frame object
  frame={frame}
  //Adding scale collection
  scales={scales}
  >
  </EJ.LinearGauge>,
```



```
document.getElementById('LinearGauge1')
);
```

Execute the above code to render the following output.

![[/js/LinearGauge/Custom-labelsimages/Custom-labelsimg1.png)

## Basic Customization

### Appearance

- You can customize custom labels using the properties like **textAngle**, **color** and **font**. The API **textAngle** is used to display the custom labels in the specified angles and **color** attribute is used to display the custom labels in specified color. You can use **value** attribute to set the text value in the custom labels.
- To display the custom labels, set **showCustomLabels** as 'true'. Font option is also available on the custom labels. The basic three properties of fonts such as size, family and style can be achieved by **size**, **fontStyle** and **fontFamily**. You can adjust the opacity of the label with the property **opacity** and the value of opacity lies between 0 and 1.

## HTML

```
<div id="LinearGauge1"></div>
```

## JAVASCRIPT

```
"use strict";
var scales = [{
  backgroundColor: "transparent",
  border: { color: "transparent", width: 0 },
  showMarkerPointers: false, showBarPointers: true,
  showCustomLabels: true,
  //Adding bar pointer collection
  barPointers: [
    {
      width: 10, backgroundColor: "#8BABFF",
      value: 91, placement: "near", distanceFromScale: 30
    }
  ],
  //Adding tick collection
  ticks: [{
    type: "majorinterval", width: 2,
    color: "#8c8c8c", distanceFromScale: { x: 7, y: 0 }
  },
  {
    type: "minorinterval", width: 1, height: 6,
    color: "#8c8c8c", distanceFromScale: { x: 7, y: 0 }
  }
  ],
  //Adding custom labels
  customLabels: [{
    position: { x: 55, y: 87 },
    value: "Mathematics Mark",
    color: "Red",
    textAngle: 30,
    opacity: 0.5
  }
  ]
  }
];
```

```

    ]]
  }];
  var frame = {
    innerWidth: 8,
    outerWidth: 10,
    // For setting back ground Image URL
    backgroundImageUrl: "Gauge_linear_light.png"
  };
  ReactDOM.render(
    <EJ.LinearGauge id="lineargauge" frame = {frame} scales = {scales}
    enableAnimation = {false} value = {78} height = {500} width = {200}
    labelColor = "Grey">
    </EJ.LinearGauge>,
    document.getElementById('LinearGauge1')
  );

```

Execute the above code to render the following output.

![[/js/LinearGauge/Custom-labelsimages/Custom-labelsimg2.png)

### Locating the CustomLabels

To set the location of the custom label in **Linear Gauge**, **position** property is used. You can position the custom labels in horizontal and vertical axis using **X** and **Y** axis respectively.

### HTML

```
<div id="LinearGauge1"></div>
```

### JAVASCRIPT

```

"use strict";
var scales = [{
  backgroundColor: "transparent",
  border: { color: "transparent", width: 0 },
  showMarkerPointers: false, showBarPointers: true,
  showCustomLabels: true,
  //Adding bar pointer collection
  barPointers: [
    {
      width: 10, backgroundColor: "#8BABFF",
      value: 91, placement: "near", distanceFromScale: 30
    }
  ],
  //Adding ticks collection
  ticks: [{
    type: "majorinterval", width: 2,
    color: "#8c8c8c", distanceFromScale: { x: 7, y: 0 }
  },
  {
    type: "minorinterval", width: 1, height: 6,
    color: "#8c8c8c", distanceFromScale: { x: 7, y: 0 }
  }
  ],
  //Adding custom label collection
  customLabels: [
    { value: "Mathematics Mark", position: { x: 55, y: 87 } }
  ]
}];

```

```

var frame = {
  innerWidth: 8,
  outerWidth: 10,
  // For setting back ground Image URL
  backgroundImageUrl: "Gauge_linear_light.png"
};
ReactDOM.render(
  <EJ.LinearGauge id="lineargauge" frame = {frame} scales = {scales}
  enableAnimation = {false} value = {78} height = {500} width = {200}
  labelColor = "Grey" >
  </EJ.LinearGauge>,
  document.getElementById('LinearGauge1')
);

```

Execute the above code to render the following output.

![[/js/LinearGauge/Custom-labelsimages/Custom-labelsimg3.png]

### Multiple Custom Labels

You can set multiple custom labels in a single **Linear Gauge** by adding an array of custom label objects. Refer the following code example for multiple custom label functionality.

#### HTML

```
<div id="LinearGauge1"></div>
```

#### JAVASCRIPT

```

"use strict";
var scales = [{
  backgroundColor: "transparent",
  border: { color: "transparent", width: 0 },
  showMarkerPointers: false, showBarPointers: true,
  showCustomLabels: true,
  //Adding bar pointer collection
  barPointers: [
    {
      width: 10, backgroundColor: "#8BABFF",
      value: 91, placement: "near", distanceFromScale: 30
    }
  ],
  //Adding ticks collection
  ticks: [{
    type: "majorinterval", width: 2,
    color: "#8c8c8c", distanceFromScale: { x: 7, y: 0 }
  },
  {
    type: "minorinterval", width: 1, height: 6,
    color: "#8c8c8c", distanceFromScale: { x: 7, y: 0 }
  }
  ],
  //Adding custom label collection
  customLabels: [
    {
      value: "Mathematics Mark", position: { x: 55, y: 87 },
      color: "Red"
    }
  ],

```

```

{
  value: "Marks in %", position: { x: 15, y: 57 },
  color: "Red", textAngle: 90
}]
}];
var frame = {
  innerWidth: 8,
  outerWidth: 10,
  // For setting back ground Image URL
  backgroundImageUrl: "Gauge_linear_light.png"
};
ReactDOM.render(
  <EJ.LinearGauge id="lineargauge" frame = {frame} scales = {scales}
  enableAnimation = {false} value = {78} height = {500} width = {200}
  labelColor = "Grey">
  </EJ.LinearGauge>,
  document.getElementById('LinearGauge1')
);

```

Execute the above code to render the following output.

![[/js/LinearGauge/Custom-labelsimages/Custom-labelsimg4.png)

## Indicators

Indicators simply indicates the current status of the pointer. Indicators are in several formats such as in shape format, textual format and image format.

### Setting Dimension

- You can enable indicators by setting **showIndicators** to 'true' in scale collection. The **height** and **width** property for the indicators are used to specify the area allocated to the indicator for the width and height respectively.
- You can use the position collection to position the indicators along **X** and **Y** axis. **X** specifies horizontal position in indicators whereas **Y** specifies vertical position in indicators. Indicators are of several types such as, dimensions like circle, rectangle, rounded rectangle, text and image. By using the **type** property it can be applied. For image type **imageUrl** property is used.

## HTML

```
<div id="LinearGauge1"></div>
```

## JAVASCRIPT

```

"use strict";
var scales = [{
  width: 0,
  border: { color: "transparent", width: 0 },
  minimum: 0,
  maximum: 300,
  minorIntervalValue: 5,
  majorIntervalValue: 30,
  showBarPointers: false,
  showIndicators: true,
  //Adding marker pointer collection

```

```

markerPointers: [{
  width: 10, length: 10,
  backgroundColor: "Grey", distanceFromScale: 12
}],
//Adding ticks collection
ticks: [{
  type: "majorinterval", width: 2,
  color: "#8c8c8c", distanceFromScale: { x: 7, y: 0 }
},
{
  type: "minorinterval", width: 1, height: 6,
  color: "#8c8c8c", distanceFromScale: { x: 7, y: 0 }
}],
//Adding indicator collection
indicators: [{
  height: 10,
  width: 10,
  type: "circle",
  position: { x: 50, y: 100 }
}]
}];
var frame = {
  innerWidth: 8,
  outerWidth: 10,
  // For setting back ground Image URL
  backgroundImageUrl: "Gauge_linear_light.png"
};
ReactDOM.render(
  <EJ.LinearGauge id="lineargauge" frame = {frame} scales = {scales}
  enableAnimation = {false}
  value = {78}>
</EJ.LinearGauge>,
  document.getElementById('LinearGauge1')
);

```

Execute the above code to render the following output.



### State Ranges

State ranges are used to specify the indicator behavior in the certain region. **startValue** and **endValue** are used to set the range bound for the pointer. Whenever the pointer crosses the specified region, the indicator attributes are applied for the ranges.

### HTML

```
<div id="LinearGauge1"></div>
```

### JAVASCRIPT

```

"use strict";
var scales = [{
  width: 0,
  border: { color: "transparent", width: 0 },
  minimum: 0,
  maximum: 300,

```

```

minorIntervalValue: 5,
majorIntervalValue: 30,
showRanges: true,
showBarPointers: false,
showIndicators: true,
//Adding marker pointer collection
markerPointers: [{
width: 10, length: 10, backgroundColor: "Grey",
distanceFromScale: 12
}],
//Adding tick collection
ticks: [{
type: "majorinterval", width: 2,
color: "#8c8c8c", distanceFromScale: { x: 7, y: 0 }
},
{
type: "minorinterval", width: 1, height: 6,
color: "#8c8c8c", distanceFromScale: { x: 7, y: 0 }
}],
//Adding range collection
ranges: [{
startWidth: 5, endWidth: 5, startValue: 0, endValue: 200,
backgroundColor: "#94C361", border: {
color: "#94C361", width: 1
}
},
{
startWidth: 5, endWidth: 5, startValue: 200, endValue: 250,
backgroundColor: "#F9CF67", border: {
color: "#F9CF67", width: 1
}
},
{
startWidth: 5, endWidth: 5, startValue: 250, endValue: 300,
backgroundColor: "#F89B83", border: {
color: "#F89B83", width: 1
}
}],
//Adding indicator collection
indicators: [{
height: 10, width: 10, type: "circle", position: { x: 50, y: 100 } },
//Adding State ranges collection
stateRanges: [
{
backgroundColor: "#02A258", endValue: 200,
startValue: 0, borderColor: "#02A258"
},
{
backgroundColor: "Grey", endValue: 300,
startValue: 200, borderColor: "Grey"
}
]
}];
var frame = {
innerWidth: 8,
outerWidth: 10,

```

```
// For setting back ground Image URL
backgroundImageUrl: "Gauge_linear_light.png"
};
ReactDOM.render(
  <EJ.LinearGauge id="lineargauge" frame = {frame} scales = {scales}
  enableAnimation = {false} value = {78}>
</EJ.LinearGauge>,
  document.getElementById('LinearGauge1')
);
```

Execute the above code to render the following output.



Linear Gauge with indicator state ranges

### Color and Appearance

The **backgroundColor** and **borderColor** sets the appearance behavior for the indicators. You can apply this only if it lies within the state ranges. Otherwise default behavior will be applied.

### HTML

```
<div id="LinearGauge1"></div>
```

### JAVASCRIPT

```
"use strict";
var scales = [{
  width: 0,
  border: { color: "transparent", width: 0 },
  minimum: 0,
  maximum: 300,
  minorIntervalValue: 5,
  majorIntervalValue: 30,
  showBarPointers: false,
  showIndicators: true,
  //Adding marker pointer collection
  markerPointers: [{
    width: 10, length: 10,
    backgroundColor: "Grey", distanceFromScale: 12
  }],
  //Adding ticks collection
  ticks: [{
    type: "majorinterval", width: 2,
    color: "#8c8c8c", distanceFromScale: { x: 7, y: 0 }
  },
  {
    type: "minorinterval", width: 1, height: 6,
    color: "#8c8c8c", distanceFromScale: { x: 7, y: 0 }
  }],
  //Adding indicator collection
  indicators: [{
    height: 10, width: 10, type: "circle", position: { x: 50, y: 100 },
    stateRanges: [{
      backgroundColor: "#91B64E", endValue: 300,
      startValue: 0, borderColor: "#91B64E"
    }
  ]
}];
```

```

    ]]
  ]];
  var frame = {
    innerWidth: 8,
    outerWidth: 10,
    // For setting back ground Image URL
    backgroundImageUrl: "Gauge_linear_light.png"
  };
  ReactDOM.render(
    <EJ.LinearGauge id="lineargauge" frame = {frame} scales = {scales}
    enableAnimation = {false} value = {78}>
    </EJ.LinearGauge>,
    document.getElementById('LinearGauge1')
  );

```

Execute the above code to render the following output.



### Font options

The basic font options available for the textual type indicators in the **Linear Gauge** such as Size, font style and font family are achieved by the properties **size**, **fontStyle** and **fontFamily**.

### HTML

```
<div id="LinearGauge1"></div>
```

### JAVASCRIPT

```

"use strict";
var scales = [{
  width: 0,
  border: { color: "transparent", width: 0 },
  minimum: 0,
  maximum: 300,
  minorIntervalValue: 5,
  majorIntervalValue: 30,
  showRanges: true,
  showBarPointers: false,
  showIndicators: true,
  //Adding marker pointer collection
  markerPointers: [{
    width: 10, length: 10,
    backgroundColor: "Grey", distanceFromScale: 12
  }],
  //Adding tick collection
  ticks: [{
    type: "majorinterval", width: 2,
    color: "#8c8c8c", distanceFromScale: { x: 7, y: 0 }
  },
  {
    type: "minorinterval", width: 1, height: 6,
    color: "#8c8c8c", distanceFromScale: { x: 7, y: 0 }
  }],
  //Adding range collection

```



```

ranges: [{
  startWidth: 5, endWidth: 5, startValue: 0, endValue: 200,
  backgroundColor: "#94C361", border: {
    color: "#94C361", width: 1
  }
},
{
  startWidth: 5, endWidth: 5, startValue: 200, endValue: 250,
  backgroundColor: "#F9CF67", border: {
    color: "#F9CF67", width: 1
  }
},
{
  startWidth: 5, endWidth: 5, startValue: 250, endValue: 300,
  backgroundColor: "#F89B83", border: {
    color: "#F89B83", width: 1
  }
}
],
//Adding indicator collection
indicators: [{
  type: "text", textLocation: { x: 50, y: 100 },
  //Adding font option
  font: { size: "12px", fontFamily: "Arial", fontType: "Bold" },
  //Adding state ranges collection
  stateRanges: [{
    startValue: 0, endValue: 200,
    text: "Safe", textColor: "#94C361"
  },
  {
    startValue: 200, endValue: 250,
    text: "Caution", textColor: "#F9CF67"
  },
  {
    startValue: 250, endValue: 300,
    text: "Danger", textColor: "#F89B83"
  }
]
}];
var frame = {
  innerWidth: 8,
  outerWidth: 10,
  // For setting back ground Image URL
  backgroundImageUrl: "Gauge_linear_light.png"
};
ReactDOM.render(
  <EJ.LinearGauge id="lineargauge" frame = {frame} scales = {scales}
  enableAnimation = {false} value = {78}>
  </EJ.LinearGauge>,
  document.getElementById('LinearGauge1')
);

```

Execute the above code to render the following output.



## Multiple Indicator

You can set multiple indicators in a single **Linear Gauge** by adding an array of indicator objects. Refer the following code example for multiple indicator functionality.

### HTML

```
<div id="LinearGauge1"></div>
```

### JAVASCRIPT

```
"use strict";
var scales = [{
  width: 0,
  border: { color: "transparent", width: 0 },
  minimum: 0,
  maximum: 300,
  minorIntervalValue: 5,
  majorIntervalValue: 30,
  showRanges: true,
  showBarPointers: false,
  showIndicators: true,
  //Adding marker pointer collection
  markerPointers: [{
    width: 10, length: 10, backgroundColor: "Grey",
    distanceFromScale: 12
  }],
  //Adding ticks collection
  ticks: [{
    type: "majorinterval", width: 2,
    color: "#8c8c8c", distanceFromScale: { x: 7, y: 0 }
  },
  {
    type: "minorinterval", width: 1, height: 6,
    color: "#8c8c8c", distanceFromScale: { x: 7, y: 0 }
  }],
  //Adding ranges collection
  ranges: [
    {
      startWidth: 5, endWidth: 5, startValue: 0, endValue: 200,
      backgroundColor: "#94C361", border: {
        color: "#94C361", width: 1
      }
    },
    {
      startWidth: 5, endWidth: 5, startValue: 200, endValue: 250,
      backgroundColor: "#F9CF67", border: {
        color: "#F9CF67", width: 1
      }
    },
    {
      startWidth: 5, endWidth: 5, startValue: 250, endValue: 300,
      backgroundColor: "#F89B83", border: {
        color: "#F89B83", width: 1
      }
    }
  ],
}
```

```
//Adding indicator collection
indicators: [
  //Adding indicator 1
  {
    height: 10, width: 10, type: "circle", position: { x: 30, y: 100 },
    //Adding state ranges collection
    stateRanges: [{
      backgroundColor: "#02A258", endValue: 200,
      startValue: 0, borderColor: "#02A258"
    },
    {
      backgroundColor: "Grey", endValue: 300,
      startValue: 200, borderColor: "Grey"
    }]
  },
  //Adding indicator 2
  {
    height: 10, width: 10, type: "circle", position: { x: 70, y: 100 },
    stateRanges: [{
      backgroundColor: "Grey", endValue: 200,
      startValue: 0, borderColor: "Grey"
    },
    {
      backgroundColor: "Red", endValue: 300,
      startValue: 200, borderColor: "Red"
    }]
  }
];
var frame = {
  innerWidth: 8,
  outerWidth: 10,
  // For setting back ground Image URL
  backgroundImageUrl: "Gauge_linear_light.png"
};
ReactDOM.render(
  <EJ.LinearGauge id="lineargauge" frame = {frame} scales = {scales}
  enableAnimation = {false} value = {78}>
</EJ.LinearGauge>,
  document.getElementById('LinearGauge1')
);
```

Execute the above code to render the following output.



## Exporting

**Linear Gauge** has an exporting feature that converts **Gauge** control into image format and then export in client side. The method API **exportImage** is used to export the **LinearGauge**. It has two arguments such as **file name** and **file format** to specify the file name and file formats. For exporting refer the following code example.

## HTML

```
<div id="LinearGauge1"></div>
<button id="btnSubmit">Export</button>
<div id=" fileName ">FileName </div>
```

```
<div id=" fileFormat ">FileFormat </div>
<select id="fileFormat">
<option value="JPEG">JPEG</option>
<option value="PNG">PNG</option>
</select>
```

## JAVASCRIPT

```
"use strict";
//Adding scale collection
var scales= [{
width: 4, border: { color: "transparent", width: 0 }, showBarPointers:
false, showRanges: true, length: 310,
position: { x: 52, y: 50 }, markerPointers: [{
value: 50, length: 10, width: 10, backgroundColor: "#4D4D4D", border: {
color: "#4D4D4D" }
}],
//Adding label collection
labels: [{ font: { size: "11px", fontFamily: "Segoe UI", fontStyle: "bold"
}, distanceFromScale: { x: -13 } }],
//Adding tick collection
ticks: [{ type: "majorinterval", width: 1, color: "#8c8c8c" }],
//Adding range collection
ranges: [{
endValue: 60,
startValue: 0,
backgroundColor: "#F6B53F",
border: { color: "#F6B53F" }, startWidth: 4, endWidth: 4
}, {
endValue: 100,
startValue: 60,
backgroundColor: "#E94649",
border: { color: "#E94649" }, startWidth: 4, endWidth: 4
}]
}];
ReactDOM.render(
<EJ.LinearGauge id="default"
width={450}
labelColor= "#8c8c8c"
load="loadGaugeTheme"
scales={scale}
>
</EJ.LinearGauge>,
document.getElementById('lineargauge-default')
);
function buttonclickevent() {
var FileName = $("#fileName").val();
var FileFormat = $("#fileFormat").ejDropDownList("option", "value");
$("#default").ejLinearGauge("exportImage", FileName, FileFormat);
}
```

Execute the above code to render the following output.

![[/js/LinearGauge/Exportingimages/Exportingimg1.png]

## Methods

### *destroy()*

destroy the linear gauge all events bound using this.\_on will be unbind automatically and bring the control to pre-init state.

### **HTML**

```
<div id="linear"></div>
```

### **JAVASCRIPT**

```
ReactDOM.render(  
  <EJ.LinearGauge id="default"></EJ.LinearGauge>,  
  document.getElementById('linear')  
);  
function LinearGaugeMethod() {  
  var linearObj = $("#default").data("ejLinearGauge");  
  linearObj.destroy();  
};
```

### *exportImage()*

To export Image

### **HTML**

```
<div id="linear"></div>
```

### **JAVASCRIPT**

```
ReactDOM.render(  
  <EJ.LinearGauge id="default"></EJ.LinearGauge>,  
  document.getElementById('linear')  
);  
function LinearGaugeMethod() {  
  var linearObj = $("#default").data("ejLinearGauge");  
  linearObj.exportImage();  
};
```

### *getBarDistanceFromScale()*

To get Bar Distance From Scale in number

### **HTML**

```
<div id="linear"></div>
```

### **JAVASCRIPT**

```
ReactDOM.render(  
  <EJ.LinearGauge id="default"></EJ.LinearGauge>,  
  document.getElementById('linear')  
);  
function LinearGaugeMethod() {  
  var linearObj = $("#default").data("ejLinearGauge");  
  linearObj.getBarDistanceFromScale();  
};
```

```
};
```

### *getBarPointerValue()*

To get Bar Pointer Value in number

#### **HTML**

```
<div id="linear"></div>
```

#### **JAVASCRIPT**

```
ReactDOM.render(  
  <EJ.LinearGauge id="default"></EJ.LinearGauge>,  
  document.getElementById('linear')  
)  
;  
function LinearGaugeMethod() {  
  var linearObj = $("#default").data("ejLinearGauge");  
  linearObj.getBarPointerValue();  
};
```

### *getBarWidth()*

To get Bar Width in number

#### **HTML**

```
<div id="linear"></div>
```

#### **JAVASCRIPT**

```
ReactDOM.render(  
  <EJ.LinearGauge id="default"></EJ.LinearGauge>,  
  document.getElementById('linear')  
)  
;  
function LinearGaugeMethod() {  
  var linearObj = $("#default").data("ejLinearGauge");  
  linearObj.getBarWidth();  
};
```

### *getCustomLabelAngle()*

To get CustomLabel Angle in number

#### **HTML**

```
<div id="linear"></div>
```

#### **JAVASCRIPT**

```
ReactDOM.render(  
  <EJ.LinearGauge id="default"></EJ.LinearGauge>,  
  document.getElementById('linear')  
)  
;  
function LinearGaugeMethod() {  
  var linearObj = $("#default").data("ejLinearGauge");  
  linearObj.getCustomLabelAngle();  
};
```

```
};
```

### [\*getCustomLabelValue\(\)\*](#)

To get CustomLabel Value in string

#### **HTML**

```
<div id="linear"></div>
```

#### **JAVASCRIPT**

```
ReactDOM.render(  
  <EJ.LinearGauge id="default"></EJ.LinearGauge>,  
  document.getElementById('linear')  
)  
;  
function LinearGaugeMethod() {  
  var linearObj = $("#default").data("ejLinearGauge");  
  linearObj.getCustomLabelValue();  
}
```

### [\*getLabelAngle\(\)\*](#)

To get Label Angle in number

#### **HTML**

```
<div id="linear"></div>
```

#### **JAVASCRIPT**

```
ReactDOM.render(  
  <EJ.LinearGauge id="default"></EJ.LinearGauge>,  
  document.getElementById('linear')  
)  
;  
function LinearGaugeMethod() {  
  var linearObj = $("#default").data("ejLinearGauge");  
  linearObj.getLabelAngle();  
}
```

### [\*getLabelPlacement\(\)\*](#)

To get LabelPlacement in number

#### **HTML**

```
<div id="linear"></div>
```

#### **JAVASCRIPT**

```
ReactDOM.render(  
  <EJ.LinearGauge id="default"></EJ.LinearGauge>,  
  document.getElementById('linear')  
)  
;  
function LinearGaugeMethod() {  
  var linearObj = $("#default").data("ejLinearGauge");  
  linearObj.getLabelPlacement();  
}
```

```
};
```

### *getLabelStyle()*

To get LabelStyle in number

#### **HTML**

```
<div id="linear"></div>
```

#### **JAVASCRIPT**

```
ReactDOM.render(  
  <EJ.LinearGauge id="default"></EJ.LinearGauge>,  
  document.getElementById('linear')  
)  
;  
function LinearGaugeMethod() {  
  var linearObj = $("#default").data("ejLinearGauge");  
  linearObj.getLabelStyle();  
};
```

### *getLabelXDistanceFromScale()*

To get Label XDistance From Scale in number

#### **HTML**

```
<div id="linear"></div>
```

#### **JAVASCRIPT**

```
ReactDOM.render(  
  <EJ.LinearGauge id="default"></EJ.LinearGauge>,  
  document.getElementById('linear')  
)  
;  
function LinearGaugeMethod() {  
  var linearObj = $("#default").data("ejLinearGauge");  
  linearObj.getLabelXDistanceFromScale();  
};
```

### *getLabelYDistanceFromScale()*

To get PointerValue in number

#### **HTML**

```
<div id="linear"></div>
```

#### **JAVASCRIPT**

```
ReactDOM.render(  
  <EJ.LinearGauge id="default"></EJ.LinearGauge>,  
  document.getElementById('linear')  
)  
;  
function LinearGaugeMethod() {  
  var linearObj = $("#default").data("ejLinearGauge");  
  linearObj.getLabelYDistanceFromScale();  
};
```



```
};
```

### [getMajorIntervalValue\(\)](#)

To get Major Interval Value in number

#### **HTML**

```
<div id="linear"></div>
```

#### **JAVASCRIPT**

```
ReactDOM.render(  
  <EJ.LinearGauge id="default"></EJ.LinearGauge>,  
  document.getElementById('linear')  
)  
;  
function LinearGaugeMethod() {  
  var linearObj = $("#default").data("ejLinearGauge");  
  linearObj.getMajorIntervalValue();  
};
```

### [getMarkerStyle\(\)](#)

To get MarkerStyle in number

#### **HTML**

```
<div id="linear"></div>
```

#### **JAVASCRIPT**

```
ReactDOM.render(  
  <EJ.LinearGauge id="default"></EJ.LinearGauge>,  
  document.getElementById('linear')  
)  
;  
function LinearGaugeMethod() {  
  var linearObj = $("#default").data("ejLinearGauge");  
  linearObj.getMarkerStyle();  
};
```

### [getMaximumValue\(\)](#)

To get Maximum Value in number

#### **HTML**

```
<div id="linear"></div>
```

#### **JAVASCRIPT**

```
ReactDOM.render(  
  <EJ.LinearGauge id="default"></EJ.LinearGauge>,  
  document.getElementById('linear')  
)  
;  
function LinearGaugeMethod() {  
  var linearObj = $("#default").data("ejLinearGauge");  
  linearObj.getMaximumValue();  
};
```

```
};
```

### *getMinimumValue()*

To get PointerValue in number

#### **HTML**

```
<div id="linear"></div>
```

#### **JAVASCRIPT**

```
ReactDOM.render(  
  <EJ.LinearGauge id="default"></EJ.LinearGauge>,  
  document.getElementById('linear')  
)  
;  
function LinearGaugeMethod() {  
  var linearObj = $("#default").data("ejLinearGauge");  
  linearObj.getMinimumValue();  
};
```

### *getMinorIntervalValue()*

To get Minor Interval Value in number

#### **HTML**

```
<div id="linear"></div>
```

#### **JAVASCRIPT**

```
ReactDOM.render(  
  <EJ.LinearGauge id="default"></EJ.LinearGauge>,  
  document.getElementById('linear')  
)  
;  
function LinearGaugeMethod() {  
  var linearObj = $("#default").data("ejLinearGauge");  
  linearObj.getMinorIntervalValue();  
};
```

### *getPointerDistanceFromScale()*

To get Pointer Distance From Scale in number

#### **HTML**

```
<div id="linear"></div>
```

#### **JAVASCRIPT**

```
ReactDOM.render(  
  <EJ.LinearGauge id="default"></EJ.LinearGauge>,  
  document.getElementById('linear')  
)  
;  
function LinearGaugeMethod() {  
  var linearObj = $("#default").data("ejLinearGauge");  
  linearObj.getPointerDistanceFromScale();  
};
```

```
};
```

### [getPointerHeight\(\)](#)

To get PointerHeight in number

#### HTML

```
<div id="linear"></div>
```

#### JAVASCRIPT

```
ReactDOM.render(  
  <EJ.LinearGauge id="default"></EJ.LinearGauge>,  
  document.getElementById('linear')  
)  
;  
function LinearGaugeMethod() {  
  var linearObj = $("#default").data("ejLinearGauge");  
  linearObj.getPointerHeight();  
}
```

### [getPointerPlacement\(\)](#)

To get Pointer Placement in String

#### HTML

```
<div id="linear"></div>
```

#### JAVASCRIPT

```
ReactDOM.render(  
  <EJ.LinearGauge id="default"></EJ.LinearGauge>,  
  document.getElementById('linear')  
)  
;  
function LinearGaugeMethod() {  
  var linearObj = $("#default").data("ejLinearGauge");  
  linearObj.getPointerPlacement();  
}
```

### [getPointerValue\(\)](#)

To get PointerValue in number

#### HTML

```
<div id="linear"></div>
```

#### JAVASCRIPT

```
ReactDOM.render(  
  <EJ.LinearGauge id="default"></EJ.LinearGauge>,  
  document.getElementById('linear')  
)  
;  
function LinearGaugeMethod() {  
  var linearObj = $("#default").data("ejLinearGauge");  
  linearObj.getPointerValue();  
}
```

```
};
```

### [getPointerWidth\(\)](#)

To get PointerWidth in number

#### **HTML**

```
<div id="linear"></div>
```

#### **JAVASCRIPT**

```
ReactDOM.render(  
  <EJ.LinearGauge id="default"></EJ.LinearGauge>,  
  document.getElementById('linear')  
)  
;  
function LinearGaugeMethod() {  
  var linearObj = $("#default").data("ejLinearGauge");  
  linearObj.getPointerWidth();  
};
```

### [getRangeBorderWidth\(\)](#)

To get Range Border Width in number

#### **HTML**

```
<div id="linear"></div>
```

#### **JAVASCRIPT**

```
ReactDOM.render(  
  <EJ.LinearGauge id="default"></EJ.LinearGauge>,  
  document.getElementById('linear')  
)  
;  
function LinearGaugeMethod() {  
  var linearObj = $("#default").data("ejLinearGauge");  
  linearObj.getRangeBorderWidth();  
};
```

### [getRangeDistanceFromScale\(\)](#)

To get Range Distance From Scale in number

#### **HTML**

```
<div id="linear"></div>
```

#### **JAVASCRIPT**

```
ReactDOM.render(  
  <EJ.LinearGauge id="default"></EJ.LinearGauge>,  
  document.getElementById('linear')  
)  
;  
function LinearGaugeMethod() {  
  var linearObj = $("#default").data("ejLinearGauge");  
  linearObj.getRangeDistanceFromScale();  
};
```

```
};
```

### *getRangeEndValue()*

To get Range End Value in number

#### **HTML**

```
<div id="linear"></div>
```

#### **JAVASCRIPT**

```
ReactDOM.render(  
  <EJ.LinearGauge id="default"></EJ.LinearGauge>,  
  document.getElementById('linear')  
)  
;  
function LinearGaugeMethod() {  
  var linearObj = $("#default").data("ejLinearGauge");  
  linearObj.getRangeEndValue();  
}
```

### *getRangeEndWidth()*

To get Range End Width in number

#### **HTML**

```
<div id="linear"></div>
```

#### **JAVASCRIPT**

```
ReactDOM.render(  
  <EJ.LinearGauge id="default"></EJ.LinearGauge>,  
  document.getElementById('linear')  
)  
;  
function LinearGaugeMethod() {  
  var linearObj = $("#default").data("ejLinearGauge");  
  linearObj.getRangeEndWidth();  
}
```

### *getRangePosition()*

To get Range Position in number

#### **HTML**

```
<div id="linear"></div>
```

#### **JAVASCRIPT**

```
ReactDOM.render(  
  <EJ.LinearGauge id="default"></EJ.LinearGauge>,  
  document.getElementById('linear')  
)  
;  
function LinearGaugeMethod() {  
  var linearObj = $("#default").data("ejLinearGauge");  
  linearObj.getRangePosition();  
}
```

```
};
```

### *getRangeStartValue()*

To get Range Start Value in number

#### **HTML**

```
<div id="linear"></div>
```

#### **JAVASCRIPT**

```
ReactDOM.render(  
  <EJ.LinearGauge id="default"></EJ.LinearGauge>,  
  document.getElementById('linear')  
)  
;  
function LinearGaugeMethod() {  
  var linearObj = $("#default").data("ejLinearGauge");  
  linearObj.getRangeStartValue();  
}
```

### *getRangeStartWidth()*

To get Range Start Width in number

#### **HTML**

```
<div id="linear"></div>
```

#### **JAVASCRIPT**

```
ReactDOM.render(  
  <EJ.LinearGauge id="default"></EJ.LinearGauge>,  
  document.getElementById('linear')  
)  
;  
function LinearGaugeMethod() {  
  var linearObj = $("#default").data("ejLinearGauge");  
  linearObj.getRangeStartWidth();  
}
```

### *getScaleBarLength()*

To get ScaleBarLength in number

#### **HTML**

```
<div id="linear"></div>
```

#### **JAVASCRIPT**

```
ReactDOM.render(  
  <EJ.LinearGauge id="default"></EJ.LinearGauge>,  
  document.getElementById('linear')  
)  
;  
function LinearGaugeMethod() {  
  var linearObj = $("#default").data("ejLinearGauge");  
  linearObj.getScaleBarLength();  
}
```

```
};
```

### *getScaleBarSize()*

To get Scale Bar Size in number

#### **HTML**

```
<div id="linear"></div>
```

#### **JAVASCRIPT**

```
ReactDOM.render(  
  <EJ.LinearGauge id="default"></EJ.LinearGauge>,  
  document.getElementById('linear')  
)  
;  
function LinearGaugeMethod() {  
  var linearObj = $("#default").data("ejLinearGauge");  
  linearObj.getScaleBarSize();  
};
```

### *getScaleBorderWidth()*

To get Scale Border Width in number

#### **HTML**

```
<div id="linear"></div>
```

#### **JAVASCRIPT**

```
ReactDOM.render(  
  <EJ.LinearGauge id="default"></EJ.LinearGauge>,  
  document.getElementById('linear')  
)  
;  
function LinearGaugeMethod() {  
  var linearObj = $("#default").data("ejLinearGauge");  
  linearObj.getScaleBorderWidth();  
};
```

### *getScaleDirection()*

To get Scale Direction in number

#### **HTML**

```
<div id="linear"></div>
```

#### **JAVASCRIPT**

```
ReactDOM.render(  
  <EJ.LinearGauge id="default"></EJ.LinearGauge>,  
  document.getElementById('linear')  
)  
;  
function LinearGaugeMethod() {  
  var linearObj = $("#default").data("ejLinearGauge");  
  linearObj.getScaleDirection();  
};
```

```
};
```

### *getScaleLocation()*

To get Scale Location in object

#### **HTML**

```
<div id="linear"></div>
```

#### **JAVASCRIPT**

```
ReactDOM.render(  
  <EJ.LinearGauge id="default"></EJ.LinearGauge>,  
  document.getElementById('linear')  
)  
;  
function LinearGaugeMethod() {  
  var linearObj = $("#default").data("ejLinearGauge");  
  linearObj.getScaleLocation();  
}
```

### *getScaleStyle()*

To get Scale Style in string

#### **HTML**

```
<div id="linear"></div>
```

#### **JAVASCRIPT**

```
ReactDOM.render(  
  <EJ.LinearGauge id="default"></EJ.LinearGauge>,  
  document.getElementById('linear')  
)  
;  
function LinearGaugeMethod() {  
  var linearObj = $("#default").data("ejLinearGauge");  
  linearObj.getScaleStyle();  
}
```

### *getTickAngle()*

To get Tick Angle in number

#### **HTML**

```
<div id="linear"></div>
```

#### **JAVASCRIPT**

```
ReactDOM.render(  
  <EJ.LinearGauge id="default"></EJ.LinearGauge>,  
  document.getElementById('linear')  
)  
;  
function LinearGaugeMethod() {  
  var linearObj = $("#default").data("ejLinearGauge");  
  linearObj.getTickAngle();  
}
```



```
};
```

### *getTickHeight()*

To get Tick Height in number

#### **HTML**

```
<div id="linear"></div>
```

#### **JAVASCRIPT**

```
ReactDOM.render(  
  <EJ.LinearGauge id="default"></EJ.LinearGauge>,  
  document.getElementById('linear')  
)  
;  
function LinearGaugeMethod() {  
  var linearObj = $("#default").data("ejLinearGauge");  
  linearObj.getTickHeight();  
};
```

### *getTickPlacement()*

To get getTickPlacement in number

#### **HTML**

```
<div id="linear"></div>
```

#### **JAVASCRIPT**

```
ReactDOM.render(  
  <EJ.LinearGauge id="default"></EJ.LinearGauge>,  
  document.getElementById('linear')  
)  
;  
function LinearGaugeMethod() {  
  var linearObj = $("#default").data("ejLinearGauge");  
  linearObj.getTickPlacement();  
};
```

### *getTickStyle()*

To get Tick Style in string

#### **HTML**

```
<div id="linear"></div>
```

#### **JAVASCRIPT**

```
ReactDOM.render(  
  <EJ.LinearGauge id="default"></EJ.LinearGauge>,  
  document.getElementById('linear')  
)  
;  
function LinearGaugeMethod() {  
  var linearObj = $("#default").data("ejLinearGauge");  
  linearObj.getTickStyle();  
};
```

```
};
```

### *getTickWidth()*

To get Tick Width in number

#### **HTML**

```
<div id="linear"></div>
```

#### **JAVASCRIPT**

```
ReactDOM.render(  
  <EJ.LinearGauge id="default"></EJ.LinearGauge>,  
  document.getElementById('linear')  
)  
;  
function LinearGaugeMethod() {  
  var linearObj = $("#default").data("ejLinearGauge");  
  linearObj.getTickWidth();  
};
```

### *getTickXDistanceFromScale()*

To get get Tick XDistance From Scale in number

#### **HTML**

```
<div id="linear"></div>
```

#### **JAVASCRIPT**

```
ReactDOM.render(  
  <EJ.LinearGauge id="default"></EJ.LinearGauge>,  
  document.getElementById('linear')  
)  
;  
function LinearGaugeMethod() {  
  var linearObj = $("#default").data("ejLinearGauge");  
  linearObj.getTickXDistanceFromScale();  
};
```

### *getTickYDistanceFromScale()*

To get Tick YDistance From Scale in number

#### **HTML**

```
<div id="linear"></div>
```

#### **JAVASCRIPT**

```
ReactDOM.render(  
  <EJ.LinearGauge id="default"></EJ.LinearGauge>,  
  document.getElementById('linear')  
)  
;  
function LinearGaugeMethod() {  
  var linearObj = $("#default").data("ejLinearGauge");  
  linearObj.getTickYDistanceFromScale();  
};
```

```
};
```

### *scales()*

Specifies the scales.

#### **HTML**

```
<div id="linear"></div>
```

#### **JAVASCRIPT**

```
ReactDOM.render(  
  <EJ.LinearGauge id="default"></EJ.LinearGauge>,  
  document.getElementById('linear')  
)  
;  
function LinearGaugeMethod() {  
  var linearObj = $("#default").data("ejLinearGauge");  
  linearObj.scales();  
};
```

### *setBarDistanceFromScale()*

To set setBarDistanceFromScale

#### **HTML**

```
<div id="linear"></div>
```

#### **JAVASCRIPT**

```
ReactDOM.render(  
  <EJ.LinearGauge id="default"></EJ.LinearGauge>,  
  document.getElementById('linear')  
)  
;  
function LinearGaugeMethod() {  
  var linearObj = $("#default").data("ejLinearGauge");  
  linearObj.setBarDistanceFromScale();  
};
```

### *setBarPointerValue()*

To set setBarPointerValue

#### **HTML**

```
<div id="linear"></div>
```

#### **JAVASCRIPT**

```
ReactDOM.render(  
  <EJ.LinearGauge id="default"></EJ.LinearGauge>,  
  document.getElementById('linear')  
)  
;  
function LinearGaugeMethod() {  
  var linearObj = $("#default").data("ejLinearGauge");  
  linearObj.setBarPointerValue();  
};
```

```
};
```

### *setBarWidth()*

To set setBarWidth

#### **HTML**

```
<div id="linear"></div>
```

#### **JAVASCRIPT**

```
ReactDOM.render(  
  <EJ.LinearGauge id="default"></EJ.LinearGauge>,  
  document.getElementById('linear')  
)  
;  
function LinearGaugeMethod() {  
  var linearObj = $("#default").data("ejLinearGauge");  
  linearObj.setBarWidth();  
};
```

### *setCustomLabelAngle()*

To set setCustomLabelAngle

#### **HTML**

```
<div id="linear"></div>
```

#### **JAVASCRIPT**

```
ReactDOM.render(  
  <EJ.LinearGauge id="default"></EJ.LinearGauge>,  
  document.getElementById('linear')  
)  
;  
function LinearGaugeMethod() {  
  var linearObj = $("#default").data("ejLinearGauge");  
  linearObj.setCustomLabelAngle();  
};
```

### *setCustomLabelValue()*

To set setCustomLabelValue

#### **HTML**

```
<div id="linear"></div>
```

#### **JAVASCRIPT**

```
ReactDOM.render(  
  <EJ.LinearGauge id="default"></EJ.LinearGauge>,  
  document.getElementById('linear')  
)  
;  
function LinearGaugeMethod() {  
  var linearObj = $("#default").data("ejLinearGauge");  
  linearObj.setCustomLabelValue();  
};
```

```
};
```

### *setLabelAngle()*

To set setLabelAngle

#### **HTML**

```
<div id="linear"></div>
```

#### **JAVASCRIPT**

```
ReactDOM.render(  
  <EJ.LinearGauge id="default"></EJ.LinearGauge>,  
  document.getElementById('linear')  
)  
;  
function LinearGaugeMethod() {  
  var linearObj = $("#default").data("ejLinearGauge");  
  linearObj.setLabelAngle();  
}
```

### *setLabelPlacement()*

To set setLabelPlacement

#### **HTML**

```
<div id="linear"></div>
```

#### **JAVASCRIPT**

```
ReactDOM.render(  
  <EJ.LinearGauge id="default"></EJ.LinearGauge>,  
  document.getElementById('linear')  
)  
;  
function LinearGaugeMethod() {  
  var linearObj = $("#default").data("ejLinearGauge");  
  linearObj.setLabelPlacement();  
}
```

### *setLabelStyle()*

To set setLabelStyle

#### **HTML**

```
<div id="linear"></div>
```

#### **JAVASCRIPT**

```
ReactDOM.render(  
  <EJ.LinearGauge id="default"></EJ.LinearGauge>,  
  document.getElementById('linear')  
)  
;  
function LinearGaugeMethod() {  
  var linearObj = $("#default").data("ejLinearGauge");  
  linearObj.setLabelStyle();  
}
```

```
};
```

### [\*setLabelXDistanceFromScale\(\)\*](#)

To set setLabelXDistanceFromScale

#### **HTML**

```
<div id="linear"></div>
```

#### **JAVASCRIPT**

```
ReactDOM.render(  
  <EJ.LinearGauge id="default"></EJ.LinearGauge>,  
  document.getElementById('linear')  
);  
function LinearGaugeMethod() {  
  var linearObj = $("#default").data("ejLinearGauge");  
  linearObj.setLabelXDistanceFromScale();  
};
```

### [\*setLabelYDistanceFromScale\(\)\*](#)

To set setLabelYDistanceFromScale

#### **HTML**

```
<div id="linear"></div>
```

#### **JAVASCRIPT**

```
ReactDOM.render(  
  <EJ.LinearGauge id="default"></EJ.LinearGauge>,  
  document.getElementById('linear')  
);  
function LinearGaugeMethod() {  
  var linearObj = $("#default").data("ejLinearGauge");  
  linearObj.setLabelYDistanceFromScale();  
};
```

### [\*setMajorIntervalValue\(\)\*](#)

To set setMajorIntervalValue

#### **HTML**

```
<div id="linear"></div>
```

#### **JAVASCRIPT**

```
ReactDOM.render(  
  <EJ.LinearGauge id="default"></EJ.LinearGauge>,  
  document.getElementById('linear')  
);  
function LinearGaugeMethod() {  
  var linearObj = $("#default").data("ejLinearGauge");  
  linearObj.setMajorIntervalValue();  
};
```

```
};
```

### *setMarkerStyle()*

To set setMarkerStyle

#### **HTML**

```
<div id="linear"></div>
```

#### **JAVASCRIPT**

```
ReactDOM.render(  
  <EJ.LinearGauge id="default"></EJ.LinearGauge>,  
  document.getElementById('linear')  
)  
;  
function LinearGaugeMethod() {  
  var linearObj = $("#default").data("ejLinearGauge");  
  linearObj.setMarkerStyle();  
};
```

### *setMaximumValue()*

To set setMaximumValue

#### **HTML**

```
<div id="linear"></div>
```

#### **JAVASCRIPT**

```
ReactDOM.render(  
  <EJ.LinearGauge id="default"></EJ.LinearGauge>,  
  document.getElementById('linear')  
)  
;  
function LinearGaugeMethod() {  
  var linearObj = $("#default").data("ejLinearGauge");  
  linearObj.setMaximumValue();  
};
```

### *setMinimumValue()*

To set setMinimumValue

#### **HTML**

```
<div id="linear"></div>
```

#### **JAVASCRIPT**

```
ReactDOM.render(  
  <EJ.LinearGauge id="default"></EJ.LinearGauge>,  
  document.getElementById('linear')  
)  
;  
function LinearGaugeMethod() {  
  var linearObj = $("#default").data("ejLinearGauge");  
  linearObj.setMinimumValue();  
};
```

```
};
```

### *setMinorIntervalValue()*

To set setMinorIntervalValue

#### **HTML**

```
<div id="linear"></div>
```

#### **JAVASCRIPT**

```
ReactDOM.render(
  <EJ.LinearGauge id="default"></EJ.LinearGauge>,
  document.getElementById('linear')
);
function LinearGaugeMethod() {
  var linearObj = $("#default").data("ejLinearGauge");
  linearObj.setMinorIntervalValue();
};
```

### *setPointerDistanceFromScale()*

To set setPointerDistanceFromScale

#### **HTML**

```
<div id="linear"></div>
```

#### **JAVASCRIPT**

```
ReactDOM.render(
  <EJ.LinearGauge id="default"></EJ.LinearGauge>,
  document.getElementById('linear')
);
function LinearGaugeMethod() {
  var linearObj = $("#default").data("ejLinearGauge");
  linearObj.setPointerDistanceFromScale();
};
```

### *setPointerHeight()*

To set PointerHeight

#### **HTML**

```
<div id="linear"></div>
```

#### **JAVASCRIPT**

```
ReactDOM.render(
  <EJ.LinearGauge id="default"></EJ.LinearGauge>,
  document.getElementById('linear')
);
function LinearGaugeMethod() {
  var linearObj = $("#default").data("ejLinearGauge");
  linearObj.setPointerHeight();
};
```



```
};
```

### *setPointerPlacement()*

To set setPointerPlacement

#### **HTML**

```
<div id="linear"></div>
```

#### **JAVASCRIPT**

```
ReactDOM.render(  
  <EJ.LinearGauge id="default"></EJ.LinearGauge>,  
  document.getElementById('linear')  
)  
;  
function LinearGaugeMethod() {  
  var linearObj = $("#default").data("ejLinearGauge");  
  linearObj.setPointerPlacement();  
};
```

### *setPointerValue()*

To set PointerValue

#### **HTML**

```
<div id="linear"></div>
```

#### **JAVASCRIPT**

```
ReactDOM.render(  
  <EJ.LinearGauge id="default"></EJ.LinearGauge>,  
  document.getElementById('linear')  
)  
;  
function LinearGaugeMethod() {  
  var linearObj = $("#default").data("ejLinearGauge");  
  linearObj.setPointerValue();  
};
```

### *setPointerWidth()*

To set PointerWidth

#### **HTML**

```
<div id="linear"></div>
```

#### **JAVASCRIPT**

```
ReactDOM.render(  
  <EJ.LinearGauge id="default"></EJ.LinearGauge>,  
  document.getElementById('linear')  
)  
;  
function LinearGaugeMethod() {  
  var linearObj = $("#default").data("ejLinearGauge");  
  linearObj.setPointerWidth();  
};
```

```
};
```

### *setRangeBorderWidth()*

To set setRangeBorderWidth

#### **HTML**

```
<div id="linear"></div>
```

#### **JAVASCRIPT**

```
ReactDOM.render(  
  <EJ.LinearGauge id="default"></EJ.LinearGauge>,  
  document.getElementById('linear')  
)  
;  
function LinearGaugeMethod() {  
  var linearObj = $("#default").data("ejLinearGauge");  
  linearObj.setRangeBorderWidth();  
};
```

### *setRangeDistanceFromScale()*

To set setRangeDistanceFromScale

#### **HTML**

```
<div id="linear"></div>
```

#### **JAVASCRIPT**

```
ReactDOM.render(  
  <EJ.LinearGauge id="default"></EJ.LinearGauge>,  
  document.getElementById('linear')  
)  
;  
function LinearGaugeMethod() {  
  var linearObj = $("#default").data("ejLinearGauge");  
  linearObj.setRangeDistanceFromScale();  
};
```

### *setRangeEndValue()*

To set setRangeEndValue

#### **HTML**

```
<div id="linear"></div>
```

#### **JAVASCRIPT**

```
ReactDOM.render(  
  <EJ.LinearGauge id="default"></EJ.LinearGauge>,  
  document.getElementById('linear')  
)  
;  
function LinearGaugeMethod() {  
  var linearObj = $("#default").data("ejLinearGauge");  
  linearObj.setRangeEndValue();  
};
```

```
};
```

### *setRangeEndWidth()*

To set setRangeEndWidth

#### **HTML**

```
<div id="linear"></div>
```

#### **JAVASCRIPT**

```
ReactDOM.render(  
  <EJ.LinearGauge id="default"></EJ.LinearGauge>,  
  document.getElementById('linear')  
)  
;  
function LinearGaugeMethod() {  
  var linearObj = $("#default").data("ejLinearGauge");  
  linearObj.setRangeEndWidth();  
};
```

### *setRangePosition()*

To set setRangePosition

#### **HTML**

```
<div id="linear"></div>
```

#### **JAVASCRIPT**

```
ReactDOM.render(  
  <EJ.LinearGauge id="default"></EJ.LinearGauge>,  
  document.getElementById('linear')  
)  
;  
function LinearGaugeMethod() {  
  var linearObj = $("#default").data("ejLinearGauge");  
  linearObj.setRangePosition();  
};
```

### *setRangeStartValue()*

To set setRangeStartValue

#### **HTML**

```
<div id="linear"></div>
```

#### **JAVASCRIPT**

```
ReactDOM.render(  
  <EJ.LinearGauge id="default"></EJ.LinearGauge>,  
  document.getElementById('linear')  
)  
;  
function LinearGaugeMethod() {  
  var linearObj = $("#default").data("ejLinearGauge");  
  linearObj.setRangeStartValue();  
};
```

```
};
```

### *setRangeStartWidth()*

To set setRangeStartWidth

#### **HTML**

```
<div id="linear"></div>
```

#### **JAVASCRIPT**

```
ReactDOM.render(  
  <EJ.LinearGauge id="default"></EJ.LinearGauge>,  
  document.getElementById('linear')  
)  
;  
function LinearGaugeMethod() {  
  var linearObj = $("#default").data("ejLinearGauge");  
  linearObj.setRangeStartWidth();  
}
```

### *setScaleBarLength()*

To set setScaleBarLength

#### **HTML**

```
<div id="linear"></div>
```

#### **JAVASCRIPT**

```
ReactDOM.render(  
  <EJ.LinearGauge id="default"></EJ.LinearGauge>,  
  document.getElementById('linear')  
)  
;  
function LinearGaugeMethod() {  
  var linearObj = $("#default").data("ejLinearGauge");  
  linearObj.setScaleBarLength();  
}
```

### *setScaleBarSize()*

To set setScaleBarSize

#### **HTML**

```
<div id="linear"></div>
```

#### **JAVASCRIPT**

```
ReactDOM.render(  
  <EJ.LinearGauge id="default"></EJ.LinearGauge>,  
  document.getElementById('linear')  
)  
;  
function LinearGaugeMethod() {  
  var linearObj = $("#default").data("ejLinearGauge");  
  linearObj.setScaleBarSize();  
}
```

```
};
```

### *setScaleBorderWidth()*

To set `setScaleBorderWidth`

#### **HTML**

```
<div id="linear"></div>
```

#### **JAVASCRIPT**

```
ReactDOM.render(  
  <EJ.LinearGauge id="default"></EJ.LinearGauge>,  
  document.getElementById('linear')  
)  
;  
function LinearGaugeMethod() {  
  var linearObj = $("#default").data("ejLinearGauge");  
  linearObj.setScaleBorderWidth();  
}
```

### *setScaleDirection()*

To set `setScaleDirection`

#### **HTML**

```
<div id="linear"></div>
```

#### **JAVASCRIPT**

```
ReactDOM.render(  
  <EJ.LinearGauge id="default"></EJ.LinearGauge>,  
  document.getElementById('linear')  
)  
;  
function LinearGaugeMethod() {  
  var linearObj = $("#default").data("ejLinearGauge");  
  linearObj.setScaleDirection();  
}
```

### *setScaleLocation()*

To set `setScaleLocation`

#### **HTML**

```
<div id="linear"></div>
```

#### **JAVASCRIPT**

```
ReactDOM.render(  
  <EJ.LinearGauge id="default"></EJ.LinearGauge>,  
  document.getElementById('linear')  
)  
;  
function LinearGaugeMethod() {  
  var linearObj = $("#default").data("ejLinearGauge");  
  linearObj.setScaleLocation();  
}
```

```
};
```

### *setScaleStyle()*

To set setScaleStyle

#### **HTML**

```
<div id="linear"></div>
```

#### **JAVASCRIPT**

```
ReactDOM.render(  
  <EJ.LinearGauge id="default"></EJ.LinearGauge>,  
  document.getElementById('linear')  
)  
;  
function LinearGaugeMethod() {  
  var linearObj = $("#default").data("ejLinearGauge");  
  linearObj.setScaleStyle();  
};
```

### *setTickAngle()*

To set setTickAngle

#### **HTML**

```
<div id="linear"></div>
```

#### **JAVASCRIPT**

```
ReactDOM.render(  
  <EJ.LinearGauge id="default"></EJ.LinearGauge>,  
  document.getElementById('linear')  
)  
;  
function LinearGaugeMethod() {  
  var linearObj = $("#default").data("ejLinearGauge");  
  linearObj.setTickAngle();  
};
```

### *setTickHeight()*

To set setTickHeight

#### **HTML**

```
<div id="linear"></div>
```

#### **JAVASCRIPT**

```
ReactDOM.render(  
  <EJ.LinearGauge id="default"></EJ.LinearGauge>,  
  document.getElementById('linear')  
)  
;  
function LinearGaugeMethod() {  
  var linearObj = $("#default").data("ejLinearGauge");  
  linearObj.setTickHeight();  
};
```

```
};
```

### *setTickPlacement()*

To set setTickPlacement

#### **HTML**

```
<div id="linear"></div>
```

#### **JAVASCRIPT**

```
ReactDOM.render(  
  <EJ.LinearGauge id="default"></EJ.LinearGauge>,  
  document.getElementById('linear')  
)  
;  
function LinearGaugeMethod() {  
  var linearObj = $("#default").data("ejLinearGauge");  
  linearObj.setTickPlacement();  
}
```

### *setTickStyle()*

To set setTickStyle

#### **HTML**

```
<div id="linear"></div>
```

#### **JAVASCRIPT**

```
ReactDOM.render(  
  <EJ.LinearGauge id="default"></EJ.LinearGauge>,  
  document.getElementById('linear')  
)  
;  
function LinearGaugeMethod() {  
  var linearObj = $("#default").data("ejLinearGauge");  
  linearObj.setTickStyle();  
}
```

### *setTickWidth()*

To set setTickWidth

#### **HTML**

```
<div id="linear"></div>
```

#### **JAVASCRIPT**

```
ReactDOM.render(  
  <EJ.LinearGauge id="default"></EJ.LinearGauge>,  
  document.getElementById('linear')  
)  
;  
function LinearGaugeMethod() {  
  var linearObj = $("#default").data("ejLinearGauge");  
  linearObj.setTickWidth();  
}
```

```
};
```

### [setTickXDistanceFromScale\(\)](#)

To set setTickXDistanceFromScale

#### **HTML**

```
<div id="linear"></div>
```

#### **JAVASCRIPT**

```
ReactDOM.render(
  <EJ.LinearGauge id="default"></EJ.LinearGauge>,
  document.getElementById('linear')
);
function LinearGaugeMethod() {
  var linearObj = $("#default").data("ejLinearGauge");
  linearObj.setTickXDistanceFromScale();
};
```

### [setTickYDistanceFromScale\(\)](#)

To set setTickYDistanceFromScale

#### **HTML**

```
<div id="linear"></div>
```

#### **JAVASCRIPT**

```
ReactDOM.render(
  <EJ.LinearGauge id="default"></EJ.LinearGauge>,
  document.getElementById('linear')
);
function LinearGaugeMethod() {
  var linearObj = $("#default").data("ejLinearGauge");
  linearObj.setTickYDistanceFromScale();
};
```

### Events

#### [drawBarPointers](#)

Triggers while the bar pointer are being drawn on the gauge.

#### **HTML**

```
<div id="linear"></div>
```

#### **JAVASCRIPT**

```
ReactDOM.render(
  <EJ.LinearGauge id="default" drawBarPointers =
  {DrawBarPointers}></EJ.LinearGauge>,
  document.getElementById('linear')
);
```



```
function DrawBarPointers() {  
  // Do Something  
};
```

#### *drawCustomLabel*

Triggers while the customLabel are being drawn on the gauge.

#### **HTML**

```
<div id="linear"></div>
```

#### **JAVASCRIPT**

```
ReactDOM.render(  
  <EJ.LinearGauge id="default" drawCustomLabel =  
    {DrawCustomLabel}></EJ.LinearGauge>,  
  document.getElementById('linear')  
);  
function DrawCustomLabel() {  
  // Do Something  
};
```

#### *drawIndicators*

Triggers while the Indicator are being drawn on the gauge.

#### **HTML**

```
<div id="linear"></div>
```

#### **JAVASCRIPT**

```
ReactDOM.render(  
  <EJ.LinearGauge id="default" drawIndicators =  
    {DrawIndicators}></EJ.LinearGauge>,  
  document.getElementById('linear')  
);  
function DrawIndicators() {  
  // Do Something  
};
```

#### *drawLabels*

Triggers while the label are being drawn on the gauge.

#### **HTML**

```
<div id="linear"></div>
```

#### **JAVASCRIPT**

```
ReactDOM.render(  
  <EJ.LinearGauge id="default" drawLabels = {DrawLabels}></EJ.LinearGauge>,  
  document.getElementById('linear')  
);  
function DrawLabels() {
```

```
// Do Something  
};
```

### *drawMarkerPointers*

Triggers while the marker are being drawn on the gauge.

#### **HTML**

```
<div id="linear"></div>
```

#### **JAVASCRIPT**

```
ReactDOM.render(  
  <EJ.LinearGauge id="default"  
    drawMarkerPointers={DrawMarkerPointers}></EJ.LinearGauge>,  
  document.getElementById('linear')  
)  
;  
function DrawMarkerPointers() {  
  // Do Something  
};
```

### *drawRange*

Triggers while the range are being drawn on the gauge.

#### **HTML**

```
<div id="linear"></div>
```

#### **JAVASCRIPT**

```
ReactDOM.render(  
  <EJ.LinearGauge id="default" drawRange = {DrawRange}></EJ.LinearGauge>,  
  document.getElementById('linear')  
)  
;  
function DrawRange() {  
  // Do Something  
};
```

### *drawTicks*

Triggers while the ticks are being drawn on the gauge.

#### **HTML**

```
<div id="linear"></div>
```

#### **JAVASCRIPT**

```
ReactDOM.render(  
  <EJ.LinearGauge id="default" drawTicks = {DrawTicks}></EJ.LinearGauge>,  
  document.getElementById('linear')  
)  
;  
function DrawTicks() {  
  // Do Something  
};
```

*init*

Triggers when the gauge is initialized.

**HTML**

```
<div id="linear"></div>
```

**JAVASCRIPT**

```
ReactDOM.render(  
  <EJ.LinearGauge id="default" init = {Init}></EJ.LinearGauge>,  
  document.getElementById('linear')  
);  
function Init(){  
  // Do Something  
};
```

*load*

Triggers while the gauge start to Load.

**HTML**

```
<div id="linear"></div>
```

**JAVASCRIPT**

```
ReactDOM.render(  
  <EJ.LinearGauge id="default" load = {Load}></EJ.LinearGauge>,  
  document.getElementById('linear')  
);  
function Load(){  
  // Do Something  
};
```

*mouseClick*

Triggers when the left mouse button is clicked.

**HTML**

```
<div id="linear"></div>
```

**JAVASCRIPT**

```
ReactDOM.render(  
  <EJ.LinearGauge id="default" mouseClick = {MouseClick}></EJ.LinearGauge>,  
  document.getElementById('linear')  
);  
function MouseClick(){  
  // Do Something  
};
```

### *mouseClickMove*

Triggers when clicking and dragging the mouse pointer over the gauge pointer.

#### **HTML**

```
<div id="linear"></div>
```

#### **JAVASCRIPT**

```
ReactDOM.render(  
  <EJ.LinearGauge id="default"  
    mouseClickMove={MouseClickMove}></EJ.LinearGauge>,  
  document.getElementById('linear')  
);  
function MouseClickMove() {  
  // Do Something  
};
```

### *mouseClickUp*

Triggers when the mouse click is released.

#### **HTML**

```
<div id="linear"></div>
```

#### **JAVASCRIPT**

```
ReactDOM.render(  
  <EJ.LinearGauge id="default" mouseClickUp =  
    {MouseClickUp}></EJ.LinearGauge>,  
  document.getElementById('linear')  
);  
function MouseClickUp() {  
  // Do Something  
};
```

### *renderComplete*

Triggers while the rendering of the gauge completed.

#### **HTML**

```
<div id="linear"></div>
```

#### **JAVASCRIPT**

```
ReactDOM.render(  
  <EJ.LinearGauge id="default" renderComplete =  
    {RenderComplete}></EJ.LinearGauge>,  
  document.getElementById('linear')  
);  
function RenderComplete() {  
  // Do Something  
};
```

## ListBox

### Getting Started

The React **ListBox** control contains a list of selectable items. It performs exceptionally well with thousands of items and supports checkboxes, template, single and multiple selection, keyboard navigation and virtual scrolling.

### Key Features

**Data Binding:** Supports Data binding with JSON data and remote data.

**Multi Selection:** Supports multiple selection of list items.

**Virtual Scrolling:** Provides support to load its data on demand through virtual scrolling which greatly improves the application's performance.

**Template Support:** Support Template contents to render as list items

**Grouping:** Groups the set of list items with header

**Cascading:** To populate data in a ListBox based on the selection in another ListBox.

**Drag and Drop:** Supports drag and drop features for list items

This section helps to understand the getting started of the React ListBox with the step-by-step instructions.

### Create an ListBox

You can create a React application and add necessary scripts and styles with the help of the given [ReactJJs Getting Started](#) Documentation.

Define an HTML element for adding ListBox in the application and refer the JSX file created.

### HTML

```
<script src="scripts/sampleddata.js"></script>
<div id="listbox-default"></div>
<script src="app/listbox/default.js"></script>
```

To configure data for ListBox component, define data in an array and map corresponding fields to it.

### JS

```
var BikeList = [
  { empid: "bk1", text: "Aache RTR" }, { empid: "bk2", text: "CBR 150-R" }, {
  empid: "bk3", text: "CBZ Xtreme" },
  { empid: "bk4", text: "Discover" }, { empid: "bk5", text: "Dazzler" }, {
  empid: "bk6", text: "Flame" },
  { empid: "bk7", text: "Fazzer" }, { empid: "bk8", text: "FZ-S" }, { empid:
  "bk9", text: "Pulsar" },
  { empid: "bk10", text: "Shine" }, { empid: "bk11", text: "R15" }, { empid:
  "bk12", text: "Unicorn" }
];
window.listBoxFields = { id: "empid", value: "text" };
```

Create a JSX file for rendering ListBox component using <EJ.ListBox> syntax. Add required properties to it in <EJ.ListBox> tag element

## JS

```
"use strict";
ReactDOM.render(
  <div>
    <div className="lblTitle">Select a bike</div>
    <EJ.ListBox id="default" width="100%" dataSource={BikeList}
      fields={window.listBoxFields}>
    </EJ.ListBox>
  </div>,
  document.getElementById('listbox-default')
);
```

Run the above code to render the following output:



## Selection

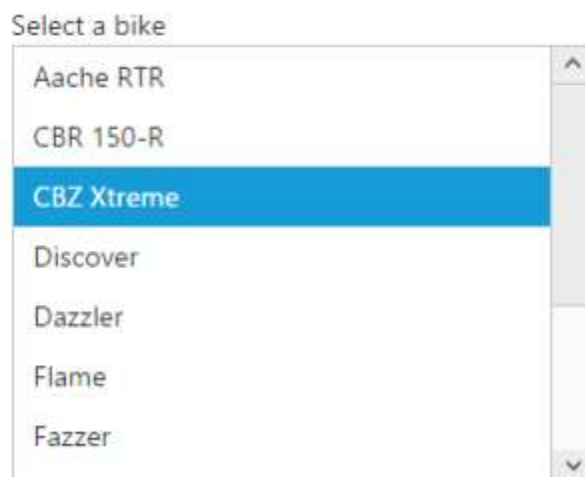
The React **ListBox** widget also supports the item selection.

Use the following code render the ListBox with Selection

## JS

```
"use strict";
ReactDOM.render(
  <div>
    <div className="lblTitle">Select a bike</div>
    <EJ.ListBox id="default" width="100%" dataSource={BikeList}
      fields={window.listBoxFields} selectedIndex="2">
    </EJ.ListBox>
  </div>,
  document.getElementById('listbox-default')
);
```

Run the above code to render the following output.



*Note: You can find the ListBox properties from the [API reference](#) document*

## ListView

### Getting Started

The **React ListView** widget builds an interactive list view interface. This control allows you to select an item from a list-like interface and provides the infrastructure to display a set of data items in different layouts or views. Lists display data, data navigation, result lists, and data entry.

### Key Features

- **AJAX Load:** Loads AJAX content in the ListView content.
- **Data binding:** Supports Data binding with JSON data and remote data.
- **Template support:** The ListView supports template content.
- **Group List:** The ListView supports group item list.
- **Check List:** The ListView supports check list.

This section helps to understand the getting started of the ReactJS ListView with the step-by-step instructions.

### Create an ListView

You can create a React application and add necessary scripts and styles with the help of the given [ReactJs Getting Started](#) Documentation.

Define an HTML element for adding ListView in the application and refer the JSX file created.

### HTML

```
<div id="listview-default"></div>
<script src="app/listview/default.js"></script>
```

Create a JSX file for rendering ListView component using <EJ.ListView> syntax. Add required properties to it in <EJ.ListBox> tag element

### JS

```
"use strict";
ReactDOM.render(
  <div className="list">
    <EJ.ListView id="default" width="100%" height="100%"
      persistSelection={true}>
      <ul>
        <li data-ej-text="Artwork"></li>
        <li data-ej-text="Abstract"></li>
        <li data-ej-text="2 Acrylic Mediums"></li>
        <li data-ej-text="Creative Acrylic"></li>
        <li data-ej-text="Modern Painting"></li>
        <li data-ej-text="Canvas Art"></li>
        <li data-ej-text="Black white"></li>
        <li data-ej-text="Children"></li>
        <li data-ej-text="Preschool Crafts"></li>
        <li data-ej-text="School-age Crafts"></li>
      </ul>
    </EJ.ListView>
  </div>,
  document.getElementById('listview-default')
);
```

Run the above code to render the following output:

Artwork
Abstract
2 Acrylic Mediums
Creative Acrylic
Modern Painting
Canvas Art
Black white
Children
Preschool Crafts
School-age Crafts

#### Add Header

We can add a header for **ListView**. Refer to the following script.

Use the following code render the ListBox with Selection

#### JS

```
"use strict";
ReactDOM.render(
  <div className="list">
    <EJ.ListView id="default" width="100%" height="100%" showHeader={true}
      headerTitle="MyFavourites" persistSelection={true}>
      <ul>
        <li data-ej-text="Artwork"></li>
        <li data-ej-text="Abstract"></li>
        <li data-ej-text="2 Acrylic Mediums"></li>
        <li data-ej-text="Creative Acrylic"></li>
        <li data-ej-text="Modern Painting"></li>
        <li data-ej-text="Canvas Art"></li>
        <li data-ej-text="Black white"></li>
        <li data-ej-text="Children"></li>
        <li data-ej-text="Preschool Crafts"></li>
      </ul>
    </EJ.ListView>
  </div>
);
```



```
<li data-ej-text="School-age Crafts"></li>
</ul>
</EJ.ListView>
</div>,
document.getElementById('listview-default')
```

Run the above code to render the following output.

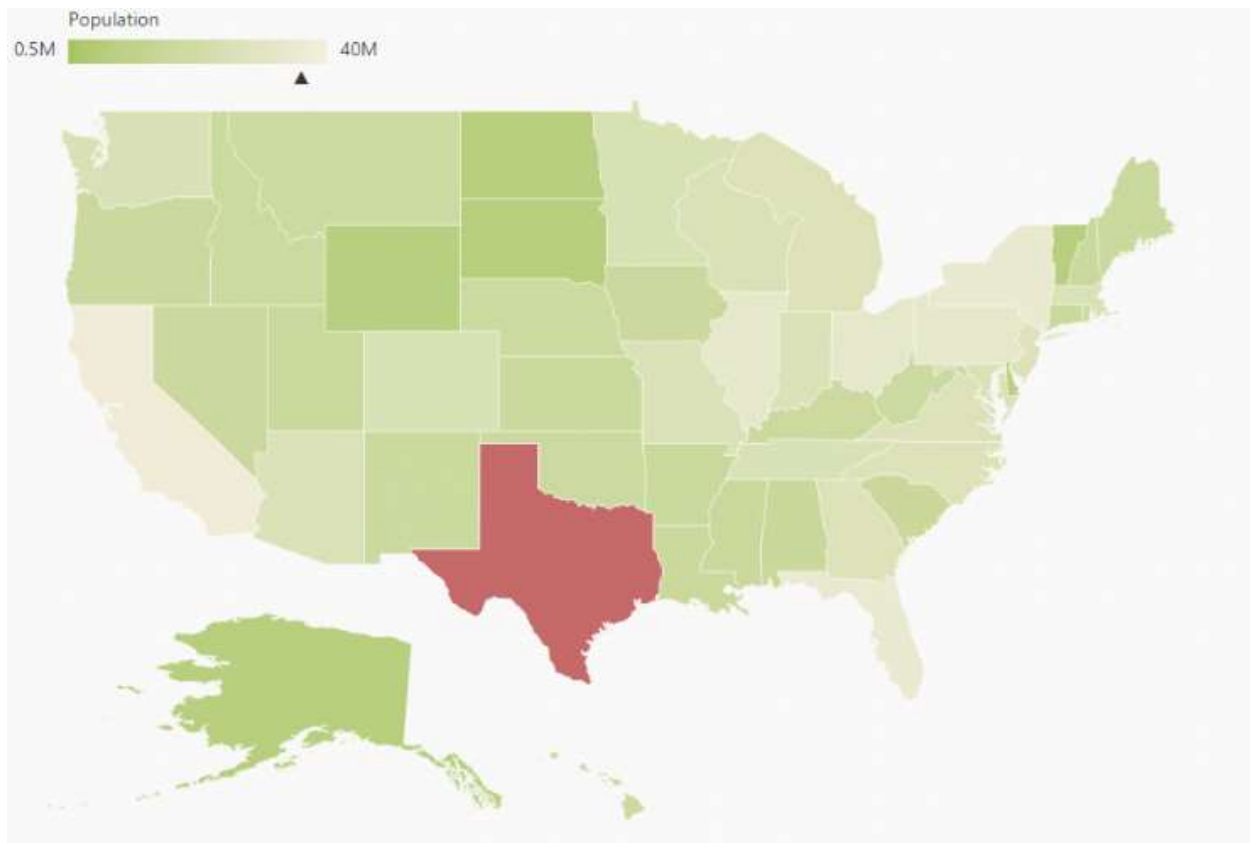
MyFavourites
Artwork
Abstract
2 Acrylic Mediums
Creative Acrylic
Modern Painting
Canvas Art
Black white
Children
Preschool Crafts
School-age Crafts

*Note: You can find the ListView properties from the [API reference](#) document*

## Maps

### Getting Started

- This section explains briefly about how to create Maps in your application with TypeScript
- You can learn how to configure Map with simple steps. In this example, you can learn how to configure USA population map with customized appearance and tooltip.



### Create a Map

You can easily create the Map widget by following the below steps

### Adding Script Reference

Create an **HTML** page and add the scripts references in the order mentioned in the following code example.

- [jQuery](#) 1.10.2 and later versions
- [jsRender](#) - to render the templates

The required ReactJS script dependencies as follows. And you can also refer [React](#) to know more about ReactJS.

- react.min.js - <http://cdn.syncfusion.com/js/assets/external/react.min.js>
- react-dom.min.js - <http://cdn.syncfusion.com/js/assets/external/react-dom.min.js>
- ej.web.react.min.js - <http://cdn.syncfusion.com/{{ site.releaseversion }}/js/common/ej.web.react.min.js>

To get started, you can use the `ej.web.all.min.js` file that encapsulates all the `ej` controls and frameworks in one single file.

### HTML

```
<!DOCTYPE html>  
<html>
```

```

<head>
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<meta name="description" content="Essential Studio for React JS">
<meta name="author" content="Syncfusion">
<title>Getting Started for Ribbon React JS</title>
<!-- Essential Studio for JavaScript theme reference -->
<link href="http://cdn.syncfusion.com/{{ site.releaseversion }}/js/web/flat-
azure/ej.web.all.min.css" rel="stylesheet" />
<!-- Essential Studio for JavaScript script references -->
<script src="http://cdn.syncfusion.com/js/assets/external/jquery-
3.0.0.min.js"></script>
<script
src="http://cdn.syncfusion.com/js/assets/external/react.min.js"></script>
<script src="http://cdn.syncfusion.com/js/assets/external/react-
dom.min.js"></script>
<script src="http://cdn.syncfusion.com/{{ site.releaseversion
}}/js/web/ej.web.all.min.js"></script>
<script src="http://cdn.syncfusion.com/{{ site.releaseversion
}}/js/common/ej.web.react.min.js"></script>
<!-- Add your custom scripts here -->
</head>
<body>
</body>
</html>

```

**Note:** 1. In production, we highly recommend you to use our [custom script generator](#) to create custom script file with required controls and its dependencies only. Also to reduce the file size further please use [GZip compression](#) in your server.

2. For themes, you can use the `ej.web.all.min.css` CDN link from the code snippet given. To add the themes in your application, please refer to [this link](#).

### Control Initialization

Control can be initialized in two ways.

- Using jsx Template
- Without using jsx Template

### Using jsx Template

By using the jsx template, we can create the HTML file and jsx file. The `.jsx` file can be convert to `.js` file and it can be referred in html page.

### Preparing Shape Data

The Shape Data collection describing geographical shape information can be obtained from [GEOJSON format shapes](#).

In this example, USA shape is used as shape data by utilizing the `United States of America.json` file in the following folder structure obtained from downloaded `Maps_GeoJSON` folder.

```
..\Maps_GeoJSON\All Countries with States
```

You can assign the complete contents in “United States of America.json” file to new JSON object. For your better understanding, a JS file “usa.js” is already created to store JSON data in JSON object “usMap”.

## JAVASCRIPT

```
var usMap = //Paste all the content copied from the JSON file//
```

### Prepare Data Source

The datasource is populated with JSON data relative to shape data and stored in JSON object. USA population as datasource is used for better understanding.

The “populationData.js” file is used to store JSON data in JSON object “populationData”.

## JAVASCRIPT

```
var populationData = [
  { name: "California", population: "38332521" },
  { name: "Texas", population: "26448193" },
  { name: "New York", population: "19651127" },
  { name: "Florida", population: "19552860" },
  { name: "Illinois", population: "12882135" },
  { name: "Pennsylvania", population: "12773801" },
  { name: "Ohio", population: "11570808" },
  { name: "Georgia", population: "9992167" },
  { name: "Michigan", population: "9895622" },
  { name: "North Carolina", population: "9848060" },
  { name: "New Jersey", population: "8899339" },
  { name: "Virginia", population: "8260405" },
  { name: "Washington", population: "6971406" },
  { name: "Massachusetts", population: "6692824" },
  { name: "Arizona", population: "6626624" },
  { name: "Indiana", population: "6570902" },
  { name: "Tennessee", population: "6495978" },
  { name: "Missouri", population: "6044171" },
  { name: "Maryland", population: "5928814" },
  { name: "Wisconsin", population: "5742713" },
  { name: "Minnesota", population: "5420380" },
  { name: "Colorado", population: "5268367" },
  { name: "Alabama", population: "4833722" },
  { name: "South Carolina", population: "4774839" },
  { name: "Louisiana", population: "4625470" },
  { name: "Kentucky", population: "4395295" },
  { name: "Oregon", population: "3930065" },
  { name: "Oklahoma", population: "3850568" },
  { name: "Puerto Rico", population: "3615086" },
  { name: "Connecticut", population: "3596080" },
  { name: "Iowa", population: "3090416" },
  { name: "Mississippi", population: "2991207" },
  { name: "Arkansas", population: "2959373" },
  { name: "Utah", population: "2900872" },
  { name: "Kansas", population: "2893957" },
  { name: "Nevada", population: "2790136" },
  { name: "New Mexico", population: "2085287" },
  { name: "Nebraska", population: "1868516" },
  { name: "West Virginia", population: "1854304" },
  { name: "Idaho", population: "1612136" },
  { name: "Hawaii", population: "1404054" },
  { name: "Maine", population: "1328302" },
  { name: "New Hampshire", population: "1323459" },
  { name: "Rhode Island", population: "1051511" },
```

```
[
  { name: "Montana", population: "1015165" },
  { name: "Delaware", population: "925749" },
  { name: "South Dakota", population: "844877" },
  { name: "Alaska", population: "735132" },
  { name: "North Dakota", population: "723393" },
  { name: "District of Columbia", population: "646449" },
  { name: "Vermont", population: "626630" },
  { name: "Wyoming", population: "582658" }
]
```

You can refer to shape data and datasource as illustrated in the HTML page,

### HTML

```
<!-- Shape data file-->
<script src="usa.js" type="text/javascript"></script>
<!-- Data source file-->
<script src="populationData.js" type="text/javascript"></script>
```

### Initialize Map

1.Create a <div> tag and set the height and width to determine the map size.

### HTML

```
<!DOCTYPE html>
<html>
<body>
<div id="map-default" style="width: 900px; height: 600px;"></div>
<script src="app/map/default.js"></script>
</body>
</html>
```

2.Initialize the Map by using the `EJ.Map` tag.

### JAVASCRIPT

```
"use strict";
ReactDOM.render(
  <div className="default">
    <EJ.Map id="map1"></EJ.Map>,
  </div>,
  document.getElementById('map-default')
);
```

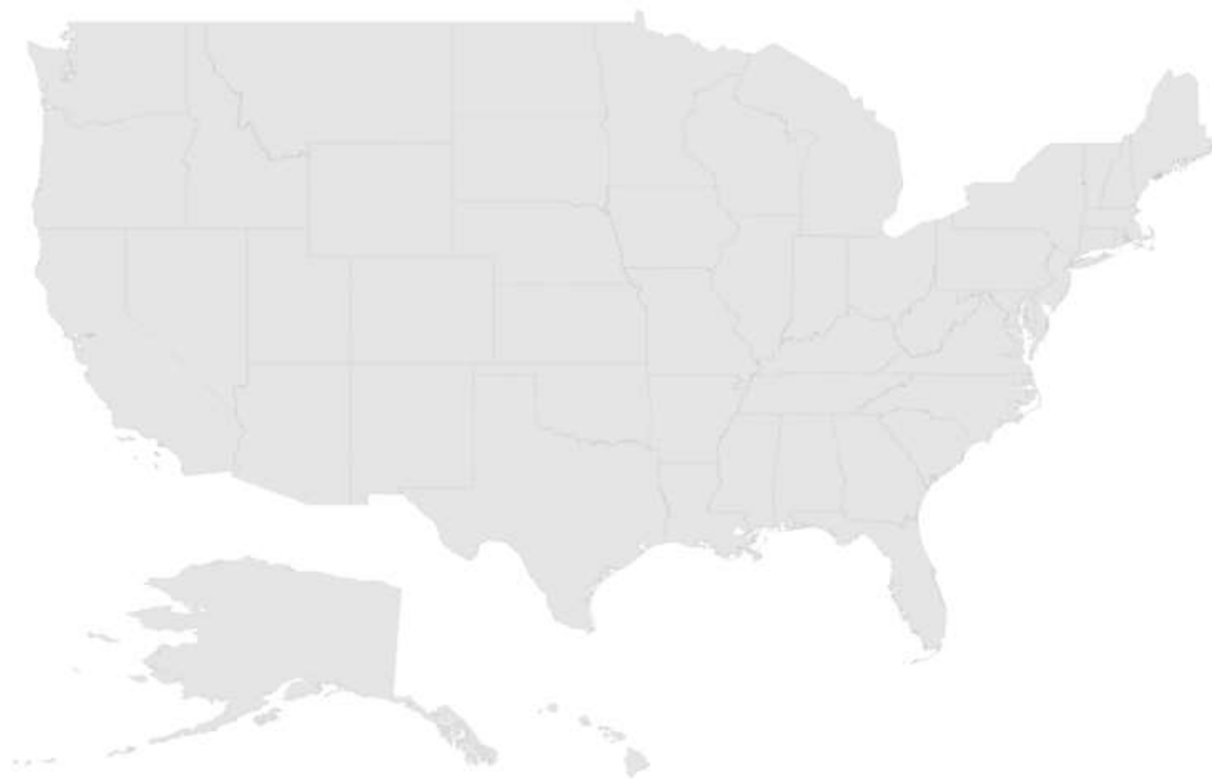
3.Add the **shapeData** property in the maps to render it in layers

### JAVASCRIPT

```
<script type="text/babel">
var layers= [
{
  shapeData: usMap
}];
<!DOCTYPE html>
<html>
```

```
<body>
<script type="text/babel">
ReactDOM.render(
<div className="default">
<EJ.Map id="map1" layers={layers}></EJ.Map>,
</div>,
document.getElementById('map-default')
);
</script>
</body>
</html>
```

The above code renders a map with default properties and shape input provided through data in layers.



### Data Binding in Map

The following properties in shape layers is used for binding data in Maps control.

- DataSource
- ShapeDataPath
- ShapePropertyPath

#### DataSource

The `dataSource` property accepts collection values as input. For example, you can provide the list of objects as input.

### Shape Data Path

The `shapeDataPath` property is used to refer the data ID in DataSource. For example, population MapData contains data ids 'Name' and 'Population'. The `shapeDataPath` and `shapePropertyPath` properties are related to each other (refer to `shapePropertyPath` for more details).

### Shape Property Path

The `shapePropertyPath` property is similar to the `shapeDataPath` that refers the column name in the Data property of shape layers to identify the shape. When the values of the `shapeDataPath` property in the `dataSource` property and the value of `shapePropertyPath` in the Data property match, then the associated object from the `dataSource` is bound to the corresponding shape.

### JAVASCRIPT

```
<script type="text/babel">
var layers= [
{
  shapeData: usMap,
  shapeDataPath: "name",
  shapePropertyPath: "name",
  dataSource: populationData
}];
<!DOCTYPE html>
<html>
<body>
<script type="text/babel">
ReactDOM.render(
<div className="default">
<EJ.Map id="map1" layers={layers}></EJ.Map>,
</div>,
document.getElementById('map-default')
);
</script>
</body>
</html>
```

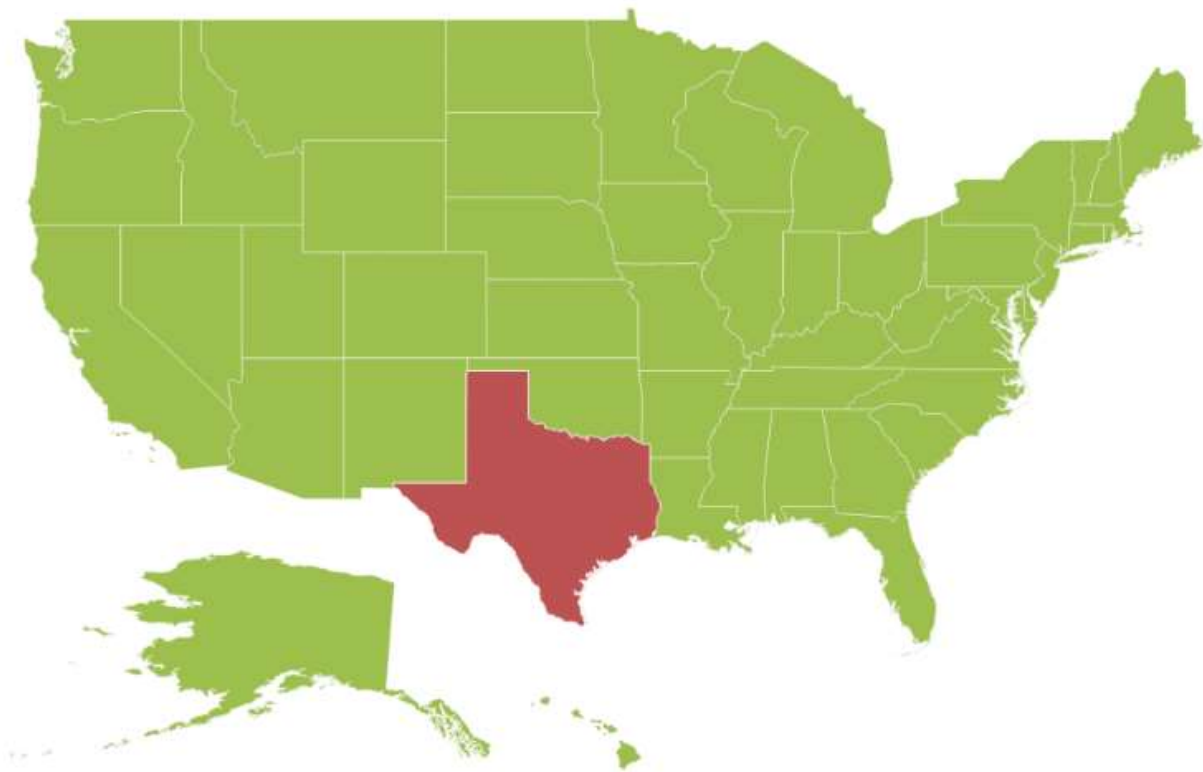
### Customize Map Appearance

You can customize the shape's color by using `fill`, `stroke` and `strokeThickness` properties in `shapeSettings`.

### JAVASCRIPT

```
<script type="text/babel">
var layers= [
{
  shapeData: usMap,
  shapeDataPath: "name",
  shapePropertyPath: "name",
  dataSource: populationData,
  enableSelection: false,
  enableMouseHover: true,
  shapeSettings: {
    fill: "#9CBF4E",
    strokeThickness: "0.5",
    stroke: "White",
    highlightStroke: "White",
```

```
highlightColor: "#BC5353",
highlightBorderWidth: "1"
}
}];
<!DOCTYPE html>
<html>
<body>
<script type="text/babel">
ReactDOM.render(
  <div className="default">
    <EJ.Map id="map1" layers={layers}></EJ.Map>,
  </div>,
  document.getElementById('map-default')
);
</script>
</body>
</html>
```



#### *Customizing Map Appearance by Range*

The Range color mapping is used to differentiate the shape's fill based on its underlying value and color ranges. The `from` and `to` properties defines the value ranges and the `gradientColors` property defines the equivalent color ranges respective to their value ranges.

**Note:** The `enableGradient` property value should be true to apply gradient colors for maps.

#### **JAVASCRIPT**

```
<script type="text/babel">
```



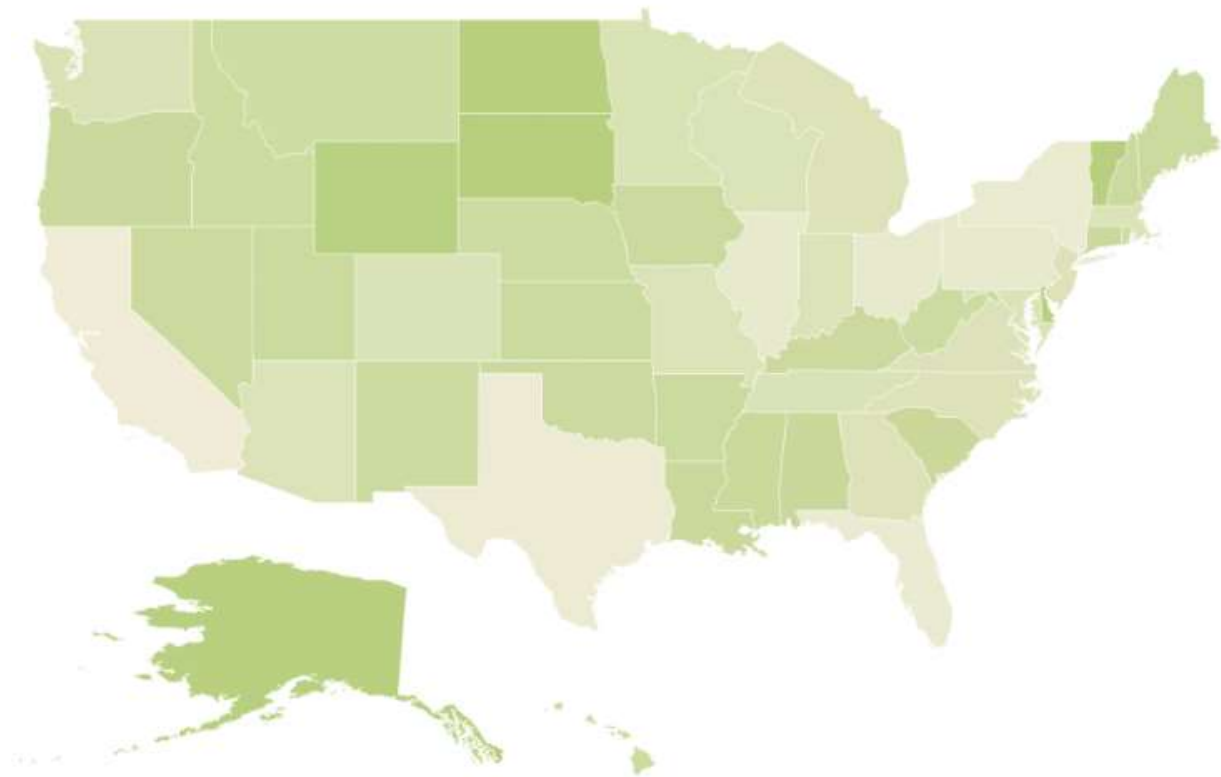
```

var layers= [
{
  shapeData: usMap,
  shapeDataPath: "name",
  shapePropertyPath: "name",
  dataSource: populationData,
  enableSelection: false,
  enableMouseHover: true,
  shapeSettings: {
    fill: "#9CBF4E",
    strokeThickness: "0.5",
    stroke: "White",
    highlightStroke: "White",
    highlightColor: "#BC5353",
    highlightBorderWidth: "1",
    valuePath: "name",
    colorValuePath: "population",
    enableGradient: true,
    colorMappings:
    {
      rangeColorMapping: [
        {
          {
            from: 500000,
            to: 1000000,
            gradientColors: ["#9CBF4E", "#B8CE7B"]
          },
          {
            from: 1000001,
            to: 5000000,
            gradientColors: ["#B8CE7B", "#CBD89A"]
          },
          {
            from: 5000001,
            to: 10000000,
            gradientColors: ["#CBD89A", "#DEE2B9"]
          },
          {
            from: 10000001,
            to: 40000000,
            gradientColors: ["#DEE2B9", "#F1ECD8"]
          }
        ]
      }
    }
  }
}];
<!DOCTYPE html>
<html>
<body>
<script type="text/babel">
ReactDOM.render(
<div className="default">
<EJ.Map id="map1" layers={layers}></EJ.Map>,
</div>,
document.getElementById('map-default')
);
</script>
</body>

```

```
</html>
```

The following screenshot illustrates a Map with gradient color property enabled.



### Enable Tooltip

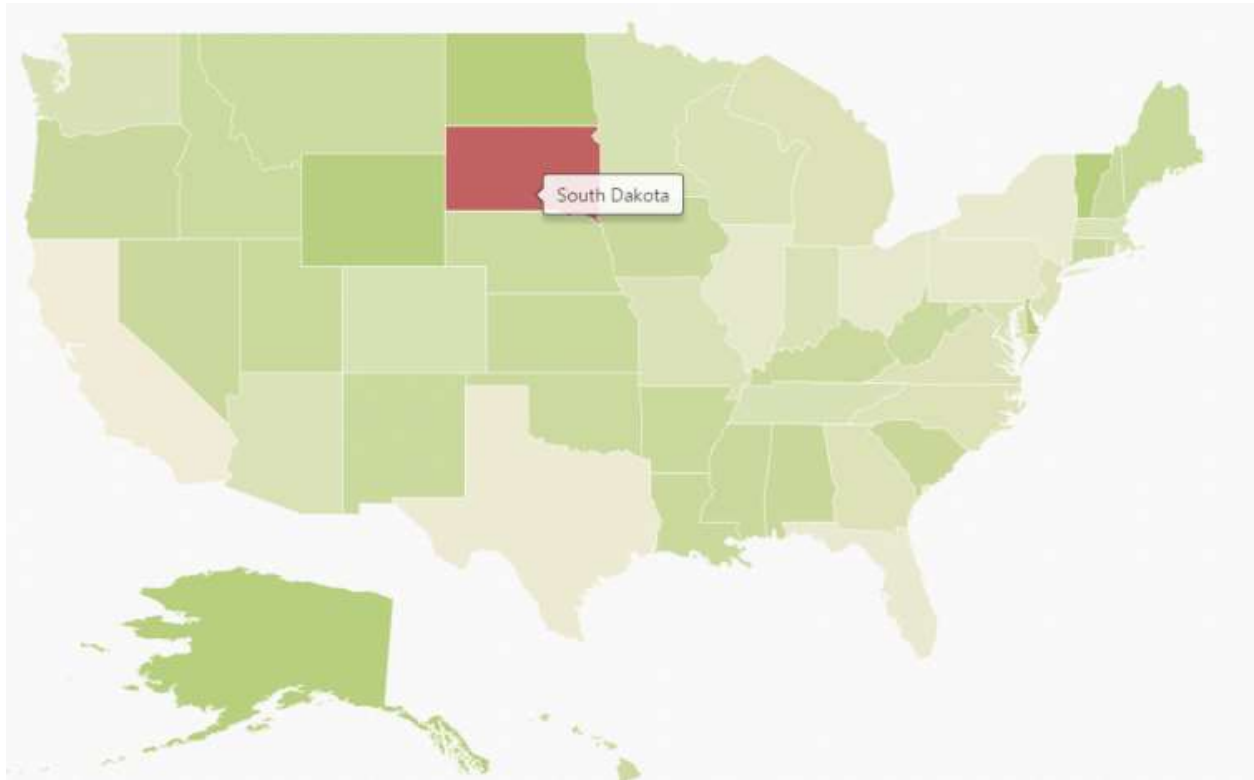
The tooltip is displayed only when `showTooltip` is set to 'true' in the shape layers. By default, it takes the property of the bound object that is referred in the `valuePath` and displays its content on hovering the corresponding shape. The `tooltipTemplate` property is used for customizing the template for tooltip.

### JAVASCRIPT

```
<script type="text/babel">
var layers= [
{
// ...
shapeSettings: {
// ...
valuePath: "name",
// ...
},
showTooltip: true
}];
<!DOCTYPE html>
<html>
<body>
<script type="text/babel">
```

```
ReactDOM.render(  
  <div className="default">  
    <EJ.Map id="map1" layers={layers}></EJ.Map>,  
  </div>,  
  document.getElementById('map-default')  
) ;  
</script>  
</body>  
</html>
```

The following screenshot illustrates a map control displaying a Tooltip.



### Legend

A Legend can be made visible by setting the `showLegend` property in `legendSetting`.

#### Interactive Legend

The legends can be made interactive with an arrow mark indicating the exact range color in the legend, when the mouse hovers the corresponding shapes. You can enable this option by setting `mode` property in `legendSettings` value as 'Interactive'. The default value of `mode` property is 'Default' to enable the normal legend.

#### Title

Use `title` property to provide title for interactive legend.

#### Label

You can use `leftLabel` and `rightLabel` property to provide left and right labels for interactive legend.

### JAVASCRIPT

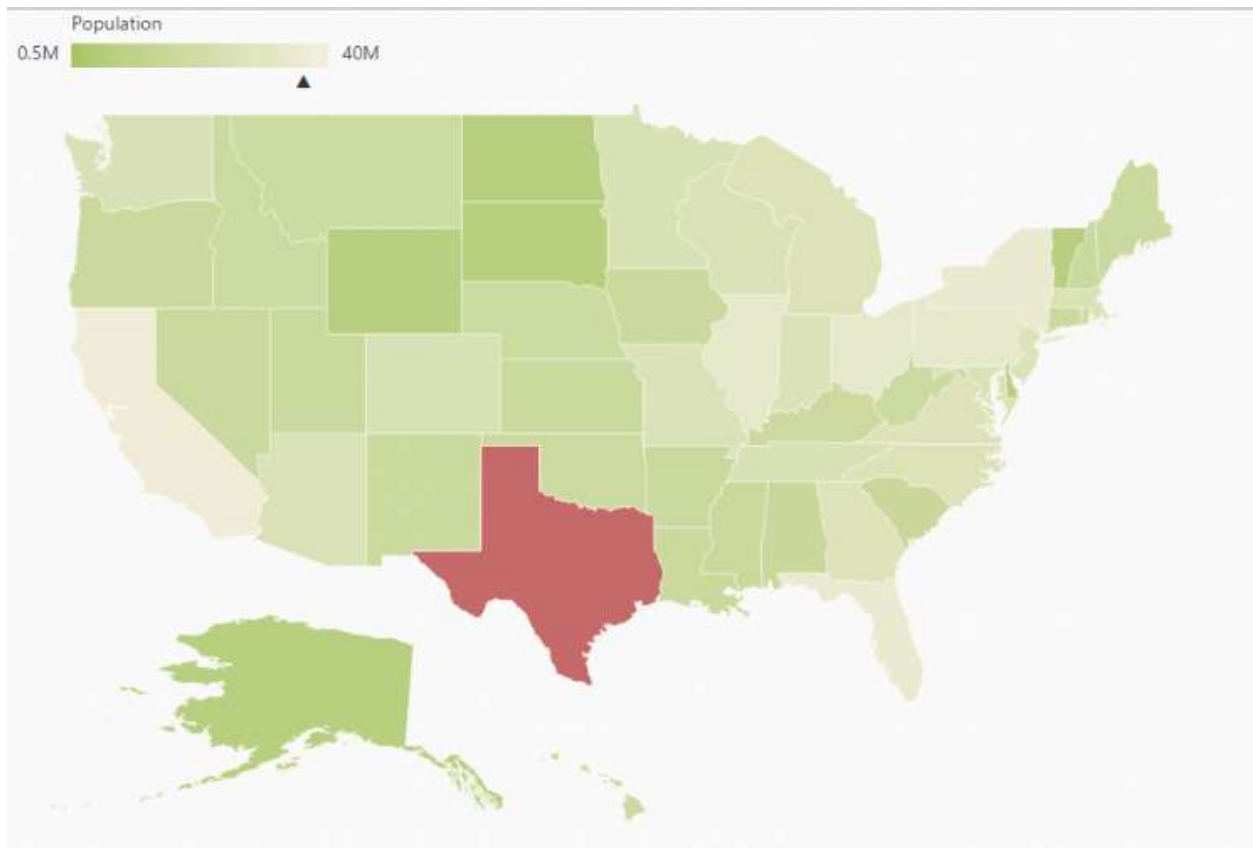
```
<script type="text/babel">
var layers= [
{
  shapeData: usMap,
  shapeDataPath: "name",
  shapePropertyPath: "name",
  dataSource: populationData,
  enableSelection: false,
  enableMouseHover: true,
  shapeSettings: {
    fill: "#9CBF4E",
    strokeThickness: "0.5",
    stroke: "White",
    highlightStroke: "White",
    highlightColor: "#BC5353",
    highlightBorderWidth: "1",
    valuePath: "population",
    enableGradient: true,
    colorMappings:
    {
      rangeColorMapping: [
        {
          from: 500000,
          to: 1000000,
          gradientColors: ["#9CBF4E", "#B8CE7B"]
        },
        {
          from: 1000001,
          to: 5000000,
          gradientColors: ["#B8CE7B", "#CBD89A"]
        },
        {
          from: 5000001,
          to: 10000000,
          gradientColors: ["#CBD89A", "#DEE2B9"]
        },
        {
          from: 10000001,
          to: 40000000,
          gradientColors: ["#DEE2B9", "#F1ECD8"]
        }
      ]
    },
    legendSettings: {
      showLegend: true,
      dockOnMap: true,
      height: 18,
      width: 190,
      mode: "interactive",
      title: "Population",
      leftLabel: "0.5M",
      rightLabel: "40M"
    }
  }
}];
<!DOCTYPE html>
<html>
<body>
```

```

<script type="text/babel">
ReactDOM.render(
  <div className="default">
    <EJ.Map id="map1" layers={layers}></EJ.Map>,
  </div>,
  document.getElementById('map-default')
);
</script>
</body>
</html>

```

The following screenshot illustrates a map displaying an interactive legend.



#### Without using jsx Template

The Map can be created from a HTML **DIV** element with the HTML **id** attribute set to it. Refer to the following code example.

#### HTML

```

<div id="map-default"></div>

```

#### JAVASCRIPT

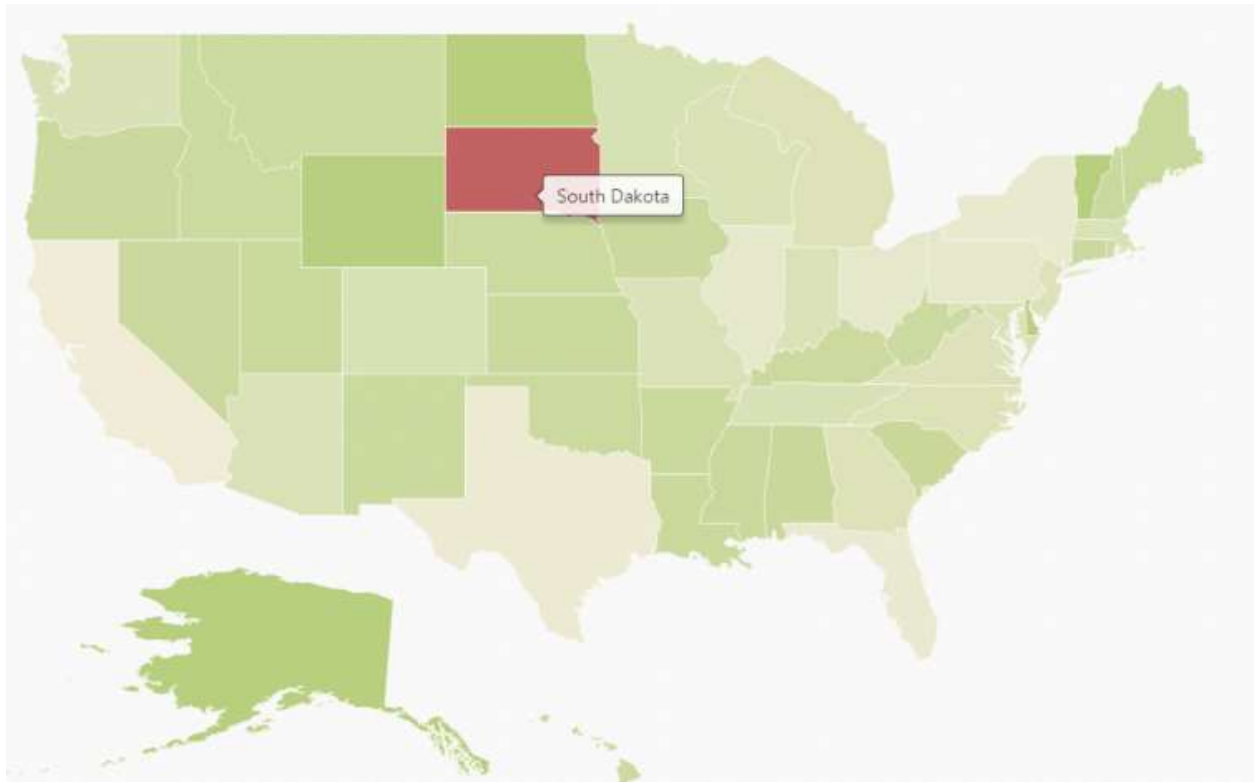
```

<script type="text/babel">
var layers= [
{
  // ...Add the required datasource , shapeDataPath ,shapePropertyPath

```

```
shapeSettings: {  
  // ...  
  valuePath: "name",  
  // ...  
},  
showTooltip: true  
}];  
ReactDOM.render(  
  React.createElement(EJ.Map, {id: "default",  
    layers: layers,  
  })  
,  
  document.getElementById('map-default')  
);  
</script>
```

On running the above code the Map will rendered along with tooltip



## Populate Data

In this section you can learn how to populate shape data for providing Data input to Map control and usage of DataSource property.

### Shape Data

The Shape Data collection describing geographical shape information can be obtained from [GEOJSON format shapes](#).

In this example, USA shape is used as shape data by utilizing the “**United States of America.json**” file in the following folder structure obtained from downloaded Maps\_GeoJSON folder.

..\ Maps\_GeoJSON\All Countries with States

You can assign the complete contents in “**United States of America.json**” file to new JSON object. For better understanding, a JS file “**usa.js**” is already created to store JSON data in JSON object “usMap”.

**[usa.js]**

### HTML

```
var usMap = //Paste all the content copied from the JSON file//
```

### Data Binding

The **Maps** control supports data binding with the **dataSource** property in the shape layers.

#### Properties

The following properties in shape layers is be used for binding data in **Maps** control,

- dataSource
- shapeDataPath
- shapePropertyPath

#### Data Source

The **dataSource** property accepts the collection values as input. For example, you can provide the list of objects as input.

#### Shape Data Path

The **shapeDataPath** property is used to refer the data ID in DataSource. For example, population MapData contains data ids ‘Name’ and ‘Population’. The **shapeDataPath** and the **shapePropertyPath** properties are related to each other (refer to **shapePropertyPath** for more details).

#### Shape Property Path

The **shapePropertyPath** property is similar to the **shapeDataPath** that refers to the column name in the **data** property of shape layers to identify the shape. When the values of the **shapeDataPath** property in the **dataSource** property and the value of **shapePropertyPath** in the **data** property match, then the associated object from the **dataSource** is bound to the corresponding shape.

The datasource is populated with JSON data relative to shape data and stored in JSON object. The USA population as datasource is used for better understanding.

The “populationData.js” file is used to store JSON data in JSON object “populationData”.

Refer both shape data and datasource as illustrated in the following code example.

**[populationData.js]**

### JAVASCRIPT

```
var populationData = [  
  { name: "California", population: "38332521" },  
  { name: "Texas", population: "26448193" },  
  { name: "New York", population: "19651127" },  
  { name: "Florida", population: "19552860" },  
  { name: "Illinois", population: "12882135" },  
  { name: "Pennsylvania", population: "12773801" },  
  { name: "Ohio", population: "11570808" },  
]
```

```

{ name: "Georgia", population: "9992167" },
{ name: "Michigan", population: "9895622" },
{ name: "North Carolina", population: "9848060" },
{ name: "New Jersey", population: "8899339" },
{ name: "Virginia", population: "8260405" },
{ name: "Washington", population: "6971406" },
{ name: "Massachusetts", population: "6692824" },
{ name: "Arizona", population: "6626624" },
{ name: "Indiana", population: "6570902" },
{ name: "Tennessee", population: "6495978" },
{ name: "Missouri", population: "6044171" },
{ name: "Maryland", population: "5928814" },
{ name: "Wisconsin", population: "5742713" },
{ name: "Minnesota", population: "5420380" },
{ name: "Colorado", population: "5268367" },
{ name: "Alabama", population: "4833722" },
{ name: "South Carolina", population: "4774839" },
{ name: "Louisiana", population: "4625470" },
{ name: "Kentucky", population: "4395295" },
{ name: "Oregon", population: "3930065" },
{ name: "Oklahoma", population: "3850568" },
{ name: "Puerto Rico", population: "3615086" },
{ name: "Connecticut", population: "3596080" },
{ name: "Iowa", population: "3090416" },
{ name: "Mississippi", population: "2991207" },
{ name: "Arkansas", population: "2959373" },
{ name: "Utah", population: "2900872" },
{ name: "Kansas", population: "2893957" },
{ name: "Nevada", population: "2790136" },
{ name: "New Mexico", population: "2085287" },
{ name: "Nebraska", population: "1868516" },
{ name: "West Virginia", population: "1854304" },
{ name: "Idaho", population: "1612136" },
{ name: "Hawaii", population: "1404054" },
{ name: "Maine", population: "1328302" },
{ name: "New Hampshire", population: "1323459" },
{ name: "Rhode Island", population: "1051511" },
{ name: "Montana", population: "1015165" },
{ name: "Delaware", population: "925749" },
{ name: "South Dakota", population: "844877" },
{ name: "Alaska", population: "735132" },
{ name: "North Dakota", population: "723393" },
{ name: "District of Columbia", population: "646449" },
{ name: "Vermont", population: "626630" },
{ name: "Wyoming", population: "582658" }
]
<!-- Shape data file-->
<script src="usa.js" type="text/javascript"></script>
<!-- Data source file-->
<script src="populationData.js" type="text/javascript"></script>

```

The JSON object “populationData” is used as `dataSource` in the following code example.

### JAVASCRIPT

```
"use strict";
```



```

var layers= [
{
  shapeData: usMap,
  shapeDataPath: "name",
  shapeIdTableField: "name",
  dataSource: populationData
}];
ReactDOM.render(
<EJ.Map id="map1" layers={layers}></EJ.Map>,
document.getElementById('maps')
);

```

## Customization

**Maps** control supports color customization to determine the exact combination of colors for shapes displayed in Maps and tooltip support to display additional information of shape data.

### Shape Settings

The `shapeSettings` defines the basic customization settings of shapes in the map.

The important property that makes an impact on shape colors is the `autoFill`. This `autoFill` property is available in the `shapeSettings`.

- `fill` - It is used to set the fill color of the shapes in the map.
- `stroke` - It is used to set the border color of the shape in the map.
- `strokeThickness` - It is used to set the border thickness of the shape in the map.
- `highlightColor` - It is used to set the mouse hover color for shapes in the map.
- `highlightBorderWidth` - It is used to set the mouse hover border width for shapes in the map.
- `selectionColor` - It is used to set the selection color for shapes in the map.

The above properties of `shapeSettings` are applied only when the `autoFill` property value is set to false. By default, the `autoFill` property value is false.

### JAVASCRIPT

```

"use strict";
var layers= [
{
  shapeData: usMap,
  enableMouseHover: true,
  shapeSettings: {
    fill: "#9CBF4E",
    strokeThickness: "0.5",
    stroke: "White",
    highlightStroke: "White",
    highlightColor: "#BC5353",
    highlightBorderWidth: "1"
  }
}];
ReactDOM.render(
<EJ.Map id="map1" layers = {layers}></EJ.Map>,
document.getElementById('maps')
);

```



### Color Mapping

The **Color Mapping** support enables the customization of shape colors based on the underlying value of shape received from bounded data.

- **colorValuePath** - It renders the field value that is to be fetched from data for each shape used for determining the shape color.
- **valuePath** - It renders the field value that is to be fetched from data for each shape. This support also provides a tree map-like impact on the map UI. The various types of Color Mapping supported in maps are listed as follows.
- **rangeColorMapping** - It is used to differentiate the shape's fill based on its underlying value and color ranges. The properties of rangeColorMapping are listed in the following table.

Property	Type	Description
from	Double	Gets or sets start value
to	Double	Gets or sets end value
color	Color	Gets or sets the colors to be applied for specific range value containing shapes when enableGradient property value is false.
label	String	Gets or sets the label for legend when mode property value is "default".
gradientColors	Array	Gets or sets the start point and end point gradient colors to be applied for specific range value containing shapes when enableGradient property value is set to true.

### JAVASCRIPT

```
"use strict";
var layers= [
{
  shapeData: usMap,
  shapeDataPath: "name",
  shapePropertyPath: "name",
  dataSource: populationData,
  shapeSettings: {
    fill: "#9CBF4E",
    strokeThickness: "0.5",
    stroke: "White",
    valuePath: 'population',
    enableGradient: true,
    colorMappings:
    {
      rangeColorMapping: [
        {
          from: 500000,
          to: 1000000,
          gradientColors: ["#9CBF4E", "#B8CE7B"]
        },
        {

```

```

from: 1000001,
to: 5000000,
gradientColors: ["#B8CE7B", "#CBD89A"]
},
{
from: 5000001,
to: 10000000,
gradientColors: ["#CBD89A", "#DEE2B9"]
},
{
from: 10000001,
to: 40000000,
gradientColors: ["#DEE2B9", "#F1ECD8"]
}]
}
}
}];
ReactDOM.render(
<EJ.Map id="map1" layers = {layers}></EJ.Map>,
document.getElementById('maps')
);

```

When the underlying object value is 700000, then the fill color of the corresponding shape is set between #9CBF4E and #B8CE7B.

When the underlying value is below any of the given sorted range or above the sorted range, then the fill is set from fill.



- **equalColorMapping** - The equalColorMapping is used to differentiate the shape's fill based on its underlying value and color. The properties of equalColorMapping is listed in the following table.

Property	Type	Description
value	String	Gets or sets the value.
color	String	Gets or sets the color for mapping.

In equal color mapping, value property contains the values of the field set in **colorValuePath** property of shape settings.

Here USA election data is considered as input datasource and stored in "electionData.js".

#### JAVASCRIPT

```

var electionData = [
{ State: "Alabama", Candidate: "Romney", Electors: 9 },
{ State: "Alaska", Candidate: "Romney", Electors: 3 },
{ State: "Arizona", Candidate: "Romney", Electors: 11 },
{ State: "Arkansas", Candidate: "Romney", Electors: 6 },
{ State: "California", Candidate: "Obama", Electors: 55 },
{ State: "Colorado", Candidate: "Obama", Electors: 9 },

```

```

{ State: "Connecticut", Candidate: "Obama", Electors: 7 },
{ State: "Delaware", Candidate: "Obama", Electors: 3 },
{ State: "District of Columbia", Candidate: "Obama", Electors: 3 },
{ State: "Florida", Candidate: "Obama", Electors: 29 },
{ State: "Georgia", Candidate: "Romney", Electors: 16 },
{ State: "Hawaii", Candidate: "Obama", Electors: 4 },
{ State: "Idaho", Candidate: "Romney", Electors: 4 },
{ State: "Illinois", Candidate: "Obama", Electors: 20 },
{ State: "Indiana", Candidate: "Romney", Electors: 11 },
{ State: "Iowa", Candidate: "Obama", Electors: 6 },
{ State: "Kansas", Candidate: "Romney", Electors: 6 },
{ State: "Kentucky", Candidate: "Romney", Electors: 8 },
{ State: "Louisiana", Candidate: "Romney", Electors: 8 },
{ State: "Maine", Candidate: "Obama", Electors: 4 },
{ State: "Maryland", Candidate: "Obama", Electors: 10 },
{ State: "Massachusetts", Candidate: "Obama", Electors: 11 },
{ State: "Michigan", Candidate: "Obama", Electors: 16 },
{ State: "Minnesota", Candidate: "Obama", Electors: 10 },
{ State: "Mississippi", Candidate: "Romney", Electors: 6 },
{ State: "Missouri", Candidate: "Romney", Electors: 10 },
{ State: "Montana", Candidate: "Romney", Electors: 3 },
{ State: "Nebraska", Candidate: "Romney", Electors: 5 },
{ State: "Nevada", Candidate: "Obama", Electors: 6 },
{ State: "New Hampshire", Candidate: "Obama", Electors: 4 },
{ State: "New Jersey", Candidate: "Obama", Electors: 14 },
{ State: "New Mexico", Candidate: "Obama", Electors: 5 },
{ State: "New York", Candidate: "Obama", Electors: 29 },
{ State: "North Carolina", Candidate: "Romney", Electors: 15 },
{ State: "North Dakota", Candidate: "Romney", Electors: 3 },
{ State: "Ohio", Candidate: "Obama", Electors: 18 },
{ State: "Oklahoma", Candidate: "Romney", Electors: 7 },
{ State: "Oregon", Candidate: "Obama", Electors: 7 },
{ State: "Pennsylvania", Candidate: "Obama", Electors: 20 },
{ State: "Rhode Island", Candidate: "Obama", Electors: 4 },
{ State: "South Carolina", Candidate: "Romney", Electors: 9 },
{ State: "South Dakota", Candidate: "Romney", Electors: 3 },
{ State: "Tennessee", Candidate: "Romney", Electors: 11 },
{ State: "Texas", Candidate: "Romney", Electors: 38 },
{ State: "Utah", Candidate: "Romney", Electors: 6 },
{ State: "Vermont", Candidate: "Obama", Electors: 3 },
{ State: "Virginia", Candidate: "Obama", Electors: 13 },
{ State: "Washington", Candidate: "Obama", Electors: 12 },
{ State: "West Virginia", Candidate: "Romney", Electors: 5 },
{ State: "Wisconsin", Candidate: "Obama", Electors: 10 },
{ State: "Wyoming", Candidate: "Romney", Electors: 3 }
]

```

## JAVASCRIPT

```

"use strict";
var layers= [
{
  shapeData: usMap,
  shapeDataPath: "State",
  shapePropertyPath: "name",
  dataSource: electionData,

```

```

shapeSettings:
{
strokeThickness: 0.5,
autoFill: false,
stroke: "white",
valuePath: "Electors",
colorValuePath: "Candidate",
colorMappings:
{
equalColorMapping:
[
{ value: "Romney", color: "#D84444" },
{ value: "Obama", color: "#316DB5" }
]
}
}
}];
ReactDOM.render(
<EJ.Map id="map1" layers = {layers}></EJ.Map>,
document.getElementById('maps')
);

```



### Color Palette

#### AutoFill

When **autoFill** property is set to true, shapes are filled with default colors from built-in palettes or custom palette.

#### JAVASCRIPT

```

"use strict";
var layers= [
{
shapeData: usMap,
shapeSettings: {
strokeThickness: 0.5,
stroke: "white",
autoFill: true
}
}
];
ReactDOM.render(
<EJ.Map id="map1" layers = {layers}></EJ.Map>,
document.getElementById('maps')
);

```



#### Color Palette

The **colorPalette** property determines whether the auto fill colors are fetched from built-in color palettes or custom palette.

The `colorPalette` property can be set with `palette1`, `palette2`, `palette3` and `custompalette` values where `palette1`, `palette2` and `palette3` are built-in color palettes and default value for this property is “`palette1`”.

The `customPalette` property is used to set an array of colors to be auto filled in shapes.

This property is enabled only when `colorPalette` property value is set to “`custompalette`”.

#### JAVASCRIPT

```
"use strict";
var layers= [
{
  ShapeSettings: {
    // ...
    autoFill: true,
    colorPalette: "custompalette",
    customPalette: ["#E51400", "#A4C400", "#730202", "#008B00", "#EF6535",
    "#1BA0E2", "#C63477", "#0050EF", "#BF004D", "#AA00FF"]
    // ...
  }
}];
ReactDOM.render(
<EJ.Map id="map1" layers = {layers}></EJ.Map>,
document.getElementById('maps')
);
```



#### Tooltip

The tooltip is displayed only when you set `showTooltip` to “`true`” in the shape layers. By default, it takes the property of the bound object that is referred in the `valuePath` and displays its content on hovering the corresponding shape.

#### JAVASCRIPT

```
"use strict";
var layers = [
{
  // ...
  ShapeSettings: {
    // ...
    valuePath: "name",
    // ...
  }
  showTooltip: true
}];
ReactDOM.render(
<EJ.Map id="map1" layers = {layers}></EJ.Map>,
document.getElementById('maps')
);
```



### Tooltip Template

The `tooltipTemplate` property is used for customizing the template for tooltip.

#### JAVASCRIPT

```
"use strict";
var layers = [
{
  // ...
  ShapeSettings: {
    // ...
    valuePath: "name",
    // ...
  }
  showTooltip: ,
  tooltipTemplate: 'myTooltip'
}];
ReactDOM.render(
  <EJ.Map id="map1" layers = {layers}></EJ.Map>,
  document.getElementById('maps')
);
```

#### HTML

```
<div id="myTooltip" style="display: none;">
<div >
<div style="height:60px;width:120px;background:#4586a0;border-radius:3px;">
<div style="margin-top:-20px;margin-left:9px;padding-top:3px">
<label style="margin-top:-20px;font-weight:normal;font-size:12px;color:white;font-family:Segoe UI;">{{:name}}</label>
</div>
<div style="height:10px;"></div>
<div style="margin-top:-10px;margin-left:9px;">
<label style="margin-top:-10px;font-weight:normal;font-size:14px;color:white;font-family:segoe ui light;">{{:population}}</label>
</div>
</div>
</div>
</div>
```

The following screenshot illustrates a map control displaying a Tooltip with template.



### Customize map background

The Map background can be customized by using the `background` property of the Map.

#### JAVASCRIPT

```
ReactDOM.render(
  <EJ.Map id="map1" background="blue"></EJ.Map>,
  document.getElementById('maps')
);
```

### Base Map Index

Specifies the index of the map to determine the shape layer to be displayed, you can use `baseMapIndex` property and the default value is 0.

#### JAVASCRIPT

```
ReactDOM.render(  
  <EJ.Map id="map1" baseMapIndex={0}></EJ.Map>,  
  document.getElementById('maps')  
) ;
```

### Center Position

Specify the `centerPosition` where map should be displayed

#### JAVASCRIPT

```
ReactDOM.render(  
  <EJ.Map id="map1" centerPosition=""></EJ.Map>,  
  document.getElementById('maps')  
) ;
```

### Label Settings

The `labelSettings` defines the basic customization settings of labels in the map.

The below properties are used for `labelSettings`

- `enableSmartLabel` - enable or disable the `enableSmartLabel` property.
- `labelLength` - set the `labelLength` property.
- `labelPath` - set the `labelPath` property.
- `showLabels` - The property specifies whether to show labels or not
- `smartLabelSize` - set the `smartLabelSize` property.

#### JAVASCRIPT

```
var mapLayers= [{  
  // ...  
  LabelSettings: {  
    enableSmartLabel: "true",  
    labelLength: '',  
    labelPath: '',  
    showLabels: 'true',  
    smartLabelSize: ''  
  },  
  showTooltip: true  
}];  
ReactDOM.render(  
  <EJ.Map id="map1" layers={mapLayers}></EJ.Map>,  
  document.getElementById('maps')  
) ;
```



## Map Elements

Map control contains a set of map elements such as shapes, bubbles, markers, legend, labels and data items that can be visualized with the customized appearance showing additional information on the map by using the bound data.

### Markers

Markers are notes used to leave some message on the map.

There are two ways to set marker for map.

1. Marker and marker template
2. Adding marker objects to map.

### Markers and Marker Template

The `markers` property has a list of objects that contains the data for Annotation. By default, it displays the bound data at the specified latitude and longitude. The `markerTemplate` property is used for customizing the template for markers.

### JAVASCRIPT

```
use strict;
var markers = [
  { latitude: 37.0000, longitude: -120.0000, city: "California" },
  { latitude: 40.7127, longitude: -74.0059, city: "New York" },
  { latitude: 42, longitude: -93, city: "Iowa" }
];
var layers = [{
  // ...
  markers: markers,
  markerTemplate: 'template'
}];
ReactDOM.render(
  <EJ.Map id="map1" layers = {layers}></EJ.Map>,
  document.getElementById('maps')
);
<div id="template" style="display: none;">
<div>
<div style="background-
image:url(http://js.syncfusion.com/demos/web/Images/map/pin.png);margin-
left:3px;height:40px;width:25px;margin-top:-15px;">
</div>
</div>
</div>
```

![[/js/Maps/Map-Elementsimages/Map-Elementsimg1.png]

### Adding Marker objects to the map

Without datasource, n number of markers can be added to shape layers with `markers` property. Each marker object contains the following list of properties.

- `label` - Text that displays some information about the annotation in text format.
- `latitude` - Latitude point determine the Y-axis position of annotation.
- `longitude` - Longitude point determine the X-axis position of annotation.

**JAVASCRIPT**

```

var markers = [
  { latitude: 37.0000, longitude: -120.0000, city: "California" },
  { latitude: 40.7127, longitude: -74.0059, city: "New York" },
  { latitude: 42, longitude: -93, city: "Iowa" }
];
var layers = [{
  // ...
  markers: markers,
  markerTemplate: 'template'
}];
ReactDOM.render(
  <EJ.Map id="map1" layers = {layers}></EJ.Map>,
  document.getElementById('maps')
);

```

**HTML**

```

<div id="template" style="display: none;">
<div>
<div style="margin-left:8px;height:45px;width:120px;margin-top:-23px;">
<label class="label1" style="color:black;margin-left:15px;font-
weight:normal">{{:city}}</label>
</div>
</div>
</div>

```



**Bubbles**

Bubbles in the **Maps** control represent the underlying data values of the map. Bubbles are scattered throughout the map shapes that contain bound values.

Bubbles are included when data binding and the `bubbleSettings` is set to the shape layers.

**Properties**

Property	Type	Description
maxValue	String	Get or sets the maximum height and width of the bubble.
minValue	String	Gets or sets the minimum height and width of the bubble.
colorValuePath	String	Get or sets the field value that is to be fetched from data for each bubble used for determining the bubble color.
valuePath	String	Gets or sets the field value that is to be fetched from data for each bubble.
colorMappings	Collection of RangeColorMapping	Gets or sets the tree map colors.

Color	String	Gets or sets the fill color for bubbles.
showTooltip	Boolean	Enable or disable the tooltip for bubbles.
tooltipTemplate	String	Gets or sets the tooltip template for bubbles.

### Add Bubbles to the Map

To add bubbles to a map, the bubble marker setting is added to the shape file layer. Create the Model and ViewModel as illustrated in the Data Binding topic and add the following code. Also set the `maxValue`, `minValue`, and `valuePath` properties as illustrated in the following code sample.

**Note:** Tooltip and Color Mappings for bubble is to be set as similar to the tooltip and color mappings set in the layers and shapeSettings. For more details, refer to the Tooltip and Color Mappings section.

### JAVASCRIPT

```
"use strict";
var layers= [
{
  shapeData: usMap,
  shapeDataPath: "name",
  shapePropertyPath: "name",
  dataSource: [
    { name: "California", population: "38332521" },
    { name: "New York", population: "19651127" },
    { name: "Iowa", population: "3090416" }
  ],
  enableMouseHover: true,
  shapeSettings: {
    fill: "#9CBF4E",
    strokeThickness: "0.5",
    stroke: "White"
  },
  bubbleSettings: {
    showBubble: true,
    minValue: "20",
    maxValue: "40",
    color: "#C99639",
    valuePath: "population"
  }
}];
var markers = [
{ latitude: 37.0000, longitude: -120.0000, city: "California" },
{ latitude: 40.7127, longitude: -74.0059, city: "New York" },
{ latitude: 42, longitude: -93, city: "Iowa" }
];
ReactDOM.render(
<EJ.Map id="map1" layers={layers}></EJ.Map>,
document.getElementById('maps')
);
```



## Legend

A legend is a key used on a map that contains swatches of symbols with descriptions. It provides valuable information for interpreting what the map is displaying and can be represented in various colors, shapes or other identifiers based on the data. It gives a breakdown of what each symbol represents throughout the map.

### Visibility

The Legends can be made visible by setting the `showLegend` property of `legendSettings` to true.

### Positioning of the Legend

The legend can be positioned in two ways.

1. Absolute Position.
2. Dock Position.

#### Absolute Position

Based on the margin values of X and Y-axes, the Map legends can be positioned with the support of `positionX` and `positionY` properties available in `legendSettings`. For positioning the legend based on margins corresponding to a map, `position` value is set as `'none'`.

#### Dock Position

The map legends can be positioned in following locations within the container.

1. `topLeft`
2. `topCenter`
3. `topRight`
4. `centerLeft`
5. `center`
6. `centerRight`
7. `bottomLeft`
8. `bottomRight`
9. `bottomCenter`
10. `bottomRight`
11. `none`

You can set this option by using `position` property in `legendSettings`.

### Legend Size

The map legend size can be modified by using the `height` and `width` properties in `legendSettings`.

### Legend for Shapes

The Layer shape type legends can be generated for each color mappings in shape settings.

**Note:** Here, Equal Color Mapping code sample for `shapeSettings` with color mappings is referred.

## JAVASCRIPT

```
"use strict";
var layers = [{
  // ...
  legendSettings:{
    showLegend:true,
```

```
position:"bottomleft",
height: 30,
width: 70,
},
// ...
}];
ReactDOM.render(
<EJ.Map id="map1" layers = {layers}></EJ.Map>,
document.getElementById('maps')
);
```



#### Interactive Legend

The legends can be made interactive with an arrow mark indicating the exact range color in the legend when the mouse hovers over the corresponding shapes. You can enable this option by setting `mode` property in `legendSettings` value as “interactive” and default value of `mode` property is “default” to enable the normal legend.

#### Title for Interactive Legend

You can provide the title for interactive legend by using `title` property in `legendSettings`.

#### Label for Interactive Legend

You can provide the left and right labels to interactive legend by using `leftLabel` and `rightLabel` properties in `legendSettings`.

**Note:** Here, Range Color Mapping code snippet for `shapeSettings` with color mappings is referred.

### JAVASCRIPT

```
"use strict";
var layers = [{
// ...
legendSettings: {
showLegend: true,
dockOnMap:true,
height: 15,
width: 150,
position: "topleft",
mode: "interactive",
title: "Population",
leftLabel: "0.5M",
rightLabel: "40M"
},
// ...
}];
ReactDOM.render(
<EJ.Map id="map1" layers = {layers}></EJ.Map>,
document.getElementById('maps')
);
```



### Bubble Legend

A bubble legend feature is used to provide the key (legend) for another map element bubble. You can activate the Bubble legend by setting the enum `type` in `legendSettings` as “bubble” and this enables you to easily identify what value a particular bubble is representing.

### JAVASCRIPT

```
"use strict";
var layers = [{
  legendSettings: {
    // ...
    type: "bubbles",
    // ...
  },
  bubbleSettings: {
    showBubble: true,
    valuePath: "population",
    minValue: 20,
    maxValue: 40,
    colorMappings:
    {
      rangeColorMapping:
      [
        {
          from: 500000,
          to: 1000000,
          color: "#9CBF4E",
          range: 10688
        },
        {
          from: 1000001,
          to: 5000000,
          color: "#45D6BD",
          range: 19390
        },
        {
          from: 5000001,
          to: 10000000,
          color: "#FF567C",
          range: 18718
        },
        {
          from: 10000001,
          to: 40000000,
          color: "#470F52",
          range: 30716
        }
      ],
    }
  },
  shapeData: usMap
}];
ReactDOM.render(
  <EJ.Map id="map1" layers = {layers}></EJ.Map>,
  document.getElementById('maps')
);
```



## User Interaction

Options like zooming, panning, and map selection enables the effective interaction on Map elements.

### Map Selection

Each shape in the Map can be selected and deselected during interaction with the shapes.

The `selectionColor` property is used to get or set the selected shape color. The `selectionStroke` and `selectionStrokeWidth` properties are used to customize the selected shape border.

You can select the shape by tapping the shape. The Single selection is enabled by the `enableSelection` property of shape layer. When `enableSelection` is set to false, the shapes cannot be selected.

### JAVASCRIPT

```
"use strict";
var layers = [{
  shapeData: usMap,
  shapeSettings: {
    fill: "#9CBF4E",
    strokeThickness: "0.5",
    stroke: "White",
    selectionStrokeWidth: 2,
    selectionStroke: "white",
    selectionColor: "#BC5353"
  },
  enableSelection: true
}];
ReactDOM.render(
  <EJ.Map id="map1" layers = {layers}></EJ.Map>,
  document.getElementById('maps')
);
```

{% include image.html url="/js/Maps/User-Interactionimages/User-Interactionimg1.png"%}

### MultiSelection

This feature enables you to select multiple Map shapes on mouse taps accompanied by the "Control" key press. To enable this feature, set the `selectionMode` property as "multiple" along with the `enableSelection` property.

### JAVASCRIPT

```
"use strict";
var layers = [{
  shapeData: usMap,
  // ...
  enableSelection: true,
  selectionMode: "multiple",
  // ..
}];
ReactDOM.render(
  <EJ.Map id="map1" layers = {layers}></EJ.Map>,
  document.getElementById('maps')
);
```

```
);
```

```
{% include image.html url="/js/Maps/User-Interactionimages/User-Interactionimg5.png"%}
```

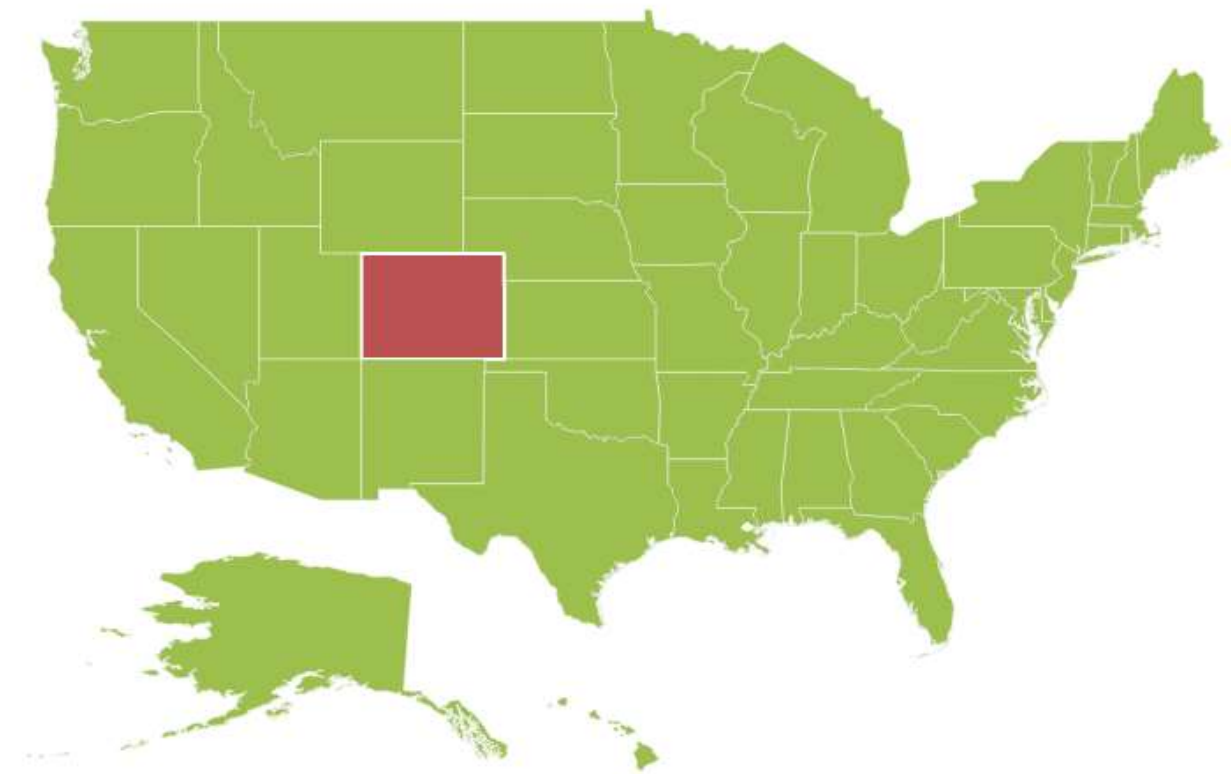
### Dragging On Selection

This feature enables you to select the shapes by dragging over the shapes. While dragging over the shapes, a rectangle is generated and the shapes that comes within the rectangle is selected.

You can enable this feature by setting the `draggingOnSelection` property in the `layers` to `true`.

### JAVASCRIPT

```
"use strict";
var layers = [{
  // ...
  draggingOnSelection: true
  // ...
}];
ReactDOM.render(
  <EJ.Map id="map1" layers = {layers}></EJ.Map>,
  document.getElementById('maps')
);
```



### Zooming

The zooming feature enables you to zoom in and out the Map to show in-depth information. It is controlled by the `level` property of the Map. When the zoom level of the Map control is increased, the Map is zoomed in. When the zoom level is decreased, then the Map is zoomed out.



### Properties

The following properties are related to the zooming feature of the **Maps** control:

1. level
2. enableZoom
3. minValue
4. maxValue

### Level

The **level** property determines the Map's scale size when zooming. The default value of **level** is 1.

**Note:** level cannot be less than 1

### EnableZoom

The **enableZoom** property enables or disables the zooming feature.

### MinValue

The **minValue** property is used to set the minimum zoom level of the Map.

### MaxValue

The **maxValue** property is used to set the maximum zoom level of the Map.

### JAVASCRIPT

```
"use strict";
var layers= [
{
  shapeData: usMap,
  shapeSettings: {
    fill: "#9CBF4E",
    strokeThickness: "0.5",
    stroke: "White",
    selectionStrokeWidth: 2,
    selectionStroke:"white",
    selectionColor: "#BC5353"
  },
}];
var zoomSettings = {
  enableZoom: true,
  minValue: 1,
  maxValue: 20,
  level: 1
};
ReactDOM.render(
  <EJ.Map id="map1" layers = {layers} zoomSettings = {zoomSettings}</EJ.Map>,
  document.getElementById('maps')
);
```

### Additional Options to Zoom the Map

Maps can be zoomed by using the following options also,

- Zoom method.
- mouse scroll.
- mouse double tap.

- shape selection
- Position

### *Zoom method*

You can zoom the Maps by using `zoom` method. The `zoom` method contains parameter zoom value. The Map can be zoomed or scaled based on zoom value parameter.

### **JAVASCRIPT**

```
$("#map").ejMap("zoom", 2);
```

### *Mouse scroll*

You can zoom the Map with mouse events by using mouse scroll. When the mouse is scrolled up, the Map is zoomed in and when the mouse is scrolled down, the Map is zoomed out.

### *Mouse double tap*

When the Map is double-tapped by using mouse, the zoom in operation is performed.



### *Shape Selection*

Map shape is zoomed to the whole Map area on the shape selected. Animation can be applied for that zooming by using the `enableAnimation` property as true.

You can enable this feature by setting `enableZoomOnSelection` property value as 'true'.

When `enableZoomOnSelection` property is set to true, then zooming by double click is muted.

### **JAVASCRIPT**

```
"use strict";
var layers= [
{
```

```

shapeData: usMap,
shapeSettings: {
  fill: "#9CBF4E",
  strokeThickness: "0.5",
  stroke: "White",
  selectionStrokeWidth: 2,
  selectionStroke: "white",
  selectionColor: "#BC5353"
},
}];
var zoomSettings = {
  enableZoomOnSelection: true
};
ReactDOM.render(
  <EJ.Map id="map1" layers = {layers} zoomSettings = {zoomSettings}</EJ.Map>,
  document.getElementById('maps')
);

```

### Positioning

Depending on the latitude and longitude, you can zoom the Map to the exact position. All locations are considered as latitude and longitude values and the exact location is considered as Map coordinates.

The `navigateTo` is a method that allows you to zoom the Map control to the given location. This method contains three attributes as follows.

Attribute	Type	Description
Latitude	Double	Latitude point of the position
Longitude	Double	Longitude point of the position
level	Double	Zoom level of the map

### HTML

```

<script type="text/javascript">
function buttonClick() {
  $("#map").ejMap("navigateTo", 13, 80, 5);
}
</script>

```

### Panning

The panning feature enables the Map navigation. The `enablePan` property is used to enable or disable the panning support.

### JAVASCRIPT

```

"use strict";
var layers= [
{
  shapeData: usMap,
  shapeSettings: {
    fill: "#9CBF4E",
    strokeThickness: "0.5",

```

```
stroke: "White",
selectionStrokeWidth: 2,
selectionStroke: "white",
selectionColor: "#BC5353"
},
enablePan: true
});
ReactDOM.render(
  <EJ.Map id="map1" layers={layers} ></EJ.Map>,
  document.getElementById('maps')
);
```

### Factor

Specifies the zoom factor for map zoom value, you can use `factor` property.

### JAVASCRIPT

```
ReactDOM.render(
  <EJ.Map id="map1" factor={1} ></EJ.Map>,
  document.getElementById('maps')
);
```

### Navigation Control

**Navigation** control is built-in with **Maps** control. With Navigation control, **Maps** can be panned in any direction and zoomed. It is possible to show or hide the NavigationControl by `enableNavigation` property.



### JAVASCRIPT

```
"use strict";
var layers= [
{
```

```

shapeData: usMap,
shapeSettings: {
  fill: "#9CBF4E",
  strokeThickness: "0.5",
  stroke: "White",
  selectionStrokeWidth: 2,
  selectionStroke: "white",
  selectionColor: "#BC5353"
},
enablePan: true
}];
var navigationControl = {enableNavigation: true};
ReactDOM.render(
  <EJ.Map id="map1" layers={layers} navigationControl = {navigationControl}
  ></EJ.Map>,
  document.getElementById('maps')
);

```

### Positions

The Navigation control can be positioned in two ways.

- Absolute Position
- Dock Position

#### Absolute Position

Based on the margin values of X and Y-axes, the navigation control can be positioned with the help of the **x** and **y** properties available in **absolutePosition**. For positioning the navigation control based on margins corresponding to a Map, **dockPosition** value is set as *'none'*.

#### Dock Position

The navigation control can be positioned in the following locations within the container.

- topLeft
- topCenter
- topRight
- centerLeft
- center
- centerRight
- bottomLeft
- bottomRight
- bottomCenter
- bottomRight
- none

You can set this option by using **dockPosition** property in **navigationControl**.

### JAVASCRIPT

```

"use strict";
var layers= [
{
  shapeData: usMap,

```

```

shapeSettings: {
  fill: "#9CBF4E",
  strokeThickness: "0.5",
  stroke: "White",
  selectionStrokeWidth: 2,
  selectionStroke: "white",
  selectionColor: "#BC5353"
},
enablePan: true
}];
var navigationControl = {
  enableNavigation: true,
  orientation: 'vertical',
  absolutePosition: {
    x: 5,
    y: 16
  },
  dockPosition: 'none'
};
ReactDOM.render(
  <EJ.Map id="map1" layers = {layers}
  navigationControl = {navigationControl} ></EJ.Map>,
  document.getElementById('maps')
);

```

### Orientation

Set the **orientation** value for navigation control.

### JAVASCRIPT

```

var navigationControl = {
  enableNavigation: true,
  orientation: 'vertical',
  absolutePosition: {
    x: 5,
    y: 16
  },
  dockPosition: 'none'
};
ReactDOM.render(
  <EJ.Map id="map1"
  navigationControl = {navigationControl} ></EJ.Map>,
  document.getElementById('maps')
);

```

### Content

Specifies the navigation control template for map, you can use **content** property.

### HTML

```

var navigationControl = {
  enableNavigation: true,
  orientation: 'vertical',
  absolutePosition: {
    x: 5,
    y: 16
  }
};

```

```
},
dockPosition: 'none',
content: ''
};
ReactDOM.render(
  <EJ.Map id="map1"
  navigationControl = {navigationControl} ></EJ.Map>,
  document.getElementById('maps')
);
```

### Animation

**Animation** is enabled or disabled using `enableAnimation` property.

#### JAVASCRIPT

```
ReactDOM.render(
  <EJ.Map id="map1"
  enableAnimation = {true} ></EJ.Map>,
  document.getElementById('maps')
);
```

### Enable Layer Change Animation

Enables or Disables the animation for layer change in map, you can use `enableLayerChangeAnimation` property and the default value is false.

#### JAVASCRIPT

```
ReactDOM.render(
  <EJ.Map id="map1"
  enableLayerChangeAnimation = {true} ></EJ.Map>,
  document.getElementById('maps')
);
```

### Responsiveness during browser resize

**Map** is made responsive when resizing the browser by using `isResponsive` property.

#### JAVASCRIPT

```
ReactDOM.render(
  <EJ.Map id="map1"
  isResponsive = {true} ></EJ.Map>,
  document.getElementById('maps')
);
```

### Layers

Map is maintained through `layers` and it can accommodate one or more layers.

### Multilayer

The Multilayer support allows you to load multiple shape files in a single container, enabling maps to display more information.

### Adding Multiple Layers in the Map

The shape layers is the core layer of the map. The multiple layers can be added in the shape layers as `subLayers` within the shape layers.

### SubLayer

The subLayer is the collection of shape layers.

In this example, World Map shape is used as shape data by utilizing the “**WorldMap.json**” file in the following folder structure obtained from downloaded Maps\_GeoJSON folder.

```
..\Maps_GeoJSON\
```

You can assign the complete contents in “**WorldMap.json**” file to new JSON object. For better understanding, a JS file “**WorldMap.js**” is already created to store JSON data in JSON object “usMap”

[usa.js]

### JAVASCRIPT

```
var world_map = //Paste all the content copied from the JSON file//
```

### JAVASCRIPT

```
"use strict";
var layers= [{
  shapeData: world_map,
  shapeSettings: {
    fill: "#9CBF4E",
    strokeThickness: "0.5",
    stroke: "White"
  },
  subLayers: [{
    shapeData: usMap,
    shapeSettings: {
      fill: "orange",
      strokeThickness: "1",
      stroke: "White"
    }
  }]
}];
ReactDOM.render(
  <EJ.Map id="map1" layers = {layers} ></EJ.Map>,
  document.getElementById('maps')
);
```



### Map Providers

**Map** control support map providers such as OpenStreetMap that can be added to any layers in maps.

### Open Street Map

OpenStreetMap is a map of the entire world. The OpenStreetMap allows you to view, edit and use geographical data in a collaborative way from any place on the Earth.

### Enable OSM

You can enable this feature by setting the `layerType` property value as "OSM".



**JAVASCRIPT**

```
"use strict";
var layers= [{
  layerType: 'osm',
  urlTemplate: 'http://a.tile.openstreetmap.org/level/tileX/tileY.png'
}];
ReactDOM.render(
  <EJ.Map id="map1" layers = {layers} ></EJ.Map>,
  document.getElementById('maps')
);
```

*URL Template*

The `urlTemplate` property determines the format of tile map. You can specify the template for the tile layer.



*Bing Map*

Bing Map is a key feature in accessing the external geospatial imagery services for deep-zoom satellite view.

*Enable Bing Maps*

You can enable this feature by defining the `layerType` as “bing”.

**JAVASCRIPT**

```
"use strict";
var layers= [{
  layerType: 'bing',
  bingMapType: "AerialWithLabel",
  key: '// ...bingMapKey'
}];
ReactDOM.render(
  <EJ.Map id="map1" layers = {layers} ></EJ.Map>,
  document.getElementById('maps')
);
```

*Key*

The bing Map key is provided as input to this `key` property. The Bing Map key can be obtained from <http://www.microsoft.com/maps/create-a-bing-maps-key.aspx>.



*Methods**navigateTo(latitude, longitude, level)*

Method for navigating to specific shape based on latitude, longitude and zoom level.

**HTML**

```
<div id="map"></div>
```

**JAVASCRIPT**

```
ReactDOM.render(
```

```
<EJ.Map id="default"></EJ.Map>,
document.getElementById('map')
);
function MapMethod() {
var mapObj = $("#default").data("ejMap");
mapObj.navigateTo();
};
```

*pan(direction)*

Method to perform map panning

#### HTML

```
<div id="map"></div>
```

#### JAVASCRIPT

```
ReactDOM.render (
<EJ.Map id="default"></EJ.Map>,
document.getElementById('map')
);
function MapMethod() {
var mapObj = $("#default").data("ejMap");
mapObj.pan();
};
```

*refresh()*

Method to reload the map.

#### HTML

```
<div id="map"></div>
```

#### JAVASCRIPT

```
ReactDOM.render (
<EJ.Map id="default"></EJ.Map>,
document.getElementById('map')
);
function MapMethod() {
var mapObj = $("#default").data("ejMap");
mapObj.refresh();
};
```

*refreshLayers()*

Method to reload the shapeLayers with updated values

#### HTML

```
<div id="map"></div>
```

#### JAVASCRIPT

```
ReactDOM.render (
```

```
<EJ.Map id="default"></EJ.Map>,
document.getElementById('map')
);
function MapMethod() {
var mapObj = $("#default").data("ejMap");
mapObj.refreshLayers();
};
```

*refreshNavigationControl(navigation)*

Method to reload the navigation control with updated values.

#### HTML

```
<div id="map"></div>
```

#### JAVASCRIPT

```
ReactDOM.render (
<EJ.Map id="default"></EJ.Map>,
document.getElementById('map')
);
function MapMethod() {
var mapObj = $("#default").data("ejMap");
mapObj.refreshNavigationControl();
};
```

*zoom(level, isAnimate)*

Method to perform map zooming.

#### HTML

```
<div id="map"></div>
```

#### JAVASCRIPT

```
ReactDOM.render (
<EJ.Map id="default"></EJ.Map>,
document.getElementById('map')
);
function MapMethod() {
var mapObj = $("#default").data("ejMap");
mapObj.zoom();
};
```

#### Events

*markerSelected*

Triggered on selecting the map markers.

#### HTML

```
<div id="map"></div>
```

#### JAVASCRIPT

```
ReactDOM.render(  
  <EJ.Map id="default" markerSelected = {MarkerSelected}></EJ.Map>,  
  document.getElementById('map')  
)  
;  
function MarkerSelected() {  
  // Do Something  
};
```

### *mouseleave*

Triggers while leaving the hovered map shape

#### **HTML**

```
<div id="map"></div>
```

#### **JAVASCRIPT**

```
ReactDOM.render(  
  <EJ.Map id="default" mouseleave = {Mouseleave}></EJ.Map>,  
  document.getElementById('map')  
)  
;  
function Mouseleave() {  
  // Do Something  
};
```

### *mouseover*

Triggers while hovering the map shape.

#### **HTML**

```
<div id="map"></div>
```

#### **JAVASCRIPT**

```
ReactDOM.render(  
  <EJ.Map id="default" mouseover = {Mouseover}></EJ.Map>,  
  document.getElementById('map')  
)  
;  
function Mouseover() {  
  // Do Something  
};
```

### *onRenderComplete*

Triggers once map render completed.

#### **HTML**

```
<div id="map"></div>
```

#### **JAVASCRIPT**

```
ReactDOM.render(  
  <EJ.Map id="default" onRenderComplete = {OnRenderComplete}></EJ.Map>,  
  document.getElementById('map')
```

```
);  
function OnRenderComplete() {  
  // Do Something  
};
```

### *panned*

Triggers when map panning ends.

### **HTML**

```
<div id="map"></div>
```

### **JAVASCRIPT**

```
ReactDOM.render(  
  <EJ.Map id="default" panned = {Panned}></EJ.Map>,  
  document.getElementById('map')  
);  
function Panned() {  
  // Do Something  
};
```

### *shapeSelected*

Triggered on selecting the map shapes.

### **HTML**

```
<div id="map"></div>
```

### **JAVASCRIPT**

```
ReactDOM.render(  
  <EJ.Map id="default" shapeSelected = {ShapeSelected}></EJ.Map>,  
  document.getElementById('map')  
);  
function ShapeSelected() {  
  // Do Something  
};
```

### *zoomedIn*

Triggered when map is zoomed-in.

### **HTML**

```
<div id="map"></div>
```

### **JAVASCRIPT**

```
ReactDOM.render(  
  <EJ.Map id="default" zoomedIn = {ZoomedIn}></EJ.Map>,  
  document.getElementById('map')  
);  
function ZoomedIn() {  
  // Do Something  
};
```

```
};
```

### zoomedOut

Triggers when map is zoomed out.

### HTML

```
<div id="map"></div>
```

### JAVASCRIPT

```
ReactDOM.render(
  <EJ.Map id="default" zoomedOut = {ZoomedOut}></EJ.Map>,
  document.getElementById('map')
);
function ZoomedOut() {
  // Do Something
};
```

[Copyright &#169; 2001 - 2015 Syncfusion Inc. All Rights Reserved](http://www.syncfusion.com/copyright)

## MaskEdit

### Getting Started

This section helps to get started of the MaskEdit component in a React application

#### Create a MaskEdit

Refer the common React Getting Started Documentation to create an application and add necessary scripts and styles for rendering our ReactJS components.

Create a JSX file and use <EJ.MaskEdit> syntax to render React MaskEdit component. Add required properties to <EJ.MaskEdit> tag element.

### HTML

```
ReactDOM.render(
  <EJ.MaskEdit>
</EJ.MaskEdit>,
  document.getElementById('mask')
);
```

Define an HTML element for adding MaskEdit in the application and refer the JSX file created.

### HTML

```
<div id="mask"></div>
<script type="text/babel" src="sample.jsx">
```

This will render an empty MaskEdit component on executing.


### Configure Properties

In the JSX, need to declare the MaskEdit properties. Refer to the following code,.

## HTML

```
ReactDOM.render(  
  <EJ.MaskEdit value="4242422424" maskFormat="99 999-99999">  
  </EJ.MaskEdit>,  
  document.getElementById('mask')  
) ;
```

Run the above code to render the following output,



*Note: You can find the MaskEdit properties from the [API reference](#) document.*

## Menu

### Overview

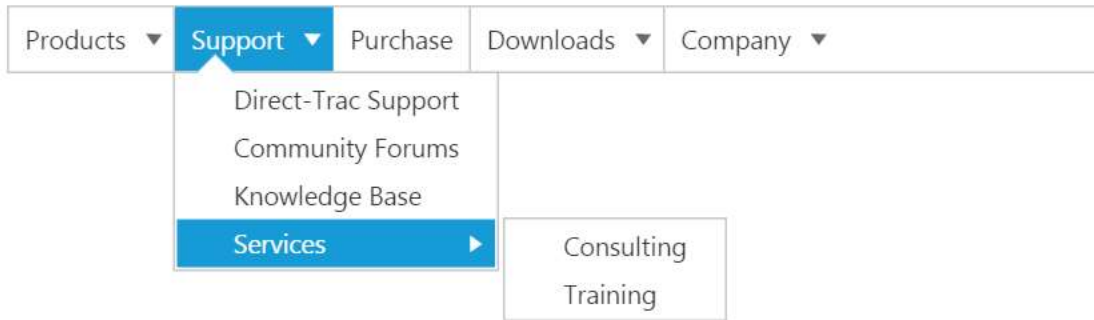
The **Menu** control supports displaying a **Menu** created from list items. The **Menu** is based on a hierarchy of **UL** and **LI** elements where the list items are rendered as sub-menu items.

### Key Features

- **UL/LI template:** Requires only **UL** and **LI** list of items as input.
- **Orientation:** Vertical and horizontal orientation support.
- **Context menu:** A **Menu** that is displayed wherever you right-click on the page or on the control.
- **Keyboard shortcuts:** Allows easy access to **Menu** items through shortcut keys.
- **Data binding:** Supports data binding **Menu** items as remote or local data.
- **Theme:** Essential JavaScript controls feature 12 built-in themes, six flat and six with gradient effects, and also supports custom skin options for user-defined themes.
- **Keyboard navigation:** You can interact with the **Menu** control using the keyboard.
- **RTL support:** This feature allows text in **Menu** items to be displayed from right to left.
- **Center Menu:** This feature allows you to center-align root **Menu** items.
- Separators and open-on-click support.

### Getting Started

This section explains briefly about how to create a **Menu** control in your application with **JavaScript**. The **Essential JavaScript Menu** supports displaying a **Menu** of list-out items. This **Menu** is based on ul-li hierarchy, where the sub-list items are rendered as the sub-menu items. The **Menu** control can also be rendered with local and remote data source. From the following guidelines, you can learn how to customize the **Menu** control for a website. In this case, **Syncfusion's** website **Menu** is discussed. The following screenshot displays the appearance of **Menu**.



### Create a Menu in React JS

**Essential JavaScript Menu** widgets are basically provided with built-in features like keyboard navigation, show and hide **Menu** items with animations, and flexible API's. From the following guidelines, you can learn how to render **Menu** control with Remote data source value.

You can create a React application and add necessary scripts and styles with the help of the given [React Getting Started Documentation](#).

Define an HTML element for adding Menu in the application and refer the JSX file.

#### HTML

```
<div id="menu-default"></div>
<script src="app/menu/default.js">
```

Create a JSX file for rendering Menu component using <EJ.Menu> syntax. Add required properties to it in <EJ.Menu> tag element

#### JAVASCRIPT

```
"use strict";
ReactDOM.render(
  <div className="imgframe">
    <EJ.Menu width="100%">
    </EJ.Menu>
  </div>,
  document.getElementById('menu-default')
);
```

Output of the above steps.

### Configure parent Menu items

Every **Menu** has a list of **Menu** items with list of sub level **Menu** items. From the following guidelines, you can learn how to initialize the root level elements of **Menu** control with Local data source value. Initialize the **Menu** with data source value as illustrated in the following code example.

#### HTML

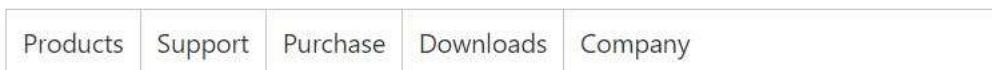


```
<div id="menu-default"></div>
<script src="app/menu/default.js">
```

## JAVASCRIPT

```
"use strict";
var data = [
{ id: 1, text: "Products", parentId: null },
{ id: 2, text: "Support", parentId: null },
{ id: 3, text: "Purchase", parentId: null },
{ id: 4, text: "Downloads", parentId: null },
{ id: 5, text: "Company", parentId: null }];
ReactDOM.render(
<div className="imgframe">
<EJ.Menu fields-dataSource={data} fields-id="id" fields-text="text" fields-
parentId="parentId" width="500px">
</EJ.Menu>
</div>,
document.getElementById('menu-default')
);
```

The following screenshot displays output.



## Initialize sub-level Menu items

Every **Menu** items have a list of sub level **Menu** items. From the following guidelines, you can learn how to initialize the sub level items of **Menu** control. The **parentId** field is used to map root level **Menu** item to its sub level **Menu** item.

The following code example describes how to initialize first level sub menu items of product **Menu** item.

## JAVASCRIPT

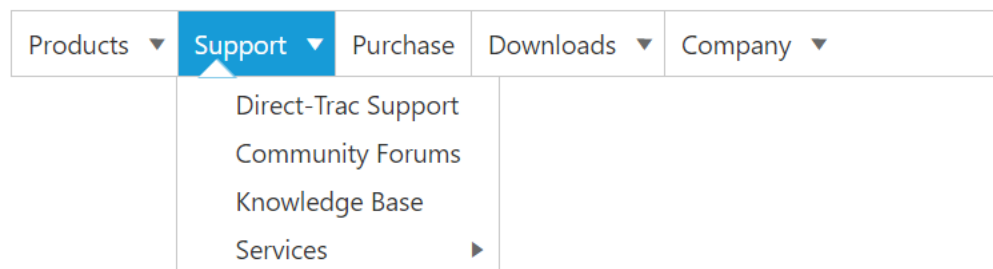
```
"use strict";
var data = [
{ id: 1, text: "Products", parentId: null },
{ id: 2, text: "Support", parentId: null },
{ id: 3, text: "Purchase", parentId: null },
{ id: 4, text: "Downloads", parentId: null },
{ id: 5, text: "Company", parentId: null },
//first level child
{ id: 11, parentId: 1, text: "ASP.NET" },
{ id: 12, parentId: 1, text: "ASP.NET MVC" },
{ id: 13, parentId: 1, text: "Mobile MVC" },
{ id: 14, parentId: 1, text: "Silverlight" },
{ id: 15, parentId: 2, text: "Direct-Trac Support" },
{ id: 16, parentId: 2, text: "Community Forums" },
{ id: 17, parentId: 2, text: "Knowledge Base" },
{ id: 18, parentId: 2, text: "Services" },
{ id: 19, parentId: 4, text: "Evaluation" },
{ id: 20, parentId: 4, text: "Free E-Books" },
{ id: 21, parentId: 4, text: "Metro Studio" },
{ id: 22, parentId: 4, text: "Latest Version" },
```

```

{ id: 23, parentId: 5, text: "Technology Resource Portal " },
{ id: 24, parentId: 5, text: "Case Studies" },
{ id: 25, parentId: 5, text: "Bouchers & Datasheets" },
{ id: 26, parentId: 5, text: "FAQ" }
];
ReactDOM.render(
<div className="imgframe">
<EJ.Menu fields-dataSource={data} width="100%">
</EJ.Menu>
</div>,
document.getElementById('menu-default')
);

```

Execute the above code example to render the following output.



### Define multiple level Menu items

You can define the sub-menu items to multiple levels in **Menu** control. You need to specify the parent Id value to render sub level **Menu** item for the **Menu** items.

To initialize multiple levels sub menu items, use the following code example.

#### JAVASCRIPT

```

"use strict";
var data = [
{ id: 1, text: "Products", parentId: null },
{ id: 2, text: "Support", parentId: null },
{ id: 3, text: "Purchase", parentId: null },
{ id: 4, text: "Downloads", parentId: null },
{ id: 5, text: "Company", parentId: null },
//first level child
{ id: 11, parentId: 1, text: "ASP.NET" },
{ id: 12, parentId: 1, text: "ASP.NET MVC" },
{ id: 13, parentId: 1, text: "Mobile MVC" },
{ id: 14, parentId: 1, text: "Silverlight" },
{ id: 15, parentId: 2, text: "Direct-Trac Support" },
{ id: 16, parentId: 2, text: "Community Forums" },
{ id: 17, parentId: 2, text: "Knowledge Base" },
{ id: 18, parentId: 2, text: "Services" },
{ id: 19, parentId: 4, text: "Evaluation" },
{ id: 20, parentId: 4, text: "Free E-Books" },
{ id: 21, parentId: 4, text: "Metro Studio" },
{ id: 22, parentId: 4, text: "Latest Version" },
{ id: 23, parentId: 5, text: "Technology Resource Portal " },
{ id: 24, parentId: 5, text: "Case Studies" },
{ id: 25, parentId: 5, text: "Bouchers & Datasheets" },

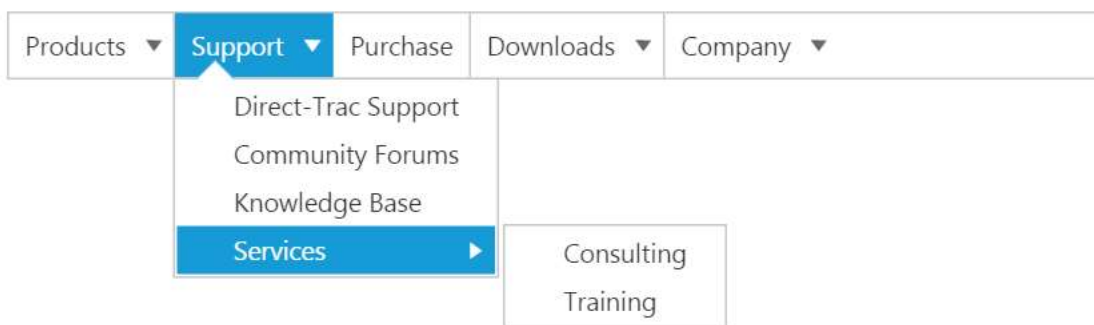
```

```

{ id: 26, parentId: 5, text: "FAQ" },
//second level child
{id: 111, parentId: 18, text: "Consulting" },
{ id: 112, parentId: 18, text: "Training" }
];
ReactDOM.render(
<div className="imgframe">
<EJ.Menu fields-dataSource={data} width="100%">
</EJ.Menu>
</div>,
document.getElementById('menu-default')
);

```

The following screenshot is the output.



By following the above mentioned steps, you can render the **Menu** control with multiple level sub items through online data source. You can simply customize the **Menu** widget in an efficient manner.

In summary of this getting started, you have now simulated the **Syncfusion** website **Menu** with **Essential JavaScript Menu**. You have utilized and learn the appearance customization etc.

By following the above mentioned steps, you can render the **Menu** control with multiple level sub items. You can simply customize the **Menu** in an efficient manner.

## Navigation Drawer

### Overview

Our Essential ReactJS Navigation Drawer component displays a sliding panel, which displays the list of view options. By default, the list of view options is not visible, but you can display it onto the left/right side of the window screen by clicking with the desired target icon.

### Key Features

- **Direction:** To change the list view options from the right or left side of the window screen for specifying the direction of the Navigation Drawer.
- **Type:** To change the transition for specifying the type of the Navigation Drawer.
- **Consider SubPage:** Specifies whether the Navigation Drawer to be rendered inside the closest sub page or not.
- **Position:** Specifies the position of the Navigation Drawer whether it's fixed or relative to the page.

## Getting Started

This section helps to get started with Essential ReactJS Navigation Drawer component.

### Create a Navigation Drawer

Refer the common ReactJS [Getting Started](#) Documentation to create an application and add necessary scripts and styles for rendering our ReactJS components.

Create a JSX file and use <EJ.NavigationDrawer> syntax to render React NavigationDrawer component. Add required properties to <EJ. NavigationDrawer > tag element.

### JAVASCRIPT

```
ReactDOM.render(  
  <EJ.NavigationDrawer id="navpanedefault"  
  </EJ.NavigationDrawer>,  
  document.getElementById('navigation')  
) ;
```

Define an HTML element for adding Navigation Drawer in the application and refer the JSX file created.

### HTML

```
<div id="navigation"></div>  
<script type="text/babel" src="sample.jsx">
```

## Configuring properties

In the JSX, need to declare the navigation drawer properties. Refer to the following code,

### JAVASCRIPT

```
ReactDOM.render(  
  <EJ.NavigationDrawer id="navpanedefault" type="overlay" direction="left"  
  enableListView={true} listViewSettings-width={220} listViewSettings-  
  height="100%"  
  listViewSettings-selectedItemIndex="0" position="normal">  
    <ul>  
      <li data-ej-text="Home"></li>  
      <li data-ej-text="People"></li>  
      <li data-ej-text="Profile"></li>  
    </ul>  
  </EJ.NavigationDrawer>,  
  document.getElementById('navigation')  
) ;
```

Create the target element as follows to display the list items by clicking target icon.

### HTML

```
<div id="targetPane">  
  <div className="e-lv">  
    <div className="e-header">  
      <div id="drawer" className="drawerIcon e-icon alignText"></div>  
    </div>  
  </div>
```

To set the target icon image and with the correct position as using the below mentioned styles.

### CSS

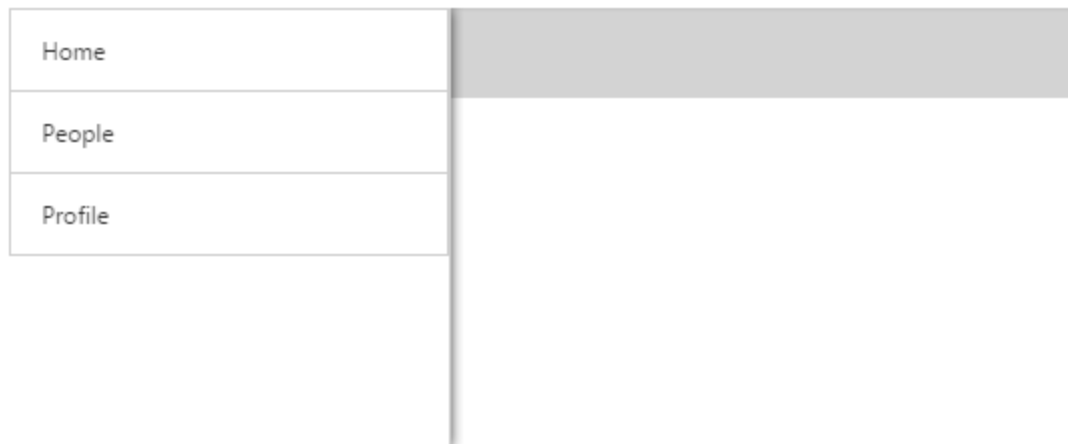
```
<style>
.drawerIcon {
background-position: center center;
background-repeat: no-repeat;
height: 45px;
width: 32px;
background-size: 100% 100%;
padding-right: 10px;
}
.drawerIcon:before {
content: "\e76b";
font-size: 24px;
height: 26px;
line-height: 24px;
}
#targetPane {
height: 220px;
position: relative;
padding: 0px;
overflow: hidden;
margin: 0px;
}
</style>
```



You can open the list items by clicking on target element using the `targetId` property.

### JAVASCRIPT

```
ReactDOM.render(
<EJ.NavigationDrawer id="navpanedefault" type="overlay" direction="left"
targetId="drawer" enableListView={true} listViewSettings-width={220}
listViewSettings-height="100%"
listViewSettings-selectedItemIndex="0" position="normal">
<ul>
<li data-ej-text="Home"></li>
<li data-ej-text="People"></li>
<li data-ej-text="Profile"></li>
</ul>
</EJ.NavigationDrawer>,
document.getElementById('navigation')
);
```



To set the images for list items of the Navigation Drawer by using the `[data-ej-imageclass]` property as follows.

### JAVASCRIPT

```
ReactDOM.render (
  <EJ.NavigationDrawer id="navpanedefault" type="overlay" direction="left"
  targetId="drawer" enableListView={true} listViewSettings-width={220}
  listViewSettings-height="100%"
  listViewSettings-selectedItemIndex="0" position="normal">
    <ul>
      <li data-ej-imageclass="e-icon e-home" data-ej-text="Home"></li>
      <li data-ej-imageclass="e-icon e-photo" data-ej-text="Photos"></li>
      <li data-ej-imageclass="e-icon e-profile" data-ej-text="Profile"></li>
    </ul>
  </EJ.NavigationDrawer>,
  document.getElementById('navigation')
);
```

To set the images with the correct position by using the mentioned styles.

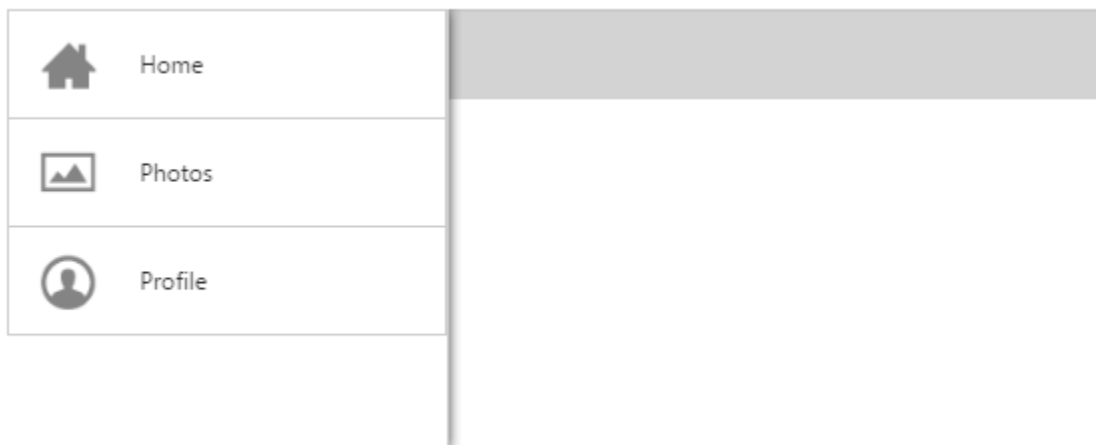
### CSS

```
<style>
@font-face {
font-family: 'ej-font';
src: url('../common-images/tools/icons.eot');
src: url('../common-images/tools/icons.eot') format('embedded-opentype'),
url('../common-images/tools/icons.woff')
format('woff'),url('../common-images/tools/icons.woff') format('woff'),
url('../common-images/tools/icons.ttf') format('truetype'),
url('../common-images/tools/icons.svg') format('svg');
font-weight: normal;
font-style: normal;
}
.e-home:before {
font-family: "ej-font";
content: "\e900";
}
```

```

.e-profile:before {
font-family: "ej-font";
content: "\e901";
}
.e-photo:before {
font-family: "ej-font";
content: "\e903";
}
.e-location:before {
font-family: "ej-font";
content: "\e905";
}
.e-people:before {
font-family: "ej-font";
content: "\e902";
}
.e-communities:before {
font-family: "ej-font";
content: "\e904";
}
.e-home, .e-profile, .e-people, .e-photo, .e-communities, .e-location {
font-size: 24px;
color: black;
}
</style>

```



To add desired page content while selecting the options in navigation drawer as follows.

### HTML

```

<!-- Home Page Content-->
<div id="Home">
The Home screen allows you to choose the specific content type displayed.
</div>
<!-- Profile Page Content-->
<div id="Profile" style="display: none">
The Profile page content is displayed.
</div>
<!-- Photos Page Content-->

```

```
<div id="Photos" style="display: none">
The Photos page content is displayed.
</div>
```

To load the appropriate content for the navigation by using the listViewSettings-mouseUp event handler of ListView component.

#### JAVASCRIPT

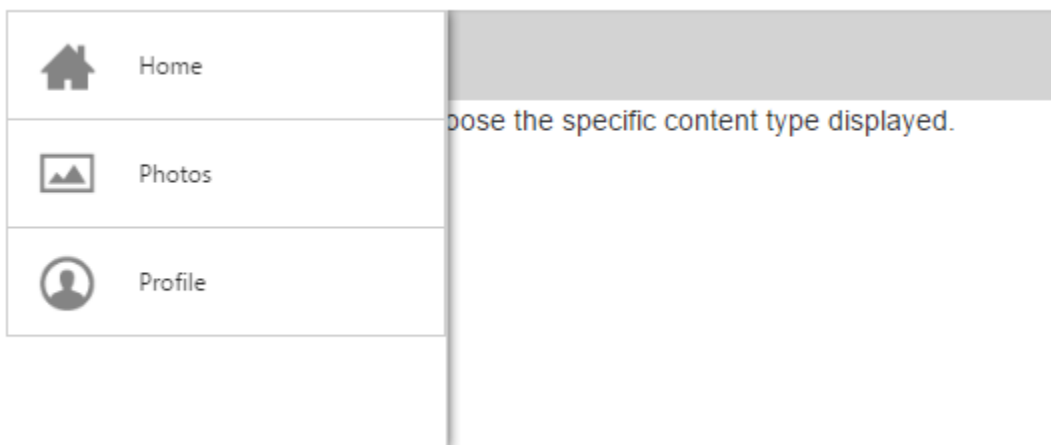
```
ReactDOM.render (
<EJ.NavigationDrawer id="navpanedefault" type="overlay" direction="left"
targetId="drawer" enableListView={true} listViewSettings-width={220}
listViewSettings-height="100%"
listViewSettings-selectedItemIndex="0" listViewSettings-
mouseUp={this.headChange} position="normal">
<ul>
<li data-ej-imageclass="e-icon e-home" data-ej-text="Home"></li>
<li data-ej-imageclass="e-icon e-photo" data-ej-text="Photos"></li>
<li data-ej-imageclass="e-icon e-profile" data-ej-text="Profile"></li>
</ul>
</EJ.NavigationDrawer>,
document.getElementById('navigation')
);
```

In the mouse, up handler, it's display the respective selected item's content.

#### JAVASCRIPT

```
headChange: function (e) {
$('#Home, #Profile, #Photos').hide(); //Hiding all other contents
$('#' + e.text).show(); //Displaying the content based on the text of item
selected
}
```

Run the above code to render the following output.



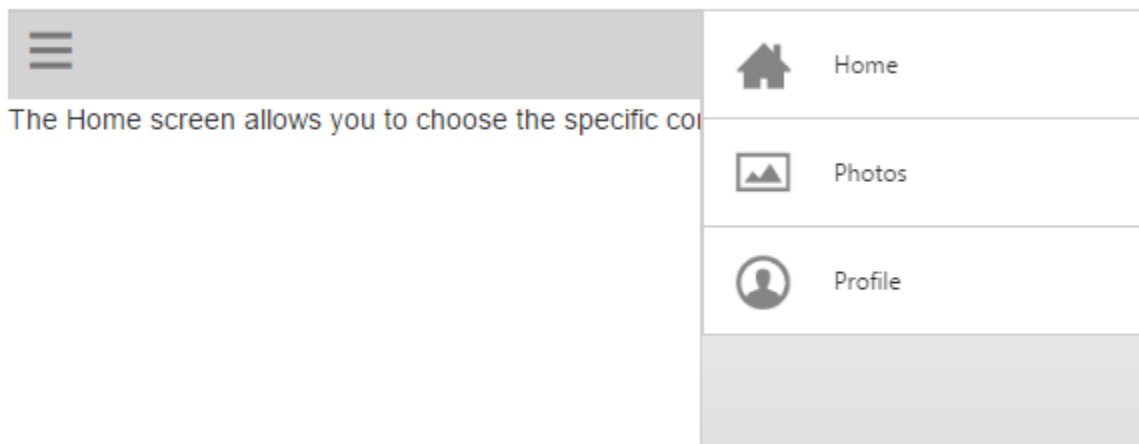


### Customize Direction

By using direction property, to change the list view open direction. The possible directions are Right, Left and the Left is default value.

#### JAVASCRIPT

```
ReactDOM.render(
  <EJ.NavigationDrawer id="navpanedefault" type="overlay" direction="right"
    targetId="drawer" enableListView={true} listViewSettings-width={220}
    listViewSettings-height="100%"
    listViewSettings-selectedItemIndex="0" listViewSettings-
    mouseUp={this.headChange} position="normal">
    <ul>
      <li data-ej-imageclass="e-icon e-home" data-ej-text="Home"></li>
      <li data-ej-imageclass="e-icon e-photo" data-ej-text="Photos"></li>
      <li data-ej-imageclass="e-icon e-profile" data-ej-text="Profile"></li>
    </ul>
  </EJ.NavigationDrawer>,
  document.getElementById('navigation')
);
```



**Note:** To get the complete API list for all the Syncfusion component's properties from the [API reference](#)

## NumericTextbox

### Getting Started

This section helps to get started of the NumericTextbox component in a React application

#### Create a NumericTextbox

Refer the common React Getting Started Documentation to create an application and add necessary scripts and styles for rendering our ReactJS components.

Create a JSX file and use <EJ.NumericTextbox> syntax to render React NumericTextbox component. Add required properties to <EJ.NumericTextbox> tag element.

#### HTML

```
ReactDOM.render(  
  <EJ.NumericTextbox>  
  </EJ.NumericTextbox>,  
  document.getElementById('numeric')  
) ;
```

Define an HTML element for adding NumericTextbox in the application and refer the JSX file created.

### HTML

```
<div id="numeric"></div>  
<script type="text/babel" src="sample.jsx">
```

This will render an empty NumericTextbox component on executing.

### Configure Properties

In the JSX, need to declare the NumericTextbox properties. Refer to the following code,.

### HTML

```
ReactDOM.render(  
  <EJ. NumericTextbox value={40} width="250px">  
  </EJ. NumericTextbox >,  
  document.getElementById('numeric')  
) ;
```

Run the above code to render the following output,



*Note: You can find the NumericTextbox properties from the [API reference](#) document.*

## PDF viewer

### Overview

PDF viewer for JavaScript is a visualization component to view PDF documents in web pages. It is powered by HTML5/JavaScript and provides various customization.

### Key Features

The key features of the **PDF viewer** control are listed as follows:

- Supports uploading PDF document into the server as byte array or as stream and supports displaying the same in the client.
- Supports for viewing password protected/encrypted PDF documents.
- Supports customizing the default toolbar or creation of a toolbar as per the application requirements.
- Supports zooming tools and viewing modes for better viewing experience.
- Supports built-in themes for enhancing appearance.
- Supports printing the PDF document.
- Supports responsive rendering while resizing the control/window.

- Supports downloading the PDF document being displayed.
- Compatible with all the modern browsers that provides support for HTML5/CSS3/JavaScript.

## Getting Started

This section explains briefly about how to integrate a **PDF viewer** control in your application with **React JS**.

### Script and CSS Reference

Create a **HTML** page and add the scripts and CSS references in the order mentioned in the following code example.

#### HTML

```
<!DOCTYPE html>
<html>
<head>
<!-- Essential Studio for JavaScript theme reference -->
<link rel="stylesheet"
href="http://cdn.syncfusion.com/14.4.0.15/js/web/bootstrap-
theme/ej.web.all.min.css" />
<!-- react script -->
<script
src="https://cdnjs.cloudflare.com/ajax/libs/react/15.2.1/react.js"></script>
<script src="https://cdnjs.cloudflare.com/ajax/libs/react/15.2.1/react-
dom.js"></script>
<script src="https://cdnjs.cloudflare.com/ajax/libs/babel-
core/5.8.34/browser.min.js"></script>
<!-- jquery script -->
<script src="https://code.jquery.com/jquery-3.0.0.min.js"></script>
<!-- Essential JS UI widget -->
<script
src="http://cdn.syncfusion.com/14.4.0.15/js/web/ej.web.all.min.js"></script>
<script
src="http://cdn.syncfusion.com/14.4.0.15/js/common/ej.web.react.min.js"></sc
ript>
<!--Add custom scripts here -->
</head>
<body>
</body>
</html>
```

**Note:** 1. In production, we highly recommend you to use our [custom script generator](#) to create custom script file with required controls and its dependencies only. Also to reduce the file size further please use [GZip compression](#) in your server.

**Note:** 2. For themes, you can use the `ej.web.all.min.css` CDN link from the code snippet given. To add the themes in your application, please refer to [this link](#).

- `react.js` and `react-dom.js` are the core files needed to create react elements.
- `browser.min.js` file is required for code transform.
- `ej.web.react.min.js` is a react-syncfusion bridge to render Syncfusion components.

### Initialize and configure the control

Control can be initialized in two ways.

- Using jsx Template
- Without using jsx Template

#### Using jsx Template

By using the jsx template, we can create the html file and jsx file. The .jsx file can be converted into .js file and it can be referred in html page.

Please refer to the code of HTML file.

#### HTML

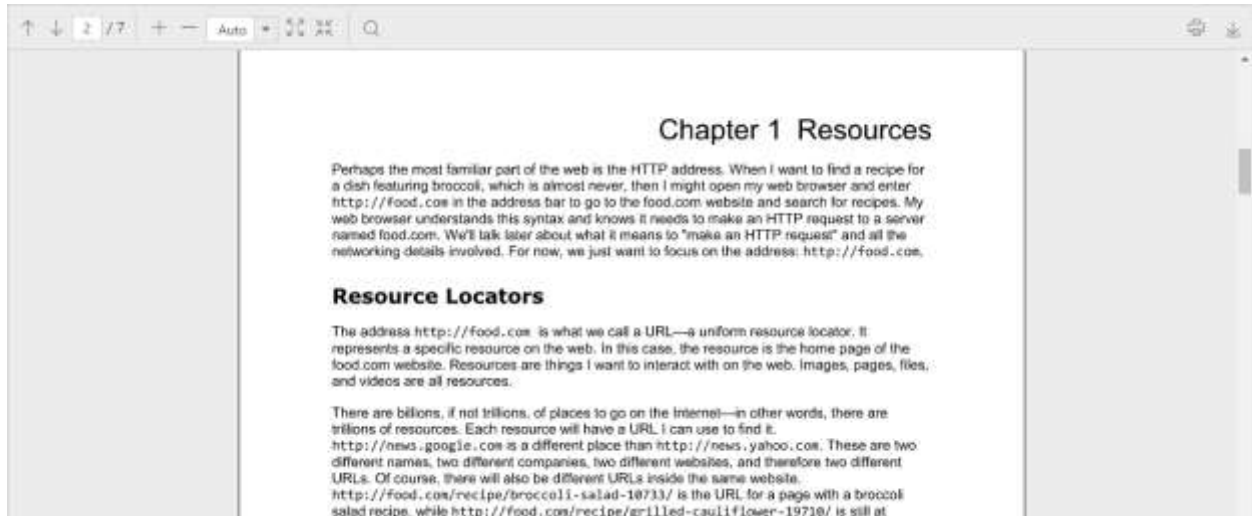
```
<div id="pdfviewerctl"></div>
<script src="app/pdfviewer/default.js"></script>
```

Initialize the PDF viewer by using the `EJ.PdfViewer` tag.

#### HTML

```
<!DOCTYPE html>
<html>
<head>
<style>
#PdfViewer {
width: 100%;
height: 550px;
}
</style>
</head>
<body>
<div id="PdfViewer1" style="width:99%;"></div>
<script type="text/babel">
var servicePath="http://js.syncfusion.com/ejservices/api/PdfViewer";
ReactDOM.render (
<EJ.PdfViewer id="PdfViewer" serviceUrl={servicePath}></EJ.PdfViewer>,
document.getElementById('PdfViewer1')
);
</script>
</body>
</html>
```

Now, the PDF viewer control is rendered with default PDF document, which used in the services.



### Without using jsx Template

PDF viewer can be created from a HTML `DIV` element with the HTML `id` attribute set to it. Refer to the following code example.

#### HTML

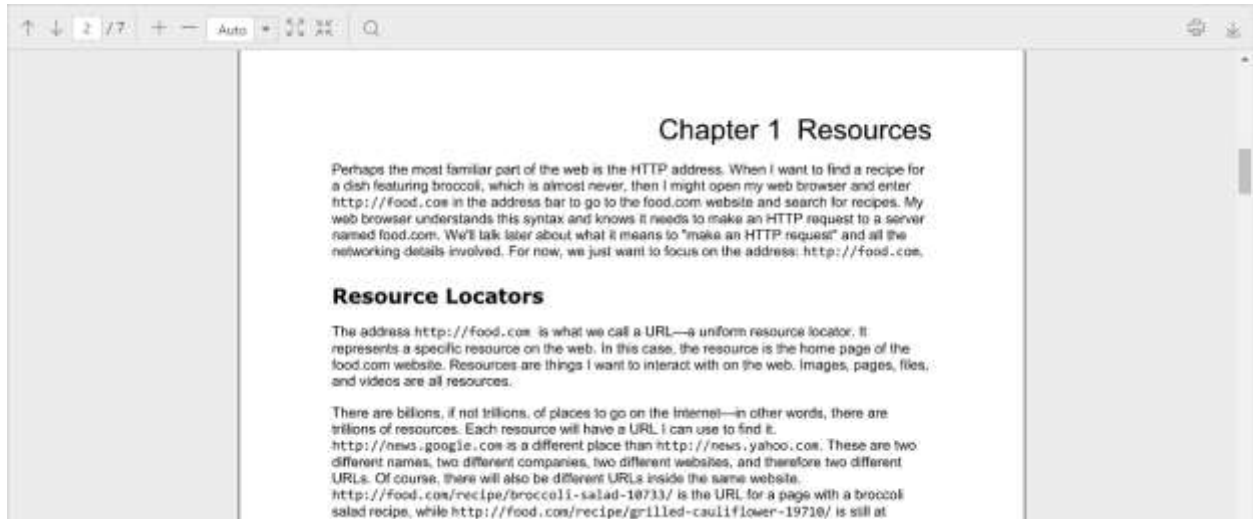
```
<body>
<div id="pdfviewer1"></div>
</body>
```

Initialize the PDF viewer control by adding the following script code to the body section of the HTML document.

#### HTML

```
<div id="pdfviewer1"></div>
<script type="text/javascript">
var service = "http://js.syncfusion.com/ejservices/api/PdfViewer";
ReactDOM.render(
  React.createElement(EJ.PdfViewer,
    {
      serviceUrl:service
    },
    document.getElementById('pdfviewer1')
  );
</script>
```

Now, the PDF viewer control will be rendered with the default PDF document, which is used in the service.



## PercentageTextbox

### Getting Started

This section helps to get started of the PercentageTextbox component in a React application

#### Create a PercentageTextbox

Refer the common React Getting Started Documentation to create an application and add necessary scripts and styles for rendering our ReactJS components.

Create a JSX file and use <EJ.PercentageTextbox> syntax to render React PercentageTextbox component. Add required properties to <EJ.PercentageTextbox> tag element.

#### HTML

```
ReactDOM.render(
  <EJ.PercentageTextbox>
</EJ.PercentageTextbox>,
  document.getElementById('percent')
);
```

Define an HTML element for adding PercentageTextbox in the application and refer the JSX file created.

#### HTML

```
<div id="percent"></div>
<script type="text/babel" src="sample.jsx">
```

This will render an empty PercentageTextbox component on executing.

#### Configure Properties

In the JSX, need to declare the PercentageTextbox properties. Refer to the following code,.

#### HTML

```
ReactDOM.render(
  <EJ.PercentageTextbox value={20} width="250px">
</EJ.PercentageTextbox>,
  document.getElementById('percent')
```

```
);
```

Run the above code to render the following output,



*Note: You can find the `PercentageTextbox` properties from the [API reference](#) document.*

## PivotChart

### Overview

The **PivotChart** control is a lightweight control that reads both **OLAP** and **Relational** datasource and visualizes it in graphical format with the ability to drill up and down.

### Key Features

The key features of the **PivotChart** control is listed as follows:

- **Data source** - Supports OLAP data binding with XML/A data sources and Relational data sources.
- **Series** - A collection of series items that contain the actual data points is displayed on the chart.
- **Legend** - A color code that helps to differentiate between chart items. A legend also has labels beside each color to indicate that it applies to information from Series 1, Series 2, and so on.
- **Drill support** - Enables you to navigate to the inner levels in the row axis.
- **Tooltip** - Displays data point values of respective chart series on mouse hover.
- **Zooming and scrolling** – Enables you to zoom into an area of the chart so the data can be viewed with more clarity.

### Getting Started

This section explains you the steps required to populate the PivotChart with data source. This section covers only the minimal features that you need to know to get started with the PivotChart.

### Script and CSS Reference

Create a **HTML** page and add the script and CSS references in the order mentioned in the following code example.

#### HTML

```
<!DOCTYPE html>
<html>
<head>
  <!-- Essential Studio for JavaScript theme reference -->
  <link rel="stylesheet"
href="http://cdn.syncfusion.com/14.3.0.49/js/web/bootstrap-
theme/ej.web.all.min.css" />
  <!-- react script -->
  <script
src="https://cdnjs.cloudflare.com/ajax/libs/react/15.2.1/react.js"></script>
  <script src="https://cdnjs.cloudflare.com/ajax/libs/react/15.2.1/react-
dom.js"></script>
  <script src="https://cdnjs.cloudflare.com/ajax/libs/babel-
core/5.8.34/browser.min.js"></script>
```

```

<!-- jquery script -->
<script src="https://code.jquery.com/jquery-3.0.0.min.js"></script>
<!-- Essential JS UI widget -->
<script
src="http://cdn.syncfusion.com/14.3.0.49/js/web/ej.web.all.min.js"></script>
<script
src="http://cdn.syncfusion.com/14.3.0.49/js/common/ej.web.react.min.js"></sc
ript>
<!--Add custom scripts here -->
</head>
<body>
</body>
</html>

```

In the above code, `ej.web.all.min.js` script reference has been added for demonstration purpose. It is not recommended to use this for deployment purpose, as its file size is larger since it contains all the widgets. Instead, you can use [CSG](#) utility to generate a custom script file with the required widgets for deployment purpose.

**Note:**

- `react.js` and `react-dom.js` are the core files needed to create react elements.
- `browser.min.js` file is required for code transform.
- `ej.web.react.min.js` is a react-syncfusion bridge to render Syncfusion components.

### Relational

This section covers the information that you need to know to populate a simple PivotChart with Relational data source.

#### Control Initialization

Add a `div` container to render the PivotChart.

#### HTML

```

<!DOCTYPE html>
<html>
<body>
<div id="PivotChart1" style="width:99%;"></div>
</body>
</html>

```

Initialize the PivotChart by using the `EJ.PivotChart` tag.

#### HTML

```

<!DOCTYPE html>
<html>
<head>
<style>
#Relational {
width: 100%;
height: 450px;
}
</style>

```



```

</head>
<body>
<div id="PivotChart1" style="width:99%;"></div>
<script type="text/babel">
ReactDOM.render(
<EJ.PivotChart id="Relational"></EJ.PivotChart>,
document.getElementById('PivotChart1')
);
</script>
</body>
</html>

```

### Populate PivotChart with Data

Let us now see how to populate the PivotChart control using a sample JSON data as shown below.

### HTML

```

<script type="text/babel">
var pivot_dataset = [
{ Amount: 100, Country: "Canada", Date: "FY 2005", Product: "Bike",
Quantity: 2, State: "Alberta" },
{ Amount: 200, Country: "Canada", Date: "FY 2006", Product: "Van", Quantity:
3, State: "British Columbia" },
{ Amount: 300, Country: "Canada", Date: "FY 2007", Product: "Car", Quantity:
4, State: "Brunswick" },
{ Amount: 150, Country: "Canada", Date: "FY 2008", Product: "Bike",
Quantity: 3, State: "Manitoba" },
{ Amount: 200, Country: "Canada", Date: "FY 2006", Product: "Car", Quantity:
4, State: "Ontario" },
{ Amount: 100, Country: "Canada", Date: "FY 2007", Product: "Van", Quantity:
1, State: "Quebec" },
{ Amount: 200, Country: "France", Date: "FY 2005", Product: "Bike",
Quantity: 2, State: "Charente-Maritime" },
{ Amount: 250, Country: "France", Date: "FY 2006", Product: "Van", Quantity:
4, State: "Essonne" },
{ Amount: 300, Country: "France", Date: "FY 2007", Product: "Car", Quantity:
3, State: "Garonne (Haute)" },
{ Amount: 150, Country: "France", Date: "FY 2008", Product: "Van", Quantity:
2, State: "Gers" },
{ Amount: 200, Country: "Germany", Date: "FY 2006", Product: "Van",
Quantity: 3, State: "Bayern" },
{ Amount: 250, Country: "Germany", Date: "FY 2007", Product: "Car",
Quantity: 3, State: "Brandenburg" },
{ Amount: 150, Country: "Germany", Date: "FY 2008", Product: "Car",
Quantity: 4, State: "Hamburg" },
{ Amount: 200, Country: "Germany", Date: "FY 2008", Product: "Bike",
Quantity: 4, State: "Hessen" },
{ Amount: 150, Country: "Germany", Date: "FY 2007", Product: "Van",
Quantity: 3, State: "Nordrhein-Westfalen" },
{ Amount: 100, Country: "Germany", Date: "FY 2005", Product: "Bike",
Quantity: 2, State: "Saarland" },
{ Amount: 150, Country: "United Kingdom", Date: "FY 2008", Product: "Bike",
Quantity: 5, State: "England" },
{ Amount: 250, Country: "United States", Date: "FY 2007", Product: "Car",
Quantity: 4, State: "Alabama" },

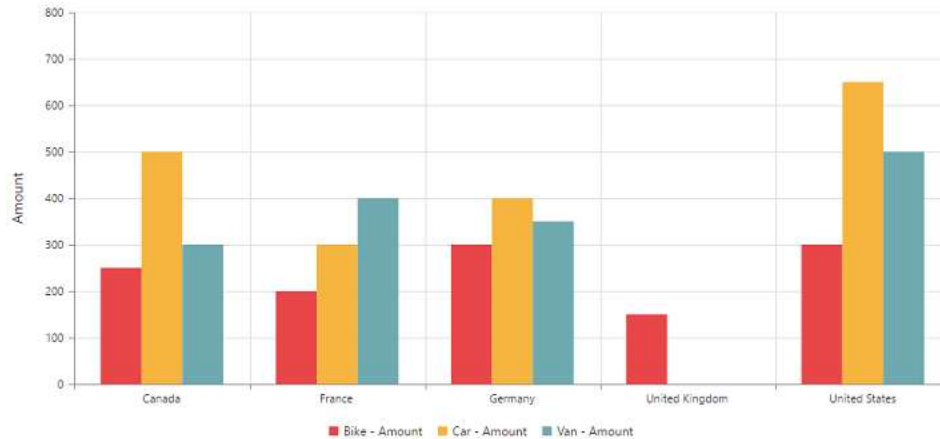
```

```

{ Amount: 200, Country: "United States", Date: "FY 2005", Product: "Van",
Quantity: 4, State: "California" },
{ Amount: 100, Country: "United States", Date: "FY 2006", Product: "Bike",
Quantity: 2, State: "Colorado" },
{ Amount: 150, Country: "United States", Date: "FY 2008", Product: "Car",
Quantity: 3, State: "New Mexico" },
{ Amount: 200, Country: "United States", Date: "FY 2005", Product: "Bike",
Quantity: 4, State: "New York" },
{ Amount: 250, Country: "United States", Date: "FY 2008", Product: "Car",
Quantity: 3, State: "North Carolina" },
{ Amount: 300, Country: "United States", Date: "FY 2007", Product: "Van",
Quantity: 4, State: "South Carolina" }
];
var pivotdataSource = {
data: pivot_dataset,
rows: [
{ fieldName: "Country", fieldCaption: "Country" },
{ fieldName: "State", fieldCaption: "State" }
],
columns: [
{ fieldName: "Product", fieldCaption: "Product" }
],
values: [
{ fieldName: "Amount", fieldCaption: "Amount" },
{ fieldName: "Quantity", fieldCaption: "Quantity" }
],
filters: []
};
var piovchart_size={ height: "460px", width: "100%" };
var def_commonSeriesOption = { enableAnimation: true, type:
ej.PivotChart.ChartTypes.Column, tooltip: { visible: true } };
$(function(){
ReactDOM.render(
<EJ.PivotChart id="Relational" dataSource= {pivotdataSource}
isResponsive={true} commonSeriesOptions= {def_commonSeriesOption} size=
{piovchart_size}></EJ.PivotChart>,
document.getElementById('PivotChart1')
);
});
</script>

```

The above code will generate a simple PivotChart with sales amount over products in different regions.



## OLAP

This section covers the information that you need to know to populate a simple PivotChart with OLAP data source.

### Control Initialization

Add a `div` container to render the PivotChart.

### HTML

```
<!DOCTYPE html>
<html>
<body>
<div id="PivotChart1" style="width:99%;"></div>
</body>
</html>
```

Initialize the PivotChart by using the `EJ.PivotChart` tag.

### HTML

```
<!DOCTYPE html>
<html>
<head>
<style>
#Olap{
width: 100%;
height: 450px;
}
</style>
</head>
<body>
<div id="PivotChart1" style="width:99%;"></div>
<script type="text/babel">
ReactDOM.render(
<EJ.PivotChart id="Olap"></EJ.PivotChart>,
document.getElementById('PivotChart1')
);
</script>
</body>
</html>
```

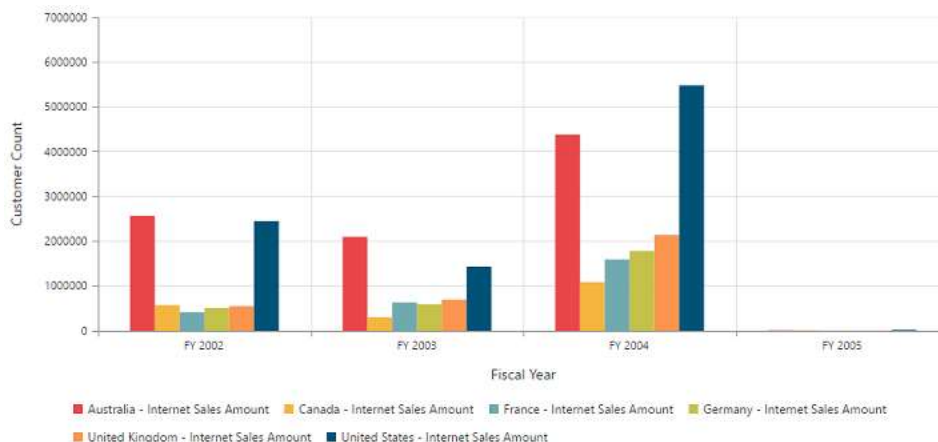
### Populate PivotChart with data

Let us now see how to populate the PivotChart control using a sample JSON data as shown below.

#### HTML

```
<script type="text/babel">
var Olap_dataSource={
  data: "http://bi.syncfusion.com/olap/msmdpump.dll",
  catalog: "Adventure Works DW 2008 SE", //"Adventure Works DW 2008 SEstandard
  Edition
  cube: "Adventure Works", rows: [{ fieldName: "[Date].[Fiscal]" }], columns:
  [{ fieldName: "[Customer].[Customer Geography]" }],
  values: [{ measures: [{ fieldName: "[Measures].[Internet Sales Amount]" }],
  axis: "columns" }]
};
var chart_size={ height: "460px", width: "100%" };
var def_commonSeriesOption={ enableAnimation: true, type:
ej.PivotChart.ChartTypes.Column, tooltip: { visible: true } };
$(function() {
  ReactDOM.render(
    <EJ.PivotChart id="Olap" dataSource= {Olap_dataSource} isResponsive={true}
    commonSeriesOptions= {def_commonSeriesOption} size=
    {chart_size}></EJ.PivotChart>,
    document.getElementById('PivotChart1')
  );
});
</script>
```

The above code will generate a simple PivotChart with internet sales amount over a period of fiscal years across different customer geographic locations.



## PivotGrid

### Overview

The **PivotGrid** control is an easily configurable, presentation-quality business control that reads both OLAP and Relational datasource, allows to create multi-dimensional views for analysis and satisfying business user needs.

### Key Features

Some of the key features of PivotGrid are as follows,

- **Data Binding:** Supports both OLAP and Relational datasource data binding in the same environment.
- **PivotTable Field List:** Lists the entire datasource bound to PivotGrid and allows UI interaction such as filtering and drag drop operation of any measure, dimension, hierarchy, level and field at runtime.
- **Grouping Bar:** Supports UI interaction at runtime through sorting, filtering and drag drop operation of the existing field items bound through report.
- **Grid Layout:** Four different layouts such as normal, top summary, no summaries and excel-like layout are supported.
- **Export:** PivotGrid can be exported to Word, PDF, CSV and Excel documents.
- PivotGrid control specific features such as hyperlink, cell selection, conditional formatting, RTL, globalization and localization are supported as well.

## Getting Started

This section explains you the steps required to populate the PivotGrid with data source. This section covers only the minimal features that you need to know to get started with the PivotGrid.

### Script and CSS Reference

Create a **HTML** page and add the script and CSS references in the order mentioned in the following code example.

#### HTML

```
<!DOCTYPE html>
<html>
<head>
  <!-- Essential Studio for JavaScript theme reference -->
  <link rel="stylesheet"
href="http://cdn.syncfusion.com/14.3.0.49/js/web/bootstrap-
theme/ej.web.all.min.css" />
  <!-- react script -->
  <script
src="https://cdnjs.cloudflare.com/ajax/libs/react/15.2.1/react.js"></script>
  <script src="https://cdnjs.cloudflare.com/ajax/libs/react/15.2.1/react-
dom.js"></script>
  <script src="https://cdnjs.cloudflare.com/ajax/libs/babel-
core/5.8.34/browser.min.js"></script>
  <!-- jquery script -->
  <script src="https://code.jquery.com/jquery-3.0.0.min.js"></script>
  <!-- Essential JS UI widget -->
  <script
src="http://cdn.syncfusion.com/14.3.0.49/js/web/ej.web.all.min.js"></script>
  <script
src="http://cdn.syncfusion.com/14.3.0.49/js/common/ej.web.react.min.js"></sc
ript>
  <!--Add custom scripts here -->
</head>
<body>
</body>
</html>
```

In the above code, `ej.web.all.min.js` script reference has been added for demonstration purpose. It is not recommended to use this for deployment purpose, as its file size is larger since it contains all the widgets. Instead, you can use [CSG](#) utility to generate a custom script file with the required widgets for deployment purpose.

**Note:**

- `react.js` and `react-dom.js` are the core files needed to create react elements.
- `browser.min.js` file is required for code transform.
- `ej.web.react.min.js` is a react-syncfusion bridge to render Syncfusion components.

### Relational

This section covers the information that you need to know to populate a simple PivotGrid with Relational data source.

#### Control Initialization

Add a `div` container to render the PivotGrid.

**HTML**

```
<!DOCTYPE html>
<html>
<body>
<div id="PivotGrid1" style="width:99%;"></div>
</body>
</html>
```

Initialize the PivotGrid by using the `EJ.PivotGrid` tag.

**HTML**

```
<!DOCTYPE html>
<html>
<head>
<style>
#Relational{
width: 800px;
height: 450px;
overflow: auto;
}
</style>
</head>
<body>
<div id="PivotGrid1" style="width:99%;"></div>
<script type="text/babel">
ReactDOM.render(
<EJ.PivotGrid id="Relational"></EJ.PivotGrid>,
document.getElementById('PivotGrid1')
);
</script>
</body>
</html>
```

*Populate PivotGrid with Data*

Let us now see how to populate the PivotGrid control using a sample JSON data as shown below.

**HTML**

```
<script type="text/babel">
var pivot_dataset = [
{ Amount: 100, Country: "Canada", Date: "FY 2005", Product: "Bike",
Quantity: 2, State: "Alberta" },
{ Amount: 200, Country: "Canada", Date: "FY 2006", Product: "Van", Quantity:
3, State: "British Columbia" },
{ Amount: 300, Country: "Canada", Date: "FY 2007", Product: "Car", Quantity:
4, State: "Brunswick" },
{ Amount: 150, Country: "Canada", Date: "FY 2008", Product: "Bike",
Quantity: 3, State: "Manitoba" },
{ Amount: 200, Country: "Canada", Date: "FY 2006", Product: "Car", Quantity:
4, State: "Ontario" },
{ Amount: 100, Country: "Canada", Date: "FY 2007", Product: "Van", Quantity:
1, State: "Quebec" },
{ Amount: 200, Country: "France", Date: "FY 2005", Product: "Bike",
Quantity: 2, State: "Charente-Maritime" },
{ Amount: 250, Country: "France", Date: "FY 2006", Product: "Van", Quantity:
4, State: "Essonne" },
{ Amount: 300, Country: "France", Date: "FY 2007", Product: "Car", Quantity:
3, State: "Garonne (Haute)" },
{ Amount: 150, Country: "France", Date: "FY 2008", Product: "Van", Quantity:
2, State: "Gers" },
{ Amount: 200, Country: "Germany", Date: "FY 2006", Product: "Van",
Quantity: 3, State: "Bayern" },
{ Amount: 250, Country: "Germany", Date: "FY 2007", Product: "Car",
Quantity: 3, State: "Brandenburg" },
{ Amount: 150, Country: "Germany", Date: "FY 2008", Product: "Car",
Quantity: 4, State: "Hamburg" },
{ Amount: 200, Country: "Germany", Date: "FY 2008", Product: "Bike",
Quantity: 4, State: "Hessen" },
{ Amount: 150, Country: "Germany", Date: "FY 2007", Product: "Van",
Quantity: 3, State: "Nordrhein-Westfalen" },
{ Amount: 100, Country: "Germany", Date: "FY 2005", Product: "Bike",
Quantity: 2, State: "Saarland" },
{ Amount: 150, Country: "United Kingdom", Date: "FY 2008", Product: "Bike",
Quantity: 5, State: "England" },
{ Amount: 250, Country: "United States", Date: "FY 2007", Product: "Car",
Quantity: 4, State: "Alabama" },
{ Amount: 200, Country: "United States", Date: "FY 2005", Product: "Van",
Quantity: 4, State: "California" },
{ Amount: 100, Country: "United States", Date: "FY 2006", Product: "Bike",
Quantity: 2, State: "Colorado" },
{ Amount: 150, Country: "United States", Date: "FY 2008", Product: "Car",
Quantity: 3, State: "New Mexico" },
{ Amount: 200, Country: "United States", Date: "FY 2005", Product: "Bike",
Quantity: 4, State: "New York" },
{ Amount: 250, Country: "United States", Date: "FY 2008", Product: "Car",
Quantity: 3, State: "North Carolina" },
{ Amount: 300, Country: "United States", Date: "FY 2007", Product: "Van",
Quantity: 4, State: "South Carolina" }
];
var pivotdataSource = {
```

```

data: pivot_dataset,
rows: [
{ fieldName: "Country", fieldCaption: "Country" },
{ fieldName: "State", fieldCaption: "State" }
],
columns: [{ fieldName: "Product", fieldCaption: "Product" }],
values: [
{ fieldName: "Amount", fieldCaption: "Amount" },
{ fieldName: "Quantity", fieldCaption: "Quantity" }
],
filters: []
};
$(function() {
ReactDOM.render(
<EJ.PivotGrid id="Relational" dataSource= {pivotdataSource}></EJ.PivotGrid>,
document.getElementById('PivotGrid1')
);
});
</script>

```

The above code will generate a simple PivotGrid with “Country” field in Row, “Product” field in Column and “Amount” field in Value section.

	Bike	Car	Van	Grand Total
	Amount	Amount	Amount	Amount
Canada	250	500	300	1050
France	200	300	400	900
Germany	300	400	350	1050
United Kingdom	150			150
United States	300	650	500	1450
Grand Total	1200	1850	1550	4600

### Apply Sorting

You can sort a field either to ascending or descending order using the “**sortOrder**” property. Sorting is applicable only for Row and Column fields. By default, fields are arranged in ascending order.

### HTML

```

<script type="text/babel">
var pivot_dataset = []; // data source
var pivotdataSource = {
data: pivot_dataset,
rows: [
{ fieldName: "Country", fieldCaption: "Country", sortOrder:
ej.PivotAnalysis.SortOrder.Descending }
],
columns: [{ fieldName: "Product", fieldCaption: "Product" }],
values: [
{ fieldName: "Amount", fieldCaption: "Amount" }

```



```

],
filters: []
};
</script>

```

	Bike	Car	Van	Grand Total
	Amount	Amount	Amount	Amount
United States	300	650	500	1450
United Kingdom	150			150
Germany	300	400	350	1050
France	200	300	400	900
Canada	250	500	300	1050
Grand Total	1200	1850	1550	4600

#### Apply Filtering

Filtering option allows you to specify a set of values that either need to be displayed or hidden. Also filtering option is applicable only for Row, Column and Filter areas.

“**filterItems**” object allow us to apply filtering to the fields using the following properties:

- **filterType** - indicates whether the values should be included or excluded.
- **values** - specify an array of values that needs to be included or excluded within the particular field.

#### HTML

```

<script type="text/babel">
var pivot_dataset = []; // data source
var pivotdataSource = {
  data: pivot_dataset,
  rows: [ { fieldName: "Country",
    fieldCaption: "Country",
    filterItems: {
      filterType: ej.PivotAnalysis.FilterType.Exclude,
      values: ["United Kingdom"]
    }
  }
],
  columns: [{ fieldName: "Product",
    fieldCaption: "Product",
    filterItems: {
      filterType: ej.PivotAnalysis.FilterType.Include,
      values: ["Bike", "Car"]
    }
  }
]
}

```

```

}],
//....
};
</script>

```

	Bike	Car	Grand Total
	Amount	Amount	Amount
Canada	250	500	750
France	200	300	500
Germany	300	400	700
United States	300	650	950
Grand Total	1050	1850	2900

#### Apply Summary Types

Allow us to specify the required summary type that PivotGrid should use in its summary cells. **“sum”** is the default summary type. Following are the summary types that are supported:

- sum
- average
- count
- min
- max

#### HTML

```

<script type="text/babel">
var pivot_dataset = []; // data source
var pivotdataSource = {
  data: pivot_dataset,
  //...
  values: [
    { fieldName: "Amount", fieldCaption: "Amount", summaryType:
ej.PivotAnalysis.SummaryType.Average },
    { fieldName: "Quantity", fieldCaption: "Quantity", summaryType:
ej.PivotAnalysis.SummaryType.Sum }
  ],
  filters: []
};
</script>

```

	Bike		Car		Grand Total	
	Amount	Quantity	Amount	Quantity	Amount	Quantity
Canada	125	5	250	8	187.5	13
France	200	2	300	3	250	5
Germany	150	6	200	7	175	13
United States	150	6	212.5	10	181.25	16
Grand Total	156.25	19	240.63	28	198.44	47

## OLAP

This section covers the information that you need to know to populate a simple PivotGrid with OLAP data source.

Add a `div` container to render the PivotGrid.

## HTML

```
<!DOCTYPE html>
<html>
<body>
<div id="PivotGrid1" style="width:99%;"></div>
</body>
</html>
```

Initialize the PivotGrid by using the `EJ.PivotGrid` tag.

## HTML

```
<!DOCTYPE html>
<html>
<head>
<style>
#Olap{
width: 800px;
height: 450px;
overflow: auto;
}
</style>
</head>
<body>
<div id="PivotGrid1" style="width:99%;"></div>
<script type="text/babel">
ReactDOM.render (
<EJ.PivotGrid id="Olap"></EJ.PivotGrid>,
document.getElementById('PivotGrid1')
);
</script>
</body>
</html>
```

### Populate PivotGrid with data

Let us now see how to populate the PivotGrid control using a sample JSON data as shown below.

#### HTML

```
<script type="text/babel">
var Olap_dataSource={
data: "http://bi.syncfusion.com/olap/msmdpump.dll",
catalog: "Adventure Works DW 2008 SE", //"Adventure Works DW 2008 Standard
Edition
cube: "Adventure Works", rows: [{ fieldName: "[Date].[Fiscal]" }], columns:
[{ fieldName: "[Customer].[Customer Geography]" }],
values: [{ measures: [{ fieldName: "[Measures].[Internet Sales Amount]" }],
axis: "columns" }]
};
$(function() {
ReactDOM.render(
<EJ.PivotGrid id="Olap" dataSource= {Olap_dataSource}></EJ.PivotGrid>,
document.getElementById('PivotGrid1')
);
});
</script>
```

The above code will generate a simple PivotGrid with “Fiscal” field in Row, “Customer Geography” field in Column and “Internet Sales Amount” field in Value section.

	► Australia	► Canada	► France	► Germany	► United Kingdom	► United States	Total
	Internet Sales Amount						
► FY 2002	\$2,568,701.39	\$573,100.97	\$414,245.32	\$513,353.17	\$550,507.33	\$2,452,176.07	\$7,072,084.24
► FY 2003	\$2,099,585.43	\$305,010.69	\$633,399.70	\$593,247.24	\$696,594.97	\$1,434,296.26	\$5,762,134.30
► FY 2004	\$4,383,479.54	\$1,088,879.50	\$1,592,880.75	\$1,784,107.09	\$2,140,388.50	\$5,483,882.67	\$16,473,618.05
► FY 2005	\$9,234.23	\$10,853.70	\$3,491.95	\$3,604.83	\$4,221.41	\$19,434.51	\$50,840.63
Total	\$9,061,000.58	\$1,977,844.86	\$2,644,017.71	\$2,894,312.34	\$3,391,712.21	\$9,389,789.51	\$29,358,677.22

### Relational

#### Grouping Bar

##### Initialization

Grouping Bar allows user to dynamically alter the report by filter, sort and remove operations in the PivotGrid control. Based on the Relational datasource and report bound to the PivotGrid control, Grouping Bar will be automatically populated. You can enable Grouping Bar option in PivotGrid by setting the [enableGroupingBar](#) property to true.

#### HTML

```
<script type="text/babel">
//...
$(function() {
ReactDOM.render(
<EJ.PivotGrid id="Relational" dataSource= {pivotdataSource}
enableGroupingBar={true}></EJ.PivotGrid>,
document.getElementById('PivotGrid1')
);
});
</script>
```

```
</script>
```

Amount X Quantity X		Product Y ^ X							
		Bike		Car		Van		Grand Total	
Country Y ^ X	State Y ^ X	Amount	Quantity	Amount	Quantity	Amount	Quantity	Amount	Quantity
▼ Canada	Quebec					100	1	100	1
	Ontario			200	4			200	4
	Manitoba	150	3					150	3
	Brunswick			300	4			300	4
	British Columbia					200	3	200	3
Canada Total		150	3	500	8	300	4	950	15

### Drag and Drop

You can alter the report on fly through drag-and-drop operation.

Date (Multiple items) Y X		Product Y ^ X							
Amount X Quantity X		Bike		Car		Van		Grand Total	
Country Y ^ X	State Y ^ X	Amount	Quantity	Amount	Quantity	Amount	Quantity	Amount	Quantity
▼ Canada	Quebec					100	1	100	1
	Ontario			200	4			200	4
	Manitoba	150	3					150	3
	Brunswick			300	4			300	4
	British Columbia					200	3	200	3

### Context Menu

You can also alter the report by using context menu.

Date (Multiple items) Y X		Product Y ^ X							
Amount X Quantity X		Bike		Car		Van		Grand Total	
Country Y ^ X	State Y ^ X	Amount	Quantity	Amount	Quantity	Amount	Quantity	Amount	Quantity
▼ Canada	Quebec					100	1	100	1
	Ontario			200	4			200	4
	Manitoba	150	3					150	3
	Brunswick			300	4			300	4
	British Columbia					200	3	200	3

### Searching Values

Search option available in Grouping Bar allows you to search a specific value that needs to be filtered from the list of values inside the filter pop-up window.

Drag field here					
Amount X		Product ▼ ▲ X		Country ▼ ▲ X	
		▼ Bike			Bike Total
		Canada	France	United States	
		Amount	Amount	Amount	Amount
▼ FY 2005	Alabama			\$34,943,100.00	\$34,943,100.00
	Alberta	\$27,077,100.00			\$27,077,100.00
	British Columbia		\$78,491,400.00		\$78,491,400.00
	Brunswick	\$26,943,600.00			\$26,943,600.00
	New York			\$40,248,600.00	\$40,248,600.00
	Quebec	\$28,211,400.00			\$28,211,400.00

Product X

bike

☒ Bike

OK Cancel

### Filtering Values

Filtering option available in Grouping Bar allows you to select a specific set of values that needs to be displayed in the PivotGrid control. At least one value needed to be in checked state while filtering otherwise "Ok" button will be disabled.

Drag field here

Amount X		Product ▼ ▲ X		Country ▼ ▲ X	
		▼ Bike			Bike Total
		Canada	France	United States	
		Amount	Amount	Amount	Amount
▼ FY 2005	Alabama			\$34,943,100.00	\$34,943,100.00
	Alberta	\$27,077,100.00			\$27,077,100.00
	British Columbia		\$78,491,400.00		\$78,491,400.00
	Brunswick	\$26,943,600.00			\$26,943,600.00
	New York			\$40,248,600.00	\$40,248,600.00
	Quebec	\$28,211,400.00			\$28,211,400.00

Product X

Search Product

- ☒ All
- ☐ Bike
- ☒ Van
- ☒ Car

OK Cancel

### Sorting Values

Sorting option available in Grouping Bar allows you to arrange headers either in ascending or descending order. Sorting option is applicable for fields available only in Row and Column region. By default, headers are sorted in ascending order. Regarding sorting indicator, up arrow denotes ascending order and down arrow denotes descending order.

Drag field here					
Amount X		Product Y ^ X		Country Y ^ X	
		▼ Bike			Bike Total
		Canada	France	United States	
		Amount	Amount	Amount	Amount
▼ FY 2005	Alabama			\$34,943,100.00	\$34,943,100.00
	Alberta	\$27,077,100.00			\$27,077,100.00
	British Columbia		\$78,491,400.00		\$78,491,400.00
	Brunswick	\$26,943,600.00			\$26,943,600.00
	New York			\$40,248,600.00	\$40,248,600.00
	Quebec	\$28,211,400.00			\$28,211,400.00

Drag field here				
Amount X		Product Y ^ X		Country Y v X
		▼ Bike		
		United States	France	Canada
		Amount	Amount	Amount
▼ FY 2005	Alabama	\$34,943,100.00		
	Alberta			\$27,077,100.00
	British Columbia		\$78,491,400.00	
	Brunswick			\$26,943,600.00
	New York	\$40,248,600.00		
	Quebec			\$28,211,400.00



*Removing Field*

Remove option available in the Grouping Bar allows you to completely remove a specific field from the PivotGrid control. Remove operation can be either achieved by clicking remove icon available inside each field or by dragging and dropping field out of Grouping Bar region.

Drag field here					
Amount X		Product Y ^ X		Country Y ^ X	
		▼ Bike			
		Canada	France	United States	Bike Total
		Amount	Amount	Amount	Amount
▼ FY 2005	Alabama			\$34,943,100.00	\$34,943,100.00
	Alberta	\$27,077,100.00			\$27,077,100.00
	British Columbia		\$78,491,400.00		\$78,491,400.00
	Brunswick	\$26,943,600.00			\$26,943,600.00
	New York			\$40,248,600.00	\$40,248,600.00
	Quebec	\$28,211,400.00			\$28,211,400.00

Drag field here					
Amount X		Country Y ^ X			
		Canada	France	United States	Grand Total
		Amount	Amount	Amount	Amount
▼ FY 2005	Alabama			\$52,716,900.00	\$52,716,900.00
	Alberta	\$37,862,700.00			\$37,862,700.00
	British Columbia		\$109,102,500.00		\$109,102,500.00
	Brunswick	\$38,304,000.00			\$38,304,000.00
	New York			\$55,157,400.00	\$55,157,400.00
	Quebec	\$39,497,700.00			\$39,497,700.00

## PivotTable Field List

### Initialization

Field List, also known as Pivot Schema Designer, allows user to add, rearrange, filter and remove fields to show data in PivotGrid exactly the way they want.

Based on the datasource, Relational, bound to the PivotGrid control, PivotTable Field List will be automatically populated with Cube Information or Field Names. PivotTable Field List provides an Excel like appearance and behavior.

In-order to initialize PivotTable Field List, first you need to define a “div” tag with an appropriate “id” attribute which acts as a container for the widget. Then you need to initialize the PivotTable Field List by using the “**EJ.PivotSchemaDesigner**” method.

### HTML

```
<html>
//..
<body>
  <!--Create a tag which acts as a container for PivotGrid-->
  <div id="PivotGrid1" style="width: 55%; height: 670px; overflow: scroll;
float: left;"></div>
  <!--Create a tag which acts as a container for PivotTable Field List-->
  <div id="PivotSchemaDesigner1" style="height:650px;width:40%;"></div>
  <script type="text/babel">
    $(function() {
      ReactDOM.render(
        <EJ.PivotSchemaDesigner id="PivotSchemaDesigner"></EJ.PivotSchemaDesigner>,
        document.getElementById('PivotSchemaDesigner1')
      );
      ReactDOM.render(
        <EJ.PivotGrid id="Relational" dataSource= {pivotdataSource}
        enableGroupingBar={true}
        pivotTableFieldListID="PivotSchemaDesigner"></EJ.PivotGrid>,
        document.getElementById('PivotGrid1')
      );
    });
  </script>
</body>
</html>
```

		Bike		Car		Van	
		Amount	Quantity	Amount	Quantity	Amount	Quantity
Canada	Alberta	100	2				
	British Columbia					200	
	Brunswick			300	4		
	Manitoba	150	3				
	Ontario			200	4		
	Quebec					100	
Canada Total		250	5	500	8	300	
France	Charente-Maritime	200	2				
	Essonne					250	
	Garonne (Haute)			300	3		
	Geis					150	
France Total		200	2	300	3	400	
Germany	Bayern					200	
	Brandenburg			250	3		
	Hamburg			150	4		

**PivotTable Field List**

Choose fields to add to report:
 

☒ Amount
 ☒ Country
 ☒ Date
 ☒ Product
 ☒ Quantity
 ☒ State

Drag fields between areas below:
 

**Report Filter**

Date

**Column Label**

Product

**Row Label**

Country

State

**Values**

Amount

Quantity

### Layout

The top portion of the layout shows Field or Cube items in a categorized way. They can be dynamically added into the report either by drag and drop option or through simple check box selection.

On item(s) selection they will be placed in Row section by default except numeric based item(s) or measures, which will alone be placed in the Value section by default.

The bottom portion of the layout is segregated as below.

- **Report Filter:** Exclusively designed to filter an item(s) placed in this particular position of the layout.
- **Value Section:** The value label usually displays the numeric value item(s) present in the report.
- **Column Section:** It is used to display item(s) as column header and values in the PivotGrid control.
- **Row Section:** It is used to display item(s) as row header and values in the PivotGrid control.

### UI Interactions

#### By Drag and Drop

You can alter the report on fly through drag-and-drop operation. You can drag any item from Field List and drop into column, row, value or filter section available at the bottom of the Field List.

**PivotTable Field List**

Choose fields to add to report:

☒ Amount  
☒ Country  
☒ Date  
☒ Product  
☒ Quantity  
☒ State

Drag fields between areas below:

Report Filter	Column Label
Date	Product
	Amount

Row Label	Values
Country	Quantity
State	Amount

[By Drag and Drop to Grid Headers](#)

You can also drag and drop elements from field list to grid headers.

Drag field here

Amount X Quantity X Product T A X

Country T A X State T A X

		Bike		Car		Van	
		Amount	Quantity	Amount	Quantity	Amount	Quantity
Canada	Alberta	100	2				
	British Columbia					200	3
	Brunswick			300	4		
	Manitoba	150	5				
	Ontario			200	4		
	Quebec					100	1
Canada Total		250	5	500	8	300	4
France	Charente-Maritime	200	2				
	Essonne					250	4
	Garonne (Haute)			300	3		
	Gers					150	2
France Total		200	2	300	3	400	6
	Byern					200	3

PivotTable Field List

Choose fields to add to report:

- ☒ Amount
- ☐ Country
- ☐ Date
- ☐ Product
- ☒ Quantity
- ☒ State

Drag fields between areas below:

Filters

Columns

Product

Rows

Country  
State

Values

Amount  
Quantity

Drag field here

Amount X Quantity X Product T A X Date T A X

Country T A X

		Bike		Bike Total	
		FY 2005	FY 2006	FY 2008	
		Amount	Quantity	Amount	Quantity
Canada		100	2		200
France		200	2		200
Germany		100	2		300
United Kingdom				150	5
United States		200	4	100	2
Grand Total		600	10	300	1200

PivotTable Field List

Choose fields to add to report:

- ☒ Amount
- ☒ Country
- ☒ Date
- ☐ Product
- ☒ Quantity
- ☐ State

Drag fields between areas below:

Filters

Columns

Product  
Date

Rows

Country

Values

Amount  
Quantity

Drag field here

Amount X Quantity X Product T A X Date T A X

Country T A X State T A X

		Bike		Bike Total		Car	
		FY 2005	FY 2006	FY 2008		FY 2005	FY 2007
		Amount	Amount	Amount	Amount	Amount	Amount
Canada	Alberta	100				100	
	British Columbia						300
	Brunswick						
	Manitoba			100	100		
	Ontario						200
	Quebec						
Canada Total		100				200	300
France	Charente-Maritime	200					
	Essonne						200
	Garonne (Haute)						
	Gers						
France Total		200				200	300

PivotTable Field List

Choose fields to add to report:

- ☒ Amount
- ☒ Country
- ☒ Date
- ☐ Product
- ☒ Quantity
- ☒ State

Drag fields between areas below:

Filters

Columns

Product  
Date

Rows

Country  
State

Values

Amount

### By Treeview Selection

You can also alter the report on fly through check and uncheck option as an alternate. By default, fields will be added to the Row Label when checked.

**PivotTable Field List**

Choose fields to add to report:

☒ Amount

☒ Country

☒ Date

☐ Product

☒ Quantity

☒ State

### By Context Menu

You can also alter the report by using context menu.

**PivotTable Field List**

Choose fields to add to report:

☒ Amount

☒ Country

☐ Date

☒ Product

☒ Quantity

☒ State

Drag fields between areas below:

**Filters**

**Columns**

Product

Add to Filter

Add to Row

Add to Column

Add to Value

**Rows**

Country

State

**Values**

Amount

Quantity

**PivotTable Field List**

Choose fields to add to report:

- ☒ Amount
- ☒ Country
- ☐ Date
- ☒ Product
- ☒ Quantity
- ☒ State

Drag fields between areas below:

Filters	Columns
	Product

Rows	Values
Country	Amount
State	Quantity

### Searching Values

Search option available in Field List allows you to search a specific value that needs to be filtered from the list of values inside the filter pop-up window.

**PivotTable Field List**

Choose fields to add to report:

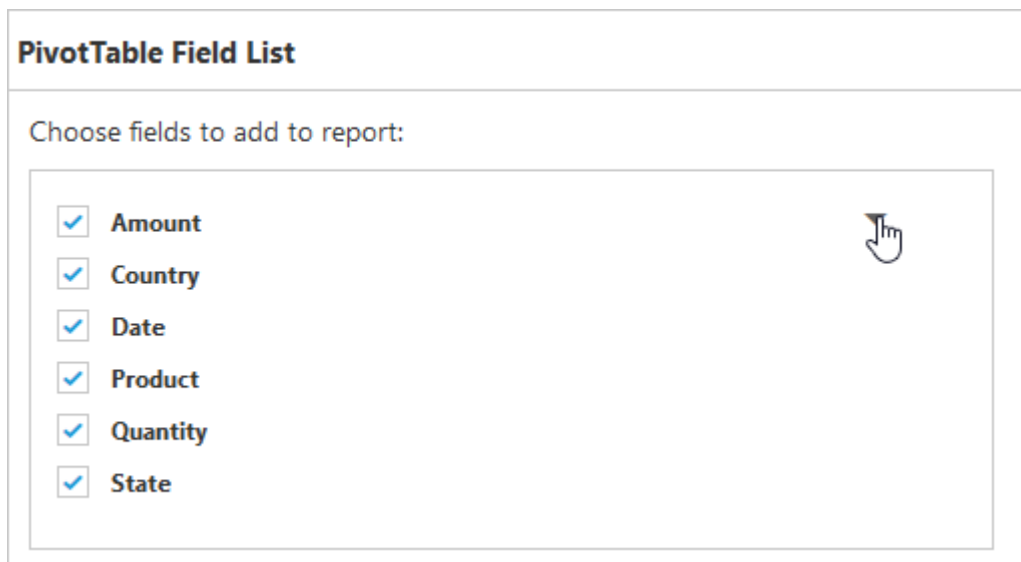
- ☒ Amount
- ☒ Country
- ☒ Date
- ☒ Product
- ☒ Quantity
- ☒ State



A dialog box titled "Amount" with a close button (X) in the top right corner. It features a text input field at the top containing the value "100". Below the input field is a list box containing a single item: a checked checkbox followed by the text "100". At the bottom of the dialog are two buttons: "OK" and "Cancel".

### Filtering

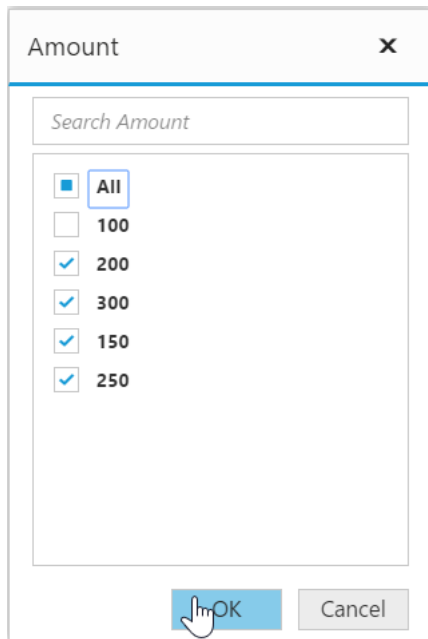
Values can be filtered by checking/unchecking the check box besides them, inside the filter pop-up window. At least one value needed to be checked state while filtering otherwise "Ok" button will be disabled.



A dialog box titled "PivotTable Field List". It contains a section labeled "Choose fields to add to report:" followed by a list of fields. Each field has a checked checkbox to its left. The fields listed are: Amount, Country, Date, Product, Quantity, and State. A mouse cursor is visible over the right side of the list.

Field	Selected
Amount	<input checked="" type="checkbox"/>
Country	<input checked="" type="checkbox"/>
Date	<input checked="" type="checkbox"/>
Product	<input checked="" type="checkbox"/>
Quantity	<input checked="" type="checkbox"/>
State	<input checked="" type="checkbox"/>





### Value Sorting

**Information:** This feature is applicable for Relational datasource only at Client Mode.

Value Sorting allows to sort columns and rows based on value fields.

The headers of the column to be sorted is given in the 'headerText' property under 'valueSortSettings' in field wise order separated by a string. The string which is used to separate the headers is given in the property 'headerDelimiters'.

### HTML

```
<script type="text/babel">
var pivot_dataset = []; // data source
var pivotdataSource = {
  data: pivot_dataset,
  rows: [{ fieldName: "Country", fieldCaption: "Country" }, { fieldName:
"State", fieldCaption: "State" }],
  columns: [{ fieldName: "Product", fieldCaption: "Product" }],
  values: [{ fieldName: "Amount", fieldCaption: "Amount" }, { fieldName:
"Quantity", fieldCaption: "Quantity" }],
  filters: []
};
var valueSort = {
  headerText: "Bike##Quantity",
  headerDelimiters: "##",
  sortOrder: ej.PivotAnalysis.SortOrder.Descending
};
$(function() {
  ReactDOM.render(
    <EJ.PivotGrid id="Relational" dataSource= {pivotdataSource}
valueSortSettings= {valueSort}></EJ.PivotGrid>,
    document.getElementById('PivotGrid1')
  );
});
</script>
```

	Bike		Car		Van		Grand Total	
	Amount	Quantity	Amount	Quantity	Amount	Quantity	Amount	Quantity
Canada	99960	1070	99260	1168	96820	925	296040	3163
France	106300	998	87300	975	93600	707	287200	2680
Germany	123850	1195	119210	1183	136860	1378	379920	3756
United Kingdom	10840	188	13780	266	17140	226	41760	680
United States	135300	1753	134990	1570	122320	1647	392610	4970
Grand Total	476250	5204	454540	5162	466740	4883	1397530	15249

	Bike		Car		Van		Grand Total	
	Amount	Quantity ▼	Amount	Quantity	Amount	Quantity	Amount	Quantity
United States	135300	1753	134990	1570	122320	1647	392610	4970
Germany	123850	1195	119210	1183	136860	1378	379920	3756
Canada	99960	1070	99260	1168	96820	925	296040	3163
France	106300	998	87300	975	93600	707	287200	2680
United Kingdom	10840	188	13780	266	17140	226	41760	680
Grand Total	476250	5204	454540	5162	466740	4883	1397530	15249

### Calculated Field

**Note:** This feature is applicable only for Relational data source.

PivotGrid provides support to insert a new calculated field based on the existing Pivot Fields either through Calculated Field dialog or code behind.

### Through UI

To insert a new calculated Field, open the Calculated Field dialog using the Grouping Bar context menu. We can define "Name" for the new Calculated Field and "Formula" can be entered by inserting required fields through Fields section. For inserting numbers and operators, you can use formula pop-up as shown in the below screen-shot.

Calculated Field

Name:

Price

▼

Add

Formula:

Amount \* 15

✕

Σ

Delete

Fields:

	+	-	*	/
	7	8	9	%
Amount	4	5	6	^
Quantity	1	2	3	(
	.	0	C	)

Insert Field

OK

Cancel

Click **Add** for adding the respective Calculated Field and **OK** to populate the PivotGrid control.

#### Through Code-behind

For client mode, Calculated Field can be created at code-behind by defining formula based on the existing Pivot Fields in the PivotGrid. To indicate a field as a calculated field we need to set [isCalculatedField](#) property to true and [formula](#) property to set the expression.

#### HTML

```
<script type="text/babel">
var pivot_dataset = []; // data source
var pivotdataSource = {
  //...
  values: [{ fieldName: "Amount", fieldCaption: "Amount" },
    {
      fieldName: "Price",
      fieldCaption: "Price",
      isCalculatedField: true,
      formula: "Amount*15"
    }
  ],
  filters: []
};
$(function() {
  ReactDOM.render(
    <EJ.PivotGrid id="Relational" dataSource= {pivotdataSource}
    enableGroupingBar={true}></EJ.PivotGrid>,

```

```
document.getElementById('PivotGrid1')
);
});
</script>
```

Drag field here								
Amount X	Price X	Product ▼ ^ X						
Country ▼ ^ X	Bike		Car		Van		Grand Total	
	Amount	Price	Amount	Price	Amount	Price	Amount	Price
Canada	250	3750	500	7500	300	4500	1050	15750
France	200	3000	300	4500	400	6000	900	13500
Germany	300	4500	400	6000	350	5250	1050	15750
United Kingdom	150	2250					150	2250
United States	300	4500	650	9750	500	7500	1450	21750
Grand Total	1200	18000	1850	27750	1550	23250	4600	69000

### Cell Editing

**Information:** This feature is applicable only for Relational data source.

Cell editing allows you to edit and alter the values in PivotGrid. The summary values will be recreated based on the edited values. By selecting multiple cells (like in cell selection feature), you can edit multiple cells at the same time.

You can enable cell editing option in PivotGrid by setting the [enableCellEditing](#) property to true.

### HTML

```
<script type="text/babel">
//..
$(function() {
  ReactDOM.render(
    <EJ.PivotGrid id="Relational" enableCellEditing={true}></EJ.PivotGrid>,
    document.getElementById('PivotGrid1')
  );
});
</script>
```

	FY 2005	FY 2006	FY 2007	FY 2008	Grand Total
	Amount	Amount	Amount	Amount	Amount
Canada	100	400	400	150	1050
France	<input type="text" value="200"/>	<input type="text" value="250"/>	<input type="text" value="300"/>	150	900
Germany	<input type="text" value="100343"/>	<input type="text" value="200"/>	<input type="text" value="400"/>	350	1050
United Kingdom	<input type="text" value="12000"/>	<input type="text"/>	<input type="text"/>	150	150
United States	400	100	550	400	1450
Grand Total	800	950	1650	1200	4600

### Sub Total Hiding

**Note:** This feature is applicable only for Relational data source.

You can hide the **Sub Total** for respective fields in rows and columns by setting the property [showSubTotal](#) to **false**

### HTML

```
<script type="text/babel">
//...
var pivotdataSource = {
  data: pivot_dataset,
  rows: [{ fieldName: "Country", fieldCaption: "Country", showSubTotal: false },
        { fieldName: "State", fieldCaption: "State" }],
  columns: [{ fieldName: "Product", fieldCaption: "Product" }],
  values: [{ fieldName: "Amount", fieldCaption: "Amount" }, { fieldName:
    "Quantity", fieldCaption: "Quantity" }],
  filters: []
};
//...
</script>
```

		Canada		France		Germany		United Kingdom		United States		Grand Total	
		Amount	Quantity	Amount	Quantity	Amount	Quantity	Amount	Quantity	Amount	Quantity	Amount	Quantity
▼ FY 2005	Bike	100	2	200	2	100	2			200	4	600	10
	Van									200	4	200	4
▼ FY 2006	Bike									100	2	100	2
	Car	200	4									200	4
▼ FY 2007	Van	200	3	250	4	200	3					650	10
	Car	300	4	300	3	250	3			250	4	1100	14
▼ FY 2008	Van	100	1			150	3			300	4	550	8
	Bike	150	3			200	4	150	5			500	12
	Car					150	4			400	6	550	10
Grand Total		1050	17	900	11	1050	19	150	5	1450	24	4600	76

### Collapse by default

**Information:** This feature is applicable only for Relational datasource.

Allows you to collapse all members displayed in the grid. You can enable collapsing all members by default in PivotGrid by setting [enableCollapseByDefault](#) property to true.

### HTML

```
<script type="text/babel">
//..
$(function() {
  ReactDOM.render(
    <EJ.PivotGrid id="Relational" dataSource= {pivotdataSource}
    enableCollapseByDefault= {true}></EJ.PivotGrid>,
    document.getElementById('PivotGrid1')
  );
});
</script>
```

	Bike	Car	Van	Grand Total
	Quantity	Quantity	Quantity	Quantity
► Canada	5	8	4	17
► France	2	3	6	11
► Germany	6	7	6	19
► United Kingdom	5			5
► United States	6	10	8	24
Grand Total	24	28	24	76

## OLAP

### KPI

Key Performance Indicators (KPIs) are business metric that help to figure out the progress of an enterprise in meeting its business goals.

The different indicators available in KPI are:

- KPI Value: A physical measure or a calculated measure.
- KPI Goal: Defines the target for the measure.
- KPI Status: Evaluates the current status of the value compared to the goal.
- KPI Trend: Evaluate the current trend of the value compared to the goal.

The “**KpiElements**” class in OLAP Base library holds the KPI name and when its object is added to an OlapReport, you can view the resultant information in PivotGrid.

### HTML

```

<!--Create a tag which acts as a container for PivotGrid-->
<div id="PivotGrid1" style="height: 350px; width: 100%; overflow:
auto"></div>
<script type="text/babel">
var Olap_dataSource={
data: "http://bi.syncfusion.com/olap/msmdpump.dll",
catalog: "Adventure Works DW 2008 SE", //"Adventure Works DW 2008 SEtandard
Edition
cube: "Adventure Works", rows: [{ fieldName: "[Date].[Fiscal]" }], columns:
[{ fieldName: "[Product].[Product Categories]" }],
values: [{ measures: [{ fieldName: "[Measures].[Internet Sales Amount]" },{
fieldName: "[Measures].[Growth in Customer Base Trend]" }], axis: "columns"
}]
};
$(function(){
ReactDOM.render(
<EJ.PivotGrid id="Olap" dataSource= {Olap_dataSource}></EJ.PivotGrid>,
document.getElementById('PivotGrid1')
);
});
</script>

```

	► Accessories		► Bikes	
	Internet Sales Amount	Growth in Customer Base Trend	Internet Sales Amount	Growth in Customer Base Trend
► FY 2002		▲	\$7,072,084.24	▲
► FY 2003		▲	\$5,762,134.30	●
► FY 2004	\$667,015.32	▲	\$15,483,926.11	●
► FY 2005	\$33,744.64	◆		◆
Total	\$700,759.96	▲	\$28,318,144.65	▲

### Named Sets

Named sets is a multidimensional expression (MDX) that returns a set of dimension members, which can be created by combining cube data, arithmetic operators, numbers and functions.

You can bind the Named Sets in PivotGrid by setting it's unique name in the [fieldName](#) property either in row or column axis and [isNamedSets](#) boolean property to "true".

### HTML

```

<!--Create a tag which acts as a container for PivotGrid-->
<div id="PivotGrid1" style="height: 350px; width: 100%; overflow:
auto"></div>
<script type="text/babel">
var Olap_dataSource={
data: "http://bi.syncfusion.com/olap/msmdpump.dll",
catalog: "Adventure Works DW 2008 SE", //"Adventure Works DW 2008 SEtandard
Edition
cube: "Adventure Works", rows: [{ fieldName: "[Date].[Fiscal]" }], columns:
[{ fieldName: "[Core Product Group]", isNamedSets: true }],

```

```

values: [{ measures: [{ fieldName: "[Measures].[Internet Sales Amount]" },
axis: "columns" }]
};
$(function() {
ReactDOM.render(
<EJ.PivotGrid id="Olap" dataSource= {Olap_dataSource}></EJ.PivotGrid>,
document.getElementById('PivotGrid1')
);
});
</script>

```

	Accessories	Bikes				Clothing
		Mountain Bikes	Road Bikes	Touring Bikes	Total	
	Internet Sales Amount	Internet Sales Amount	Internet Sales Amount	Internet Sales Amount	Internet Sales Amount	Internet Sales Amount
► FY 2002		\$1,341,121.04	\$5,730,963.20		\$7,072,084.24	
► FY 2003		\$2,214,410.01	\$3,547,724.28		\$5,762,134.30	
► FY 2004	\$667,015.32	\$6,397,228.51	\$5,241,896.55	\$3,844,801.05	\$15,483,926.11	\$322,676.62
► FY 2005	\$33,744.64					\$17,095.99
Total	\$700,759.96	\$9,952,759.56	\$14,520,584.04	\$3,844,801.05	\$28,318,144.65	\$339,772.61

## Grouping Bar

### Initialization

Grouping Bar allows user to dynamically alter the report by filter and remove operations in the PivotGrid control. Based on the OLAP datasource and report bound to the PivotGrid control, Grouping Bar will be automatically populated. You can enable Grouping Bar option in PivotGrid by setting the [enableGroupingBar](#) property to true.

### HTML

```

<script type="text/babel">
//...
$(function() {
ReactDOM.render(
<EJ.PivotGrid id="Olap" dataSource= {Olap_dataSource}
enableGroupingBar={true}></EJ.PivotGrid>,
document.getElementById('PivotGrid1')
);
});
</script>

```



Drag field here							
Internet Sales A...	Customer Geography	Measures					
	Australia	Canada	France	Germany	United Kingdom	United States	Total
Date.Fiscal	Internet Sales Amount						
FY 2002	\$2,568,701.39	\$573,100.97	\$414,245.32	\$513,353.17	\$550,507.33	\$2,452,176.07	\$7,072,084.24
FY 2003	\$2,099,585.43	\$305,010.69	\$633,399.70	\$593,247.24	\$696,594.97	\$1,434,296.26	\$5,762,134.30
FY 2004	\$4,383,479.54	\$1,088,879.50	\$1,592,880.75	\$1,784,107.09	\$2,140,388.50	\$5,483,882.67	\$16,473,618.05
FY 2005	\$9,234.23	\$10,853.70	\$3,491.95	\$3,604.83	\$4,221.41	\$19,434.51	\$50,840.63
Total	\$9,061,000.58	\$1,977,844.86	\$2,644,017.71	\$2,894,312.34	\$3,391,712.21	\$9,389,789.51	\$29,358,677.22

### Drag and Drop

You can alter the report on fly through drag-and-drop operation.

Drag field here							
Internet Sales A...	Customer Geography	Measures					
	Australia	Canada	France	Germany	United Kingdom	United States	Total
Date.Fiscal	Internet Sales Amount						
FY 2002	\$2,568,701.39	\$573,100.97	\$414,245.32	\$513,353.17	\$550,507.33	\$2,452,176.07	\$7,072,084.24
FY 2003	\$2,099,585.43	\$305,010.69	\$633,399.70	\$593,247.24	\$696,594.97	\$1,434,296.26	\$5,762,134.30
FY 2004	\$4,383,479.54	\$1,088,879.50	\$1,592,880.75	\$1,784,107.09	\$2,140,388.50	\$5,483,882.67	\$16,473,618.05
FY 2005	\$9,234.23	\$10,853.70	\$3,491.95	\$3,604.83	\$4,221.41	\$19,434.51	\$50,840.63
Total	\$9,061,000.58	\$1,977,844.86	\$2,644,017.71	\$2,894,312.34	\$3,391,712.21	\$9,389,789.51	\$29,358,677.22

### Context Menu

You can also alter the report by using context menu.

Drag field here							
Internet Sales A...	Customer Geography	Measures					
	Australia	Canada	France	Germany	United Kingdom	United States	Total
Date.Fiscal	Internet Sales Amount						
FY 2002	\$2,568,701.39	\$573,100.97	\$414,245.32	\$513,353.17	\$550,507.33	\$2,452,176.07	\$7,072,084.24
FY 2003	\$2,099,585.43	\$305,010.69	\$633,399.70	\$593,247.24	\$696,594.97	\$1,434,296.26	\$5,762,134.30
FY 2004	\$4,383,479.54	\$1,088,879.50	\$1,592,880.75	\$1,784,107.09	\$2,140,388.50	\$5,483,882.67	\$16,473,618.05
FY 2005	\$9,234.23	\$10,853.70	\$3,491.95	\$3,604.83	\$4,221.41	\$19,434.51	\$50,840.63
Total	\$9,061,000.58	\$1,977,844.86	\$2,644,017.71	\$2,894,312.34	\$3,391,712.21	\$9,389,789.51	\$29,358,677.22

### Searching Values

Search option available in Grouping Bar allows you to search a specific value that needs to be filtered from the list of values inside the filter pop-up window.

Drag field here

Internet Sales A...	Customer Geography	Measures
---------------------	--------------------	----------

	► Australia	► Canada	► France	► Germany	► United Kingdom	► United States	Total
--	-------------	----------	----------	-----------	------------------	-----------------	-------

Date: Fiscal ▼

	Internet Sales Amount						
► FY 2002	\$2,568,701.39	\$573,100.97	\$414,245.32	\$513,353.17	\$550,507.33	\$2,452,176.07	\$7,072,084.24
► FY 2003	\$2,099,585.43	\$305,010.69	\$633,399.70	\$593,247.24	\$696,594.97	\$1,434,296.26	\$5,762,134.30
► FY 2004	\$4,383,479.54	\$1,088,879.50	\$1,592,880.75	\$1,784,107.09	\$2,140,388.50	\$5,483,882.67	\$16,473,618.05
► FY 2005	\$9,234.23	\$10,853.70	\$3,491.95	\$3,604.83	\$4,221.41	\$19,434.51	\$50,840.63
Total	\$9,061,000.58	\$1,977,844.86	\$2,644,017.71	\$2,894,312.34	\$3,391,712.21	\$9,389,789.51	\$29,358,677.22

Customer Geography

aus

► ☒ Australia

OK Cancel

### Filtering Values

Filtering option available in Grouping Bar allows you to select a specific set of values that needs to be displayed in the PivotGrid control. At least one value needed to be in checked state while filtering otherwise "Ok" button will be disabled.

Drag field here

Internet Sales A...	Customer Geography	Measures
---------------------	--------------------	----------

	► Australia	► Canada	► France	► Germany	► United Kingdom	► United States	Total
--	-------------	----------	----------	-----------	------------------	-----------------	-------

Date: Fiscal ▼

	Internet Sales Amount						
► FY 2002	\$2,568,701.39	\$573,100.97	\$414,245.32	\$513,353.17	\$550,507.33	\$2,452,176.07	\$7,072,084.24
► FY 2003	\$2,099,585.43	\$305,010.69	\$633,399.70	\$593,247.24	\$696,594.97	\$1,434,296.26	\$5,762,134.30
► FY 2004	\$4,383,479.54	\$1,088,879.50	\$1,592,880.75	\$1,784,107.09	\$2,140,388.50	\$5,483,882.67	\$16,473,618.05
► FY 2005	\$9,234.23	\$10,853.70	\$3,491.95	\$3,604.83	\$4,221.41	\$19,434.51	\$50,840.63
Total	\$9,061,000.58	\$1,977,844.86	\$2,644,017.71	\$2,894,312.34	\$3,391,712.21	\$9,389,789.51	\$29,358,677.22

Customer Geography

Search Customer Geography

- ☒ (All)
- ☐ Australia
- ☒ Canada
- ☒ France
- ☒ Germany
- ☒ United Kingdom
- ☒ United States

OK Cancel

### Removing Field

Remove option available in the Grouping Bar allows you to completely remove a specific field from the PivotGrid control. Remove operation can be either achieved by clicking remove icon available inside each field or by dragging and dropping field out of Grouping Bar region.

Drag field here							
Internet Sales A...	Customer Geography	Measures					
	► Australia	► Canada	► France	► Germany	► United Kingdom	► United States	Total
Date:Fiscal	Internet Sales Amount						
► FY 2002	\$2,568,701.39	\$573,100.97	\$414,245.32	\$513,353.17	\$550,507.33	\$2,452,176.07	\$7,072,084.24
► FY 2003	\$2,099,585.43	\$305,010.69	\$633,399.70	\$593,247.24	\$696,594.97	\$1,434,296.26	\$5,762,134.30
► FY 2004	\$4,383,479.54	\$1,088,879.50	\$1,592,880.75	\$1,784,107.09	\$2,140,388.50	\$5,483,882.67	\$16,473,618.05
► FY 2005	\$9,234.23	\$10,853.70	\$3,491.95	\$3,604.83	\$4,221.41	\$19,434.51	\$50,840.63
Total	\$9,061,000.58	\$1,977,844.86	\$2,644,017.71	\$2,894,312.34	\$3,391,712.21	\$9,389,789.51	\$29,358,677.22

Drag field here	
Internet Sales A...	Measures
Date:Fiscal	Internet Sales Amount
► FY 2002	\$7,072,084.24
► FY 2003	\$5,762,134.30
► FY 2004	\$16,473,618.05
► FY 2005	\$50,840.63
Total	\$29,358,677.22

## PivotTable Field List

### Initialization

Field List, also known as Pivot Schema Designer, allows user to add, rearrange, filter and remove fields to show data in PivotGrid exactly the way they want.

Based on the datasource, OLAP, bound to the PivotGrid control, PivotTable Field List will be automatically populated with Cube Information or Field Names. PivotTable Field List provides an Excel like appearance and behavior.

In-order to initialize PivotTable Field List, first you need to define a “div” tag with an appropriate “id” attribute which acts as a container for the widget. Then you need to initialize the PivotTable Field List by using the “**EJ.PivotSchemaDesigner**” method.

### HTML

```
<html>
//...
<body>
<!--Create a tag which acts as a container for PivotGrid-->
```

```

<div id="PivotGrid1" style="width: 55%; height: 670px; overflow: scroll;
float: left;"></div>
<!--Create a tag which acts as a container for PivotTable Field List-->
<div id="PivotSchemaDesigner1" style="height:650px;width:40%;">
</div>
<script type="text/babel">
//...
$(function() {
ReactDOM.render(
<EJ.PivotSchemaDesigner id="PivotSchemaDesigner"></EJ.PivotSchemaDesigner>,
document.getElementById('PivotSchemaDesigner1')
);
ReactDOM.render(
<EJ.PivotGrid id="Olap" dataSource= {Olap_dataSource}
enableGroupingBar={true}
pivotTableFieldListID="PivotSchemaDesigner"></EJ.PivotGrid>,
document.getElementById('PivotGrid1')
);
});
</script>
</html>

```

	Australia	Canada	France	Germany	United Kingdom	United States	Total
Internet Sales Amount							
FY 2002	\$2,568,701.39	\$573,100.97	\$414,345.32	\$513,353.17	\$550,507.33	\$2,452,176.07	\$7,072,084.24
FY 2003	\$2,099,585.43	\$305,010.09	\$633,399.70	\$593,247.24	\$686,594.97	\$1,434,290.26	\$5,762,154.30
FY 2004	\$4,383,479.54	\$1,088,879.53	\$1,532,880.75	\$1,784,107.09	\$2,140,388.50	\$5,489,862.67	\$16,473,618.05
FY 2005	\$9,234.23	\$10,853.70	\$3,491.95	\$3,604.83	\$4,221.41	\$19,434.51	\$50,540.63
Total	\$9,061,000.56	\$1,977,844.86	\$2,644,017.71	\$2,894,512.34	\$3,591,712.21	\$9,589,789.51	\$29,558,677.22

### PivotTable Field List

Choose fields to add to report:

- Measures
- Account
- Customer
- Date

Drag fields between areas below:

Report Filter	Column Label
	[Customer] [Customer Geography]
	Measures
Row Label	Values
[Date] [Fiscal]	[Measures] [Internet Sales Amount]

### Layout

The top portion of the layout shows Field or Cube items in a categorized way. They can be dynamically added into the report either by drag and drop option or through simple check box selection.

On item(s) selection they will be placed in Row section by default except numeric based item(s) or measures, which will alone be placed in the Value section by default.

The bottom portion of the layout is segregated as below.

- Report Filter: Exclusively designed to filter an item(s) placed in this particular position of the layout.
- Value Section: The value label usually displays the numeric value item(s) present in the report.

- Column Section: It is used to display item(s) as column header and values in the PivotGrid control.
- Row Section: It is used to display item(s) as row header and values in the PivotGrid control.

### UI Interactions

#### By Drag and Drop

You can alter the report on fly through drag-and-drop operation. You can drag any item from Field List and drop into column, row, value or filter section available at the bottom of the Field List.

**PivotTable Field List**

Choose fields to add to report:

Customer
Location
Demographic
Contacts
Other
Customer Geography

Drag fields between areas below:

Report Filter

Column Label
Measures
+ Customer Geography

Row Label
Fiscal

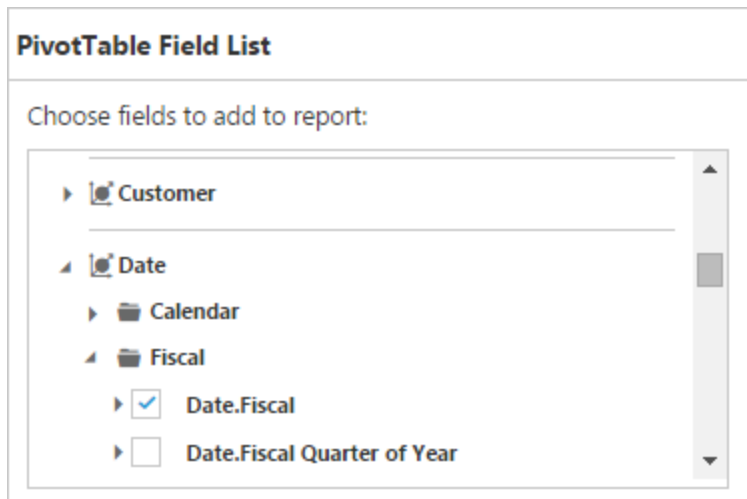
Values
[Internet Sales Amount]

☐ Defer Layout Update

Update

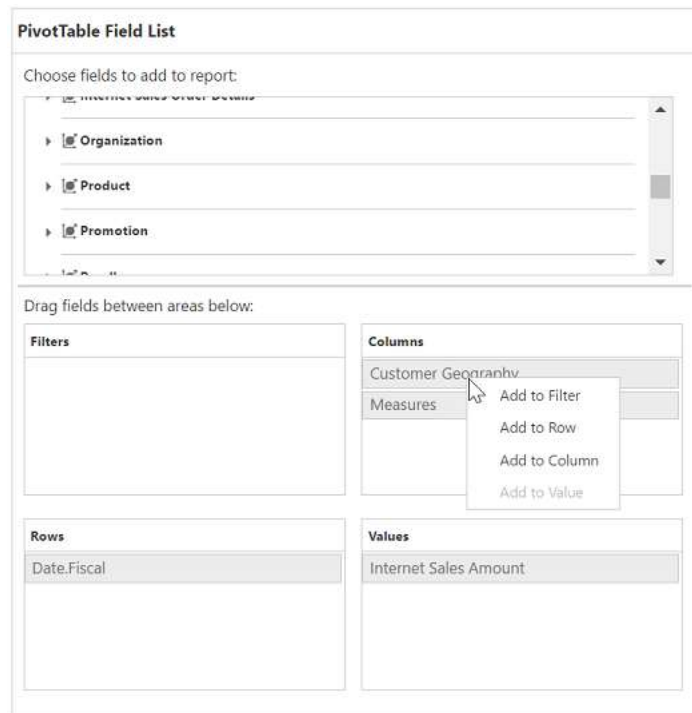
#### By Treeview Selection

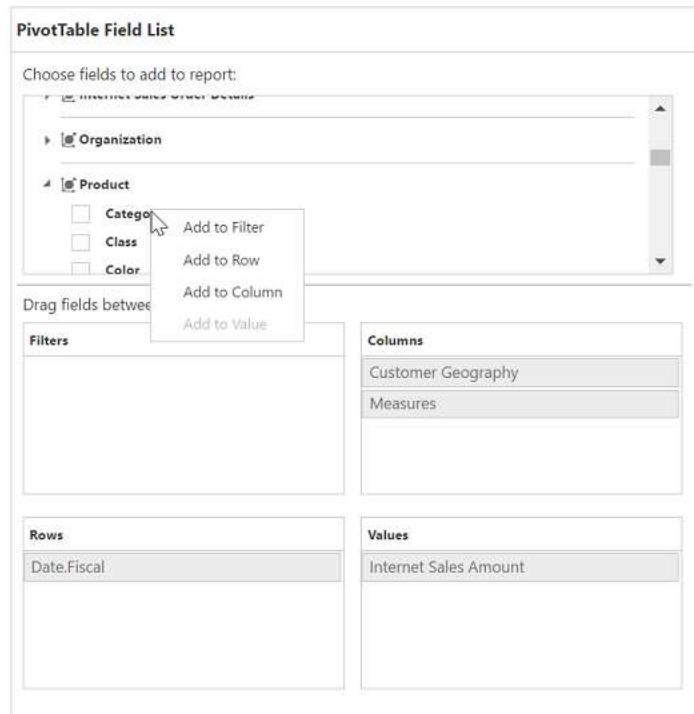
You can also alter the report on fly through check and uncheck option as an alternate. By default, fields will be added to the Row Label when checked.



### By Context Menu

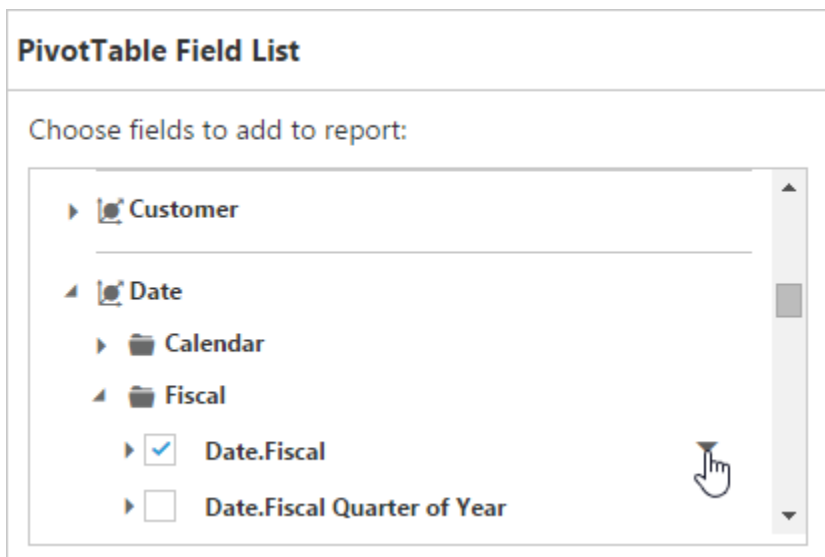
You can also alter the report by using context menu.





### Searching Values

Search option available in Field List allows you to search a specific value that needs to be filtered from the list of values inside the filter pop-up window.

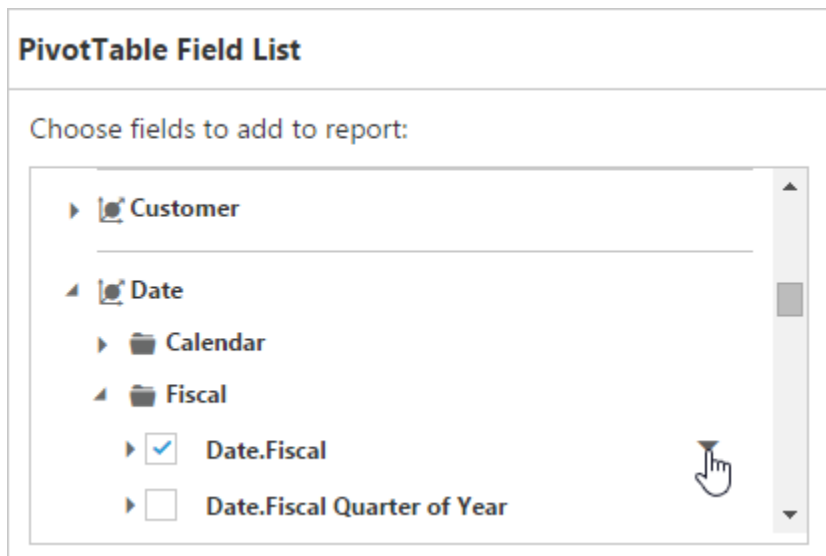






### Filtering

Values can be filtered by checking/unchecking the check box besides them, inside the filter pop-up window. At least one value needed to be checked state while filtering otherwise "Ok" button will be disabled.





## Common

### PivotGrid: Elements

#### Hyperlink

The PivotGrid control supports hyperlink option to link data for each individual cell. Hyperlink can be enabled separately for row header, column header, value and summary cells. Following are the respective properties:

- [enableColumnHeaderHyperlink](#) - Enables hyperlink for column headers.
- [enableRowHeaderHyperlink](#) - Enables hyperlink for row headers.
- [enableSummaryCellHyperlink](#) - Enables hyperlink for summary cell.
- [enableValueCellHyperlink](#) - Enables hyperlink for value cell.

Also hyperlink option provides separate events for row header, column header, value and summary cells as mentioned below.

- [columnHeaderHyperlinkClick](#) - Returns column header information through event on hyperlink click.
- [rowHeaderHyperlinkClick](#) - Returns row header information through event on hyperlink click.
- [summaryCellHyperlinkClick](#) - Returns summary cell information through event on hyperlink click.
- [valueCellHyperlinkClick](#) - Returns value cell information through event on hyperlink click.

#### HTML

```
<script type="text/babel">
//...
var hyperlinkSettings= {
  enableValueCellHyperlink: true,
  enableRowHeaderHyperlink: true,
  enableColumnHeaderHyperlink: true,
  enableSummaryCellHyperlink: true
}
```

```

$(function() {
  ReactDOM.render(
    <EJ.PivotGrid id="PivotGrid" hyperlinkSettings= {hyperlinkSettings}
    valueCellHyperlinkClick= "CellClickEvent" rowHeaderHyperlinkClick=
    "CellClickEvent" columnHeaderHyperlinkClick= "CellClickEvent"
    summaryCellHyperlinkClick= "CellClickEvent"></EJ.PivotGrid>,
    document.getElementById('PivotGrid1')
  );
});
</script>
<script type="text/javascript">
function CellClickEvent(evt) {
  alert("Cell click event is fired.");
}
</script>

```

	► Australia	► Canada	► France	► Germany	► United Kingdom	► United States	Total
	Internet Sales Amount						
► FY 2002	\$2,568,701.39	\$573,100.97	\$414,245.32	\$513,353.17	\$550,507.33	\$2,452,176.07	\$7,072,084.24
► FY 2003	\$2,099,585.43	\$305,010.69	\$633,399.70	\$593,247.24	\$696,594.97	\$1,434,296.26	\$5,762,134.30
► FY 2004	\$4,383,479.54	\$1,088,879.50	\$1,562,880.75	\$1,784,107.09	\$2,140,388.50	\$5,483,882.67	\$16,473,618.05
► FY 2005	\$9,234.23	\$10,853.70	\$3,491.95	\$3,604.83	\$4,221.41	\$19,434.51	\$50,840.63
Total	\$9,061,000.58	\$1,977,844.86	\$2,644,017.71	\$2,894,312.34	\$3,391,712.21	\$9,389,789.51	\$29,358,677.22

### Selection

You can select a particular range of value cells from PivotGrid and manipulate/display them. Cell selection is applicable only for value cells and you can enable this functionality by setting [enableCellSelection](#) property to true.

The [cellSelection](#) event would be triggered as soon as the selection process is over, that is, when the mouse left click is released. The event argument contains a collection of JSON records and header values, which contains information about the selected cells.

### HTML

```

<script type="text/babel">
//...
$(function() {
  ReactDOM.render(
    <EJ.PivotGrid id="PivotGrid" enableCellSelection= {true} cellSelection=
    "valueCellClick"></EJ.PivotGrid>,
    document.getElementById('PivotGrid1')
  );
});
</script>
<script type="text/javascript">
valueCellClick = function(evt) {
  //This event lets you to perform required operation with the selected range
  of cells.
  cellvalue = evt.JSONRecords;

```

```

rowheaders = evt.rowHeader;
colheaders = evt.columnHeader;
}
</script>

```

	► Australia	► Canada	► France	► Germany	► United Kingdom	► United States	Total
Internet Sales Amount							
► FY 2002	\$2,568,701.39	\$573,100.97	\$414,245.32	\$513,353.17	\$550,507.33	\$2,452,176.07	\$7,072,084.24
► FY 2003	\$2,099,585.43	\$305,010.69	\$633,399.70	\$593,247.24	\$696,594.97	\$1,434,296.26	\$5,762,134.30
► FY 2004	\$4,383,479.54	\$1,088,879.50	\$1,592,880.75	\$1,784,107.09	\$2,140,388.50	\$5,483,882.67	\$16,473,618.05
► FY 2005	\$9,234.23	\$10,853.70	\$3,491.95	\$3,604.83	+	\$4,221.41	\$19,434.51
Total	\$9,061,000.58	\$1,977,844.86	\$2,644,017.71	\$2,894,312.34	\$3,391,712.21	\$9,389,789.51	\$29,358,677.22

### Cell Context

Cell context allows user to perform any custom operation on cell right-click. For example, you can create and display context menu on cell right-click.

Cell context is enabled by setting the [enableCellContext](#) property to true. The [cellContext](#) event would be raised as soon as right-click is done providing cell information through event argument.

### HTML

```

<script type="text/babel">
//...
$(function() {
ReactDOM.render(
<EJ.PivotGrid id="PivotGrid" enableCellContext= {true} cellContext=
"cell_RightClick"></EJ.PivotGrid>,
document.getElementById('PivotGrid1')
);
});
</script>
<script type="text/javascript">
cell_RightClick = function(evt) {
//You can write your code here
}
</script>

```

### Conditional Formatting

Conditional formatting in PivotGrid allows user to highlight particular cells with certain color, font-style, font-family etc. based on the condition they have met. Also the condition can be applied for certain Measure alone.

Conditional formatting is enabled by setting [enableConditionalFormatting](#) property to true and the formatting dialog is launched when “**createConditionalDialog**” method is invoked.

### HTML

```

<div id="PivotGrid1" style="height: 350px; width: 100%; overflow:
auto"></div>

```

```

<button id="Button1">Apply formatting</button>
<script type="text/babel">
  //...
  $(function() {
    ReactDOM.render(
      <EJ.PivotGrid id="PivotGrid" enableConditionalFormatting= {true}>
    </EJ.PivotGrid>,
    document.getElementById('PivotGrid1')
    );
  });
</script>
<script type="text/javascript">
  $( "#Button1" ).click(function() {
    var pGridObj = $('#PivotGrid').data("ejPivotGrid");
    if (pGridObj.model.enableConditionalFormatting) {
      pGridObj.createConditionalDialog();
    }
  });
</script>

```

Conditional Formatting ✕

Conditional Type Less Than ▼

Edit Condition Add New ▼ ✕

Value1 \$55555

Value2

Back Color Violet ▼

Border Range 3 ▼

Border Color Yellow ▼

Border Style Dotted ▼

Font Style Algerian ▼

Font Size 12 ▼

Measures All ▼

OK

Cancel

	► Australia	► Canada	► France	► Germany	► United Kingdom	► United States	Total
Internet Sales Amount							
► FY 2002	\$2,568,701.39	\$573,100.97	\$414,245.32	\$513,353.17	\$550,507.33	\$2,452,176.07	\$7,072,084.24
► FY 2003	\$2,099,585.43	\$305,010.89	\$633,399.70	\$593,247.24	\$696,594.97	\$1,434,296.26	\$5,762,134.30
► FY 2004	\$4,383,479.54	\$1,088,879.50	\$1,592,880.75	\$1,784,107.09	\$2,140,388.50	\$5,483,882.67	\$16,473,618.05
► FY 2005	\$9,234.23	\$10,653.70	\$3,491.95	\$3,604.63	\$4,221.41	\$19,434.51	\$50,640.63
Total	\$9,061,000.58	\$1,977,844.86	\$2,644,017.71	\$2,894,312.34	\$3,391,712.21	\$9,389,789.51	\$29,358,677.22

### Export

We can export the PivotGrid with highlighted particular cells along with its formatting styles.

### LIMITATIONS FOR WORD:

The following border styles are not supported

- Solid
- Groove
- Ridge

## LIMITATIONS FOR PDF:

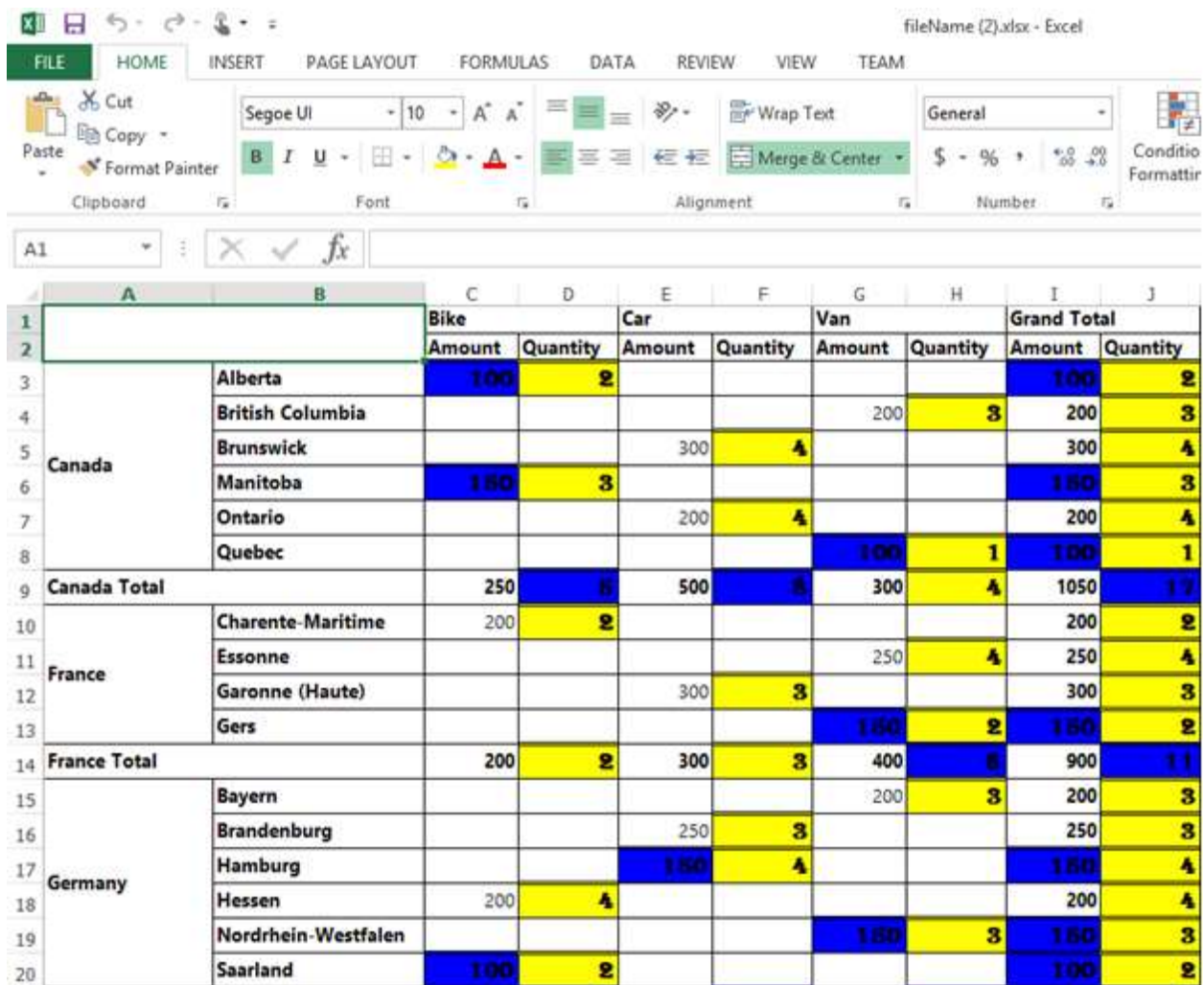
Border styles are not applicable.

## LIMITATIONS FOR EXCEL:

The following border styles are alone supported

- Dashed
- Dotted
- Double

Also border size is not supported.



		Bike		Car		Van		Grand Total	
		Amount	Quantity	Amount	Quantity	Amount	Quantity	Amount	Quantity
Canada	Alberta	100	2					100	2
	British Columbia					200	3	200	3
	Brunswick			300	4			300	4
	Manitoba	180	3					180	3
	Ontario			200	4			200	4
	Quebec					100	1	100	1
Canada Total		250	9	500	8	300	4	1050	17
France	Charente-Maritime	200	2					200	2
	Essonne					250	4	250	4
	Garonne (Haute)			300	3			300	3
	Gers					150	2	150	2
France Total		200	2	300	3	400	6	900	11
Germany	Bayern					200	3	200	3
	Brandenburg			250	3			250	3
	Hamburg			150	4			150	4
	Hessen	200	4					200	4
	Nordrhein-Westfalen					150	3	150	3
	Saarland	100	2					100	2

## Number Format

Allows us to specify the required number format that PivotGrid should use in its values by setting the format option. Following number formats that are supported:

- number
- decimal
- currency
- percentage
- date
- time
- scientific
- accounting
- fraction

## RELATIONAL

### JS

```
<script type="text/babel">
var pivot_dataset = []; // data source
var pivotdataSource = {
  data: pivot_dataset,
  //...
  values: [
    { fieldName: "Amount", fieldCaption: "Amount", format: "currency" },
    { fieldName: "Quantity", fieldCaption: "Quantity", format: "decimal" }
  ]
};
</script>
```

	Bike		Car		Van		Grand Total	
	Amount	Quantity	Amount	Quantity	Amount	Quantity	Amount	Quantity
► Canada	\$250.00	5.00	\$500.00	8.00	\$300.00	4.00	\$1,050.00	17.00
► France	\$200.00	2.00	\$300.00	3.00	\$400.00	6.00	\$900.00	11.00
► Germany	\$300.00	6.00	\$400.00	7.00	\$350.00	6.00	\$1,050.00	19.00
► United Kingdom	\$150.00	5.00					\$150.00	5.00
► United States	\$300.00	6.00	\$650.00	10.00	\$500.00	8.00	\$1,450.00	24.00
Grand Total	\$1,200.00	24.00	\$1,850.00	28.00	\$1,550.00	24.00	\$4,600.00	76.00

## OLAP

### HTML

```
<script type="text/babel">
var Olap_dataSource={
  //...
  values: [{ measures: [{ fieldName: "[Measures].[Internet Sales Amount]",
    format: "percent" //Specify the format here
  }],
  axis: "columns"
}]
};
</script>
```

	► Australia	► Canada	► France	► Germany	► United Kingdom	► United States	Total
Internet Sales Amount							
► FY 2002	256,870,139 %	57,310,097 %	41,424,532 %	51,335,317 %	55,050,733 %	245,217,607 %	707,208,424 %
► FY 2003	209,958,543 %	30,501,069 %	63,339,970 %	59,324,724 %	69,659,497 %	143,429,626 %	576,213,430 %
► FY 2004	438,347,954 %	108,887,950 %	159,288,075 %	178,410,709 %	214,038,850 %	548,388,267 %	1,647,361,805 %
► FY 2005	923,423 %	1,085,370 %	349,195 %	360,483 %	422,141 %	1,943,451 %	5,084,063 %
Total	906,100,058 %	197,784,486 %	264,401,771 %	289,431,234 %	339,171,221 %	938,978,951 %	2,935,867,722 %

## Save and Load Report

Allows you to save the current report of PivotGrid and render the control with the saved report later.

### Save Report to Local Storage

To save the current report of PivotGrid to local storage, we need to call the [saveReport](#) method in PivotGrid.

### HTML

```
<div id="PivotGrid1"> </div>
<button id="btnSave">save</button>
<script type="text/babel">
//...
$(function() {
  ReactDOM.render(
    <EJ.PivotGrid id="PivotGrid" saveReport= "saveToLocal"></EJ.PivotGrid>,
    document.getElementById('PivotGrid1')
  );
});
</script>
<script type="text/javascript">
$( "#btnSave" ).click(function() {
  var pGridObj = $('#PivotGrid').data("ejPivotGrid");
  url = "",
  name = "report",
  storage = "local",
  pGridObj.saveReport(name, storage, url);
});
function saveToLocal(args) {
  localStorage.setItem("report", JSON.stringify(args.report));
}
</script>
```

### Load Report from Local Storage

To load the stored report of PivotGrid from local storage, we need to call the [loadReport](#) method in PivotGrid.

### HTML

```
<div id="PivotGrid1"> </div>
<button id="btnLoad">load</button>
<script type="text/babel">
//...
$(function() {
  ReactDOM.render(
    <EJ.PivotGrid id="PivotGrid" loadReport= "loadFromLocal"></EJ.PivotGrid>,
```



```

document.getElementById('PivotGrid1')
);
});
</script>
<script type="text/javascript">
$( "#btnLoad" ).click(function() {
var pGridObj = $('#PivotGrid').data("ejPivotGrid");
url = "",
name = "report",
storage = "local",
pGridObj.loadReport(name, storage, url);
});
function loadFromLocal(args) {
args.targetControl.model.dataSource =
JSON.parse(localStorage.getItem("report"));
}
</script>

```

## Grid Layout

### Normal Layout

A layout in which summary cells are positioned at the bottom of each parent member and their child members appear next to them. Normal layout is the default layout in PivotGrid control. The enumeration property [layout](#) needs to be set to “**ej.PivotGrid.Layout.Normal**” in-order to view PivotGrid in normal layout.

### HTML

```

<script type="text/babel">
//...
$(function() {
ReactDOM.render(
<EJ.PivotGrid id="PivotGrid" layout=
{ej.PivotGrid.Layout.Normal}></EJ.PivotGrid>,
document.getElementById('PivotGrid1')
);
});
</script>

```

		► Australia	► Canada	► France	► Germany	► United Kingdom	► United States	Total
		Internet Sales Amount						
► FY 2002		\$2,568,701.39	\$573,100.97	\$414,245.32	\$513,353.17	\$550,507.33	\$2,452,176.07	\$7,072,084.24
▼ FY 2003	► H1 FY 2003	\$894,630.70	\$195,331.22	\$281,268.39	\$245,662.66	\$332,670.05	\$775,069.93	\$2,724,632.94
	► H2 FY 2003	\$1,204,954.73	\$109,679.47	\$352,131.31	\$347,584.58	\$363,924.93	\$659,226.34	\$3,037,501.36
	Total	\$2,099,585.43	\$305,010.69	\$633,399.70	\$593,247.24	\$696,594.97	\$1,434,296.26	\$5,762,134.30
► FY 2004		\$4,383,479.54	\$1,088,879.50	\$1,592,880.75	\$1,784,107.09	\$2,140,388.50	\$5,483,882.67	\$16,473,618.05
► FY 2005		\$9,234.23	\$10,853.70	\$3,491.95	\$3,604.83	\$4,221.41	\$19,434.51	\$50,840.63
Total		\$9,061,000.58	\$1,977,844.86	\$2,644,017.71	\$2,894,312.34	\$3,391,712.21	\$9,389,789.51	\$29,358,677.22

### No Summaries Layout

**Information:** This feature is applicable only for OLAP datasource.

A layout in which summary cells are completely hidden and the child members appear next to their parent member. The enumeration property [layout](#) needs to be set to “**ej.PivotGrid.Layout.NoSummaries**” in-order to view PivotGrid without summaries.

### HTML

```
<script type="text/babel">
//...
$(function() {
  ReactDOM.render(
    <EJ.PivotGrid id="PivotGrid" layout=
    {ej.PivotGrid.Layout.NoSummaries}></EJ.PivotGrid>,
    document.getElementById('PivotGrid1')
  );
});
</script>
```

		► Australia	► Canada	► France	► Germany	► United Kingdom	► United States
		Internet Sales Amount					
► FY 2002		\$2,568,701.39	\$573,100.97	\$414,245.32	\$513,353.17	\$550,507.33	\$2,452,176.07
▼ FY 2003	► H1 FY 2003	\$894,630.70	\$195,331.22	\$281,268.39	\$245,662.66	\$332,670.05	\$775,069.93
	► H2 FY 2003	\$1,204,954.73	\$109,679.47	\$352,131.31	\$347,584.58	\$363,924.93	\$659,226.34
► FY 2004		\$4,383,479.54	\$1,088,879.50	\$1,592,880.75	\$1,784,107.09	\$2,140,388.50	\$5,483,882.67
► FY 2005		\$9,234.23	\$10,853.70	\$3,491.95	\$3,604.83	\$4,221.41	\$19,434.51

### Excel-like Layout

A layout in which summary cells are positioned besides each parent member and their child members appear next to them. The enumeration property [layout](#) needs to be set to “**ej.PivotGrid.Layout.ExcelLikeLayout**” in-order to view PivotGrid in excel-like layout.

### HTML

```
<script type="text/babel">
//...
$(function() {
  ReactDOM.render(
    <EJ.PivotGrid id="PivotGrid" layout=
    {ej.PivotGrid.Layout.ExcelLikeLayout}></EJ.PivotGrid>,
    document.getElementById('PivotGrid1')
  );
});
</script>
```

	► Australia	► Canada	► France	► Germany	► United Kingdom	► United States	Total
Internet Sales Amount							
► FY 2002	\$2,568,701.39	\$573,100.97	\$414,245.32	\$513,353.17	\$550,507.33	\$2,452,176.07	\$7,072,084.24
▼ FY 2003	\$2,099,585.43	\$305,010.69	\$633,399.70	\$593,247.24	\$696,594.97	\$1,434,296.26	\$5,762,134.30
► H1 FY 2003	\$894,630.70	\$195,331.22	\$281,268.39	\$245,662.66	\$332,670.05	\$775,069.93	\$2,724,632.94
► H2 FY 2003	\$1,204,954.73	\$109,679.47	\$352,131.31	\$347,584.58	\$363,924.93	\$659,226.34	\$3,037,501.36
► FY 2004	\$4,383,479.54	\$1,088,879.50	\$1,592,880.75	\$1,784,107.09	\$2,140,388.50	\$5,483,882.67	\$16,473,618.05
► FY 2005	\$9,234.23	\$10,853.70	\$3,491.95	\$3,604.83	\$4,221.41	\$19,434.51	\$50,840.63
Grand Total	\$9,061,000.58	\$1,977,844.86	\$2,644,017.71	\$2,894,312.34	\$3,391,712.21	\$9,389,789.51	\$29,358,677.22

### Grand Total Hiding

Grand Total Hiding can be classified into three categories.

- Row Grand Total Hiding
- Column Grand Total Hiding
- Both

### Row Grand Total Hiding

You can hide the **Grand Total** in row alone by setting the property `enableRowGrandTotal` to `false`

### HTML

```
<script type="text/babel">
//...
$(function() {
  ReactDOM.render(
    <EJ.PivotGrid id="PivotGrid" enableRowGrandTotal= {false}></EJ.PivotGrid>,
    document.getElementById('PivotGrid1')
  );
});
</script>
```

		Canada		France		Germany		United Kingdom		United States		Grand Total	
		Amount	Quantity	Amount	Quantity	Amount	Quantity	Amount	Quantity	Amount	Quantity	Amount	Quantity
▼ FY 2005	Bike	100	2	200	2	100	2			200	4	600	10
	Van									200	4	200	4
FY 2005 Total		100	2	200	2	100	2			400	8	800	14
▼ FY 2006	Bike									100	2	100	2
	Car	200	4									200	4
	Van	200	3	250	4	200	3					650	10
FY 2006 Total		400	7	250	4	200	3			100	2	950	16
▼ FY 2007	Car	300	4	300	3	250	3			250	4	1100	14
	Van	100	1			150	3			300	4	550	8
FY 2007 Total		400	5	300	3	400	6			550	8	1650	22
▼ FY 2008	Bike	150	3			200	4	150	5			500	12
	Car					150	4			400	6	550	10
	Van			150	2							150	2
FY 2008 Total		150	3	150	2	350	8	150	5	400	6	1200	24

### Column Grand Total Hiding

You can hide the **Grand Total** in column alone by setting the property [enableColumnGrandTotal](#) to `false`

### HTML

```
<script type="text/babel">
//...
$(function() {
  ReactDOM.render(
    <EJ.PivotGrid id="PivotGrid" enableColumnGrandTotal=
    {false}></EJ.PivotGrid>,
    document.getElementById('PivotGrid1')
  );
});
</script>
```

		Canada		France		Germany		United Kingdom		United States			
		Amount	Quantity	Amount	Quantity	Amount	Quantity	Amount	Quantity	Amount	Quantity		
▼ FY 2005	Bike	100	2	200	2	100	2			200	4		
	Van									200	4		
FY 2005 Total		100	2	200	2	100	2			400	8		
▼ FY 2006	Bike									100	2		
	Car	200	4										
	Van	200	3	250	4	200	3						
FY 2006 Total		400	7	250	4	200	3			100	2		
▼ FY 2007	Car	300	4	300	3	250	3			250	4		
	Van	100	1			150	3			300	4		
FY 2007 Total		400	5	300	3	400	6			550	8		
▼ FY 2008	Bike	150	3			200	4	150	5				
	Car					150	4			400	6		
	Van			150	2								
FY 2008 Total		150	3	150	2	350	8	150	5	400	6		
Grand Total		1050	17	900	11	1050	19	150	5	1450	24		

*Both*

You can hide the **Grand Total** in both row and column by setting the property [enableGrandTotal](#) to **false**

**HTML**

```
<script type="text/babel">
//...
$(function() {
  ReactDOM.render(
    <EJ.PivotGrid id="PivotGrid" enableGrandTotal= {false}></EJ.PivotGrid>,
    document.getElementById('PivotGrid1')
  );
});
</script>
```

		Canada		France		Germany		United Kingdom		United States	
		Amount	Quantity	Amount	Quantity	Amount	Quantity	Amount	Quantity	Amount	Quantity
▼ FY 2005	Bike	100	2	200	2	100	2			200	4
	Van									200	4
FY 2005 Total		100	2	200	2	100	2			400	8
▼ FY 2006	Bike									100	2
	Car	200	4								
	Van	200	3	250	4	200	3				
FY 2006 Total		400	7	250	4	200	3			100	2
▼ FY 2007	Car	300	4	300	3	250	3			250	4
	Van	100	1			150	3			300	4
FY 2007 Total		400	5	300	3	400	6			550	8
▼ FY 2008	Bike	150	3			200	4	150	5		
	Car					150	4			400	6
	Van			150	2						
FY 2008 Total		150	3	150	2	350	8	150	5	400	6

*Advanced Filtering & Sorting*

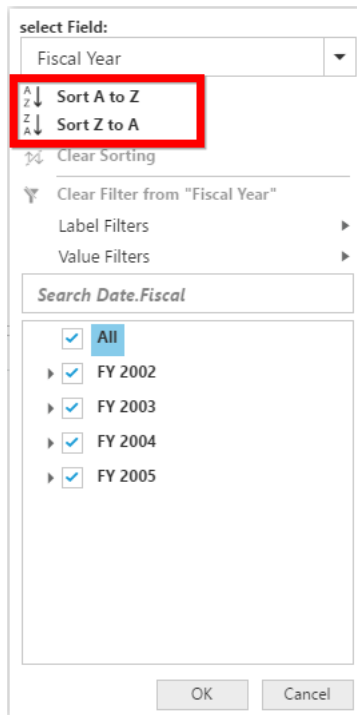
It allows to filter and sort the field members in PivotGrid. You can enable the Advanced Filtering and Sorting option in PivotGrid by setting the [enableAdvancedFilter](#) property to true.

**HTML**

```
<script type="text/babel">
//...
$(function() {
  ReactDOM.render(
    <EJ.PivotGrid id="PivotGrid" enableGroupingBar={true} enableAdvancedFilter=
    {true}></EJ.PivotGrid>,
    document.getElementById('PivotGrid1')
  );
});
</script>
```

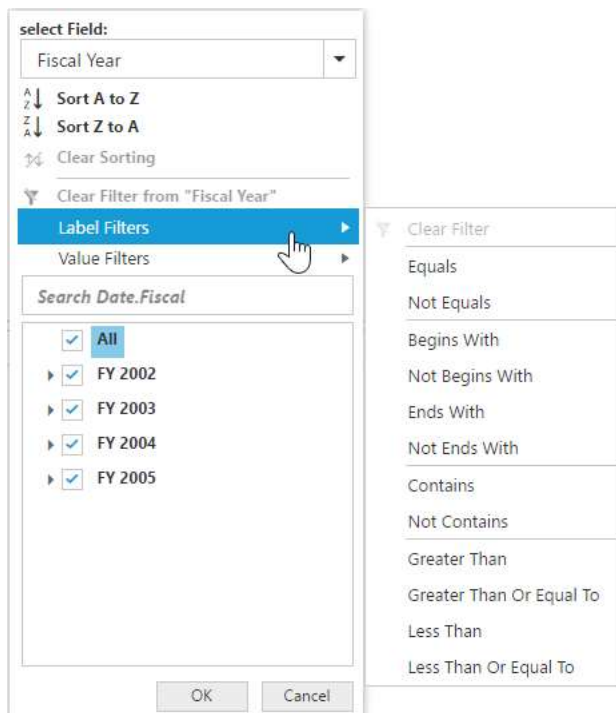
*Sorting*

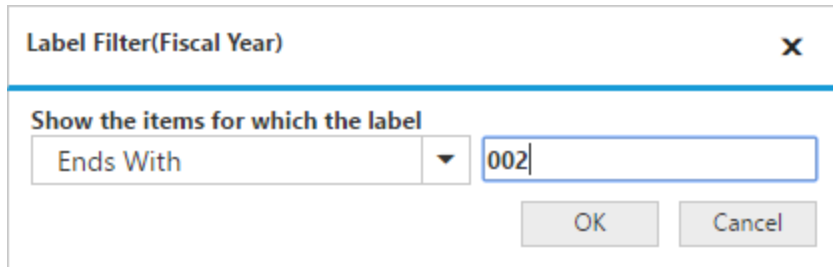
Sorting provides an option to sort the members of a field either in ascending or descending order.



### Label Filtering

Label filtering provides an option to filter the members of a field purely based on their caption.





**Label Filter(Fiscal Year)** [X]

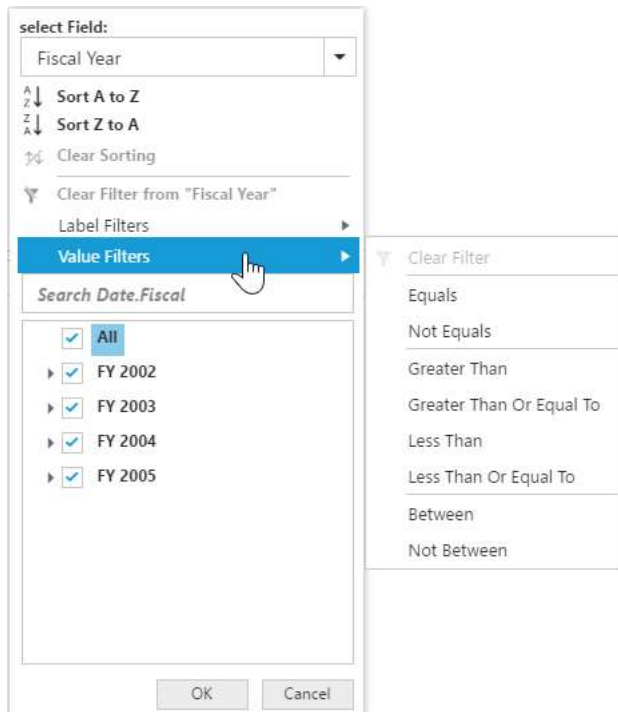
Show the items for which the label

Ends With [002]

OK Cancel

### Value Filtering

Value filtering provides an option to filter members based on the total values of the appropriate measure between the members of the level.



select Field: Fiscal Year

Sort A to Z  
Sort Z to A  
Clear Sorting

Clear Filter from "Fiscal Year"

Label Filters

**Value Filters**

Search Date.Fiscal

☒ All

☒ FY 2002

☒ FY 2003

☒ FY 2004

☒ FY 2005

Clear Filter

Equals

Not Equals

Greater Than

Greater Than Or Equal To

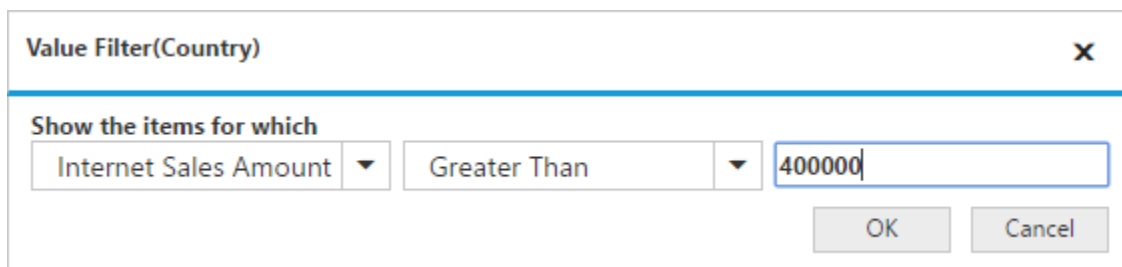
Less Than

Less Than Or Equal To

Between

Not Between

OK Cancel



**Value Filter(Country)** [X]

Show the items for which

Internet Sales Amount Greater Than 400000

OK Cancel

### Exporting

The PivotGrid control can be exported to the following file formats.

- Excel
- Word
- PDF
- CSV

The PivotGrid control can be exported by invoking “**exportPivotGrid**” method, with an appropriate export option as parameter.

#### JSON Export

##### HTML

```
<div id="PivotGrid1" style="height: 350px; width: 100%; overflow:
auto"></div>
<button id="btnExport">Export</button>
<script type="text/babel">
//...
$(function() {
ReactDOM.render(
<EJ.PivotGrid id="PivotGrid" dataSource= {pivotdataSource}></EJ.PivotGrid>,
document.getElementById('PivotGrid1')
);
});
</script>
<script type="text/javascript">
$( "#btnExport" ).click(function() {
var pGridObj = $('#PivotGrid').data("ejPivotGrid");
pGridObj.exportPivotGrid("http://js.syncfusion.com/ejservices/api/PivotGrid/
Olap/ExcelExport", "fileName");
});
</script>
```

#### Excel Export

User can export the contents of PivotGrid to an Excel document for future archival, references and analysis purposes.

To achieve Excel export, service URL and file name is sent as the parameter.

##### JAVASCRIPT

```
$( "#btnExport" ).click(function() {
var pGridObj = $('#PivotGrid').data("ejPivotGrid");
pGridObj.exportPivotGrid("http://js.syncfusion.com/ejservices/api/PivotGrid/
Olap/ExcelExport", "fileName");
});
```

#### Word Export

User can export the contents of PivotGrid to a Word document for future archival, references and analysis purposes.

To achieve Word export, service URL and file name is sent as the parameter.

##### JAVASCRIPT

```
$( "#btnExport" ).click(function() {
var pGridObj = $('#PivotGrid').data("ejPivotGrid");
pGridObj.exportPivotGrid("http://js.syncfusion.com/ejservices/api/PivotGrid/
Olap/WordExport", "fileName");
});
```



### PDF Export

User can export the contents of PivotGrid to a PDF document for future archival, references and analysis purposes.

To achieve PDF export, service URL and file name is sent as the parameter.

#### JAVASCRIPT

```
$( "#btnExport" ).click(function() {
var pGridObj = $('#PivotGrid').data("ejPivotGrid");
pGridObj.exportPivotGrid("http://js.syncfusion.com/ejservices/api/PivotGrid/Olap/PDFExport", "fileName");
});
```

### CSV Export

User can export the contents of PivotGrid to a CSV document for future archival, references and analysis purposes.

To achieve CSV export, service URL and file name is sent as the parameter.

#### JAVASCRIPT

```
$( "#btnExport" ).click(function() {
var pGridObj = $('#PivotGrid').data("ejPivotGrid");
pGridObj.exportPivotGrid("http://js.syncfusion.com/ejservices/api/PivotGrid/Olap/CSVExport", "fileName");
});
```

### Customize the export document name

For customizing file name, we need to send file name as parameter to the **exportPivotGrid** method along with service URL.

#### JAVASCRIPT

```
$( "#btnExport" ).click(function() {
var pGridObj = $('#PivotGrid').data("ejPivotGrid");
pGridObj.exportPivotGrid("http://js.syncfusion.com/ejservices/api/PivotGrid/Olap/ExcelExport", "fileName");
});
```

### Exporting Customization

You can add title and description to the exporting document by using title and description property obtained in the "beforeExport" event.

#### HTML

```
<div id="PivotGrid1" style="height: 350px; width: 100%; overflow:
auto"></div>
<button id="btnExport">Export</button>
<script type="text/babel">
//...
$(function() {
ReactDOM.render(
<EJ.PivotGrid id="PivotGrid" dataSource= {pivotdataSource}
beforeExport="Exporting"></EJ.PivotGrid>,
document.getElementById('PivotGrid1')
```

```

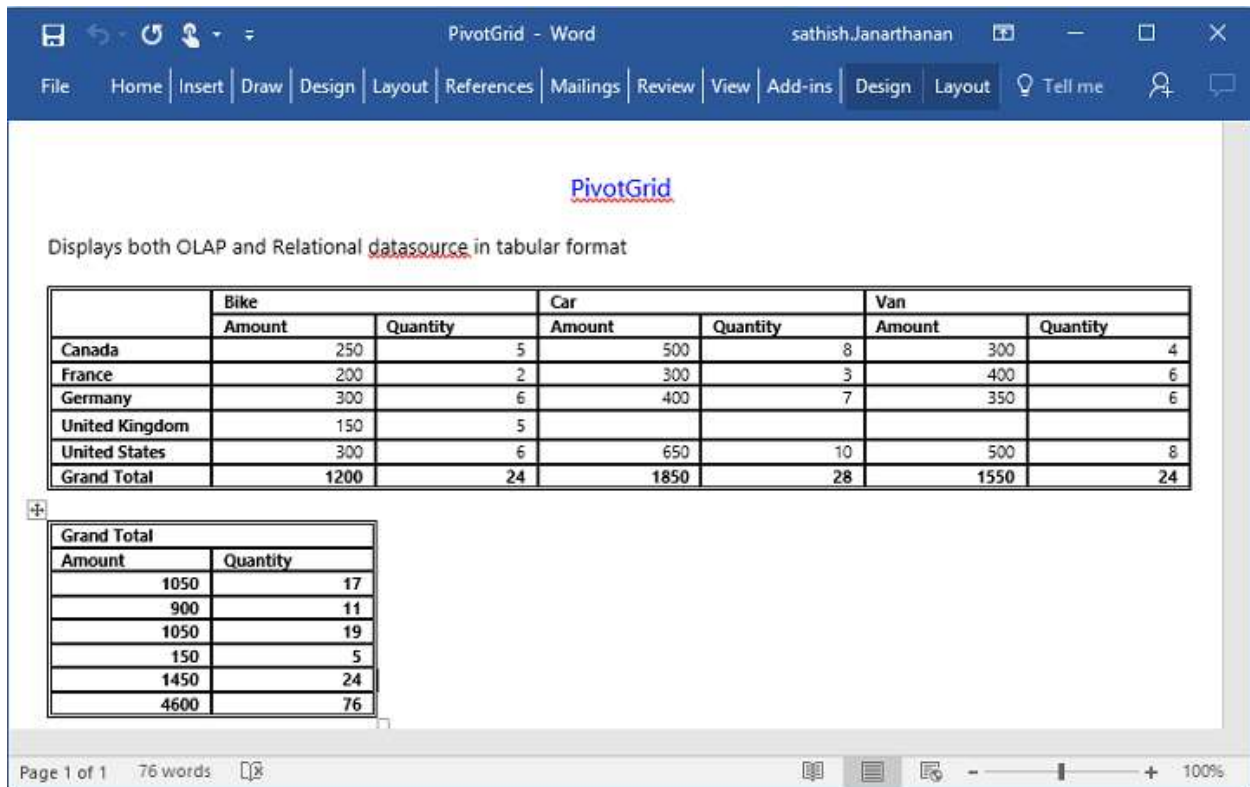
);
});
</script>
<script type="text/javascript">
$( "#btnExport" ).click(function() {
var pGridObj = $('#PivotGrid').data("ejPivotGrid");
pGridObj.exportPivotGrid("http://js.syncfusion.com/ejservices/api/PivotGrid/Olap/ExcelExport", "fileName");
});
function Exporting(args){
args.title = "PivotGrid";
args.description = "Displays both OLAP and Relational datasource in tabular format";
}
</script>

```

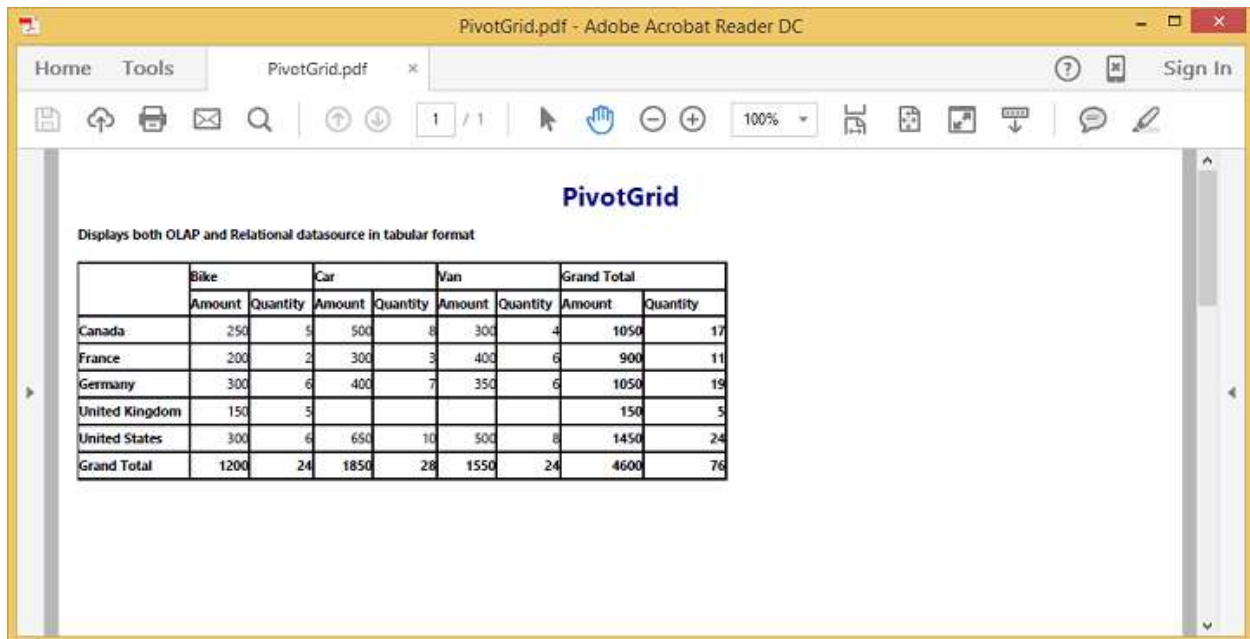
The below screenshot shows the PivotGrid control exported to Excel document.

	Bike		Car		Van		Grand Total	
	Amount	Quantity	Amount	Quantity	Amount	Quantity	Amount	Quantity
Canada	250	5	500	8	300	4	1050	17
France	200	2	300	3	400	6	900	11
Germany	300	6	400	7	350	6	1050	19
United Kingdom	150	5					150	5
United States	300	6	650	10	500	8	1450	24
Grand Total	1200	24	1850	28	1550	24	4600	76

The below screenshot shows the PivotGrid control exported to Word document.



The below screenshot shows the PivotGrid control exported to PDF document.



The below screenshot shows the PivotGrid control exported to CSV document.

	A	B	C	D	E	F	G	H	I	J
1					PivotGrid					
2										
3	Displays both OLAP and Relational datasource in tabular format									
4										
5		Bike	Bike	Car	Car	Van	Van	Grand Tot	Grand Total	
6		Amount	Quantity	Amount	Quantity	Amount	Quantity	Amount	Quantity	
7	Canada	250	5	500	8	300	4	1050	17	
8	France	200	2	300	3	400	6	900	11	
9	Germany	300	6	400	7	350	6	1050	19	
10	United Kir	150	5					150	5	
11	United Sta	300	6	650	10	500	8	1450	24	
12	Grand Tot	1200	24	1850	28	1550	24	4600	76	
13										
14										
15										

### Frozen Header

Allows you to freeze the header of the Grid so that it will be always visible when scrolling the content with a large number of rows or columns providing a precise view.

### HTML

```
<div id="PivotGrid1"></div>
<script type="text/babel">
//...
var frozenHeaderSettings= {enableFrozenHeaders : true} //To Freeze the both
Row and Column headers
$(function() {
ReactDOM.render(
<EJ.PivotGrid id="PivotGrid"
frozenHeaderSettings={frozenHeaderSettings}></EJ.PivotGrid>,
document.getElementById('PivotGrid1')
);
});
</script>
```

			► FY 2003	► FY 2004	► FY 2005	Total
		es Amount				
► Canada	► Bikes	00.97	\$305,010.69	\$945,190.73		\$
	► Clothing			\$49,582.46	\$3,582.16	\$
	Total	00.97	\$305,010.69	\$1,088,879.50	\$10,853.70	\$
► France	► Accessories			\$60,878.53	\$2,528.25	\$
	► Bikes	45.32	\$633,399.70	\$1,505,930.70		\$
	► Clothing			\$26,071.52	\$963.70	\$
	Total	45.32	\$633,399.70	\$1,592,880.75	\$3,491.95	\$

We can also freeze the row/column headers individually by setting the below properties.

#### HTML

```
<script type="text/babel">
//...
var frozenHeaderSettings= {enableFrozenRowHeaders : true} //To Freeze the
Row headers only
$(function() {
ReactDOM.render(
<EJ.PivotGrid id="PivotGrid"
frozenHeaderSettings={frozenHeaderSettings}></EJ.PivotGrid>,
document.getElementById('PivotGrid1')
);
});
</script>
```

		2002	► FY 2003	► FY 2004	► FY 2005	^
		net Sales Amount				
► Australia	► Accessories			\$132,860.61	\$5,830.02	
	► Bikes	\$2,568,701.39	\$2,099,585.43	\$4,183,763.19		
	► Clothing			\$66,855.74	\$3,404.21	
	Total	\$2,568,701.39	\$2,099,585.43	\$4,383,479.54	\$9,234.23	
► Canada	► Accessories			\$96,106.31	\$7,271.54	
	► Bikes	\$573,100.97	\$305,010.69	\$943,190.73		
	► Clothing			\$49,582.46	\$3,582.16	▼
						>

**HTML**

```

<script type="text/babel">
  //...
  var frozenHeaderSettings= {enableFrozenColumnHeaders : true}  //To Freeze
  the Column headers only
  $(function() {
    ReactDOM.render(
      <EJ.PivotGrid id="PivotGrid"
        frozenHeaderSettings={frozenHeaderSettings}></EJ.PivotGrid>,
      document.getElementById('PivotGrid1')
    );
  });
</script>

```

		► FY 2002	► FY 2003	► FY 2004	► FY 2005	
		Internet Sales Amount				
	Total	\$2,568,701.39	\$2,099,585.43	\$4,383,479.54		^
	► Accessories			\$96,106.31		
	► Bikes	\$573,100.97	\$305,010.69	\$943,190.73		
	► Clothing			\$49,582.46		
	Total	\$573,100.97	\$305,010.69	\$1,088,879.50		
	► Accessories			\$60,878.53		
	► Bikes	\$414,245.32	\$633,399.70	\$1,505,930.70		v
						< >

### Drill Through

Drill-through retrieves the raw items that are used to create a specific cell. To enable drill-through support, set [enableDrillThrough](#) property to true. Raw items are obtained through the [drillThrough](#) event, using which user can bind them to an external widget for precise view.

### OLAP

**Note:** Drill-through is supported in PivotGrid only when we configure and enable drill-through action at the Cube.

	Australia	Canada	France	Germany	United Kingdom	United States	Total
	Sales Amount						
► 2005	9061000.58440184	1977844.86209998	2644017.71430033	2894312.33820041	3391712.21090071	9389789.51080357	29358677.2207068
► 2006	9061000.58440184	1977844.86209998	2644017.71430033	2894312.33820041	3391712.21090071	9389789.51080357	29358677.2207068
► 2007	9061000.58440184	1977844.86209998	Value: 2644017.71430033	4312.33820041	3391712.21090071	9389789.51080357	29358677.2207068
► 2008	9061000.58440184	1977844.86209998	Row: 2006	4312.33820041	3391712.21090071	9389789.51080357	29358677.2207068
► 2009	9061000.58440184	1977844.86209998	Column: France-Sales Amount	4312.33820041	3391712.21090071	9389789.51080357	29358677.2207068
► 2010	9061000.58440184	1977844.86209998	2644017.71430033	2894312.33820041	3391712.21090071	9389789.51080357	29358677.2207068
► 2011	9061000.58440184	1977844.86209998	2644017.71430033	2894312.33820041	3391712.21090071	9389789.51080357	29358677.2207068
Total	9061000.58440184	1977844.86209998	2644017.71430033	2894312.33820041	3391712.21090071	9389789.51080357	29358677.2207068

On clicking any value cell, the "Drill Through Information" dialog will be opened. It consists of a Grid with data which are associated with the measure values of clicked value cell. In this example, the measure behind the respective cell is "Sales Amount" and the values of the dimensions which are associated with this measure are alone displayed in the Grid.

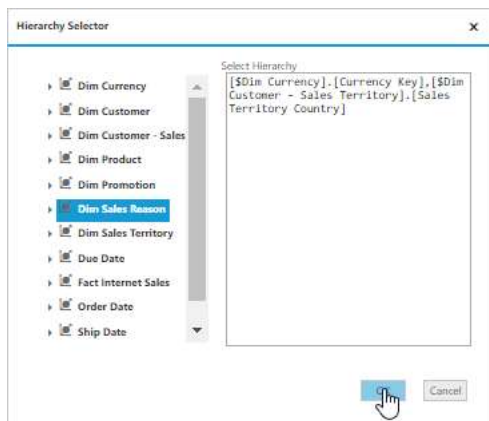
Drill Through Information

[Internet Sales]-[Revision Number]	[Internet Sales]-[Order Quantity]	[Internet Sales]-[Unit Price]	[Internet Sales]-[Extended Amount]	[Internet Sales]-[Unit Price Discount Pct]	[Internet Sales]-[Discount Amount]	[Internet Sales]-[Product Standard Cost]	[Internet Sales]-[Total Product Cost]	[Internet Sales]-[Sales Amount]	[Internet Sales]-[Tax Amt]	[Internet Sales]-[Freight]	[Internet Sales]-[Fact Internet Sales Count]
1	1	3578.27	3578.27	0	0	2171.2942	2171.2942	3578.27	286.2616	89.4568	1
1	1	3578.27	3578.27	0	0	2171.2942	2171.2942	3578.27	286.2616	89.4568	1
1	1	3578.27	3578.27	0	0	2171.2942	2171.2942	3578.27	286.2616	89.4568	1
1	1	3578.27	3578.27	0	0	2171.2942	2171.2942	3578.27	286.2616	89.4568	1
1	1	3578.27	3578.27	0	0	2171.2942	2171.2942	3578.27	286.2616	89.4568	1
1	1	3374.99	3374.99	0	0	1898.0944	1898.0944	3374.99	249.9992	84.3748	1
1	1	699.0982	699.0982	0	0	413.1463	413.1463	699.0982	55.9279	17.4775	1
1	1	3578.27	3578.27	0	0	2171.2942	2171.2942	3578.27	286.2616	89.4568	1

1 of 953 pages (7620 items)

Hierarchy Selector

On clicking the "Hierarchy Selector" button which is displayed below the Grid, the "Hierarchy Selector" dialog will be opened. It consists of the dimensions which are associated with the measure of the clicked value cell. In this example, the measure behind the respective cell is "Sales Amount" and the dimensions associated with this measure are alone displayed in the dialog.



By dragging and dropping the respective hierarchies and finally clicking "OK" button, drill through MDX query will be framed and executed internally and provides back the raw items through "drillThrough" event. In this example, we have bound the raw items obtained to our ejGrid widget. Please refer the code sample and screen-shot below.

### HTML

```

<!--Create a tag which acts as a container for PivotGrid-->
<div id="PivotGrid1"></div>
<script type="text/babel">
//...
$(function() {
  ReactDOM.render(
    <EJ.PivotGrid id="Olap" enableDrillThrough= {true} drillThrough=
    "drilledData"></EJ.PivotGrid>,
    document.getElementById('PivotGrid1')
  );
});
</script>
<script type="text/javascript">
function drilledData(args) {
  $(".e-dialog, .clientDialog, .tableDlg").remove();
  gridData = JSON.parse(args.data);
}

```



```

var dialogContent = ej.buildTag("div#" + this._id + "_tableDlg.tableDlg",
$(("<div id=\"Grid1\"></div>"))[0].outerHTML;
var dialogFooter = ej.buildTag("div",
ej.buildTag("button#btnOK.dialogBtnOK", "Hierarchy Selector")[0].outerHTML,
{ "float": "right", "margin": "-5px 0 6px 0" })[0].outerHTML;
ejDialog = ej.buildTag("div#clientDialog.clientDialog", dialogContent +
dialogFooter, { "opacity": "1" }).attr("title", "Drill Through
Information")[0].outerHTML;
$(ejDialog).appendTo("#" + this._id);
$("#btnOK").ejButton().css({ margin: "30px 0 20px 0" });
$("#Grid1").ejGrid({
dataSource: gridData,
allowPaging: true,
allowTextWrap: true,
pageSettings: { pageSize: 8 }
});
this.element.find(".clientDialog").ejDialog({ width: "70%", content: "#" +
this._id, enableResize: false, close: ej.proxy(ej.Pivot.closePreventPanel,
this) });
var pivotGrid = $("#" + this._id).data("ejPivotGrid");
$("#btnOK").click(function () {
ej.Pivot.createHierarchySelector(pivotGrid);
});
}
</script>

```

Drill Through Information

[Internet Sales]-[Revision Number]	[Internet Sales]-[Order Quantity]	[Internet Sales]-[Unit Price]	[Internet Sales]-[Extended Amount]	[Internet Sales]-[Unit Price Discount Pct]	[Internet Sales]-[Discount Amount]	[Internet Sales]-[Product Standard Cost]	[Internet Sales]-[Total Product Cost]	[Internet Sales]-[Sales Amount]	[Internet Sales]-[Tax Amt]	[Internet Sales]-[Freight]	[Internet Sales]-[Fact Internet Sales Count]	[SDim Currency]-[Currency Key]	[SDim Customer]-[Customer Key]
1	1	3399.99	3399.99	0	0	1912.1544	1912.1544	3399.99	271.9992	84.9998	1	39	28389
1	1	3578.27	3578.27	0	0	2171.2942	2171.2942	3578.27	286.2616	89.4568	1	39	11606
1	1	3578.27	3578.27	0	0	2171.2942	2171.2942	3578.27	286.2616	89.4568	1	39	11591
1	1	3578.27	3578.27	0	0	2171.2942	2171.2942	3578.27	286.2616	89.4568	1	39	11592
1	1	3578.27	3578.27	0	0	2171.2942	2171.2942	3578.27	286.2616	89.4568	1	39	11599
1	1	3578.27	3578.27	0	0	2171.2942	2171.2942	3578.27	286.2616	89.4568	1	39	11601
1	1	3578.27	3578.27	0	0	2171.2942	2171.2942	3578.27	286.2616	89.4568	1	39	11607
1	1	699.0982	699.0982	0	0	413.1463	413.1463	699.0982	55.9279	17.4775	1	39	17956

1 of 695 pages (5558 items)

Hierarchy Selector

### Relational

To enable drill-through support, set [enableDrillThrough](#) property to true. Raw items are obtained through the [drillThrough](#) event.

### HTML

```

<div id="PivotGrid1" style="width:99%;"></div>
<script type="text/babel">
//...
$(function() {
ReactDOM.render(
<EJ.PivotGrid id="Relational" enableDrillThrough= {true} drillThrough=
"drillData"></EJ.PivotGrid>,
document.getElementById('PivotGrid1')
);
});
</script>

```

```

<script type="text/javascript">
function drillData(args) {
gridData = args.selectedData;
var dialogContent = ej.buildTag("div#Grid1", {height:"50px"})[0].outerHTML;
ejDialog = ej.buildTag("div#clientDialog.clientDialog", dialogContent, {
"opacity": "1" }).attr("title", "Drill Through Information")[0].outerHTML;
$(ejDialog).appendTo("#" + this._id);
this.element.find(".clientDialog").ejDialog({ width: "70%", height: "100%",
content: "#" + this._id, enableResize: false, close:
ej.proxy(ej.Pivot.closePreventPanel, this) });
$("#Grid1").ejGrid({
dataSource: gridData,
});
}
</script>

```

Drill Through Information					
Product	Date	Country	State	Quantity	Amount
Bike	FY 2005	Canada	British Columbia	11	293700
Bike	FY 2005	Canada	British Columbia	3	87300
Bike	FY 2005	Canada	British Columbia	7	195300
Bike	FY 2005	Canada	British Columbia	10	270000
Bike	FY 2005	Canada	British Columbia	1	29700
Bike	FY 2005	Canada	British Columbia	6	169200

### Column Resizing

Allows you to resize the column by changing its width while holding and dragging the column border using the mouse.

You can enable column resizing option in PivotGrid by setting the [enableColumnResizing](#) property to true.

### HTML

```

<script type="text/babel">
//...
$(function() {
ReactDOM.render(
<EJ.PivotGrid id="PivotGrid" enableColumnResizing= {true}></EJ.PivotGrid>,
document.getElementById('PivotGrid1')
);
});
</script>

```

	► Australia	► Canada	► France		► Germany
	Internet Sales Amount ◀ ▶				
► FY 2002	\$2,568,701.39	\$573,100.97	\$414,245.32		\$513,353.17
► FY 2003	\$2,099,585.43	\$305,010.69	\$633,399.70		\$593,247.24
► FY 2004	\$4,383,479.54	\$1,088,879.50	\$1,592,880.75		\$1,784,107.09
► FY 2005	\$9,234.23	\$10,853.70	\$3,491.95		\$3,604.83
Total	\$9,061,000.58	\$1,977,844.86	\$2,644,017.71		\$2,894,312.34

### Styling

By default, PivotGrid supports **17** different themes.

- default-theme
- flat-azure-dark
- fat-lime
- flat-lime-dark
- flat-saffron
- flat-saffron-dark
- gradient-azure
- gradient-azure-dark
- gradient-lime
- gradient-lime-dark
- gradient-saffron
- gradient-saffron-dark
- bootstrap-theme
- high-contrast-01
- high-contrast-02
- material
- office-365

You can also customize the appearance of the following component manually.

- PivotGrid
- Grouping Bar
- PivotTable Field List

To change the appearance of PivotGrid, apply the below customized CSS.

### CSS

```
.e-pivotgrid table {
color: #565656;
background-color: White;
}
.e-pivotgrid th,
.e-pivotgrid td {
border: solid 1px;
```

```

border-color: #c4c4c4;
}
.e-pivotgrid .value {
background-color: White;
}
.e-pivotgrid .summary {
background-color: aqua !important;
color: #565656;
}
.e-pivotgrid .colheader, .e-pivotgrid .rowheader {
color: #5c5c5c;
background: white;
background-repeat: repeat;
}
.e-pivotgrid .colheader:hover, .e-pivotgrid .rowheader:hover {
color: white;
background: #91aa29;
background-repeat: repeat;
}

```

	► Australia	► Canada	► France	► Germany	► United Kingdom	► United States	Total
Internet Sales Amount							
► FY 2002	\$2,568,701.39	\$573,100.97	\$414,245.32	\$513,353.17	\$550,507.33	\$2,452,176.07	\$7,072,084.24
► FY 2003	\$2,099,585.43	\$305,010.69	\$633,399.70	\$593,247.24	\$696,594.97	\$1,434,296.26	\$5,762,134.30
► FY 2004	\$4,383,479.54	\$1,088,879.50	\$1,592,880.75	\$1,784,107.09	\$2,140,388.50	\$5,483,882.67	\$16,473,618.05
► FY 2005	\$9,234.23	\$10,853.70	\$3,491.95	\$3,604.83	\$4,221.41	\$19,434.51	\$50,840.63
Total	\$9,061,000.58	\$1,977,844.86	\$2,644,017.71	\$2,894,312.34	\$3,391,712.21	\$9,389,789.51	\$29,358,677.22

To change the appearance of Grouping Bar, apply the below customized CSS.

### CSS

```

.e-pivotgrid .summary {
background-color: aqua !important;
color: #565656;
}
.e-pivotgrid .grpRow, .e-pivotgrid .values, .e-pivotgrid .columns, .e-
pivotgrid .drag {
background: green;
}
.e-pivotgrid .pivotButton .e-btn.e-select, .e-pivotgrid .rows .pivotButton,
.e-pivotgrid .values .pivotButton, .e-pivotgrid .columns .pivotButton, .e-
pivotgrid .drag .pivotButton {
background: blanchedalmond;
}
.e-pivotgrid .colheader, .e-pivotgrid .rowheader {
color: #5c5c5c;
background: white;
background-repeat: repeat;
}
.e-pivotgrid .colheader:hover, .e-pivotgrid .rowheader:hover {

```

```
color: white;
background: #91aa29;
background-repeat: repeat;
}
```

Drag field here						
Amount X	Country ▼ ^ X					
	Canada	France	Germany	United Kingdom	United States	Grand Total
Date ▼ ^ X	Amount	Amount	Amount	Amount	Amount	Amount
FY 2005	\$41,108,400.00	\$42,427,200.00	\$38,198,700.00	\$37,514,100.00	\$39,838,500.00	\$199,086,900.00
FY 2006	\$17,652,600.00	\$11,579,700.00	\$13,868,400.00	\$12,230,700.00	\$9,963,000.00	\$65,294,400.00
FY 2007	\$8,056,200.00	\$6,064,500.00	\$6,915,600.00	\$8,112,000.00	\$8,535,000.00	\$37,683,300.00
FY 2008	\$3,929,100.00	\$3,324,900.00	\$5,891,100.00	\$3,035,700.00	\$4,911,900.00	\$21,092,700.00
FY 2009	\$1,123,200.00	\$2,352,900.00	\$1,810,200.00	\$2,290,200.00	\$1,907,400.00	\$9,483,900.00
Grand Total	\$71,869,500.00	\$65,749,200.00	\$66,684,000.00	\$63,182,700.00	\$65,155,800.00	\$332,641,200.00

To change the appearance of PivotTable Field List, apply the below customized CSS.

### CSS

```
.e-pivotschemadesigner, .e-pivotschemadesigner .fieldTable {
background-color: white;
color: black;
}
.e-pivotschemadesigner .e-treeview .e-active {
background: white;
}
.e-pivotschemadesigner .e-chkbox-wrap .e-chk-image.e-stop, .e-chkbox-wrap
.e-chk-image.e-checkmark {
color:green
}
.e-pivotschemadesigner .subheadText, .e-pivotschemadesigner .centerHead, .e-
pivotschemadesigner .schemaFieldTree.e-treeview .e-text{
color: black;
}
.e-pivotschemadesigner .pivotButton .pvtBtn {
background: green;
color: white;
}
```

### PivotTable Field List

Choose fields to add to report:

☒ Product
 ☒ Date
 ☒ Country
 ☒ State
 ☐ Quantity
 ☒ Amount

Drag fields between areas below:

<div>Report Filter</div>	<div>Column Label</div> <div>Product</div> <div>Country</div>
<div>Row Label</div> <div>Date</div> <div>State</div>	<div>Values</div> <div>Amount</div>

☐ Defer Layout Update
 Update

You can also customize the appearance of the Pivot component using our Theme Studio utility. To know more about Theme Studio [click here](#).

## Localization and Globalization

### Localization in PivotGrid

You can localize the PivotGrid controls text with a collection of localized strings using [ej.PivotGrid.Locale](#) for different cultures. By default, the PivotGrid control is localized in “**en-US**” culture.

## HTML

```

<!--Create a tag which acts as a container for PivotGrid-->
<div id="PivotGrid1" style="width: 55%; height: 670px; overflow: scroll;
float: left;"></div>
<!--Create a tag which acts as a container for PivotTable Field List-->

```

```

<div id="PivotSchemaDesigner1" style="height:650px;width:40%;"></div>
<script type="text/babel">
//...
$(function() {
ReactDOM.render(
<EJ.PivotSchemaDesigner id="PivotSchemaDesigner" locale = "fr-FR"
></EJ.PivotSchemaDesigner>,
document.getElementById('PivotSchemaDesigner1')
);
ReactDOM.render(
<EJ.PivotGrid id="PivotGrid" locale = "fr-FR" ></EJ.PivotGrid>,
document.getElementById('PivotGrid1')
);
});
</script>
<script type="text/javascript">
ej.PivotSchemaDesigner.Locales["fr-FR"] = {
ClearFilter: "Effacer le filtre",
SelectField:"s lectionnez Champ",
Measures: "Mesures",
Warning: "Avertissement",
AlertMsg:"Le champ que vous d placez ne peut pas  tre plac  dans cette zone
du rapport",
Goal: "Goal",
Status:"Status",
Trend:"Trend",
...
...
};
ej.PivotGrid.Locales["fr-FR"] = {
Sort:"Tri",
SelectField: "s lectionnez Champ",
LabelFilterLabel:"Afficher les  l ments pour lesquels l' tiquette",
ValueFilterLabel:"Afficher les  l ments pour lesquels",
...
...
};
</script>

```

The following table lists the default keywords in French culture for PivotGrid.

Keyword	Values
Sort	Tri
SelectField	s�lectionnez Champ
LabelFilterLabel	Afficher les �l�ments pour lesquels l'�tiquette
ValueFilterLabel	Afficher les �l�ments pour lesquels
LabelFilters	Filtres d'�tiquetage
BeginsWith	Commence par
NotBeginsWith	Non Commence pa

EndsWith	se termine par
NotEndsWith	Non termine avec
Contains	Contient
NotContains	Ne contient pas
ValueFilters	Filtres de valeur
ClearFilter	Clear Filter
Equals	Equals
NotEquals	Not Equals
GreaterThan	Supérieur
GreaterThanOrEqualTo	supérieur ou égal
LessThan	Less Than
LessThanOrEqualTo	Moins ou égal
Between	Entre
NotBetween	Non Entre
AddToFilter	Ajouter un filtre
AddToRow	Ajouter à la rangée
AddToColumn	Ajouter à la colonne
AddToValues	Ajouter aux valeurs
Warning	Avertissement
Error	Error
GroupingBarAlertMsg	Le champ que vous déplacez ne peut pas être placé dans cette zone du rapport
Measures	Mesures
Expand	Développer
Collapse	Réduire
ToolTipRow	Row
ToolTipColumn	Colonne
ToolTipValue	Value
NoValue	Pas de valeur
SeriesPage	Series Page



CategoricalPage	Catégorique page
DragFieldHere	champ de glisser ic
ColumnArea	Drop colonne ici
RowArea	Drop ligne ic
ValueArea	Lâche valeurs ici
Close	Fermer
OK	OK
Cancel	Annuler
Remove	Supprimer
Goal	Goal
Status	Status
Trend	Trend
Value	value
ConditionalFormattingErrorMsg	La valeur donnée ne correspond pas
ConditionalFormattingConformMsg	Etes-vous sûr de vouloir supprimer le format sélectionné?
EnterOperand1	Entrez Opérande1
EnterOperand2	Entrez Opérande2
ConditionalFormatting	Mise en forme conditionnelle
Condition	Type conditionnel
Value1	Value1
Value2	Value2
Editcondition	Modifier Condition
AddNew	Ajouter
Format	Format
BackColor	Back Color
Borderrange	Border Range
Borderstyle	Border Style
FontSize	Font Size
Fontstyle	aille de la police

Bordercolor	Couleur Bordure
AliceBlue	AliceBlue
Black	Noir
Blue	Bleu
Brown	Brown
Gold	Gold
Green	Green
Lime	Lime
Maroon	Bordeaux
Orange	Orange
Pink	Pink
Red	rouge
Violet	Violet
White	Blanc
Yellow	Jaune
Solid	Solid
Dashed	pointillÃ©e
Dotted	Dotted
Double	Double
Groove	Groove
Inset	Encart
Outset	Outset
Ridge	Ridge
None	Aucun
Algerian	AlgÃ©rie
Arial	Arial
BodoniMT	Bodoni MT
BritannicBold	Britannic Bold
Cambria	Cambria

Calibri	Calibri
CourierNew	Courier New
DejaVuSans	DejaVu Sans
Forte	Forte
Gerogia	Gerogia
Impact	Impact
SegoeUI	Segoe UI
Tahoma	Tahoma
TimesNewRoman	Times New Roman
Verdana	Verdana
CubeDimensionBrowser	Navigateur Dimension Cube
SelectHierarchy	Sélectionnez Hiérarchie
CalculatedField	Champ calculé
Name	nom
Add:	Ajouter:
Formula	Formule:
Delete	Supprimer
Fields	Champs
CalculatedFieldNameNotFound	Prénom CalculatedField est introuvable
InsertField	Insérer un champ
EmptyField	S'il vous plaît entrez le nom de champ calculé ou la formule
NotValid	formule donnée est pas valide
NotPresent	champ Valeur utilisé dans toute la formule de champ calculé est pas présent dans le PivotGrid
Confirm	champ calculé avec le même nom existe déjà en raison de vouloir remplacer.?
CalcValue	Champ calculé peut être inséré que dans le champ de la zone de valeur
Total	Total
GrandTotal	GrandTotal

DoesNotBeginsWith	N'a pas commence par
DoesNotEndsWith	Ne se termine par
DoesNotContains	Ne contient
DoesNotEquals	N'est pas Ã©gale
IsGreaterThan	Est supÃ©rieure Ã
IsGreaterThanOrEqualTo	Est supÃ©rieure ou Ã©gale Ã
IsLessThan	Est infÃ©rieure Ã
IsLessThanOrEqualTo	Est infÃ©rieure ou Ã©gale Ã
NumberFormatting	Formatage des chiffres
FrozenHeaders	En-tÃªtes congelÃ©
CellSelection	SÃ©lection de cellules
CellContext	Contexte cellulaire
ColumnResize	Redimensionner la colonne
ExcelLikeLayout	Comme la mise en page d'Excel
FrozenHeader	En-tÃªte congelÃ©e
AdvancedFiltering	Filtrage avancÃ©
Amount	QuantitÃ©
Quantity	Quantity
Measures	Mesures visant
NumberFormats	Les formats de nombre
Exporting	L'exportation
FileName	Nom de fichier
ToolTip	ExtrÃ©mitÃ© de l'outil
RTL	RTL
CollapseByDefault	Par dÃ©faut l'effondrement
EnableDisablePaging	Enalbe / DÃ©sactiver la pagination
PagingOptions	Options d'appel
CategoricalPageSize	Taille de page catÃ©gorique
SeriesPageSize	Taille de page sÃ©rie

HyperLink	Hyper Link
CellEditing	Montage de cellules
GroupingBar	Bar de groupement
SummaryCustomization	Personnalisation Sommaire
SummaryTypes	Types Sommaire
SummaryType	Type de statistique
EnableRowHeaderHyperlink	Activer l'en-tête de ligne HyperLink
EnableColumnHeaderHyperlink	Activer l'en-tête de colonne HyperLink
EnableValueCellHyperlink	Activer la cellule Valeur HyperLink
EnableSummaryCellHyperlink	Activer la cellule de synthèse HyperLink
HideGrandTotal	Masquer Grand Total
HideSubTotal	Masquer Sous-total
Both	Les deux
Sum	Somme
Average	La moyenne
Count	Count
Min	Min
Max	Max
Excel	Excel
Word	Mot
PDF	PDF
CSV	CSV
MultipleItems	Plusieurs Commentaires
All	Tous les
Search	Recherchez

The following table lists the default keywords in French culture for PivotTable Field List.

Keywords	Values
PivotTableFieldList	Liste de champs de tableau croisé dynamique
ChooseFieldsToAddToReport	Choisissez champs à ajouter à Signaler"

DragFieldBetweenAreasBelow	Faites glisser terrain entre les zones ci-dessous
ReportFilter	Rapport Filtre
ColumnLabel	colonne Étiquette
RowLabel	Label Row
Values	Valeurs
ClearFilter	Effacer le filtre
SelectField	sélectionnez Champ
Measures	Mesures
Warning	Avertissement
AlertMsg	Le champ que vous déplacez ne peut pas être placé dans cette zone du rapport
Goal	Goal
Status	Status
Trend	Trend
AddToFilter	Ajouter à filtrer
AddToRow	Ajouter à la rangée
AddToColumn	Ajouter à la colonne
AddToValues	Ajouter à la valeur
DeferLayoutUpdate	Différer Mise à jour
Update	Mettre à jour
OK	OK
Cancel	Annuler
Close	Fermer
DoesNotBeginsWith	N'a pas commence par
DoesNotEndsWith	Ne se termine par
DoesNotContains	Ne contient
DoesNotEquals	N'est pas égaux
IsGreaterThan	Est supérieure à
IsGreaterThanOrEqualTo	Est supérieure ou égale à
IsLessThan	Est inférieure à

IsLessThanOrEqualTo	Est inférieure ou égale à
Sort	Trier
LabelFilterLabel	Afficher les éléments pour lesquels l'étiquette
ValueFilterLabel	Afficher les éléments pour lesquels
LabelFilters	Les filtres de l'étiquette
BeginsWith	Commence par
NotBeginsWith	Commence pas avec
EndsWith	Se termine par
NotEndsWith	Pas une fin avec
Contains	Contient
NotContains	Contient pas
ValueFilters	Les filtres de valeur
ClearFilter	Clear Filter
Equals	Est égal à
NotEquals	Pas égaux
GreaterThan	Plus de
GreaterThanOrEqualTo	Supérieure ou égale à
LessThan	Moins de
LessThanOrEqualTo	Inférieure ou égale à
Between	Entre
NotBetween	Pas entre
Search	Recherchez

The following table lists the default keywords in French culture for Pivot Pager.

Keywords	Values
SeriesPage	SÃ©rie Page
CategoricalPage	CatÃ©gorique Page
Error	Error
OK	OK
Close	Fermer

PageCountErrorMsg	Entrez le numéro de page valide
-------------------	---------------------------------

### Localization and Globalization of Cube Info (Client Mode)

Content displayed within the PivotGrid control are obtained from the OLAP Cube. So following are the steps that needs to be done to get the localized and globalized Cube content.

- To get localized data from OLAP Cube, we need to set “**Locale Identifier**” in the connection string to a specific culture in the “**data**” property present inside “**dataSource**”.
- To bind the globalized content in PivotGrid control, we need to set “**locale**” property to a specific culture and want to refer specific culture file in the sample.

**Note:** Culture files are present under “[installed drive]:\Users\[user name]\AppData\Local\Syncfusion\EssentialStudio\X.X.X\JavaScript\samples\web\scripts\cultures”.

### HTML

```
//1036 refers to "fr-FR" culture.
<script type="text/babel">
var Olap_dataSource={
data: "http://bi.syncfusion.com/olap/msmdpump.dll; Locale Identifier=1036;",
....
};
$(function() {
ReactDOM.render(
<EJ.PivotGrid id="Olap" dataSource= {Olap_dataSource} locale: "fr-FR"></EJ.PivotGrid>,
document.getElementById('PivotGrid1')
);
});
</script>
```

	► Australie	► Canada	► France	► Allemagne	► Royaume-Uni	► États-Unis	Total
	Montant des Ventes (Internet)						
► FY 2002	2 568 701,39 €	573 100,97 €	414 245,32 €	513 353,17 €	550 507,33 €	2 452 176,07 €	7 072 084,24 €
► FY 2003	2 099 585,43 €	305 010,69 €	633 399,70 €	593 247,24 €	696 594,97 €	1 434 296,26 €	5 762 134,30 €
► FY 2004	4 383 479,54 €	1 088 879,50 €	1 592 880,75 €	1 784 107,09 €	2 140 388,50 €	5 483 882,67 €	16 473 618,05 €
► FY 2005	9 234,23 €	10 853,70 €	3 491,95 €	3 604,83 €	4 221,41 €	19 434,51 €	50 840,63 €
Total	9 061 000,58 €	1 977 844,86 €	2 644 017,71 €	2 894 312,34 €	3 391 712,21 €	9 389 789,51 €	29 358 677,22 €

### Localization and Globalization of Relational Info (Client Mode)

Content displayed within the PivotGrid control are obtained from the Relational datasource. So following are the steps that needs to be done to get localized as well as globalized content.

- To get the localized content, the Relational datasource must have localized headers in them which will be directly applied to PivotGrid.



- To globalize the values appear in PivotGrid , we need to set “**format**” and “**locale**” property to a specific culture and want to refer specific culture file in the sample.

**Note:** Culture files are present under “[installed drive]:\Users\ [user name]\AppData\Local\Syncfusion\EssentialStudio\X.X.X.X\JavaScript\samples\web\scripts\cultures”

### HTML

```
<script type="text/babel">
//...
$(function() {
ReactDOM.render(
<EJ.PivotGrid id="PivotGrid" locale = "fr-FR" ></EJ.PivotGrid>,
document.getElementById('PivotGrid1')
);
});
</script>
```

	Bike	Car	Van	Grand Total
	Amount	Amount	Amount	Amount
FY 2005	1.329543E+008	2.907120E+007	3.706140E+007	1.990869E+008
FY 2006	5.239890E+007		1.289550E+007	6.529440E+007
FY 2007	2.920320E+007		8.480100E+006	3.768330E+007
FY 2008	1.529940E+007		5.793300E+006	2.109270E+007
FY 2009	6.059400E+006		3.424500E+006	9.483900E+006
Grand Total	2.359152E+008	2.907120E+007	6.765480E+007	3.326412E+008

### RTL

You can render our PivotGrid control from Right to Left by setting [enableRTL](#) property to true.

### HTML

```
<script type="text/babel">
//...
$(function() {
ReactDOM.render(
<EJ.PivotGrid id="PivotGrid" enableRTL = {true} ></EJ.PivotGrid>,
document.getElementById('PivotGrid1')
);
});
</script>
```

Total	United States ◀	United Kingdom ◀	Germany ◀	France ◀	Canada ◀	Australia ◀	
Internet Sales Amount							
\$7,072,084.24	\$2,452,176.07	\$550,507.33	\$513,353.17	\$414,245.32	\$573,100.97	\$2,568,701.39	FY 2002 ◀
\$5,762,134.30	\$1,434,296.26	\$696,594.97	\$593,247.24	\$633,399.70	\$305,010.69	\$2,099,585.43	FY 2003 ◀
\$16,473,618.05	\$5,483,882.67	\$2,140,388.50	\$1,784,107.09	\$1,592,880.75	\$1,088,879.50	\$4,383,479.54	FY 2004 ◀
\$50,840.63	\$19,434.51	\$4,221.41	\$3,604.83	\$3,491.95	\$10,853.70	\$9,234.23	FY 2005 ◀
\$29,358,677.22	\$9,389,789.51	\$3,391,712.21	\$2,894,312.34	\$2,644,017.71	\$1,977,844.86	\$9,061,000.58	Total

## Responsive

PivotGrid and PivotTable Field list control supports responsive rendering based on the target device (desktop & tablet) resolution. It supports resolution upto 1024x600. You can enable responsiveness in PivotGrid by setting [isResponsive](#) property to true.

On resizing the browser, the PivotTable Field list will get collapse and an icon will appear on the left-hand side of the browser. User can toggle its view and perform UI interaction.

## HTML

```
<script type="text/babel">
//...
$(function() {
  ReactDOM.render(
    <EJ.PivotGrid id="PivotGrid" isResponsive={true}></EJ.PivotGrid>,
    document.getElementById('PivotGrid1')
  );
});
</script>
```

	► Australia	► Canada	► France	► Germany	► United Kingdom	► United States	Total
Internet Sales Amount							
► FY 2002	\$2,568,701.39	\$573,100.97	\$414,245.32	\$513,353.17	\$550,507.33	\$2,452,176.07	\$7,072,084.24
► FY 2003	\$2,099,585.43	\$305,010.69	\$633,399.70	\$593,247.24	\$696,594.97	\$1,434,296.26	\$5,762,134.30
► FY 2004	\$4,383,479.54	\$1,088,879.50	\$1,592,880.75	\$1,784,107.09	\$2,140,388.50	\$5,483,882.67	\$16,473,618.05
► FY 2005	\$9,234.23	\$10,853.70	\$3,491.95	\$3,604.83	\$4,221.41	\$19,434.51	\$50,840.63
Total	\$9,061,000.58	\$1,977,844.86	\$2,644,017.71	\$2,894,312.34	\$3,391,712.21	\$9,389,789.51	\$29,358,677.22

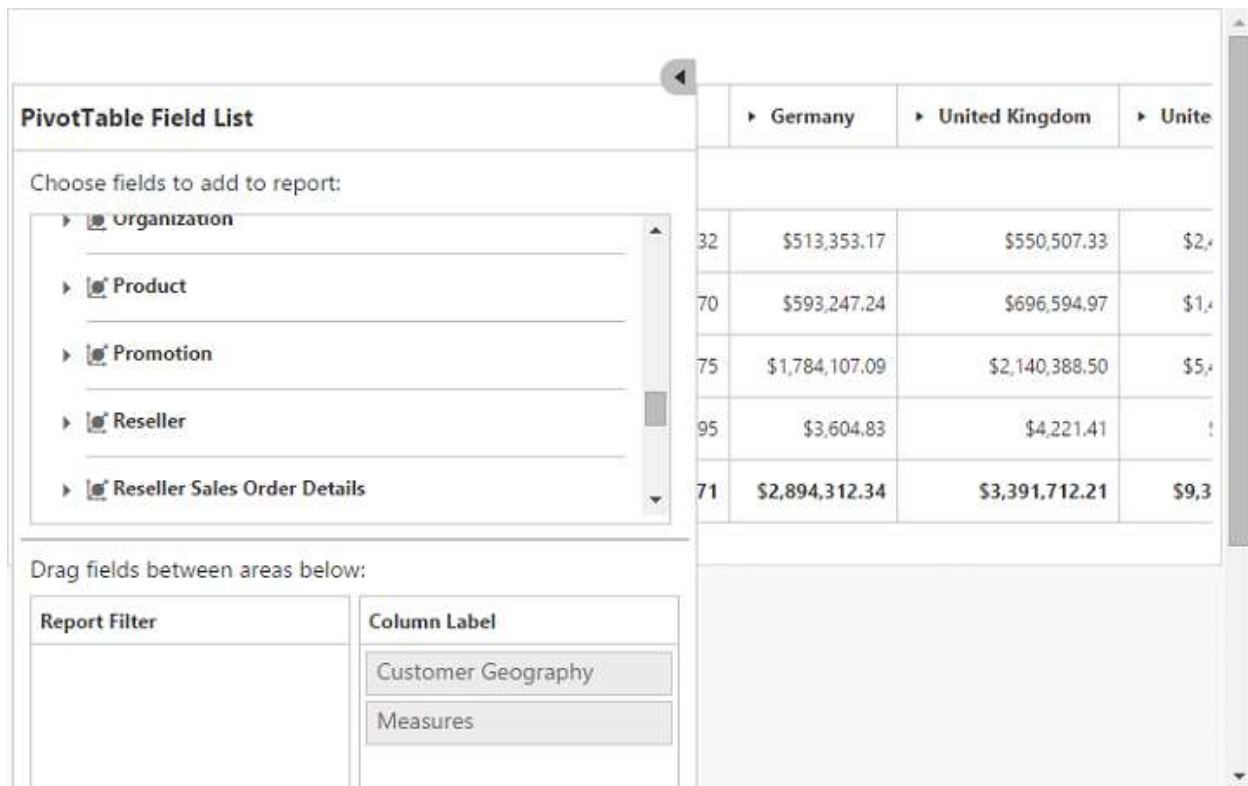
## Normal PivotGrid

	► Australia	► Canada	► France	► Germany	► United Kingdom	► United States	Total
Internet Sales Amount							
► FY 2002	\$2,568,701.39	\$573,100.97	\$414,245.32	\$513,353.17	\$550,507.33	\$2,452,176.07	\$7,072,084.24
► FY 2003	\$2,099,585.43	\$305,010.69	\$633,399.70	\$593,247.24	\$696,594.97	\$1,434,296.26	\$5,762,134.30
► FY 2004	\$4,383,479.54	\$1,088,879.50	\$1,592,880.75	\$1,784,107.09	\$2,140,388.50	\$5,483,882.67	\$16,473,618.05
► FY 2005	\$9,234.23	\$10,853.70	\$3,491.95	\$3,604.83	\$4,221.41	\$19,434.51	\$50,840.63
Total	\$9,061,000.58	\$1,977,844.86	\$2,644,017.71	\$2,894,312.34	\$3,391,712.21	\$9,389,789.51	\$29,358,677.22

*Responsive PivotGrid*

	► Australia	► Canada	► France	► Germany	► United Kingdom	► Unite
Internet Sales Amount						
► FY 2002	\$2,568,701.39	\$573,100.97	\$414,245.32	\$513,353.17	\$550,507.33	\$2,452,176.07
► FY 2003	\$2,099,585.43	\$305,010.69	\$633,399.70	\$593,247.24	\$696,594.97	\$1,434,296.26
► FY 2004	\$4,383,479.54	\$1,088,879.50	\$1,592,880.75	\$1,784,107.09	\$2,140,388.50	\$5,483,882.67
► FY 2005	\$9,234.23	\$10,853.70	\$3,491.95	\$3,604.83	\$4,221.41	\$19,434.51
Total	\$9,061,000.58	\$1,977,844.86	\$2,644,017.71	\$2,894,312.34	\$3,391,712.21	\$9,389,789.51

*Responsive PivotTable Field List - Collapsed*



*Responsive PivotTable Field List - Expanded*

### ToolTip

Allows you to display the details of the cell on hovering value cells. By default, tooltip is enabled. You can disable tooltip in PivotGrid by setting the [enableToolTip](#) property to false.

### HTML

```
<script type="text/babel">
//..
$(function() {
  ReactDOM.render(
    <EJ.PivotGrid id="PivotGrid" enableToolTip={false}></EJ.PivotGrid>,
    document.getElementById('PivotGrid1')
  );
});
</script>
```

### ToolTip Animation

The PivotGrid provides option to animate tooltip displayed in the grid. The animation enhances the appearance of tooltip by displaying it slowly. You can enable animation in tooltip by setting [enableToolTipAnimation](#) property to true.

### HTML

```
<script type="text/babel">
//..
$(function() {
  ReactDOM.render(
    <EJ.PivotGrid id="PivotGrid" enableToolTipAnimation={true}></EJ.PivotGrid>,
    document.getElementById('PivotGrid1')
  );
});
</script>
```

```
document.getElementById('PivotGrid1')
);
});
</script>
```

	Bike	Car	Van	Grand Total
	Quantity	Quantity	Quantity	Quantity
▶ Canada	5	8	4	17
▶ France	2	3	6	11
▶ Germany	6	<div>Value: 3 Row: France Total Column: Car-Quantity</div>	6	19
▶ United Kingdom	5			5
▶ United States	6		8	24
Grand Total	24	28	24	76

## PivotGauge

### Overview

The **PivotGauge** control is a lightweight control that reads both **OLAP** and **Relational** datasource.

### Key Features

Some of the key features of PivotGauge are as follows,

- **Data source:** Supports OLAP data binding with **XML/A** data sources and Relational data sources.
- **Tooltip:** Displays the value and goal information in the tooltip.
- **Multiple gauges and Layouts:** Support to customize the layout while rendering multiple controls.
- **Frame types:** Built-in frame types provide a rich appearance of control.
- **Indicators:** Displays the active/inactive state of PivotGauge.
- **Ranges:** Highlighting the range of values in scale.
- **Pointers:** Points the actual value and goal information.
- **Header labels:** Support to show or hide header labels and indicators.

### Getting Started

This section explains you the steps required to populate the PivotGauge with data source. This section covers only the minimal features that you need to know to get started with the PivotGauge.

## Script and CSS Reference

Create a **HTML** page and add the script and CSS references in the order mentioned in the following code example.

### HTML

```
<!DOCTYPE html>
<html>
<head>
  <!-- Essential Studio for JavaScript theme reference -->
  <link rel="stylesheet"
href="http://cdn.syncfusion.com/14.3.0.49/js/web/bootstrap-
theme/ej.web.all.min.css" />
  <!-- react script -->
  <script
src="https://cdnjs.cloudflare.com/ajax/libs/react/15.2.1/react.js"></script>
  <script src="https://cdnjs.cloudflare.com/ajax/libs/react/15.2.1/react-
dom.js"></script>
  <script src="https://cdnjs.cloudflare.com/ajax/libs/babel-
core/5.8.34/browser.min.js"></script>
  <!-- jquery script -->
  <script src="https://code.jquery.com/jquery-3.0.0.min.js"></script>
  <!-- Essential JS UI widget -->
  <script
src="http://cdn.syncfusion.com/14.3.0.49/js/web/ej.web.all.min.js"></script>
  <script
src="http://cdn.syncfusion.com/14.3.0.49/js/common/ej.web.react.min.js"></sc
ript>
  <!--Add custom scripts here -->
</head>
<body>
</body>
</html>
```

In the above code, `ej.web.all.min.js` script reference has been added for demonstration purpose. It is not recommended to use this for deployment purpose, as its file size is larger since it contains all the widgets. Instead, you can use [CSG](#) utility to generate a custom script file with the required widgets for deployment purpose.

#### Note:

- `react.js` and `react-dom.js` are the core files needed to create react elements.
- `browser.min.js` file is required for code transform.
- `ej.web.react.min.js` is a react-syncfusion bridge to render Syncfusion components.

## Relational

This section covers the information that you need to know to populate a simple PivotGauge with Relational data source.

### Control Initialization

Add a `div` container to render the PivotGauge.

### HTML

```

<!DOCTYPE html>
<html>
<body>
<div id="PivotGauge1" style="width:99%;"></div>
</body>
</html>

```

Initialize the PivotGauge by using the `EJ.PivotGauge` tag.

### HTML

```

<!DOCTYPE html>
<html>
<head>
<style>
#Relational{
width: 95%;
height: 450px;
overflow: auto;
}
</style>
</head>
<body>
<div id="PivotGauge1" style="width:99%;"></div>
<script type="text/babel">
ReactDOM.render(
<EJ.PivotGauge id="Relational"></EJ.PivotGauge>,
document.getElementById('PivotGauge1')
);
</script>
</body>
</html>

```

### *Populate PivotGauge with Data*

Let us now see how to populate the PivotGauge control using a sample JSON data as shown below.

### HTML

```

<script type="text/babel">
var pivot_dataset = [
{ Amount: 100, Country: "Canada", Date: "FY 2005", Product: "Bike",
Quantity: 2, State: "Alberta" },
{ Amount: 200, Country: "Canada", Date: "FY 2006", Product: "Van", Quantity:
3, State: "British Columbia" },
{ Amount: 300, Country: "Canada", Date: "FY 2007", Product: "Car", Quantity:
4, State: "Brunswick" },
{ Amount: 150, Country: "Canada", Date: "FY 2008", Product: "Bike",
Quantity: 3, State: "Manitoba" },
{ Amount: 200, Country: "Canada", Date: "FY 2006", Product: "Car", Quantity:
4, State: "Ontario" },
{ Amount: 100, Country: "Canada", Date: "FY 2007", Product: "Van", Quantity:
1, State: "Quebec" },
{ Amount: 200, Country: "France", Date: "FY 2005", Product: "Bike",
Quantity: 2, State: "Charente-Maritime" },
{ Amount: 250, Country: "France", Date: "FY 2006", Product: "Van", Quantity:
4, State: "Essonne" },

```

```

{ Amount: 300, Country: "France", Date: "FY 2007", Product: "Car", Quantity:
3, State: "Garonne (Haute)" },
{ Amount: 150, Country: "France", Date: "FY 2008", Product: "Van", Quantity:
2, State: "Gers" },
{ Amount: 200, Country: "Germany", Date: "FY 2006", Product: "Van",
Quantity: 3, State: "Bayern" },
{ Amount: 250, Country: "Germany", Date: "FY 2007", Product: "Car",
Quantity: 3, State: "Brandenburg" },
{ Amount: 150, Country: "Germany", Date: "FY 2008", Product: "Car",
Quantity: 4, State: "Hamburg" },
{ Amount: 200, Country: "Germany", Date: "FY 2008", Product: "Bike",
Quantity: 4, State: "Hessen" },
{ Amount: 150, Country: "Germany", Date: "FY 2007", Product: "Van",
Quantity: 3, State: "Nordrhein-Westfalen" },
{ Amount: 100, Country: "Germany", Date: "FY 2005", Product: "Bike",
Quantity: 2, State: "Saarland" },
{ Amount: 150, Country: "United Kingdom", Date: "FY 2008", Product: "Bike",
Quantity: 5, State: "England" },
{ Amount: 250, Country: "United States", Date: "FY 2007", Product: "Car",
Quantity: 4, State: "Alabama" },
{ Amount: 200, Country: "United States", Date: "FY 2005", Product: "Van",
Quantity: 4, State: "California" },
{ Amount: 100, Country: "United States", Date: "FY 2006", Product: "Bike",
Quantity: 2, State: "Colorado" },
{ Amount: 150, Country: "United States", Date: "FY 2008", Product: "Car",
Quantity: 3, State: "New Mexico" },
{ Amount: 200, Country: "United States", Date: "FY 2005", Product: "Bike",
Quantity: 4, State: "New York" },
{ Amount: 250, Country: "United States", Date: "FY 2008", Product: "Car",
Quantity: 3, State: "North Carolina" },
{ Amount: 300, Country: "United States", Date: "FY 2007", Product: "Van",
Quantity: 4, State: "South Carolina" }
];
var pivotdataSource = {
data: pivot_dataset,
cube: "",
rows: [
{ fieldName: "Country", fieldCaption: "Country" },
{ fieldName: "State", fieldCaption: "State" }
],
columns: [
{ fieldName: "Product", fieldCaption: "Product" }
],
values: [
{ fieldName: "Amount", fieldCaption: "Amount" },
{ fieldName: "Quantity", fieldCaption: "Quantity" }
],
filters: []
};
var pivotgauge_labelsettings = {decimalPlaces: 2};
var PivotGauge_customLables = [{
position: {
x: 180,
y: 290
},
font: {
size: "10px",

```



```
fontFamily: "Segoe UI",
fontStyle: "Normal"
}, {
color: "#666666"
}, {
position: {
x: 180,
y: 320
},
font: {
size: "10px",
fontFamily: "Segoe UI",
fontStyle: "Normal"
},
color: "#666666"
}, {
position: {
x: 180,
y: 150
},
font: {
size: "12px",
fontFamily: "Segoe UI",
fontStyle: "Normal"
},
color: "#666666"
}];
var PivotGauge_ranges = [{
distanceFromScale: -5,
backgroundColor: "#fc0606",
border: {
color: "#fc0606"
}
}, {
distanceFromScale: -5
}];
var pivotgauge_labels = [{
color: "#8c8c8c"
}]
var pivotgauge_ticks = [{
type: "major",
distanceFromScale: 2,
height: 16,
width: 1,
color: "#8c8c8c"
}, {
type: "minor",
height: 6,
width: 1,
distanceFromScale: 2,
color: "#8c8c8c"
}];
var pivotgauge_pointers = [{
showBackNeedle: true,
backNeedleLength: 20,
length: 120,
width: 7
```

```

    }];
    var pivotguage_scale = [{
    showRanges: true,
    radius: 150,
    showScaleBar: true,
    size: 1,
    border: {
    width: 0.5
    },
    showIndicators: true,
    showLabels: true,
    pointers: pivotguage_pointers,
    ticks: pivotguage_ticks,
    labels: pivotguage_labels,
    ranges: PivotGauge_ranges,
    customLabels: PivotGauge_customLables
    }];
    $(function() {
    ReactDOM.render(
    <EJ.PivotGauge id="Relational" dataSource= {pivotdataSource}
    labelFormatSettings= {pivotguage_lablesetings} enableTooltip= {true}
    isResponsive= {true} load: {"loadGaugeTheme"} backgroundColor=
    {"transparent"} scales= {pivotguage_scale}></EJ.PivotGauge>,
    document.getElementById('PivotGauge1')
    );
    });
</script>

```

The above code will generate a simple PivotGauge as shown in below figure.



## OLAP

This section covers the information that you need to know to populate a simple PivotGauge with OLAP data source.

### Control Initialization

Add a `div` container to render the PivotGauge.

#### HTML

```
<!DOCTYPE html>
<html>
<body>
<div id="PivotGauge1" style="width:99%;"></div>
</body>
</html>
```

Initialize the PivotGauge by using the `EJ.PivotGauge` tag.

#### HTML

```
<!DOCTYPE html>
<html>
<head>
<style>
#Olap{
width: 95%;
height: 450px;
overflow: auto;
}
</style>
</head>
<body>
<div id="PivotGauge1" style="width:99%;"></div>
<script type="text/babel">
ReactDOM.render(
<EJ.PivotGauge id="Olap"></EJ.PivotGauge>,
document.getElementById('PivotGauge1')
);
</script>
</body>
</html>
```

### Populate PivotGauge with data

Let us now see how to populate the PivotGauge control using a sample JSON data as shown below.

#### HTML

```
<script type="text/babel">
var Olap_dataSource={
data: "http://bi.syncfusion.com/olap/msmdpump.dll",
catalog: "Adventure Works DW 2008 SE",
cube: "Adventure Works",
rows: [
{
fieldName: "[Date].[Fiscal]",
```

```

filterItems: { filterType: "include", values: ["[Date].[Fiscal].[Fiscal
Year].&#38;[2004]"] }
},
],
columns: [
{
fieldName: "[Customer].[Customer Geography]",
filterItems: { filterType: "include", values: ["[Customer].[Customer
Geography].[Country].&#38;[Australia]"] }
},
],
values: [
{
measures: [
{
fieldName: "[Measures].[Internet Sales Amount]"
},
{
fieldName: "[Measures].[Internet Revenue Status]"
},
{
fieldName: "[Measures].[Internet Revenue Trend]"
},
{
fieldName: "[Measures].[Internet Revenue Goal]"
},
],
axis: ej.PivotGauge.AxisName.Columns
}
]
];
var pivotgauge_lablesettings = {decimalPlaces: 2};
var PivotGauge_customLables = [{
position: {
x: 180,
y: 290
},
font: {
size: "10px",
fontFamily: "Segoe UI",
fontStyle: "Normal"
},
color: "#666666"
}, {
position: {
x: 180,
y: 320
},
font: {
size: "10px",
fontFamily: "Segoe UI",
fontStyle: "Normal"
},
color: "#666666"
}, {
position: {
x: 180,

```

```
y: 150
},
font: {
  size: "12px",
  fontFamily: "Segoe UI",
  fontStyle: "Normal"
},
color: "#666666"
}];
var PivotGauge_ranges = [{
  distanceFromScale: -5,
  backgroundColor: "#fc0606",
  border: {
    color: "#fc0606"
  }
}, {
  distanceFromScale: -5
}];
var pivotgauge_labels = [{
  color: "#8c8c8c"
}]
var pivotgauge_ticks = [{
  type: "major",
  distanceFromScale: 2,
  height: 16,
  width: 1,
  color: "#8c8c8c"
}, {
  type: "minor",
  height: 6,
  width: 1,
  distanceFromScale: 2,
  color: "#8c8c8c"
}];
var pivotgauge_pointers = [{
  showBackNeedle: true,
  backNeedleLength: 20,
  length: 120,
  width: 7
}, {
  type: "marker",
  markerType: ej.datavisualization.CircularGauge.MarkerType.Diamond,
  distanceFromScale: 5,
  placement: "center",
  backgroundColor: "#29A4D9",
  length: 25,
  width: 15
}];
var pivotgauge_scale = [{
  showRanges: true,
  radius: 150,
  showScaleBar: true,
  size: 1,
  border: {
    width: 0.5
  },
  showIndicators: true,
```

```

showLabels: true,
pointers: pivotgauge_pointers,
ticks: pivotgauge_ticks,
labels: pivotgauge_labels,
ranges: PivotGauge_ranges,
customLabels: PivotGauge_customLables
}];
$(function() {
ReactDOM.render(
<EJ.PivotGauge id="Olap" dataSource= {Olap_dataSource} labelFormatSettings=
{pivotgauge_lablesettings} enableTooltip= {true} isResponsive= {true} load:
{"loadGaugeTheme"} backgroundColor= {"transparent"} scales=
{pivotgauge_scale}></EJ.PivotGauge>,
document.getElementById('PivotGauge1')
);
});
</script>

```

The above code will generate a simple PivotGauge as shown in below figure.



## PivotTreeMap

### Overview

The **PivotTreeMap** control lets the user to visualize OLAP data in the form of nested nodes in hierarchical order with the ability to drill up and down.

### Key Features

Some of the key features of PivotTreeMap are as follows,

- **Data Source** - Binds the PivotTreeMap control with XML/A data sources.
- **Drill Support** - Enables you to navigate through the inner levels of a hierarchy elements.
- **Color Mapping** - Allows user to differentiate leaf nodes using various color codes either based on their value or members.

- **Legend** - Differentiates the color code based on value range (from minimum to maximum).

## Getting Started

This section explains you the steps required to populate the PivotTreeMap with data source. This section covers only the minimal features that you need to know to get started with the PivotTreeMap.

## Script and CSS Reference

Create a **HTML** page and add the script and CSS references in the order mentioned in the following code example.

### HTML

```
<!DOCTYPE html>
<html>
<head>
<!-- Essential Studio for JavaScript theme reference -->
<link rel="stylesheet"
href="http://cdn.syncfusion.com/14.3.0.49/js/web/bootstrap-
theme/ej.web.all.min.css" />
<!-- react script -->
<script
src="https://cdnjs.cloudflare.com/ajax/libs/react/15.2.1/react.js"></script>
<script src="https://cdnjs.cloudflare.com/ajax/libs/react/15.2.1/react-
dom.js"></script>
<script src="https://cdnjs.cloudflare.com/ajax/libs/babel-
core/5.8.34/browser.min.js"></script>
<!-- jquery script -->
<script src="https://code.jquery.com/jquery-3.0.0.min.js"></script>
<!-- jsrender script -->
<script
src="http://cdn.syncfusion.com/js/assets/external/jsrender.min.js"></script>
<!-- Essential JS UI widget -->
<script
src="http://cdn.syncfusion.com/14.3.0.49/js/web/ej.web.all.min.js"></script>
<script
src="http://cdn.syncfusion.com/14.3.0.49/js/common/ej.web.react.min.js"></sc
ript>
<!--Add custom scripts here -->
</head>
<body>
</body>
</html>
```

In the above code, `ej.web.all.min.js` script reference has been added for demonstration purpose. It is not recommended to use this for deployment purpose, as its file size is larger since it contains all the widgets. Instead, you can use [CSG](#) utility to generate a custom script file with the required widgets for deployment purpose.

### Note:

- `react.js` and `react-dom.js` are the core files needed to create react elements.
- `browser.min.js` file is required for code transform.
- `ej.web.react.min.js` is a react-syncfusion bridge to render Syncfusion components.

*Control Initialization*

Add a `div` container to render the PivotTreeMap.

**HTML**

```
<!DOCTYPE html>
<html>
<body>
<div id="PivotTreeMap1" style="width:99%;"></div>
</body>
</html>
```

Initialize the PivotTreeMap by using the `EJ.PivotTreeMap` tag.

**HTML**

```
<!DOCTYPE html>
<html>
<head>
<style>
#Olap{
width: 100%;
height: 450px;
}
</style>
</head>
<body>
<div id="PivotTreeMap1" style="width:99%;"></div>
<!-- Tooltip can be localized here -->
<script id="tooltipTemplate" type="application/jsrender">
<div style="background:White; color:black; font-size:12px; font-
weight:normal; border: 1px solid #4D4D4D; white-space: nowrap;border-radius:
2px; margin-right: 25px; min-width: 110px;padding-right: 5px; padding-left:
5px; padding-bottom: 2px ;width: auto; height: auto;">
<div>Measure(s) : {{:~Measures(#data)}}</div><div>Row :
{{:~Row(#data)}}</div><div>Column : {{:~Column(#data)}}</div><div>Value :
{{:~Value(#data)}}</div>
</div>
</script>
<script type="text/babel">
ReactDOM.render(
<EJ.PivotTreeMap id="Olap"></EJ.PivotTreeMap>,
document.getElementById('PivotTreeMap1')
);
</script>
</body>
</html>
```

*Populate PivotTreeMap with data*

Let us now see how to populate the PivotTreeMap control using a sample JSON data as shown below.

**HTML**

```
<script type="text/babel">
var Olap_dataSource={
data: "http://bi.syncfusion.com/olap/msmdpump.dll",
```



```

catalog: "Adventure Works DW 2008 SE", //"Adventure Works DW 2008 SEstandard
Edition
cube: "Adventure Works", rows: [{ fieldName: "[Date].[Fiscal]" }], columns:
[{ fieldName: "[Customer].[Customer Geography]" }],
values: [{ measures: [{ fieldName: "[Measures].[Internet Sales Amount]" }],
axis: "columns" }]
};
$(function() {
ReactDOM.render(
<EJ.PivotTreeMap id="Olap" dataSource= {Olap_dataSource}></EJ.PivotTreeMap>,
document.getElementById('PivotTreeMap1')
);
});
</script>

```

The above code will generate a simple PivotTreeMap with internet sales amount over a period of fiscal years across different customer geographic locations.



## ProgressBar

### Overview

**Essential JavaScript ProgressBar** control is a graphical control element

<http://en.wikipedia.org/wiki/Graphicalcontrol element> used to visualize the changing status of an extended operation. It is available in **JavaScript** product. **ProgressBar** provides an interactive way to display the progression of the task. You can configure the item size, orientation and the display text on the **ProgressBar** control.

### Key Features

Some important features of **ProgressBar** are as follows:

- **Value support** — you can set a numeric value for the **ProgressBar** status.
- **Percentage support** — you can set a percentage value for the control.

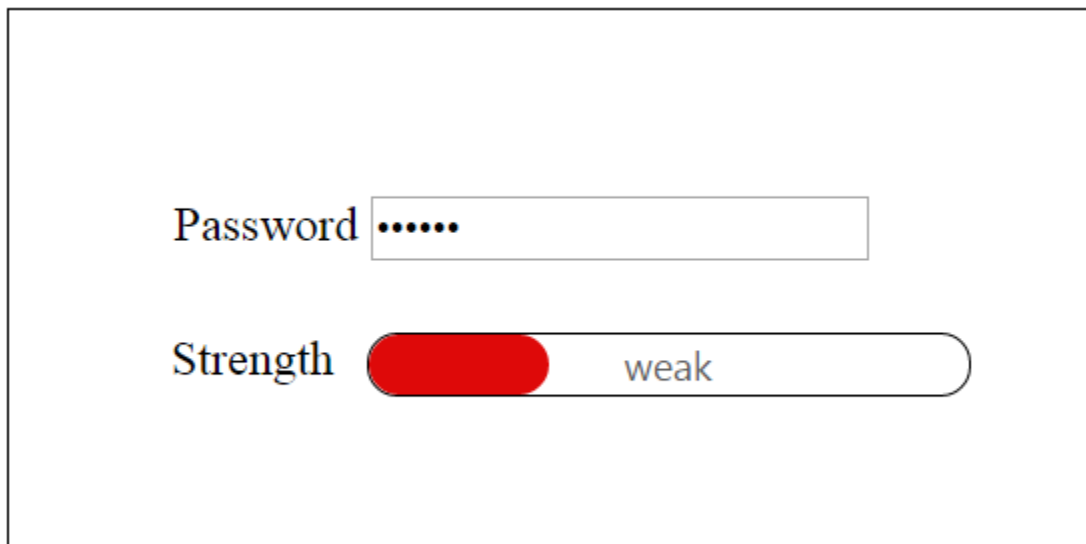
- **CustomText** — **ProgressBar** control allows you to set the text value that is to be displayed inside the control.
- **StateMaintenance** — the progress value is maintained even after the page is refreshed to resume the current progress of the operation.
- **Range** — to set the minimum and maximum values of the **ProgressBar** control
- **Theme** — **ProgressBar** control consists of 12 built-in themes (6 – metro and 6 – gradient effects), and also supports custom skins to set user-defined themes.

## Getting Started

This section briefly describes how to create a **ProgressBar** control using **Javascript** and learn its features.

**Essential JavaScript ProgressBar** displays a **ProgressBar** within a web page that allows you to show the progress of an event. Here, you can learn how to customize the progress and color of the **ProgressBar** in a real-time application to indicate the strength of the password, where the progress changes with respect to the change in length of the password. This helps you to validate the password is typed.

The following screenshot shows the **ProgressBar**.



## Create a ProgressBar

**Essential JavaScript ProgressBar** widget is created using a simple **<div>** element. This element provides built-in features that allow you to change the progress, size and text of the control.

You can create a React application and add necessary scripts and styles with the help of the given [React Getting Started Documentation](#).

Create a JSX file for rendering ProgressBar component using **<EJ.ProgressBar>** syntax. Add required properties to it in **<EJ.ProgressBar>** tag element

## JS

```
var DefaultProgressbar = React.createClass({
  render: function () {
    return (
      <div id="progressbar_default">
        <div className="imgframe">
          <span className="txt">Water Purification</span>
        </div>
      </div>
    );
  }
});
```

```

<EJ.ProgressBar value={45} height="20" text="45 %">
</EJ.ProgressBar>
</div>
</div>
);
}
});
ReactDOM.render(<DefaultProgressbar />,
document.getElementById('progressbar-default'));

```

Define an HTML element for adding ProgressBar in the application and refer the JSX file.

### HTML

```

<div id="progressbar-default"></div>
<script src="app/progressbar/default.js">

```

Add **<input>** element inside the **<body>** tag of your file to create a **ProgressBar**.

### HTML

```

<div style="content-container-fluid">
<div class="row">
<div class="cols-sample-area">
<div class="frame">
<div class="wrap_up">
<!--Initializing password field-->
<label for="startButton">Password</label>
<input type="password" id="password" style="border-radius:0px"/>
</div>
<div class="control">
<!--initializing ProgressBar control-->
<div id="progressBar"></div>
</div>
</div>
</div>
</div>
</div>

```

It also includes a Password field and through that the progress of the **ProgressBar** can be controlled

Initialize **ProgressBar** in script.

### JAVASCRIPT

```

var DefaultProgressbar = React.createClass({
  componentDidMount: function () {
    $("#progressbar-default").ejProgressBar({
      height: 20,
      value: 30, /*Specify the initial value of the progress in percentage*/
      width: 200,
    });
    progresObj = $("#progressbar-default").data("ejProgressBar");
    progresObj.option("text", "weak");
    $(".e-progress").css({ "background-color": "#DE0909", "border-radius":"10px"
  });

```

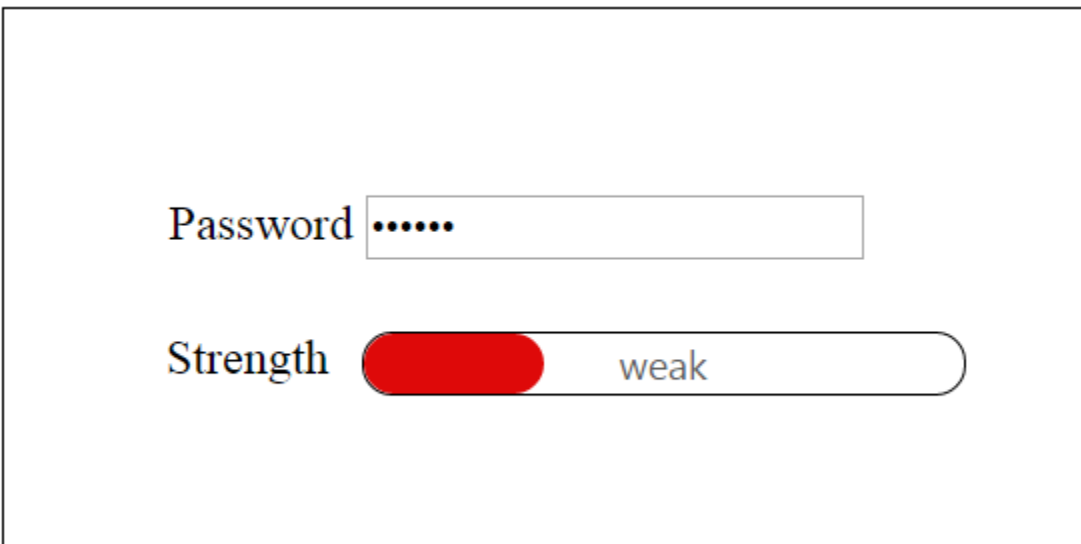
```

$(".e-progressbar").css({ "border-radius": "10px", "border": "1px solid black" });
render: function () {
  return (
    <div id="progressbar_default">
      <div classname="imgframe"></div>
    </div>
  );
}
});
ReactDOM.render(<defaultprogressbar />,
  document.getElementById('progressbar-default'));

```

Here, you can initialize the properties of the **ProgressBar** such as height, value, width, text that is applied to the control by default.

The following screenshot displays a **ProgressBar** control.



Include the following code within the **<head>** tag to change the page layout.

### CSS

```

<style type="text/css" class="cssStyles">
/*applying styles */
.frame {
border: 1px solid #BBBCBB;
border-radius: 10px 10px 10px 10px;
padding: 50px 60px;
margin-top: 40px;
width: 400px;
margin-left: 400px;
}
.control {
margin-bottom: 5px;
margin-left: 230px;
}
.wrap_up {

```

```
margin-left: 105px;
font-size: 18px;
}
#progressBar {
margin-top: 10px;
}
</style>
```

### Progress Control using Length of the Password Field

In real-time scenario, the progress of **ProgressBar** is changed according to the length of text in the password field by binding the change in the properties of control and checking the length of the password field.

Add the following code example inside the **<script>** tag of your **HTML** file.

#### JAVASCRIPT

```
var DefaultProgressbar = React.createClass({
  componentDidMount: function () {
    $("#progressbar-default").ejProgressBar({
      height: 20,
      value: 30, /*Specify the initial value of the progress in percentage*/
      width: 200,
    });
    progresObj = $("#progressbar-default").data("ejProgressBar");
    progresObj.option("text", "weak");
    $(".e-progress").css({ "background-color": "#DE0909", "border-radius": "10px" });
    $(".e-progressbar").css({ "border-radius": "10px", "border": "1px solid black" });
    var progresObj, buttonObj, k = 10, timer = window.clearInterval(timer), i = 0, obj;
    $(document).keypress(function () { //To capture the keypress inside the document
      i = $("#password").val().length;
      if (i < 5)
        weak();
      else if (i == 5 || i < 7)
        Strong();
      else if (i == 7 || i > 7) {
        var pwd = $("#password").val();
        very_strong();
      }
    });
    function Strong() { //Change the width and text of the progress ...
      called when the length is greater than 5
      progresObj.option("text", "strong");
      progresObj.option("percentage", k + 50);
      $(".e-progress").css("background-color", "#0055FF");
      $(".e-progressbar").css("color", "#000000");
    }
    function very_strong() { //Change the width and text of the progress ...
      called when the length is greater than 7
      progresObj.option("text", "Very strong");
      progresObj.option("percentage", k + 90);
      $(".e-progress").css("background-color", "Green");
    }
  }
});
```

```

$(".e-progressbar").css("color", "#000000");
}
function weak() {      //Change the width and text of the progress... called
when the length is less than 5
progresObj.option("text", "Weak");
progresObj.option("percentage", k+20 );
$(".e-progress").css("background-color", "#DE0909");
$(".e-progressbar").css("border-radius", "10px");
}
},
render: function () {
return (
<div id="progressbar_default">
<div classname="imgframe"></div>
</div>
);
}
});
ReactDOM.render(<defaultprogressbar />,
document.getElementById('progressbar-default'));

```


You can calculate length of the password and call the appropriate function that changes the percentage property of **ProgressBar**.

- The **weak()** function changes the text inside the ProgressBar to **Weak** and percentage to 30, that is invoked when the length of the text is less than 5.
- The **strong()** function changes the text inside the ProgressBar to **Strong** and percentage to 60, that is invoked when the length of the text exceeds 5.
- The **very\_strong()** function changes the text inside the ProgressBar to **Very Strong** and percentage to 100, that is invoked when the length of the text exceeds 7 and the text contains a symbol in it.

You can change themes or appearance of the ProgressBar as required.


The final output is displayed as follows.

Password

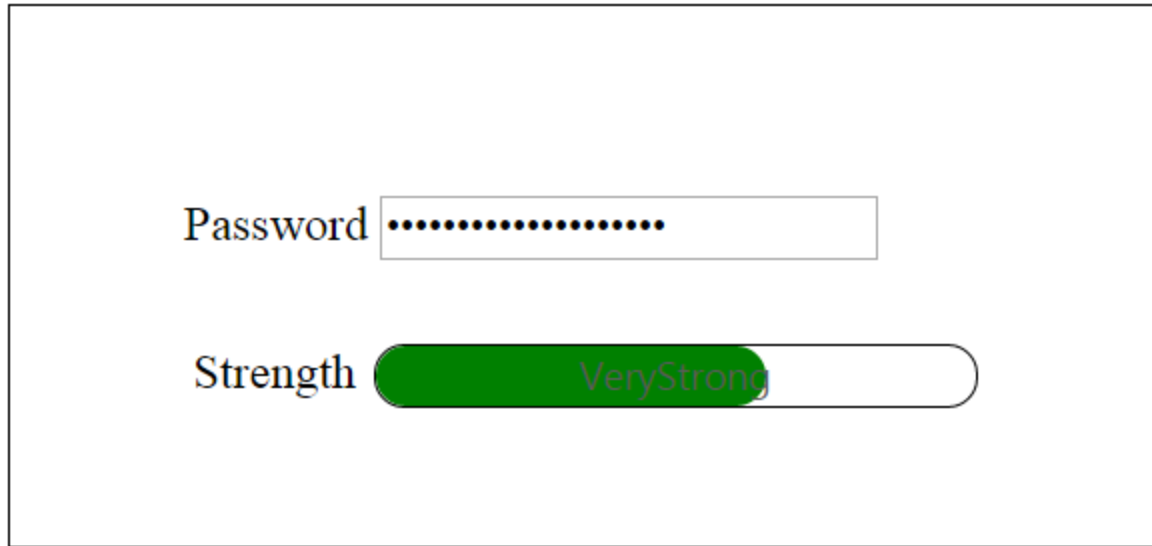
Strength  weak

The first form shows a password field with 5 dots and a strength indicator. The strength bar is a rounded rectangle with a red fill covering approximately 25% of its width, followed by a white fill. The word "weak" is displayed in a light gray font to the right of the bar.

Password

Strength  Strong

The second form shows a password field with 8 dots and a strength indicator. The strength bar is a rounded rectangle with a blue fill covering approximately 75% of its width, followed by a white fill. The word "Strong" is displayed in a light gray font to the right of the bar.



You can also bind an event at the start and finish of a ProgressBar by using the start, complete and change properties of the ProgressBar.

## RadialMenu

### Overview

Our Essential ReactJS RadialMenu component is a context that represents the menu items are arranged in a circular order with a centric button element in it. By default, center button only is visible. The Radial Menu displays the root level menu item with rotational animation effects on clicking the center menu button. You can close it either by clicking anywhere in the document or by clicking the center button where the root level items are displayed.

### Key Features

- **Nested Menu:** Supports to render the multiple levels of sub-menu items.
- **Image Customization:** Enables to customize images for all levels of the menu item.
- **Item Customization:** Supports to customize Radial Menu items with badges and slider settings.
- **Dimension:** Allows to customize Radial Menu radius and position.

### Getting Started

This section helps to get started of the RadialMenu component in a React application.



Model-view-controller (MVC) is a software architecture pattern which separates the representation of information from the user's interaction with it. The model consists of information data, business rules, logic, and functions. A view can be any output representation of data, such as a bar chart for management and a tabular representation for the model or view. The central id is the controller and in addition to dividing the application into three kinds of components, the controller mediates input, converting it to commands for the model or view. A controller can send commands through a document. It can also send commands through a document. A model notifies its associated views to produce updated output. An implementation of MVC omits the controller if the platform does not support them. A view requests from the model the data to produce an output representation to the user.



### Create a RadialMenu

The following steps guide you to add a RadialMenu component.

Refer the common React JS [Getting Started Documentation](#) to create an application and add necessary scripts and styles for rendering our React JS components.

Create a JSX file for rendering RadialMenu component using <EJ.RadialMenu> syntax. Add required properties to it in <EJ.RadialMenu> tag element.

#### HTML

```
"use strict";
ReactDOM.render(
  <EJ.RadialMenu id="default" imageClass="imageclass"
  backImageClass="backimageclass" targetElementId="radialtarget1" >
</EJ.RadialMenu>
  document.getElementById('radialmenu-default')
);
```

Define an HTML element for adding RadialMenu in the application and refer the JSX file created.

#### HTML

```
<div id="radialmenu-default"></div>
<script type="text/babel" src="sample.jsx">
```

This will render an empty RadialMenu component on executing.

*Note: You can find the RadialMenu properties from the [API reference](#) document*

### Configure Items

To configure items for RadialMenu component, define RadialMenu items using 'li' element in JSX. You can set the images for each item by giving the image URL with the data-ej-imageUrl attribute in the inner list element and text with data-ej-text attribute for the Item.

Refer to the following code example. Initialize Radial Menu component with items and set its target content as follows.

### HTML

```
<EJ.RadialMenu id="defaultradialmenu" imageClass="imageclass"
backImageClass="backimageclass" targetElementId="radialtarget1">
<ul>
<li data-ej-imageurl="content/images/radialmenu/font.png" data-ej-
text="Bold"></li>
<li data-ej-imageurl="content/images/radialmenu/fl.png" data-ej-
text="Italic"></li>
<li data-ej-imageurl="content/images/radialmenu/redo.png" data-ej-
text="Redo"></li>
<li data-ej-imageurl="content/images/radialmenu/undo.png" data-ej-
text="Undo"></li>
</ul>
</EJ.RadialMenu>
```

Refer to the following code example to add target content to the **RadialMenu**. You need to perform any actions while selecting the RTE content, you need to add **Select** and **change** events in RTE.

### HTML

```
<div id="radialtarget1">
<EJ.RTE id="rteSample4" width="100%" height="400" change={this.rteChange}
select={this.radialShow} showToolbar={false} showContextMenu={false}
value={content}>
</EJ.RTE>
</div>
```

Define the RTE content by using value property of RTE. Declare the value property in <EJ.RTE> attribute and define the content of RTE in JSX.

### JAVASCRIPT

```
var content = "Model-view-controller (MVC) is a software architecture
pattern which separates the representation of information from the user's
interaction with it. The model consists of application data, business rules,
logic, and functions. A view can be any output representation of data, such
as a chart or a diagram. Multiple views of the same data are possible, such
as a bar chart for management and a tabular view for accountants. The
controller mediates input, converting it to commands for the model or
view. The central ideas behind MVC are code reusability and in addition to
dividing the application into three kinds of components, the MVC design
defines the interactions between them. A controller can send commands to its
associated view to change the view's presentation of the model (e.g., by
scrolling through a document). It can also send commands to the model to
update the model's state (e.g., editing a document). A model notifies its
associated views and controllers when there has been a change in its state.
This notification allows the views to produce updated output, and the
controllers to change the available set of commands. A passive
implementation of MVC omits these notifications, because the application
does not require them or the software platform does not support them. A view
requests from the model the information that it needs to generate an output
representation to the user.";
```

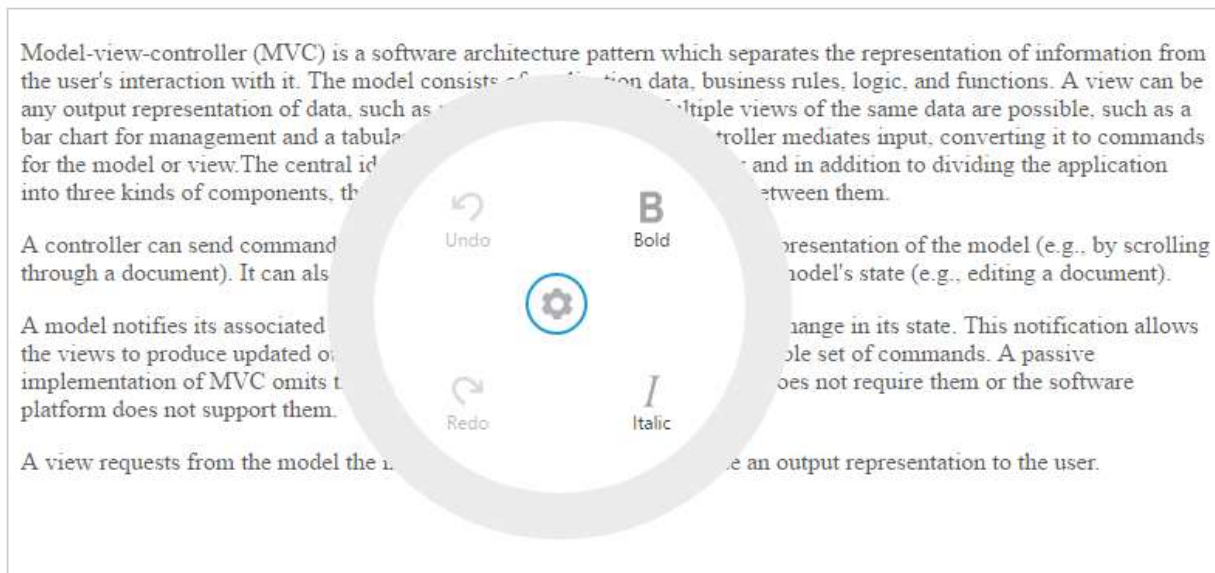
### Displaying RadialMenu

You can display the Radial Menu by performing desired action on the target content while selecting the text inside the target. Therefore, call the **radialShow** event to perform the select action of the RTE content. Refer to the following code example and add it to event handler function.

#### JAVASCRIPT

```
radialShow: function (e) {
    var target = $("#radialtarget1"), radialRadius = 150, radialDiameter = 2 *
    radialRadius,
    // To get Iframe positions
    iframeY = target.offset().top + e.event.clientY, iframeX =
    target.offset().left + e.event.clientX,
    // To set Radial Menu position within target
    x = iframeX > target.width() - radialRadius ? target.width() -
    radialDiameter : (iframeX > radialRadius ? iframeX - radialRadius : 0),
    y = (iframeY > target.height() - radialRadius ? target.height() -
    radialDiameter : (iframeY > radialRadius ? iframeY - radialRadius : 0)) +
    radialRadius;
    $('#defaulttrradialmenu').ejRadialMenu("setPosition", x, y);
    $('iframe').contents().find('body').blur();
},
```

Run the above code and select any text inside the target. The settings icon is displayed. Click that icon to render the following output.



### RadialMenu item functionalities

You can set the functionalities for each item and define click event by using **Click** event of RadialMenu. Refer to the following code example. Define the click event for Radial Menu component as follows.

#### HTML

```

<EJ.RadialMenu id="defaultradialmenu" imageClass="imageclass"
backImageClass="backimageclass" targetElementId="radialtarget1"
click={this.onItemClick}>
<ul>
<li data-ej-imageurl="content/images/radialmenu/font.png" data-ej-
text="Bold"></li>
<li data-ej-imageurl="content/images/radialmenu/fl.png" data-ej-
text="Italic"></li>
<li data-ej-imageurl="content/images/radialmenu/redo.png" data-ej-
text="Redo"></li>
<li data-ej-imageurl="content/images/radialmenu/undo.png" data-ej-
text="Undo"></li>
</ul>
</EJ.RadialMenu>

```

Refer to the following code example to add functionalities for each items in event handler for items click and you can enable items in RadialMenu by using **Change** event in JSX.

### JAVASCRIPT

```

rteChange: function (e) {
$('#defaultradialmenu').ejRadialMenu("enableItem", "Undo");
},
onItemClick: function (e) {
var rteObj = $('#rteSample4').data("ejRTE");
var itemName = (ej.isNullOrUndefined(e.text)) ? "none" :
e.text.toLowerCase();
switch (itemName) {
case "bold":
rteObj.executeCommand("bold");
break;
case "italic":
rteObj.executeCommand("italic");
break;
case "undo":
rteObj.executeCommand("undo");
break;
case "redo":
rteObj.executeCommand("redo");
break;
}
}
}

```

Run the above code and select any text inside the target. The settings icon is displayed. Click that icon to render the RadialMenu component. Click **bold** item in RadialMenu component, to render the following output.

Model-view-controller (MVC) is a software architecture pattern which separates the representation of information from the user's interaction with it. The model consists of application data, business rules, logic, and functions. A view can be any output representation of data, such as a chart or a table. Multiple views of the same data are possible, such as a bar chart for management and a tabular view for finance. The controller mediates input, converting it to commands for the model or view. The central ideas behind MVC are: in addition to dividing the application into three kinds of components, the MVC pattern also provides a set of guidelines for how to use them.

A controller can send commands to the model (e.g., by scrolling through a document). It can also send commands to the view (e.g., by editing a document).

A model notifies its associated view of a change in its state. This notification allows the views to produce updated output representations. An implementation of MVC omits these notifications if the platform does not support them.

A view requests from the model the information it needs to produce an output representation to the user.



## RadialSlider

### Overview

Our Essential ReactJS RadialSlider component that provides an optimized interface for selecting a numeric value using touch interface. This allows the user to select a value or range of values in a circular motion.

### Key Features

- **Angle Support:** Provides start and end angle level view of RadialSlider.
- **Animation:** Offers animation effect for the RadialSlider handle.
- **Image Customization:** Supports customize the images of the inner circle in Radial Slider.
- **Dimension:** Allows to change the radius and Stroke width of RadialSlider.
- **Accuracy:** Provides way to select accurate numeric value and customize the display values

### Getting Started

This section helps you to understand the getting started of the RadialSlider component with the step-by-step instructions

#### Create a simple Radial Slider

Refer the common React's Getting Started Documentation to create an application and add necessary scripts and styles for rendering our React JS components.

Create a JSX file for rendering RadialSlider component using <EJ.RadialSlider> syntax. Add required properties to it in <EJ. RadialSlider > tag element

#### JAVASCRIPT

```
ReactDOM.render(
  <EJ.RadialSlider id="defaultslider" radius={150}
  innerCircleImageUrl="http://js.syncfusion.com/demos/web/content/images/radialslider/chevron-right.png">
  </EJ.RadialSlider>,
  document.getElementById('RadialSlider-default')
```

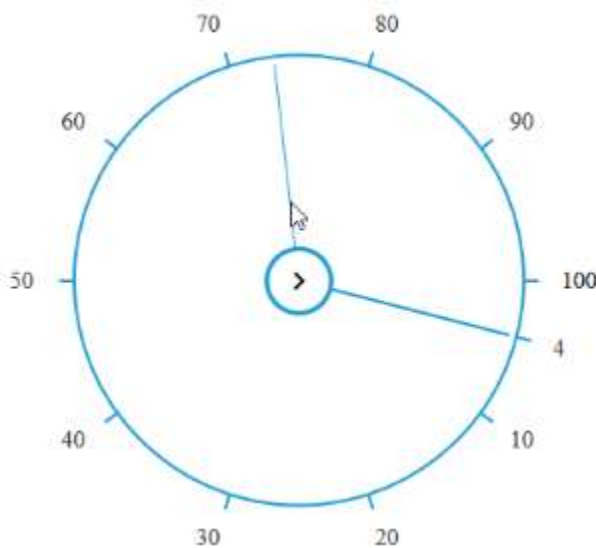
```
);
```

Define an HTML element for adding RadialSlider in the application and refer the JSX file created with script type "text/babel".

### HTML

```
<div id="RadialSlider-default"></div>
<script type="text/babel" src="sample.jsx">
```

This will render a RadialSlider component on executing.



*Note: You can find the RadialSlider properties from the [API reference](#) document*

## RadioButton

### Overview

The **RadioButton** component allows you to pick an option from multiple options to perform an action based on selection.

### Key Features

- **Trendy Look** : Rich appearance with theme Support
- **RTL** : Supports for right to left alignment
- **Easy Customization**: The customization of **RadioButton** is made simple
- **Themes**: Supports 17 built-in themes (6 – flat and 6 – gradient effects), and also support custom skin options to set user-defined themes.
- **Responsive and Touch support**: Fits in all range of devices and supports touch devices.

### GettingStarted

This section discloses the details on how to render and configure RadioButton component in a ReactJS application.

### Create a RadioButton

Create an HTML page and refer the necessary script and CSS dependency files in your application with the help of given [getting started documentation](#)

With ReactJS, the components can be initialized in two ways.

1. Using jsx Template
2. Without using jsx Template

#### Using jsx Template

You can render EJ components by using JSX template, wherein this JSX file will be converted to its equivalent JS file.

1. Create an HTML file and add a div element and give it an ID.

#### HTML

```
<body>
<div id="container"></div>
</body>
```

2. Create a JSX file and initialize RadioButton component by using the below code snippet.

#### HTML

```
ReactDOM.render (
  <div>
    <br />
    Category
    <br />
    <br />
    <table >
      <tr>
        <td>
          <EJ.RadioButton id="fresher" text="Fresher" name="category"
            value="fresher"></EJ.RadioButton>
        </td>
        <td colspan="2">
          <EJ.RadioButton id="experienced" text="Experienced" name="category"
            value="experienced" checked={true}></EJ.RadioButton>
        </td>
      </tr>
    </table>
    <br />
    Experienced
    <br />
    <br />
    <table>
      <tr>
        <td>
          <EJ.RadioButton id="exp1" text="1+ years" name="experienced" value="1+
            years" checked={true}></EJ.RadioButton>
```

```

</td>
<td colspan="2">
<EJ.RadioButton id="exp2" text="2.5+ years" name="experienced" value="2.5+
years"></EJ.RadioButton>
</td>
<td colspan="2">
<EJ.RadioButton id="exp5" text="2.5+ years" name="experienced" value="5+
years"></EJ.RadioButton>
</td>
</tr>
</table>
</div>,
document.getElementById('container')
);

```

3. Refer the JSX file created in last step in the HTML file as given below.

### HTML

```

<body>
<div id="dtp"></div>
<!-- Radiobutton.jsx created in previous step-->
<script type="text/babel" src="RadioButton.jsx">
</script>
</body>

```

Now the jsx file will be compiled into its equivalent Javascript file by means of Babel.

Execute the above code to render RadioButton component.

Category

☐ Fresher ☒ Experienced

Experienced

☒ 1+ years ☐ 2.5+years ☐ 5+years

### Without using jsx Template

The RadioButton component can be initialized without using JSX template.

1. Create an HTML page and render a  
element with an ID set to it.



## HTML

```
Category
<table>
<tr>
<td>
<div id="radio1"></div>
</td>
<td>
<div id="radio2"></div>
</td>
</tr>
</table>
```

2. Render the RadioButton component by using the below mentioned code snippet.

## JAVASCRIPT

```
ReactDOM.render(
  React.createElement(EJ.RadioButton, {
    id: "radio_btn1",
    name: "category",
    value: "fresher",
    text: "Fresher",
    checked: true
  }),
  document.getElementById('radio1')
);
ReactDOM.render(
  React.createElement(EJ.RadioButton, {
    id: "radio_btn2",
    name: "category",
    value: "experienced",
    text: "Experienced"
  }),
  document.getElementById('radio2')
);
```

Run the above code snippet to get the following output.

Category

☒ Fresher ☐ Experienced

## RangeNavigator

### Getting Started

This section explains you the steps required to populate the RangeNavigator with data, add data labels, tooltips and title to the Chart. This section covers only the minimal features that you need to know to get started with the RangeNavigator.

### Create your RangeNavigator

This section encompasses on how to configure the ejRangeNavigator and update the chart control for RangeNavigator's selected range. It also helps you to learn how to pass the required data to RangeNavigator and customize the scale and selected range for your requirements. In this example, you will look at the steps to configure a RangeNavigator to analyze sales of a product for a particular quarter in a year.



### Adding Script Reference

Create an **HTML** page and add the scripts references in the order mentioned in the following code example.

- [jQuery](#) 1.10.2 and later versions

The required ReactJS script dependencies as follows. And you can also refer [React](#) to know more about react js.

- react.min.js - <http://cdn.syncfusion.com/js/assets/external/react.min.js>
- react-dom.min.js - <http://cdn.syncfusion.com/js/assets/external/react-dom.min.js>
- ej.web.react.min.js - <http://cdn.syncfusion.com/{{ site.releaseversion }}/js/common/ej.web.react.min.js>

To get started, you can use the `ej.web.all.min.js` file that encapsulates all the `ej` controls and frameworks in one single file.

### HTML

```
<!DOCTYPE html>
```

```

<html>
<head>
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<meta name="description" content="Essential Studio for React JS">
<meta name="author" content="Syncfusion">
<title>Getting Started for Ribbon React JS</title>
<!-- Essential Studio for JavaScript theme reference -->
<link href="http://cdn.syncfusion.com/{{ site.releaseversion }}/js/web/flat-
azure/ej.web.all.min.css" rel="stylesheet" />
<!-- Essential Studio for JavaScript script references -->
<script src="http://cdn.syncfusion.com/js/assets/external/jquery-
3.0.0.min.js"></script>
<script
src="http://cdn.syncfusion.com/js/assets/external/react.min.js"></script>
<script src="http://cdn.syncfusion.com/js/assets/external/react-
dom.min.js"></script>
<script src="http://cdn.syncfusion.com/{{ site.releaseversion
}}/js/web/ej.web.all.min.js"></script>
<script src="http://cdn.syncfusion.com/{{ site.releaseversion
}}/js/common/ej.web.react.min.js"></script>
<!-- Add your custom scripts here -->
</head>
<body>
</body>
</html>

```

**Note:** 1. In production, we highly recommend you to use our [custom script generator](#) to create custom script file with required controls and its dependencies only. Also to reduce the file size further please use [GZip compression](#) in your server.

2. For themes, you can use the `ej.web.all.min.css` CDN link from the code snippet given. To add the themes in your application, please refer to [this link](#).

### Control Initialization

Control can be initialized in two ways.

- Using jsx Template
- Without using jsx Template

### Using jsx Template

By using the jsx template, we can create the html file and jsx file. The `.jsx` file can be convert to `.js` file and it can be referred in html page.

### Configure ejRangeNavigator

Getting started with your ejRangeNavigator is simple. You can initialize the ejRangeNavigator by setting its range values.

1. Create a

tag.

### HTML

```

<!DOCTYPE html>
<html>

```

```
<body>
<div id="rangeNavigator-default" ></div>
<script src="app/rangeNavigator/default.js"></script>
</body>
</html>
```

2. Initialize the RangeNavigator by using the `EJ.RangeNavigator` tag.

#### JAVASCRIPT

```
<script type="text/babel">
use strict";
var rangeSettings={
start: "2010/1/1", end: "2010/12/31"
};
ReactDOM.render(
<div className="default">
ReactDOM.render(
<div className="default">
<EJ.RangeNavigator id="rangenavigator1" rangeSettings
={rangeSettings}></EJ.RangeNavigator>,
</div>,
document.getElementById('rangenavigator-default')
);
document.getElementById('rangenavigator-default')
);
</script>
```

The following Screen shot displays the RangeNavigator with a range from 2010 January 1st to December 31st.



#### Add series

To add a series to **RangeNavigator**, you need to set **dataSource** property, as given in the following code example.

You can create data source for RangeNavigator as follows.

#### JAVASCRIPT

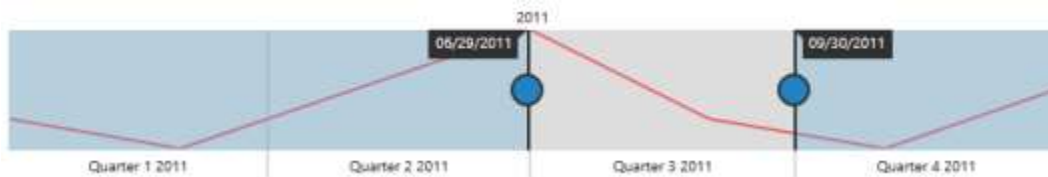
```
var data = [{ "xDate": new Date(2011, 0, 1), "yValue": 10 },
{ "xDate": new Date(2011, 2, 1), "yValue": 5 },
{ "xDate": new Date(2011, 4, 1), "yValue": 15 },
{ "xDate": new Date(2011, 6, 1), "yValue": 25 },
{ "xDate": new Date(2011, 8, 1), "yValue": 10 },
{ "xDate": new Date(2011, 10, 1), "yValue": 5 },
{ "xDate": new Date(2011, 12, 1), "yValue": 15 }];
```

Now, add the `dataSource` to the `RangeNavigator` and provide the field name to get the values from the `dataSource` in `xName` and `yName` options

### JAVASCRIPT

```
<script type="text/babel">
var series = [
{
type: 'line',
dataSource: data, xName: "XValue", yName: "YValue",
}
];
<!DOCTYPE html>
<html>
<body>
<script type="text/babel">
ReactDOM.render(
<div className="default">
<EJ.RangeNavigator id="rangenavigator1"
series={series}></EJ.RangeNavigator>,
</div>,
document.getElementById('rangenavigator-default')
);
</script>
</body>
</html>
```

The following screenshot displays the `RangeNavigator` with the type series as “line”.



### Enable tooltip

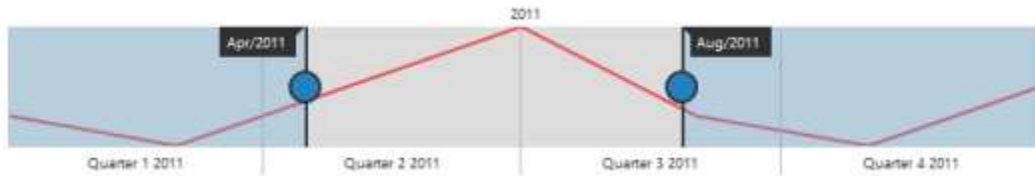
Tooltip can be customized for `RangeNavigator` using `tooltip` option. You can also use `TooltipDisplayMode` option in `tooltip` to display the tooltip “always” or “ondemand” (displays tooltip only while dragging the sliders). You can also specify label format for tooltip using `LabelFormat`.

### JAVASCRIPT

```
<script type="text/babel">
var tooltipSettings= {
visible: true, labelFormat: "MMM/yyyy", tooltipDisplayMode: "always"
};
<!DOCTYPE html>
<html>
<body>
<script type="text/babel">
ReactDOM.render(
<div className="default">
<EJ.RangeNavigator id="rangenavigator1" tooltipSettings =
{tooltipSettings}></EJ.RangeNavigator>,
</div>,
document.getElementById('rangenavigator-default')
```

```
);
</script>
</body>
</html>
```

The following screen shot displays the label format Tooltip in RangeNavigator:



### Update Chart

You can use ejRangeNavigator with controls such as chart and grid to view the range of data selected in ejRangeNavigator.

In order to update chart, whenever the selected range changes in ejRangeNavigator, you need to use rangeChanged event of ejRangeNavigator and then update the chart with the selected data in this event.

You can create a chart with line series using the following code sample.

Create a <div> tag with an id for rendering the chart.

### HTML

```
<body>
<div id="chart-default"></div>
</body>
```

### JAVASCRIPT

```
<script type="text/babel">
function onChartLoaded(sender) {
var chartObj = $("#chart").data("ejChart");
if (chartObj !== null) {
chartObj.model.primaryXAxis.zoomPosition = sender.zoomPosition;
chartObj.model.primaryXAxis.zoomFactor = sender.zoomFactor;
}
$("#default_chart_sample_0").ejChart("redraw");
};
ReactDOM.render(
<div className="default">
<EJ.RangeNavigator id="rangenavigator1" rangeChanged = {onChartLoaded}
dataSource=data xName="XValue" yName="YValue"></EJ.RangeNavigator>,
</div>,
document.getElementById('rangenavigator-default')
);
var title = { text: "Sales Analysis" };
var legend = { visible: true, position: 'top' };
var primaryYAxis = {
title: { text: "Sales(Million)" }
};
var series = [
```

```

{
  name: 'Product A', type: 'line',
  dataSource: data, xName: "xDate", yName: "yValue"
}
];
ReactDOM.render(
  <div className="default">
    <EJ.Chart id="chart1" title = {title} legend = {legend} primaryYAxis =
    {primaryYAxis} series = {series}></EJ.Chart>
  </div>,
  document.getElementById('chart-default')
);
</script>
</body>
</html>

```

The following screenshot displays how the RangeNavigator is updated when the selected range is changed.



### Set value type

RangeNavigator can also be used with numerical values. You can specify the data type using `ValueType` option.

First let's create a `DataSource` for Chart Series with integer Values.

### JAVASCRIPT

```

var Data = [
  { "xDate": 0, "yValue": 10 },
  { "xDate": 50, "yValue": 5 },

```

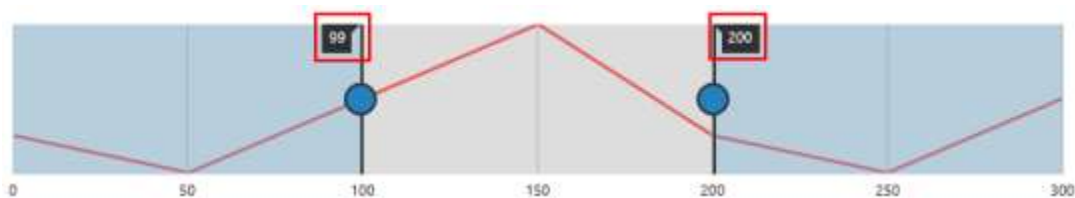
```
{ "xDate": 100, "yValue": 15 },
{ "xDate": 150, "yValue": 25 },
{ "xDate": 200, "yValue": 10 },
{ "xDate": 250, "yValue": 5 },
{ "xDate": 300, "yValue": 15 },
];
```

Now, you can set the dataSource for Chart Series and valueType property to “numeric” as given in the following code example.

### JAVASCRIPT

```
<script type="text/babel">
var series = [
{
type: 'line',
dataSource: data, xName: "XValue", yName: "YValue",
}
];
<!DOCTYPE html>
<html>
<body>
<script type="text/babel">
ReactDOM.render(
<div className="default">
<EJ.RangeNavigator id="rangenavigator1" series={series}
valueType="numeric"></EJ.RangeNavigator>,
</div>,
document.getElementById('rangenavigator-default')
);
</script>
</body>
</html>
```

The following screenshot displays the RangeNavigator with numerical values:



### Without using jsx Template

The Range Navigator can be created from a HTML DIV element with the HTML id attribute set to it. Refer to the following code example.

### HTML

```
<div id="rangenavigator-default"></div>
```

### JAVASCRIPT

```
<script type="text/babel">
var Data = [
```

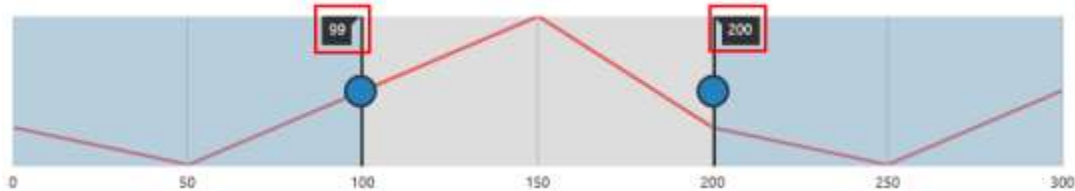


```

{ "xDate": 0, "yValue": 10 },
{ "xDate": 50, "yValue": 5 },
{ "xDate": 100, "yValue": 15 },
{ "xDate": 150, "yValue": 25 },
{ "xDate": 200, "yValue": 10 },
{ "xDate": 250, "yValue": 5 },
{ "xDate": 300, "yValue": 15 },
];
var series = [
{
type: 'line',
dataSource: data, xName: "XValue", yName: "YValue",
}
];
ReactDOM.render(
React.createElement(EJ.RangeNavigator, {id: "defaulttrangenaviagator1",
valueType: "numeric",
series: series,
}
),
document.getElementById('rangenavigator-default')
);
</script>
</body>
</html>

```

The following screenshot displays the RangeNavigator with numerical values:



### Range Types

**RangeNavigator** control is designed to visualize large number of data and navigate to particular data from the large collection at ease. The data for the **RangeNavigator** is either numeric values or **DateTime** values and the **valueType** property in **RangeNavigator** indicates the type of the data that should be passed for the control. By default the **valueType** of **RangeNavigator** is **DateTime**

- Numeric
- DateTime

### Numeric Type

**RangeNavigator** is also used with numeric data and the **valueType** for this data is **"numeric"**

### HTML

```

ReactDOM.render (
<EJ.RangeNavigator id="rangenavigator1" valueType =
"numeric"></EJ.RangeNavigator>,
document.getElementById('rangenavigator')
);

```

The following screenshot displays the **RangeNavigator** with numeric data.



### DateTime

By default the **valueType** of the **RangeNavigator** is “datetime” and represents the **DateTime** values.

### HTML

```
ReactDOM.render(  
  <EJ.RangeNavigator id="rangenavigator1" valueType =  
    "datetime"></EJ.RangeNavigator>,  
  document.getElementById('rangenavigator')  
) ;
```



### DateTime Intervals

The **DateTime** range type contains an **intervalType** property that sets the **DateTime** interval to one of the following:

- Years
- Quarters
- Months
- Weeks
- Days
- Hours

By default **intervalType** for higher level labels are **years** and for lower level labels its **quarters**.

### HTML

```
"use strict";  
var labelSettings =  
{  
  higherLevel:  
  {  
    intervalType: 'years',  
  },  
  lowerLevel:  
  {  
    intervalType: 'quarters',  
  }  
};  
ReactDOM.render(  
  <EJ.RangeNavigator id="rangenavigator1" labelSettings = {labelSettings}  
></EJ.RangeNavigator>,  
  document.getElementById('rangenavigator')  
) ;
```



### Range Padding

**Range Padding** adds padding for range in **RangeNavigator**. It allows you to space the grid lines in the **RangeNavigator**. By default, this property is set to **none**.

#### Numeric

The **rangePadding** property allows you to customize the automatic range calculation using the default auto range calculation for **RangeNavigator**.

#### HTML

```
ReactDOM.render (
  <EJ.RangeNavigator id="range1"      valueType ="numeric"
  rangePadding = 'none ' ></EJ.RangeNavigator>,
  document.getElementById('range')
);
```

#### None

By default, the **rangePadding** for numerical range is none. The range is calculated from the minimum value to the maximum value of data in the **RangeNavigator**.

The following screenshot illustrates a **RangeNavigator** with **rangePadding** set to none.



#### Additional

When you set the **rangePadding** for numerical range to **Additional**, range is padded with an interval.

The following screenshot illustrates a **RangeNavigator** with **rangePadding** set to additional.



#### Normal

In normal **rangePadding**, automatic range calculation differs based on the data.

The following screenshot illustrates **RangeNavigator** with **rangePadding** set to normal



#### Round

Round **rangePadding** for a numerical range rounds the range of the control to the nearest possible value that is divisible by the interval.

The following screenshot illustrates a **RangeNavigator** with **rangePadding** set to **Round**.



#### Padding

The gap between the container and the **RangeNavigator** can be specified using **padding** property.

#### HTML

```
ReactDOM.render (
  <EJ.RangeNavigator id="range1" padding ={15}></EJ.RangeNavigator>,
  document.getElementById('range')
);
```

### AllowSnapping

An **allowSnapping** property toggles the placement of slider exactly on the place it left or on the nearest interval.

#### HTML

```
ReactDOM.render(  
  <EJ.RangeNavigator id="range1" allowSnapping={true}></EJ.RangeNavigator>,  
  document.getElementById('range')  
) ;
```

### Responsive

Set **isResponsive** value to make the **RangeNavigator** responsive on resize.

#### HTML

```
ReactDOM.render(  
  <EJ.RangeNavigator id="range1" isResponsive={true}></EJ.RangeNavigator>,  
  document.getElementById('range')  
) ;
```

### Auto Resizing

Enable **enableAutoResizing** option to resize the **RangeNavigator**.

#### HTML

```
ReactDOM.render(  
  <EJ.RangeNavigator id="range1" enableAutoResizing  
    ={true}></EJ.RangeNavigator>,  
  document.getElementById('range')  
) ;
```

### Customize range Navigator border

**RangeNavigator** provides options to customize the **color**, **opacity** and **width** of range navigator border.

#### HTML

```
var border = {  
  color:'blue',  
  width:2,  
  opacity:0.5  
};  
ReactDOM.render(  
  <EJ.RangeNavigator id="range1" border={border}></EJ.RangeNavigator>,  
  document.getElementById('range')  
) ;
```

### Customize size of range navigator

The **height** and **width** of **RangeNavigator** can be customized using **sizeSettings** property.

#### HTML

```
ReactDOM.render(  
  <EJ.RangeNavigator id="range1" sizeSettings={sizeSettings}></EJ.RangeNavigator>,  
  document.getElementById('range')  
) ;
```

```
<EJ.RangeNavigator id="range1" width={50}
height={50}></EJ.RangeNavigator>,
document.getElementById('range')
);
```

### DateTime

Using the default range calculation for **RangeNavigator**, the **rangePadding** property allows you to customize the range.

### HTML

```
ReactDOM.render(
<EJ.RangeNavigator id="range1" valueType="dateTime"
rangePadding = 'none' ></EJ.RangeNavigator>,
document.getElementById('range')
);
```

### None

By default, the **rangePadding** for **DateTime** range is none. The range is calculated from the minimum value to the maximum value of data in the **RangeNavigator**

The following screenshot illustrates a **RangeNavigator** with **rangePadding** set to none.



### Round

Round **rangePadding** for a **DateTime** range rounds the range of the control to the nearest possible value.

The following screenshot illustrates a **RangeNavigator** with **rangePadding** set to Round.



### Customize axis range of navigator

**RangeNavigator** calculates the range automatically based on the values of series data points. However you can explicitly specify the range using the **start**, **end** properties in **rangeSettings** that is not possible when data is provided.

The following code example renders a **RangeNavigator** with a range from 2010 January 1st to 2013 January 1st.

### HTML

```
var rangeSettings = {
start: "2010/1/1", end: "2012/13/1"
};
ReactDOM.render(
<EJ.RangeNavigator id="range1" rangeSettings = {rangeSettings}
></EJ.RangeNavigator>,
document.getElementById('range')
);
```



### Populate Data

When you provide data to **RangeNavigator**, it produces limited set of data. You can populate the **RangeNavigator** with data using the **dataSource** and **series** properties.

#### Add series to the RangeNavigator

The **Series** property provides access to a collection of all series that are defined explicitly within a **RangeNavigator** control. Each series is assigned with type and name. It contains collection of data point, each point contains x value and y values. You can add data points to the series through **dataSource** property.

### HTML

```
"use strict";
var rangeSettings = {
  start: "2010/1/1", end: "2010/12/31"
};
var series = [{
  type: 'line',
  opacity: 0.5, fill: '#69D2E7',
  border: { color: 'transparent', width: 2 }
}];
ReactDOM.render(
  <EJ.RangeNavigator id="rangenavigator1" load = {onchartload} series={series}
  rangeSettings = {rangeSettings} ></EJ.RangeNavigator>,
  document.getElementById('rangenavigator')
);
var data;
function onchartload(sender) {
  data = GetData();
  sender.model.series[0].dataSource = data.Open;
  sender.model.series[0].xName= "XValue";
  sender.model.series[0].yName= "YValue";
};
function GetData() {
  var series1 = [];
  var value = 100;
  for (var i = 1; i < 360; i++) {
    if (Math.random() > .5) {
      value += Math.random();
    } else {
      value -= Math.random();
    }
    var point1 = { XValue: new Date(2010, 0, i), YValue: value };
    series1.push(point1);
  }
  data = { Open: series1};
  return data;
};
```

The following screenshot illustrates the **RangeNavigator** that is populated with data using **dataSource** property in series.

![[/js/RangeNavigator/Populate-Dataimages/Populate-Dataimg1.png]

### Behavior Customization

**RangeNavigator** allows you to customize the control using events. You can change the range for selected data of the **RangeNavigator** using events.

#### Deferred update

If you set **enableDeferredUpdate** to true, the **rangeChanged** event gets fired after dragging and dropping the slider. By default the **enableDeferredUpdate** is true. If **enableDeferredUpdate** is false then the **rangeChanged** event gets fired while dragging the slider.

#### JAVASCRIPT

```

"use strict";
ReactDOM.render(
  <EJ.RangeNavigator id="default_rn_sample_0"
  enableDeferredUpdate="true"
  >
</EJ.RangeNavigator>,
  document.getElementById('range')
);

```



#### Handle Events

##### loaded: function

This event is handled when the **RangeNavigator** gets loaded. A parameter **sender** is passed to the handler. Using **sender.model**, you can access the RangeNavigator properties.

#### JAVASCRIPT

```

function loaded(sender) {
  sender.model.isResponsive = false;
};
ReactDOM.render(
  <EJ.RangeNavigator id="default_rn_sample_0"
  loaded={loaded}
  >
</EJ.RangeNavigator>,
  document.getElementById('range')
);

```

##### rangeChanged: function

This event gets fired whenever the selected range changes in **RangeNavigator**. A parameter **sender** is passed to the handler. Using **sender.selectedRangeSettings**, you can access the start and end value of range for the selected region.

#### JAVASCRIPT

```

"use strict";
function rangeChanged(sender) {
  console.log(sender.selectedRangeSettings.start);
};
ReactDOM.render(
  <EJ.RangeNavigator id="default_rn_sample_0"
  rangeChanged={rangeChanged}

```

```
>
</EJ.RangeNavigator>,
document.getElementById('range')
);
```

### Use of ZoomCoordinates

**RangeNavigator** is used along with the controls like chart and grid to view the selected data. To update chart/grid, whenever the selected range changes in **RangeNavigator**, you can use **rangeChanged** event of **RangeNavigator** and then update the chart/grid with the selected data in this event.

You can easily update the data for chart by assigning the **zoomFactor** and **zoomPosition** of the **RangeNavigator** to the chart axis.

### JAVASCRIPT

```
"use strict";
// setting zoom factor and position for chart axis in rangeChanged event.
function onChartLoaded(sender) {
var chartObj = $("#container").data("ejChart");
if (chartObj != null) {
chartObj.model.axes[0].zoomPosition = sender.zoomPosition;
chartObj.model.axes[0].zoomFactor = sender.zoomFactor;
}
$("#container").ejChart("redraw");
};
ReactDOM.render(
<EJ.RangeNavigator id="default_rn_sample_0"
//..
rangeChanged={onChartLoaded}
//..
>
</EJ.RangeNavigator>,
document.getElementById('range')
);
```



### Thumb Template

You can customize Thumb template by using **leftThumbTemplate** and **rightThumbTemplate** property. You can add the required template as a "div" element with an "id" to the web page and assign the id or assign the HTML string to this property under **navigatorStyleSettings**.

### HTML

```
<script type="text/x-jsrender" id="left" >
<svg height="24" width="32" style="fill:#DD4A4A;stroke:black;">
<path d="M2 2 L2 22 L22 22 L32 12 L22 2 Z" />
</svg>
</script>
<script type="text/x-jsrender" id="right">
<svg height="24" width="32" style="fill:#DD4A4A;stroke:black;">
<path d="M2 12 L12 22 L32 22 L32 2 L12 2 Z" />
</svg>
</script>
```



**JAVASCRIPT**

```

"use strict";
var navigatorStyleSettings={
  leftThumbTemplate: 'left',
  rightThumbTemplate: 'right',
};
ReactDOM.render(
  <EJ.RangeNavigator id="default_rn_sample_0"
  //...
  navigatorStyleSettings={navigatorStyleSettings}
  //...
  >
  </EJ.RangeNavigator>,
  document.getElementById('range')
);

```

The following screenshot displays the **RangeNavigator** using thumb template.

![[/js/RangeNavigator/Behavior-Customizationimages/Behavior-Customizationimg3.png]

**Value Axis Settings**

You can customize the line, **Font Size**, **gridline**, **tickline**, **range**, **RangePadding** and visibility of **RangeNavigator** axis.

To enable the visibility of axis line, you need to set **Visible** property of **AxisLine** in **ValueAxisSettings**.

You can customize the axis range by specifying **Min**, **Max** and **Interval** for **Range** property.

The **MajorGridLines** can be enabled by specifying **Visible** property. The **Size**, **Width** and **Visible** property of **MajorTickLines** is used to customize the axis tick lines.

The visibility of **ValueAxisSettings** is enabled by setting **Visible** property as true.

**JAVASCRIPT**

```

var valueAxisSettings= {
  Min:10 ,
  Max: 50 ,
  Interval: 5,
  MajorTickLines:{
    visible:'true'
  },
  Size:2,
  Width:2,
  MajorGridLines:{
    visible:'true'
  },
};
ReactDOM.render(
  <EJ.RangeNavigator id="default_rn_sample_0"
  //...
  valueAxisSettings={valueAxisSettings}
  //...
  >
  </EJ.RangeNavigator>,
  document.getElementById('range')
);

```

```
);
```

### Selected Range Settings

The start and end range values of selected range can be customized using **Start** and **End** property of **SelectedRangeSettings**.

#### JAVASCRIPT

```
var selectedRangeSettings= {
  start:'',
  end:''
};
ReactDOM.render(
  <EJ.RangeNavigator id="default_rn_sample_0"
  //..
  selectedRangeSettings={selectedRangeSettings}
  //..
  >
  </EJ.RangeNavigator>,
  document.getElementById('range')
);
```

### Appearance and Styling

**JavaScript RangeNavigator** is enriched with lots of customization options for labels, gridlines and slider to develop high quality graphic rich control.

#### Customize labels

The labels are found along the range, displaying the value of the data it correspond, both on (higher level label) and below (lower level label) the **RangeNavigator**. **RangeNavigator** labels are further customized using **"font"** property in label Settings.

#### JAVASCRIPT

```
"use strict";
var labelSettings= {
  // customizing higher level labels.
  higherLevel: {
    style: {
      font: {
        color: '#ff0000',
        style: 'Normal',
        size: '12px',
        opacity: 1,
        weight: 'regular'
      },
    },
  },
  // customizing lower level labels.
  lowerLevel: {
    style: {
      font: {
        color: '#ff0000',
        style: 'Normal',
        size: '12px',
        opacity: 1,

```

```

weight: 'regular'
},
},
}
};
// ...
var rangeSettings = {
start: "2010/1/1", end: "2010/12/31"
};
ReactDOM.render(
<EJ.RangeNavigator id="range1" labelSettings = {labelSettings}
rangeSettings = {rangeSettings}></EJ.RangeNavigator>,
document.getElementById('range')
);

```



### Label Placement

Labels in **RangeNavigator** are placed inside or outside of the control. You can customize both the higher and lower level labels using **labelPlacement** property in label setting of **RangeNavigator**. By default **labelPlacement** is "outside" for the both higher and lower level labels.

The higher level labels font **Color**, **FontFamily**, **FontStyle**, **FontWeight**, **Opacity** and **Size** can be customized using **HigherLevel** property.

The lower level labels font **Color**, **FontFamily**, **FontStyle**, **FontWeight**, **Opacity** and **Size** can be customized using **LowerLevel** property.

The following screen shot illustrates both the lower and higher level labels that are placed outside the control with **labelPlacement** specified as outside.

### HTML

```

"use strict";
var labelSettings= {
higherLevel: {
labelPlacement: "inside",
},
lowerLevel: {
labelPlacement: "inside"
}
};
// ...
var rangeSettings = {
start: "2010/1/1", end: "2010/12/31"
};
ReactDOM.render(
<EJ.RangeNavigator id="range1" labelSettings = {labelSettings}
rangeSettings = {rangeSettings}></EJ.RangeNavigator>,
document.getElementById('range')
);

```

The following screenshot illustrates a **RangeNavigator** with labels inside the control after specifying the **labelPlacement** as inside.



### Customize NavigatorStyleSettings

RangeNavigator is customized using **NavigatorStyleSettings** properties. You can customize the selected and unselected region color using **SelectedRegionColor** and **UnselectedRegionColor**, **SelectedRegionOpacity** and **UnselectedRegionOpacity** in **NavigatorStyleSettings** and the thumb of the slider using **ThumbColor**, **ThumbRadius** and **ThumbStroke** in **NavigatorStyleSettings**. **MajorGridLineStyle** and **MinorGridLineStyle** are used to customize the major grid line **Color**, **Visible** property and minor gridline **Color** and **Visible**. You can customize the **Background**, **Opacity** and **Border Color**, **DashArray** and **Width** of **navigatorStyleSettings**.

### Customize Labels

The visibility of labels are enabled by setting **Visible** in higher level and **Visible** in lower level. The labels can be aligned by specifying **HorizontalAlignment** of higher level style and **HorizontalAlignment** of lower level style.

You can customize the **Border Color** and **Width**, **Fill**, **GridLineStyle Color**, **DashArray** and **Width**, **Position** property of higher level labels in **labelSettings**.

You can also customize the **Border Color** and **Width**, **Fill**, **GridLineStyle Color**, **DashArray** and **Width**, **Position** property for lower level labels of **labelSettings**.

## JAVASCRIPT

```
"use strict";
var navigatorStyleSettings = {
  unselectedRegionColor: "white",
  selectedRegionColor: "#5EABDE",
  thumbColor: "white",
  thumbRadius: 10,
  thumbStroke: "#303030",
  background: "transparent",
  border: {
    color: "black",
    width: 3
  },
  majorGridLineStyle: {
    color: "transparent"
  },
  minorGridLineStyle: {
    color: "transparent"
  }
};
var labelSettings= {
  higherLevel: {
    style: {
      font: {
        color: 'black',
        size: '13px',
        opacity: 1
      },
      horizontalAlignment: "left"
    },
    intervalType: 'years',
    labelPlacement: "inside"
```

```

},
lowerLevel: {
  style: {
    font: {
      color: 'black',
      size: '12px',
      opacity: 1
    },
    horizontalAlignment: "center"
  },
  intervalType: 'quarters',
  labelPlacement: "inside"
}
};
ReactDOM.render(
  <EJ.RangeNavigator id="range1" labelSettings = {labelSettings}
  navigatorStyleSettings = {navigatorStyleSettings}></EJ.RangeNavigator>,
  document.getElementById('range')
);

```

![[/js/RangeNavigator/Appearance-And-Stylingimages/Appearance-And-Stylingimg3.png)

### Themes

**RangeNavigator** theme is a set of pre-defined options that are applied to the control before each **RangeNavigator** is instantiated. Following predefined themes are available in JavaScript **RangeNavigator**.

1. flat light
2. flat dark
3. gradient light
4. gradient dark
5. azure
6. azure dark
7. lime
8. lime dark
9. saffron
10. saffron dark
11. gradient azure
12. gradient azure dark
13. gradient lime
14. gradient lime dark
15. gradient saffron
16. gradient saffron dark

### JAVASCRIPT

```

"use strict";
ReactDOM.render(
  <EJ.RangeNavigator id="range1" theme = 'saffron'></EJ.RangeNavigator>,
  document.getElementById('range')
);

```



### Tooltip

**RangeNavigator** provides **Tooltip** support for sliders. Sliders are used to select data at particular range in the **RangeNavigator** control. **Tooltips** for sliders display the selected start and end **DateTime** values.

### Customization

**RangeNavigator** provides support for you to customize the text display in the tooltip and background using **tooltipSettings** property. You can change font family, font color, font style, font weight. By default **"Segoe UI"** font family is set to tooltip text.

Tooltip visibility can be enabled or disabled using **visible** property.

- You can change background color of tooltip using **backgroundColor** property.
- You can customize the **color**, **family**, **fontStyle**, **opacity**, **size** and **weight** of tooltip text in **Font** property.

### HTML

```
"use strict";
var tooltipSettings = {
  visible: true,
  backgroundColor: "black",
  // To customize the tooltip text
  font: {
    color: 'red',
    family: 'Segoe UI',
    style: 'Normal',
    size: '12px',
    opacity: 1,
    weight: 'regular'
  }
};
var rangeSettings = {
  start: "2010/1/1", end: "2010/12/31"
};
ReactDOM.render(
  <EJ.RangeNavigator id="rangenavigator1" rangeSettings = {rangeSettings}
  tooltipSettings = {tooltipSettings}></EJ.RangeNavigator>,
  document.getElementById('rangenavigator')
);
```



### Label Format

By default, the **tooltip** texts are automatically determined based on the data points. To make it readable and understandable you can format the **tooltip** text. For **DateTime** data, all globalized format are supported. By default the **labelFormat** is "MM/dd/yyyy".

Some of the **labelFormat** for **DateTime** data area as follows:

- 'MMM, yyyy'
- 'dd, MMM'
- 'dd/MM/yyyy'

- 'dd, hh:mm'
- 'hh:mm:ss'
- 'hh:mm:ss:tt'

### JAVASCRIPT

```
var tooltipSettings = { labelFormat: 'MMM, yyyy' };
ReactDOM.render(
  <EJ.RangeNavigator id="rangenavigator1" tooltipSettings =
    {tooltipSettings}></EJ.RangeNavigator>,
  document.getElementById('rangenavigator')
);
```



#### Tooltip display mode

By default the **tooltip** for RangeNavigator gets displayed. You can change this behavior using the **tooltipDisplayMode** property in the tooltip and it takes the following values.

Value	Description
always	Tooltip get displayed for RangeNavigator always.
onDemand	Tooltip get displayed only when we move the slider.

### JAVASCRIPT

```
var tooltipSettings = { tooltipDisplayMode: "onDemand" };
ReactDOM.render(
  <EJ.RangeNavigator id="rangenavigator1" tooltipSettings =
    {tooltipSettings}></EJ.RangeNavigator>,
  document.getElementById('rangenavigator')
);
```



#### Globalization & Localization

**RangeNavigator** supports Localization and Globalization to customize the labels based on a culture specific to a country. The culture defines specific information for the number formats, week and month names, date and time formats etc.

#### Localization

**Localization** is the process of customizing the user interface based on a culture specific to a particular country or region in order to display regional data. The culture is represented by a unique string, for example, —en-US| for U.S. English and —fr-FR| for French (common), this is achieved by creating a JavaScript file “**rangeNavigatorSource.fr-FR.js**” and setting the equivalent word as illustrated in the following code sample.

### HTML

```
ej.datavisualization.RangeNavigator.locale["fr-FR"] = {
  intervals: {
    //string to display the intervals on RangeNavigator
```

```

quarter: {longQuarters: "Trimestre", shortQuarters: "T"},
week: { longWeeks: "Semaine", shortWeeks: "S" }
}
}

```

**Localization** is the key feature that provides solutions globally with the help of localized control.

### JAVASCRIPT

```

"use strict";
var rangeSettings = {
  start: "2010/1/1", end: "2010/12/31"
};
ReactDOM.render(
  <EJ.RangeNavigator id="rangenavigator1" locale = 'fr-FR'
  rangeSettings = {rangeSettings} ></EJ.RangeNavigator>,
  document.getElementById('rangenavigator')
);

```



### RTL

**Right-to-Left** or **RTL** describes the ability of application to handle and responds you to communicate with a right-to-left language, like Arabic or Japanese. **enableRTL** property is used to change the rendering format to **“Right to Left”**, by default it renders from **“Left to Right”** in **RangeNavigator**.

### HTML

```

"use strict";
var rangeSettings = {
  start: "2010/1/1", end: "2010/12/31"
};
ReactDOM.render(
  <EJ.RangeNavigator id="rangenavigator1" enableRTL={true} locale = 'fr-FR'
  rangeSettings = {rangeSettings} ></EJ.RangeNavigator>,
  document.getElementById('rangenavigator')
);

```



## User Interactions

### Highlight

EjRangeNavigator provides highlighting supports to the intervals on mouse hover. To enable the highlighting option, set the [enable](#) property to true in the [highlightSettings](#) of [navigatorStyleSettings](#).

### HTML

```

"use strict";
var rangeSettings = {
  start: "2010/1/1", end: "2010/12/31"
};
var navigatorStyleSettings = {
  //...
  highlightSettings:{
    // enable the highlight settings
  }
};

```



```

enable: true
}
//...
};
ReactDOM.render(
  <EJ.RangeNavigator id="rangenavigator1"
    navigatorStyleSettings={navigatorStyleSettings} rangeSettings =
    {rangeSettings} ></EJ.RangeNavigator>,
  document.getElementById('rangenavigator')
);

```



[Click](#) here to view the highlight and selections online demo sample.

*Customize the highlight style*

To customize the highlighted intervals, use color, border and opacity options in the [highlightSettings](#).

### HTML

```

"use strict";
var rangeSettings = {
  start: "2010/1/1", end: "2010/12/31"
};
var navigatorStyleSettings = {
  //...
  highlightSettings: {
    // enable the highlight settings
    enable: true ,
    color: '#006fa0',
    border: {
      color: 'red' , width: 2
    }
  }
  //...
};
ReactDOM.render(
  <EJ.RangeNavigator id="rangenavigator1"
    navigatorStyleSettings={navigatorStyleSettings} rangeSettings =
    {rangeSettings}></EJ.RangeNavigator>,
  document.getElementById('rangenavigator')
);

```



### Selection

EjRangeNavigator provides selection supports to the intervals by, clicking and dragging the highlighted intervals. To enable the selection option, set the [enable](#) property to true in the [selectionSettings](#).

### HTML

```

"use strict";
var rangeSettings = {
  start: "2010/1/1", end: "2010/12/31"
};
var navigatorStyleSettings = {

```

```
//...
selectionSettings:{
// enable the selection settings
enable: true
}
//...
};
ReactDOM.render(
<EJ.RangeNavigator id="rangenavigator1"
navigatorStyleSettings={navigatorStyleSettings} rangeSettings =
{rangeSettings} ></EJ.RangeNavigator>,
document.getElementById('rangenavigator')
);
```



[Click](#) here to view the highlight and selections online demo sample.

*Customize the selection style*

To customize the selected intervals, use color, border and opacity options in the selectionSettings.

#### HTML

```
"use strict";
var rangeSettings = {
start: "2010/1/1", end: "2010/12/31"
};
var navigatorStyleSettings = {
//...
selectionSettings:{
// enable the selection settings
enable: true ,
color: '#27e8e5',
border:{
color: 'red' , width: 2
}
}
//...
};
ReactDOM.render(
<EJ.RangeNavigator id="rangenavigator1"
navigatorStyleSettings={navigatorStyleSettings} rangeSettings =
{rangeSettings} ></EJ.RangeNavigator>,
document.getElementById('rangenavigator')
);
```



#### Scrollbar

- To render the Scrollbar in RangeNavigator, you need to enable [enableScrollbar](#) option.
- [scrollRangeSettings](#) of rangenavigator [start](#) and [end](#) value is used to set the minimum and maximum datasource value to be added in the rangenavigator.
- Based on the scrollRangeSettings *start*, *end* value and dataSource *start*, *end* value scrollbar will be adjust.

- When you change the scrollbar position, [scrollEnd](#) event returns the current position of start and end range value.

**HTML**

```
"use strict";
var scrollRangeSettings= {
  start: new Date(2010, 0, 1),
  end: new Date(2011, 10, 31)
};
var rangeSettings = {
  start: "2010/1/1", end: "2010/12/31"
};
ReactDOM.render(
  <EJ.RangeNavigator id="rangenavigator1" enableScrollbar = {true}
  scrollRangeSettings = {scrollRangeSettings} rangeSettings =
  {rangeSettings}></EJ.RangeNavigator>,
  document.getElementById('rangenavigator')
);
```



[Click](#) here to view scrollbar online demo sample.

**Methods**

[\\_destroy\(\)](#)

destroy the range navigator widget

**HTML**

```
<div id="range"></div>
```

**JAVASCRIPT**

```
ReactDOM.render(
  <EJ.RangeNavigator id="default"></EJ.RangeNavigator>,
  document.getElementById('range')
);
function RangeNavigatorMethod() {
  var rangeObj = $("#default").data("ejRangeNavigator");
  rangeObj.destroy();
};
```

**Events**

[load](#)

Fires on load of range navigator.

**HTML**

```
<div id="range"></div>
```

**JAVASCRIPT**

```
ReactDOM.render(
```

```
<EJ.RangeNavigator id="default" load = {Load}></EJ.RangeNavigator>,
document.getElementById('range')
);
function Load() {
  // Do Something
};
```

#### *loaded*

Fires after range navigator is loaded.

#### **HTML**

```
<div id="range"></div>
```

#### **JAVASCRIPT**

```
ReactDOM.render(
  <EJ.RangeNavigator id="default" loaded = {Loaded}></EJ.RangeNavigator>,
  document.getElementById('range')
);
function Loaded() {
  // Do Something
};
```

#### *rangeChanged*

Fires on changing the range of range navigator.

#### **HTML**

```
<div id="range"></div>
```

#### **JAVASCRIPT**

```
ReactDOM.render(
  <EJ.RangeNavigator id="default" rangeChanged =
  {RangeChanged}></EJ.RangeNavigator>,
  document.getElementById('range')
);
function RangeChanged() {
  // Do Something
};
```

#### *scrollChanged*

Fires on changing the scrollbar position of range navigator.

#### **HTML**

```
<div id="range"></div>
```

#### **JAVASCRIPT**

```
ReactDOM.render(
  <EJ.RangeNavigator id="default" scrollChanged =
  {ScrollChanged}></EJ.RangeNavigator>,
  document.getElementById('range')
);
function ScrollChanged() {
  // Do Something
};
```

```
document.getElementById('range')
);
function ScrollChanged() {
  // Do Something
};
```

#### *scrollStart*

Fires on when starting to change the scrollbar position of range navigator.

#### **HTML**

```
<div id="range"></div>
```

#### **JAVASCRIPT**

```
ReactDOM.render(
  <EJ.RangeNavigator id="default" scrollStart =
    {ScrollStart}></EJ.RangeNavigator>,
  document.getElementById('range')
);
function ScrollStart() {
  // Do Something
};
```

#### *selectedRangeStart*

Fires on when starting to change the slider position of range navigator.

#### **HTML**

```
<div id="range"></div>
```

#### **JAVASCRIPT**

```
ReactDOM.render(
  <EJ.RangeNavigator id="default" selectedRangeStart =
    {SelectedRangeStart}></EJ.RangeNavigator>,
  document.getElementById('range')
);
function SelectedRangeStart() {
  // Do Something
};
```

#### *selectedRangeEnd*

Fires when the selection ends in the range navigator

#### **HTML**

```
<div id="range"></div>
```

#### **JAVASCRIPT**

```
ReactDOM.render(
  <EJ.RangeNavigator id="default" selectedRangeEnd =
    {SelectedRangeEnd}></EJ.RangeNavigator>,
  document.getElementById('range')
);
```

```
document.getElementById('range')
);
function SelectedRangeEnd() {
// Do Something
};
```

### scrollEnd

Fires on changes ending the scrollbar position of range navigator.

### HTML

```
<div id="range"></div>
```

### JAVASCRIPT

```
ReactDOM.render(
<EJ.RangeNavigator id="default" scrollEnd =
"ScrollEnd"></EJ.RangeNavigator>,
document.getElementById('range')
);
function ScrollEnd() {
// Do Something
};
```

## Rating

### Overview

Essential **JavaScriptRating** control provides an intuitive **Rating** experience that enables you to select a number of stars that represent a Rating. You can configure the item size, orientation and the number of displayed items in the **Rating** control. You can also customize the **Rating** star image by using custom CSS.

### Key Features

Some important features of Chart for Essential **JavaScript** are as follows:

- **Precision support** - You can set the three different precisions like full, half, and exact.
- **Orientation support** - Rating control can be displayed in horizontal or vertical direction.
- **Read-only mode** - Rating control can be rendered in the read-only mode to enable or disable user interaction.
- **Reset** - You can reset the Rating value by using the Reset button.
- **Tooltip** - You can set a tooltip for displaying the currently selected Rating value.
- **Theme** - Rating control consists of 12 built-in themes ( 6 – metro and 6 – gradient effects), and also supports custom skins to set user-defined themes.

### Getting Started

This section explains briefly about how to create a **Rating** control in your application with **JavaScript**. **Essential JavaScript Rating** helps to select the number of stars that represent Rating. Here, you can learn how to create **Rating** control in a real-time movie download application and also learn how to rate the application.

The following screenshot illustrates the functionality of a Rating widget with a Rating range of 0 to 5.



### Create a Rating Widget in React JS

**Essential JavaScript Rating** widget is provided with built-in features such as precision, orientation and flexible API's. You can create the **Rating** widget by using input textbox element as follows:

You can create a React application and add necessary scripts and styles with the help of the given [React Getting Started Documentation](#).

Define an HTML element for adding Rating in the application and refer the JSX file.

#### HTML

```
<div id="rating-default"></div>
<script src="app/rating/default.js">
```

Create a JSX file for rendering Rating component using <EJ.Rating> syntax. Add required properties to it in <EJ.Rating> tag element

#### JS

```
var DefaultRating = React.createClass({
  render: function () {
    return (
      <div id="rating_default">
        <ej.tab width="100%">
          <ul>
            <li><a href="#steelman">Man of Steel</a></li>
            <li><a href="#worldwar">World War Z</a></li>
            <li><a href="#unive">Monsters University</a></li>
          </ul>
          <div id="steelman">
            <table>
              <tr>
                <td classname="movies-img" valign="top">
                  
                </td>
                <td valign="top">
                  <div>
                    <span classname="movie-header">Man of Steel</span><br />
                    Rating :
```

```

<br />
<ej.rating value={3}></ej.rating><br />
<span>Movie Info:</span>
<p>
A young boy learns that he has extraordinary powers and is not of this
Earth.
</p>
</div>
</td>
</tr>
</table>
</div>
<div id="woldwar">
<table>
<tr>
<td classname="movies-img" valign="top">

</td>
<td valign="top">
<div>
<span classname="movie-header">World War Z</span><br />
Rating :
<br />
<ej.rating value={4}></ej.rating><br />
<span>Movie Info:</span>
<p>
The story revolves around United Nations employee Gerry Lane (Pitt).
</p>
</div>
</td>
</tr>
</table>
</div>
<div id="unive">
<table>
<tr>
<td classname="movies-img" valign="top">

</td>
<td valign="top">
<div>
<span classname="movie-header">Monsters University</span><br />
Rating :
<br />
<ej.rating value={4}></ej.rating><br />
<span>Movie Info:</span>
<p>
Mike Wazowski and James P. Sullivan are an inseparable pair, but that wasn't
always the case.
</p>
</div>
</td>
</tr>
</table>
</div>
</ej.tab>
</div>

```



```
);
}
});
ReactDOM.render(
  <defaultRating />, document.getElementById('rating-default'));

```

Apply the following styles to show the Rating widget in the horizontal order.

### CSS

```
<style type="text/css" class="cssStyles">
.movies-img {
width: 125px;
}
.movie-header {
font-size: 20px;
font-weight: 600;
}
</style>

```

The following screenshot displays a Rating widget.



### Set the Min and Max Value

In a real-time scenario, you can extend the range by using the properties **minValue** and **maxValue** in the **Rating** widget.

### HTML

```
<div id="rating-default"></div>
<script src="app/rating/default.js">

```

### JS

```
var DefaultRating = React.createClass({
  render: function () {
    return (
      <div id="rating_precision">
        <div class="frame">

```

```

<table id="table">
<tr>
<td valign="top">
Rate :
</td>
<td>
<EJ.Rating value={4} minValue={2} maxValue={10} ></EJ.Rating>
</td>
</tr>
</table>
</div>
</div>
);
}
});
ReactDOM.render(<DefaultRating />, document.getElementById('rating-
default'));

```

The above code example displays the following output.



### Set Precision

In a real-time movie **Rating** scenario, you can set precision in between two whole numbers, such as 2.5 or 3.7 and it is done using the property **precision** by changing the value to **ej.Rating.Precision.Half** or **ej.Rating.Precision.Exact**.

### HTML

```

<div id="rating-precision"></div>
<script src="app/rating/default.js">

```

### JS

```

var PrecisionRating = React.createClass({
  render: function () {
    return (
      <div id="rating_precision">
        <div class="frame">
          <table id="table">
            <tr>
              <td valign="top">
                Full Precision :
              </td>
              <td>
                <EJ.Rating value={4} ></EJ.Rating>
              </td>
            </tr>
            <tr>
              <td valign="top">
                Half Precision :
              </td>
              <td>
                <EJ.Rating value={3.5} precision="half" ></EJ.Rating>
              </td>
            </tr>
          </table>
        </div>
      </div>
    );
  }
});

```

```

</td>
</tr>
<tr>
<td valign="top">
Exact Precision :
</td>
<td>
<EJ.Rating value={3.7} precision="exact" ></EJ.Rating>
</td>
</tr>
</table>
</div>
</div>
);
}
});
ReactDOM.render(<PrecisionRating />, document.getElementById('rating-precision'));

```

The above code example displays the following output.



You can also add additional functionalities such as orientation, event tracer and API's to the **Rating**.

## ReportDesigner

### Overview

The **Report Designer** control for Javascript helps to create and edit reports in open Report Definition Language (RDL) specification by products like Microsoft SQL Server Reporting Services. It comes with a wide range of report items to transform data into meaningful information and quickly build the reports with the help of following features.

#### Key features

**Data Sources** --- Supports connection to major data providers such as **Microsoft SQL Server, Oracle, OLEDB** and **ODBC** for exploring data and design reports with a wide range of data sources.

**User friendly Environment** --- Provides an effective design area, configuration options, and drag-and-drop facilities to make it easy for business users to compose reports.

**Report items** --- All interactive report items that are commonly used in business reports is built-in, including charts, grids, pivot grids, subreports, textboxes, images, lines, and rectangles for better visual representation of data.

**Report Parameter** --- Supports parameter to specify the data to filter in a report, connect related reports together and vary report presentation.

**Expression** --- Expressions are used throughout the report definition in parameters, queries, filters and report item properties to perform additional operations such as mathematical computation, conditional formatting, inspection, conversions, and more.

### Getting Started

This section explains briefly about how to create a ReportDesigner control in your application with **ReactJS**.

## Script and CSS Reference

- [jQuery](#) 1.10.2 and later versions
- [jsRender](#) - to render the templates

The required ReactJS script dependencies as follows. And you can also refer [React](#) to know more about ReactJS.

- react.min.js - <http://cdn.syncfusion.com/js/assets/external/react.min.js>
- react-dom.min.js - <http://cdn.syncfusion.com/js/assets/external/react-dom.min.js>
- ej.web.react.min.js - <http://cdn.syncfusion.com/{{ site.releaseversion }}/js/common/ej.web.react.min.js>

## External Dependency - Code Mirror

In report designer to edit the SQL queries with syntax highlighter need to refer the below code mirror scripts and themes.

### HTML

```
<link
href="https://cdnjs.cloudflare.com/ajax/libs/codemirror/5.37.0/codemirror.min.css" rel="stylesheet" />
<link
href="https://cdnjs.cloudflare.com/ajax/libs/codemirror/5.37.0/addon/hint/show-hint.min.css" rel="stylesheet" />
<script
src="https://cdnjs.cloudflare.com/ajax/libs/codemirror/5.37.0/codemirror.min.js" type="text/javascript"></script>
<script
src="https://cdnjs.cloudflare.com/ajax/libs/codemirror/5.37.0/addon/hint/show-hint.min.js" type="text/javascript"></script>
<script
src="https://cdnjs.cloudflare.com/ajax/libs/codemirror/5.37.0/addon/hint/sql-hint.min.js" type="text/javascript"></script>
<script
src="https://cdnjs.cloudflare.com/ajax/libs/codemirror/5.37.0/mode/sql/sql.min.js" type="text/javascript"></script>
```

Create a **HTML** page and add the below script and CSS references in the <head> tag of the html page.

### HTML

```
<!DOCTYPE html>
<html>
<head>
<!-- theme reference -->
<link rel="stylesheet"
href="http://cdn.syncfusion.com/{{site.releaseversion}}/js/web/bootstrap-theme/ej.web.all.min.css" />
<link rel="stylesheet"
href="http://cdn.syncfusion.com/{{site.releaseversion}}/js/web/bootstrap-theme/ej.reportdesigner.min.css" />
<!-- code mirror theme -->
```

```

<link
href="https://cdnjs.cloudflare.com/ajax/libs/codemirror/5.37.0/codemirror.min.css" rel="stylesheet" />
<link
href="https://cdnjs.cloudflare.com/ajax/libs/codemirror/5.37.0/addon/hint/show-hint.min.css" rel="stylesheet" />
<!-- react script -->
<script
src="http://cdn.syncfusion.com/js/assets/external/react.min.js"></script>
<script src="http://cdn.syncfusion.com/js/assets/external/react-dom.min.js"></script>
<script src="https://cdnjs.cloudflare.com/ajax/libs/babel-core/5.8.34/browser.min.js"></script>
<!-- jquery script -->
<script src="http://cdn.syncfusion.com/js/assets/external/jquery-3.0.0.min.js"></script>
<script
src="http://cdn.syncfusion.com/js/assets/external/jsrender.min.js"></script>
<!-- code mirror script -->
<script
src="https://cdnjs.cloudflare.com/ajax/libs/codemirror/5.37.0/codemirror.min.js" type="text/javascript"></script>
<script
src="https://cdnjs.cloudflare.com/ajax/libs/codemirror/5.37.0/addon/hint/show-hint.min.js" type="text/javascript"></script>
<script
src="https://cdnjs.cloudflare.com/ajax/libs/codemirror/5.37.0/addon/hint/sql-hint.min.js" type="text/javascript"></script>
<script
src="https://cdnjs.cloudflare.com/ajax/libs/codemirror/5.37.0/mode/sql/sql.min.js" type="text/javascript"></script>
<!-- Essential JS UI widget -->
<script
src="http://cdn.syncfusion.com/{{site.releaseversion}}/js/web/ej.web.all.min.js"></script>
<script
src="http://cdn.syncfusion.com/{{site.releaseversion}}/js/web/ej.reportdesigner.min.js"></script>
<script
src="http://cdn.syncfusion.com/{{site.releaseversion}}/js/common/ej.web.react.min.js"></script>
<!--Add custom scripts here -->
</head>
<body>
</body>
</html>

```

**Note:** In the above code, `ej.web.all.min.js` script reference has been added for demonstration purpose. It is not recommended to use this for deployment purpose, as its file size is larger since it contains all the widgets. Instead, you can use [CSG](#) utility to generate a custom script file with the required widgets for deployment purpose.

#### Initialize and configure the control

Control can be initialized in two ways.

- Using jsx Template
- Without using jsx Template

### Using jsx Template

While making use of jsx template, we have to create both the html and jsx files. The .jsx file should be converted into .js file using [gulp](#) command and then needs to be added as a reference in the html page.

Add a `div` container to render the ReportDesigner in **HTML** page.

#### HTML

```
<!DOCTYPE html>
<html>
<body>
<div id="reportDesigner" style="height: 590px;width: 100%;"></div>
<script src="scripts/default.js"></script>
</body>
</html>
```

Initialize the ReportDesigner by using the `EjReportDesigner` tag.

#### HTML

```
"use strict";
ReactDOM.render(
  <EJ.ReportDesigner
    id = "designerId"
    serviceUrl = {'http://js.syncfusion.com/ejservices/api/ReportDesigner'}
  </EJ.ReportDesigner>,
  document.getElementById('reportDesigner')
);
```

**Note:** The above jsx template needs to be converted from .jsx to .js extension by using [gulp](#) nuget package (refer [here](#)) and then it must be referred in the html page.

### Without using jsx Template

ReportDesigner can be created from a HTML `DIV` element with the HTML `id` attribute set to it. Refer to the following code example.

#### HTML

```
<body>
<div id="reportdesigner" style="height: 590px;width: 100%;"></div>
</body>
```

Initialize the ReportDesigner control by adding the following script code to the body section of the HTML document.

#### HTML

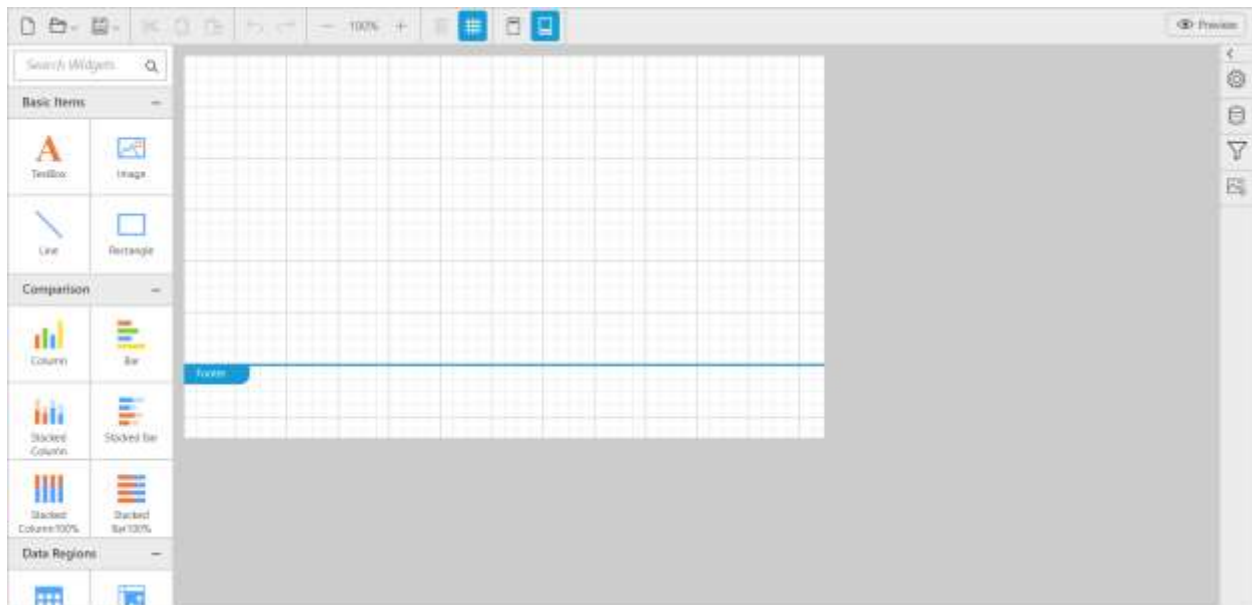
```
<script type="text/javascript">
ReactDOM.render(
  React.createElement(EJ.ReportDesigner, {
    id: "designerId",
    serviceUrl: 'http://js.syncfusion.com/demos/ejservices/api/ReportDesigner'
```

```
}},  
document.getElementById('reportdesigner')  
);  
</script>
```

**Note:** In the report designer service url, need to mention the controller name of the reporting service. To create reporting service for report designer follow the steps explained in the following link [Reporting Service](#).

### Run the Application

Run the application and you can see the ReportDesigner on the page as displayed in the following screenshot.



## ReportViewer

### Overview

The ReportViewer is a visualization control to view Microsoft SSRS RDL/RDLC files on a web page and it is powered by HTML5/JavaScript. It has support to bind DataSources/Parameters to the Reports and also supports exporting, paging, zooming and printing the report.

### Key Features

The ReportViewer supports the following features.

- DataSources (SQL Server, Oracle, MS Azure, XML, Business Object and SQL Server Compact Data Sources)
- Filters and Parameters
- Built-in Fields and Expressions
- Grouping and Sorting
- Report Items
- Table
- Matrix
- List

- Chart
- Sparkline
- DataBar
- Map
- Gauge
- Indicator
- Image
- Textbox
- Line
- Rectangle
- Subreport
- Actions: Drilldown, Drill through, Toggle and Document Map
- Printing, Exporting, Paging, FitToPage and Zooming
- Report Processing Events
- KnockOut/AngularJS Support
- Toolbar Customization

## Getting Started

This section explains briefly about how to create a ReportViewer in React JS.

### Script and CSS Reference

Create a **HTML** page and add the script and CSS references in the order mentioned in the following code example.

#### HTML

```
<!DOCTYPE html>
<html>
<head>
  <!-- Essential Studio for JavaScript theme reference -->
  <link rel="stylesheet"
href="http://cdn.syncfusion.com/14.3.0.49/js/web/bootstrap-
theme/ej.web.all.min.css" />
  <!-- react script -->
  <script
src="https://cdnjs.cloudflare.com/ajax/libs/react/15.2.1/react.js"></script>
  <script src="https://cdnjs.cloudflare.com/ajax/libs/react/15.2.1/react-
dom.js"></script>
  <script src="https://cdnjs.cloudflare.com/ajax/libs/babel-
core/5.8.34/browser.min.js"></script>
  <!-- jquery script -->
  <script src="https://code.jquery.com/jquery-3.0.0.min.js"></script>
  <!-- Essential JS UI widget -->
  <script
src="http://cdn.syncfusion.com/14.3.0.49/js/web/ej.web.all.min.js"></script>
  <script
src="http://cdn.syncfusion.com/14.3.0.49/js/common/ej.web.react.min.js"></sc
ript>
  <!--Add custom scripts here -->
</head>
<body>
</body>
</html>
```



**Note:** In the above code, `ej.web.all.min.js` script reference has been added for demonstration purpose. It is not recommended to use this for deployment purpose, as its file size is larger since it contains all the widgets. Instead, you can use [CSG](#) utility to generate a custom script file with the required widgets for deployment purpose.

- `react.js` and `react-dom.js` are the core files needed to create react elements.
- `browser.min.js` file is required for code transform.
- `ej.web.react.min.js` is a react-syncfusion bridge to render Syncfusion components.

### Initialize and configure the control

Control can be initialized in two ways.

- Using jsx Template
- Without using jsx Template

#### Using jsx Template

While making use of jsx template, we have to create both the html and jsx files. The `.jsx` file should be converted into `.js` file using [gulp](#) command and then needs to be added as a reference in the html page.

#### Control Initialization

Add a `div` container to render the ReportViewer.

#### HTML

```
<!DOCTYPE html>
<html>
<body>
<div id="groupingAggregate"></div>
<script src="scripts/default.js"></script>
</body>
</html>
```

Initialize the ReportViewer by using the `EjReportViewer` tag.

#### HTML

```
"use strict";
ReactDOM.render(
  <EJ.ReportViewer
    id = "groupingReportViewer"
    reportServiceUrl = {'http://js.syncfusion.com/ejservices/api/ReportViewer'}
    processingMode = {"remote"}
    reportPath = {'GroupingAgg.rdl'}>
  </EJ.ReportViewer>,
  document.getElementById('groupingAggregate')
);
```

**Note:** Default RDL Report will be rendered, which is used in the online service. You can obtain sample rdl/rdlc files from Syncfusion installed location

```
(%userprofile%\AppData\Local\Syncfusion\EssentialStudio\{{ site.releaseversion
}}\Common\Data\ejReportTemplate).
```

**Note:** The above jsx template needs to be converted from .jsx to .js extension by using `gulp` nuget package (refer [here](#)) and then it must be referred in the html page.

#### Run the Application

Run the sample application and you can see the ReportViewer on the page as displayed in the following screenshot.

Product Category	Product Sub Category	Quarterly Month	Sales
Bikes	Road Bikes	Q1	\$3,171,787.61
Accessories	Helmets	Q1	\$4,945.69
Clothing	Jerseys	Q1	\$9,517.33
<b>Q1 Sales</b>			<b>\$3,186,250.64</b>
Components	Road Frames	Q2	\$155,311.41
Clothing	Socks	Q2	\$1,899.62
Bikes	Mountain Bikes	Q2	\$2,416,836.61
<b>Q2 Sales</b>			<b>\$2,574,047.64</b>
Components	Forks	Q3	\$26,166.78
Clothing	Tights	Q3	\$67,088.30
Accessories	Locks	Q3	\$6,325.00
Components	Road Frames	Q3	\$957,715.19
Components	Wheels	Q3	\$288,627.83
<b>Q3 Sales</b>			<b>\$1,345,923.11</b>
Clothing	Bib-Shorts	Q4	\$35,322.87

#### ReportViewer with Grouping Aggregate Report

##### Using without jsx Template

ReportViewer can be created from a HTML `DIV` element with the HTML `id` attribute set to it. Refer to the following code example.

##### HTML

```
<body>
<div id="groupingaggregate"></div>
</body>
```

Initialize the ReportViewer control by adding the following script code to the body section of the HTML document.

##### HTML

```
<div id="groupingaggregate"></div>
<script type="text/javascript">
ReactDOM.render (
```

```

React.createElement(EJ.ReportViewer, {
  id: "groupingReportViewer",
  reportServiceUrl: 'http://js.syncfusion.com/ejservices/api/ReportViewer',
  processingMode: "remote",
  reportPath: 'GroupingAgg.rdl'
}),
document.getElementById('groupingaggregate')
);
</script>

```

Run the application and you can see the ReportViewer on the page as displayed in the following screenshot.

Product Category	Product Sub Category	Quarterly Month	Sales
Bikes	Road Bikes	Q1	\$3,171,787.61
Accessories	Helmets	Q1	\$4,945.69
Clothing	Jerseys	Q1	\$9,517.33
<b>Q1 Sales</b>			<b>\$3,186,250.64</b>
Components	Road Frames	Q2	\$155,311.41
Clothing	Socks	Q2	\$1,899.62
Bikes	Mountain Bikes	Q2	\$2,416,836.61
<b>Q2 Sales</b>			<b>\$2,574,047.64</b>
Components	Forks	Q3	\$26,166.78
Clothing	Tights	Q3	\$67,088.30
Accessories	Locks	Q3	\$6,325.00
Components	Road Frames	Q3	\$957,715.19
Components	Wheels	Q3	\$288,627.83
<b>Q3 Sales</b>			<b>\$1,345,923.11</b>
Clothing	Bib-Shorts	Q4	\$35,322.87

### ReportViewer with Grouping Aggregate Report

#### Load SSRS Server Reports

##### Using jsx Template

ReportViewer supports to load RDL/RDLC files from SSRS Server. The following steps help you to load reports from SSRS Server.

Set the `reportPath` from SSRS and SSRS `reportServerUrl` in the ReportViewer properties.

#### HTML

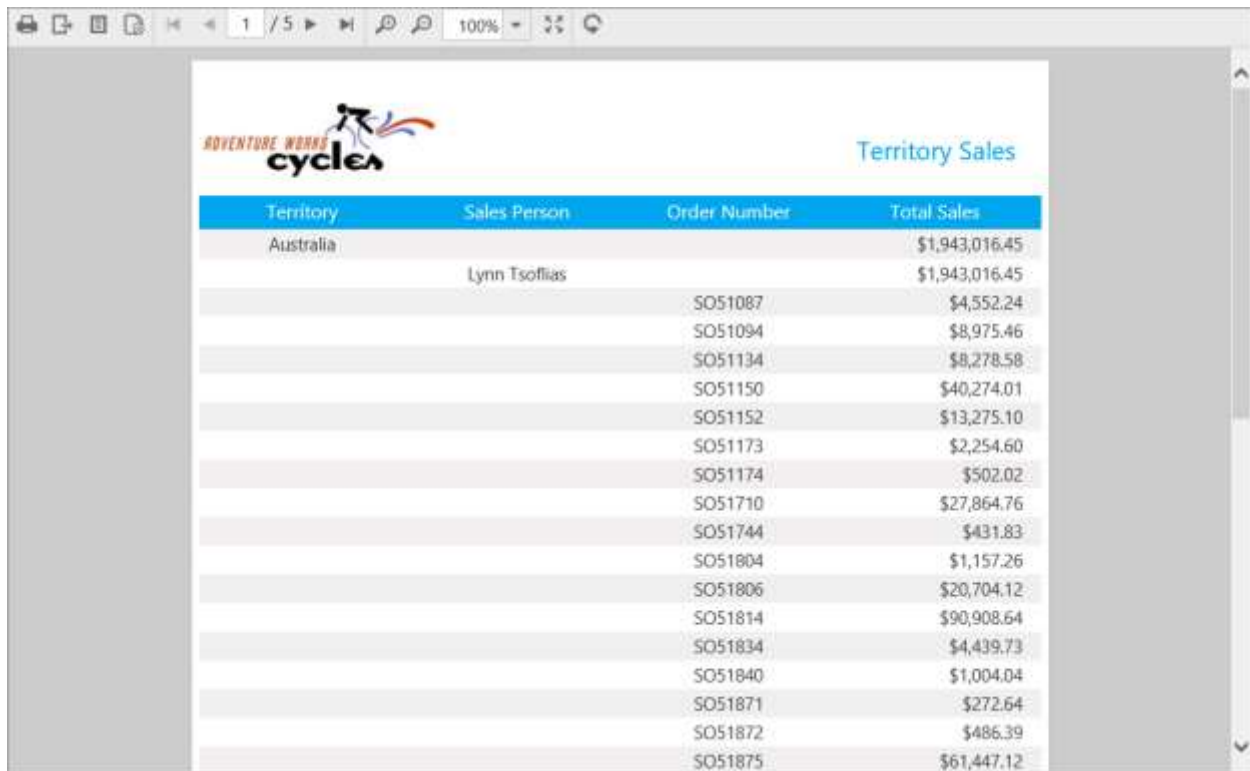
```

"use strict";
ReactDOM.render(
  <ej.reportviewer id="territoryReportViewer"
    reportserviceurl={ 'http://js.syncfusion.com/ejservices/api/ReportViewer' }
    reportserverurl={ 'http://mvc.syncfusion.com/reportserver' }

```

```
processingmode={"remote"}
reportpath={"/SSRSSamples2/Territory sales new"}>
</ej.reportviewer>,
document.getElementById('territorysales')
);
```

Run the application and you can see the ReportViewer on the page as displayed in the following screenshot.



Territory	Sales Person	Order Number	Total Sales
Australia	Lynn Tsofilias		\$1,943,016.45
		SO51087	\$4,552.24
		SO51094	\$8,975.46
		SO51134	\$8,278.58
		SO51150	\$40,274.01
		SO51152	\$13,275.10
		SO51173	\$2,254.60
		SO51174	\$502.02
		SO51710	\$27,864.76
		SO51744	\$431.83
		SO51804	\$1,157.26
		SO51806	\$20,704.12
		SO51814	\$90,908.64
		SO51834	\$4,439.73
		SO51840	\$1,004.04
		SO51871	\$272.64
		SO51872	\$486.39
		SO51875	\$61,447.12

## Report from SSRS

### Using without jsx Template

ReportViewer can be created from a HTML **DIV** element with the HTML **id** attribute set to it. Refer to the following code example.

### HTML

```
<body>
<div id="territorysales"></div>
</body>
```

Initialize the ReportViewer control by adding the following script code to the body section of the HTML document.

### HTML

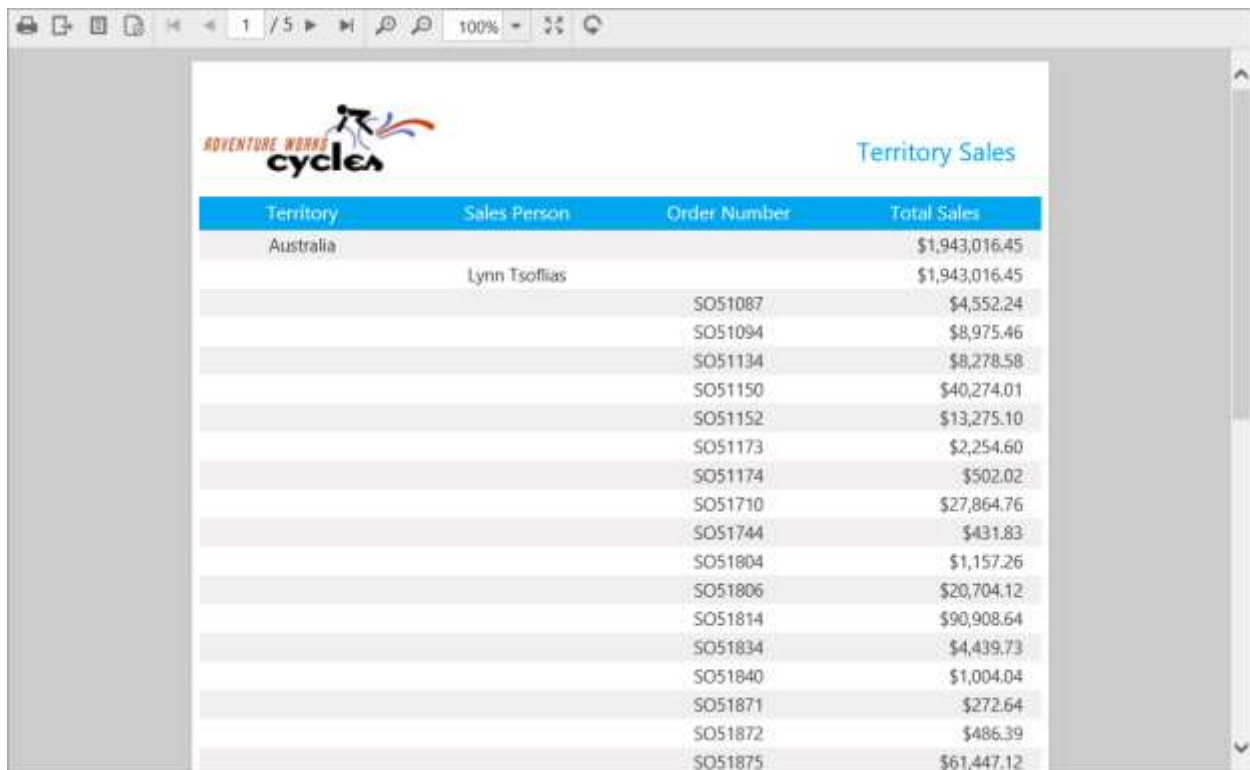
```
<div id="groupingaggregate"></div>
<script type="text/javascript">
ReactDOM.render(
React.createElement(EJ.ReportViewer, {
```

```

id: "territoryReportViewer",
reportServiceUrl: 'http://js.syncfusion.com/ejservices/api/ReportViewer',
reportServerUrl: 'http://mvc.syncfusion.com/reportserver',
processingMode: "remote",
reportPath: '/SSRSSamples2/Territory Sales new'
},
),
document.getElementById('territorysales')
);
</script>

```

Run the application and you can see the ReportViewer on the page as displayed in the following screenshot.



Territory	Sales Person	Order Number	Total Sales
Australia	Lynn Tsofilias		\$1,943,016.45
			\$1,943,016.45
		SO51087	\$4,552.24
		SO51094	\$8,975.46
		SO51134	\$8,278.58
		SO51150	\$40,274.01
		SO51152	\$13,275.10
		SO51173	\$2,254.60
		SO51174	\$502.02
		SO51710	\$27,864.76
		SO51744	\$431.83
		SO51804	\$1,157.26
		SO51806	\$20,704.12
		SO51814	\$90,908.64
		SO51834	\$4,439.73
		SO51840	\$1,004.04
		SO51871	\$272.64
		SO51872	\$486.39
		SO51875	\$61,447.12

## Report from SSRS

### Load RDLC Reports

#### Using jsx Template

The ReportViewer has data binding support to visualize the RDLC reports. The following code example helps you to bind data to ReportViewer.

Assign the RDLC report path to ReportViewer's `reportPath` property and set the data sources to the ReportViewer's `dataSources` property and specify the `processingMode` as local.

### HTML

```

"use strict";
ReactDOM.render(
  <ej.reportviewer id="areaReportViewer"
    reportserviceurl={ 'http://js.syncfusion.com/ejservices/api/ReportViewer' }

```

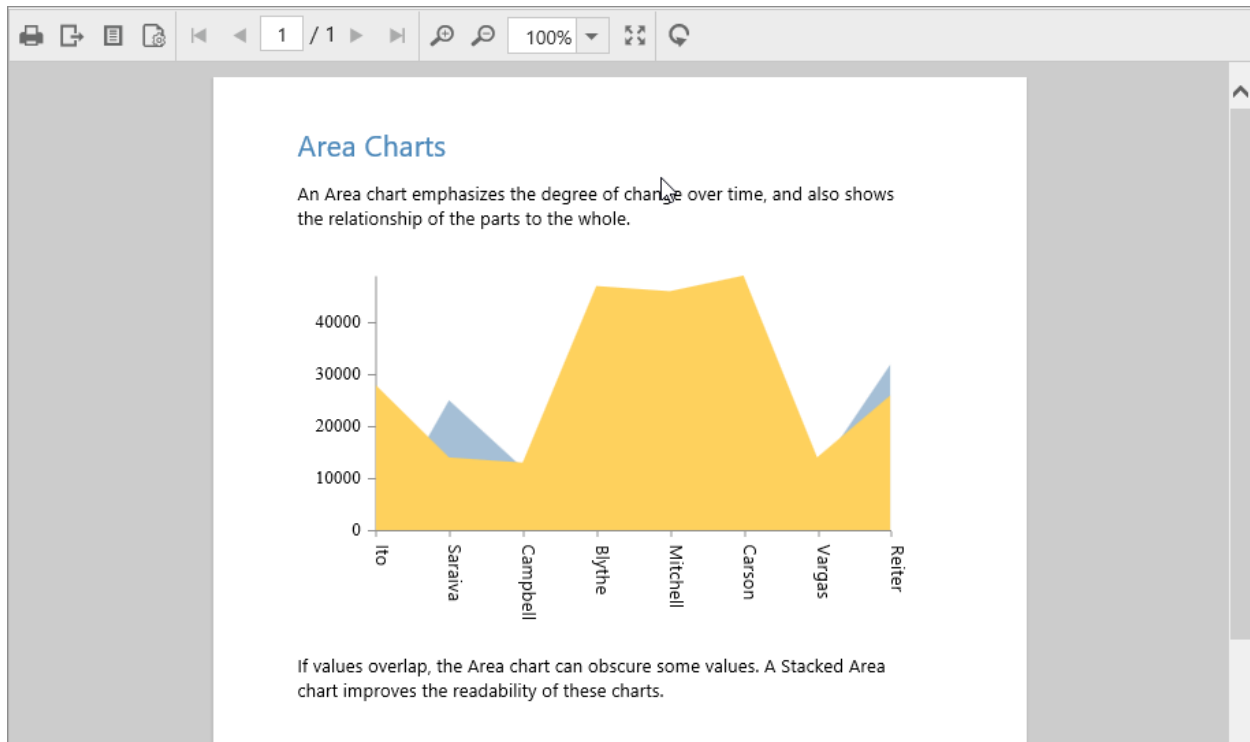
```

processingmode={"local"}
reportpath={'AreaCharts.rdlc'}
datasources={[{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 },
{ salespersonid 275, fullname 'Blythe', title 'Sales Representative', salesterritory 'North East', y2002 19000, y2003 47000, y2004 4557045 },
{ salespersonid 276, fullname 'Mitchell', title 'Sales Representative', salesterritory 'South West', y2002 28000, y2003 46000, y2004 5240075 },
{ salespersonid 277, fullname 'Carson', title 'Sales Representative', salesterritory 'Central', y2002 33000, y2003 49000, y2004 3857163 },
{ salespersonid 278, fullname 'Vargas', title 'Sales Representative', salesterritory 'Canada', y2002 11000, y2003 14000, y2004 1764938 },
{ salespersonid 279, fullname 'Reiter', title 'Sales Representative', salesterritory 'South East', y2002 32000, y2003 26000, y2004 2811012 }]],
name 'AdventureWorksXMLDataSet' }]]>
</ej.reportviewer>,
document.getElementById('areachart')
);

```

Default RDLC Report will be rendered, which is used in the online service.

Run the application and you can see the ReportViewer on the page as displayed in the following screenshot.



Area Chart RDLC Report

*Using without jsx Template*

ReportViewer can be created from a HTML **DIV** element with the HTML **id** attribute set to it. Refer to the following code example.

**HTML**

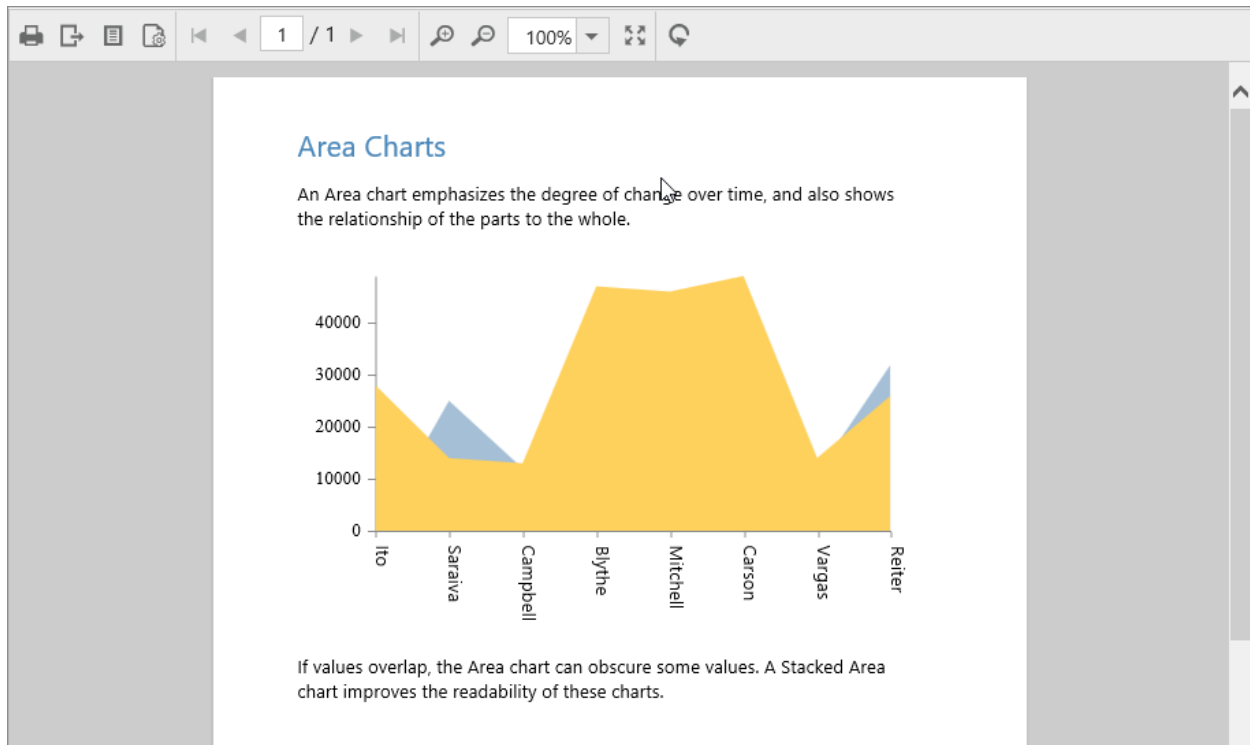
```
<body>
<div id="areachart"></div>
</body>
```

Initialize the ReportViewer control by adding the following script code to the body section of the HTML document.

**HTML**

```
<div id="areachart"></div>
<script type="text/javascript">
ReactDOM.render(
  React.createElement(EJ.ReportViewer, {
    id: "areaReportViewer",
    reportServiceUrl: 'http://js.syncfusion.com/ejservices/api/ReportViewer',
    processingMode: "local",
    reportPath: 'AreaCharts.rdlc',
    dataSources: [{
      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 },
        { SalesPersonID: 275, FullName: 'Blythe', Title: 'Sales Representative', SalesTerritory: 'North East', Y2002: 19000, Y2003: 47000, Y2004: 4557045 },
        { SalesPersonID: 276, FullName: 'Mitchell', Title: 'Sales Representative', SalesTerritory: 'South West', Y2002: 28000, Y2003: 46000, Y2004: 5240075 },
        { SalesPersonID: 277, FullName: 'Carson', Title: 'Sales Representative', SalesTerritory: 'Central', Y2002: 33000, Y2003: 49000, Y2004: 3857163 },
        { SalesPersonID: 278, FullName: 'Vargas', Title: 'Sales Representative', SalesTerritory: 'Canada', Y2002: 11000, Y2003: 14000, Y2004: 1764938 },
        { SalesPersonID: 279, FullName: 'Reiter', Title: 'Sales Representative', SalesTerritory: 'South East', Y2002: 32000, Y2003: 26000, Y2004: 2811012 }],
      name: 'AdventureWorksXMLDataSet'
    }]
  ),
  document.getElementById('areachart')
);
</script>
```

Run the application and you can see the ReportViewer on the page as displayed in the following screenshot.



### Area Chart RDLC Report

#### How To

##### Load unsupported fonts

In RDL, the defined fonts are not supported in cross-platform browsers. The unsupported fonts can be loaded through the **font-face** css rule in the style section of the application.

##### Using @font-face in Style Section

- Add the <style> tag to head section of the HTML page.
- Create the CSS rule **font-face** and then add the **font-family** and **src** resources as mentioned below.
- **font-family** -- Specifies a name that will be used as a font face value for font properties.
- **src** -- Specifies the resource containing the font data. This can be a URL to a font file location.
- The following code snippet describes the above steps.

#### HTML

```
<style>
@font-face {
font-family: Segoe UI;
src: url(segoue_ui.ttf);
}
</style>
```



## Ribbon

### Overview

The **Ribbon** control for JavaScript provides with rich customizable user interfaces like Office 2010, SharePoint 2010, and Office Web Apps 2010. The Ribbon Tab appears across the top of the page. Each Tab organizes a set of groups that has labels to identify them and also contains a set of controls and group expander.

### Key Features

- **Application Tab** – Represents the menu commands which can be used to do any operations, like file related commands. Supports Application Menu and Backstage.
- **Tabs** – Related groups are combined into Tabs for quick and easy access.
- **Contextual Tabs** - Allows to group number of Tabs based on some criteria.
- **Controls' support** - Supports Button, Split button and Dropdown List, Toggle button, and custom controls.
- **Gallery** - Convenient way to visually display related options with text/images.
- **Resize** – Provides resizing to group controls dynamically based on window size.
- **Expand and Collapse** - Ribbon can be expanded and collapsed using expand/collapse button.
- **Screen-Tip** - Supports HTML and enhanced custom tooltips for controls of Ribbon groups.
- **Theme** - Essential JavaScript controls consist of 12 built-in themes (6 – flat and 6 – gradient effects). It also supports bootstrap theme.

### Getting Started

This section explains briefly how to create a **Ribbon**.

### Script & CSS Reference

The Ribbon control has the following list of external JavaScript dependencies.

- [jQuery](#) 1.7.1 and later versions

The required ReactJS script dependencies as follows. And you can also refer [React](#) to know more about react.js.

- **react.min.js** - <http://cdn.syncfusion.com/js/assets/external/react.min.js>
- **react-dom.min.js** - <http://cdn.syncfusion.com/js/assets/external/react-dom.min.js>
- **ej.web.react.min.js** - <http://cdn.syncfusion.com/{{ site.releaseversion }}/js/common/ej.web.react.min.js>

To translate JSX to plain JavaScript, we must use `<script type="text/babel">` and refer the browser.min.js file to perform the transformation in the browser.

To get started, you can use the **ej.web.all.min.js** file that encapsulates all the **ej** controls and frameworks in one single file. So the complete boilerplate code is

### HTML

```
<!DOCTYPE html>
<html>
<head>
```

```

<meta name="viewport" content="width=device-width, initial-scale=1.0">
<meta name="description" content="Essential Studio for JavaScript">
<meta name="author" content="Syncfusion">
<title>Getting Started for Ribbon React JS</title>
<!-- Essential Studio for JavaScript theme reference -->
<link href="http://cdn.syncfusion.com/{{ site.releaseversion }}/js/web/flat-
azure/ej.web.all.min.css" rel="stylesheet" />
<!-- Essential Studio for JavaScript script references -->
<script
src="http://cdn.syncfusion.com/js/assets/external/react.min.js"></script>
<script src="http://cdn.syncfusion.com/js/assets/external/react-
dom.min.js"></script>
<script src="https://cdnjs.cloudflare.com/ajax/libs/babel-
core/5.8.34/browser.min.js"></script>
<script src="http://cdn.syncfusion.com/js/assets/external/jquery-
3.0.0.min.js"></script>
<script src="http://cdn.syncfusion.com/{{ site.releaseversion
}}/js/web/ej.web.all.min.js"></script>
<script src="http://cdn.syncfusion.com/{{ site.releaseversion
}}/js/common/ej.web.react.min.js"></script>
<!-- Add your custom scripts here -->
</head>
<body>
</body>
</html>

```

**Note:** 1. In production, we highly recommend you to use our [custom script generator](#) to create custom script file with required controls and its dependencies only. Also to reduce the file size further please use [GZip compression](#) in your server.

2. For themes, you can use the `ej.web.all.min.css` CDN link from the code snippet given. To add the themes in your application, please refer to [this link](#).

**Note:** Ribbon's sample level icons can be loaded using `ej.icons.CSS` from the location `"/Content/ej/themes/ribbon-css"`.

### Control Initialization

Control can be initialized in two ways.

- Using jsx Template
- Without using jsx Template

### Using jsx Template

By using the jsx template, we can create the html file and jsx file. The `.jsx` file can be convert to `.js` file and it can be referred in html page.

Please refer to the code of HTML file.

### HTML

```

<div id="ribbon-default"></div>
<script src="app/ribbon/default.js"></script>
<ul id="ribbonmenu1">
<li><a>FILE</a>
</li>
</ul>

```

```

<li><a>New</a></li>
<li><a>Open</a></li>
<li><a>Save</a></li>
<li><a>Save As</a></li>
<li><a>Print</a></li>
</ul>
</li>
</ul>

```

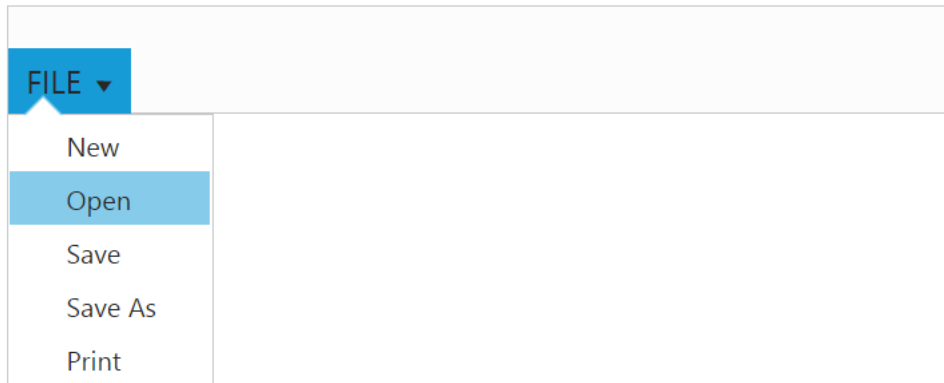
Ribbon control can be initialized with the following in HTML document.

### HTML

```

ReactDOM.render(
  <EJ.Ribbon width="50%" applicationTab-type="menu" applicationTab-
  menuItemID="ribbonmenu1">
    </EJ.Ribbon>,
  document.getElementById('ribbon-default')
);

```



**Note:** Set the required width to Ribbon, else default parent container or window width will be considered.

### Adding Tabs

Tab is a set of related groups which are combined into single item. For creating tab, id and text properties should be specified.

### HTML

```

<div id="ribbon-default"></div>
<script src="app/ribbon/default.js"></script>
<ul id="ribbonmenu1">
  <li><a>FILE</a>
  <ul>
    <li><a>New</a></li>
    <li><a>Open</a></li>
    <li><a>Save</a></li>
    <li><a>Save As</a></li>
    <li><a>Print</a></li>
  </ul>
</li>
</ul>

```

Configure the tabs bind value tab in ReactJS view-model as shown in the following code.

### HTML

```
ReactDOM.render (
  <EJ.Ribbon width="50%" applicationtab-type="menu" applicationtab-
  menuitemid="ribbonmenu1">
    <tabs>
      <tab id="home" text="HOME"></tab>
    </tabs>
  </EJ.Ribbon>
  document.getElementById('ribbon-default')
);
```



### Configuring Groups

List of controls are combined as logical groups into tab. Group alignment type as row/column, Default is row.

Create group item with text specified and add content group to Groups collection with ejButton control settings.

Configure the tabs bind value tab with group and button named as New.

### HTML

```
<div id="ribbon-default"></div>
<script src="app/ribbon/default.js"></script>
<ul id="ribbonmenu1">
  <li><a>FILE</a>
  <ul>
    <li><a>New</a></li>
    <li><a>Open</a></li>
    <li><a>Save</a></li>
    <li><a>Save As</a></li>
    <li><a>Print</a></li>
  </ul>
</li>
</ul>
```

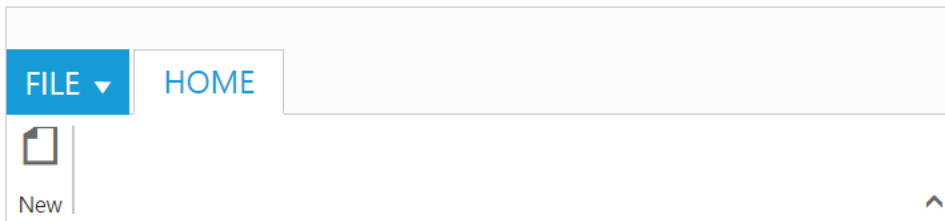
### HTML

```
ReactDOM.render (
  <EJ.Ribbon width="50%" applicationtab-type="menu" applicationtab-
  menuitemid="ribbonmenu1">
    <tabs>
      <tab id="home" text="HOME">
        <groups>
          <group text="New" alignType="rows">
```

```

<content>
<content defaults-type="button">
<groups>
<group id="new" text="New" buttonSettings-contentType="imageonly"
buttonSettings-prefixIcon="e-icon e-ribbon e-new" >
</group>
</groups>
</content>
</content>
</group>
</groups>
</tab>
</tabs>
</EJ.Ribbon>
document.getElementById('ribbon-default')
);

```



### Adding Controls to Group

Syncfusion JavaScript Controls can be added to group's content with corresponding type specified like button, split button, toggle button, dropdown list, gallery, custom, etc. Default type is button.

Configure the tabs bind value tab with groups, button, split button and dropdown controls. Also the datasource to dropdown control is configured with bind name fontFamily. Please refer to the following code snippets.

### HTML

```

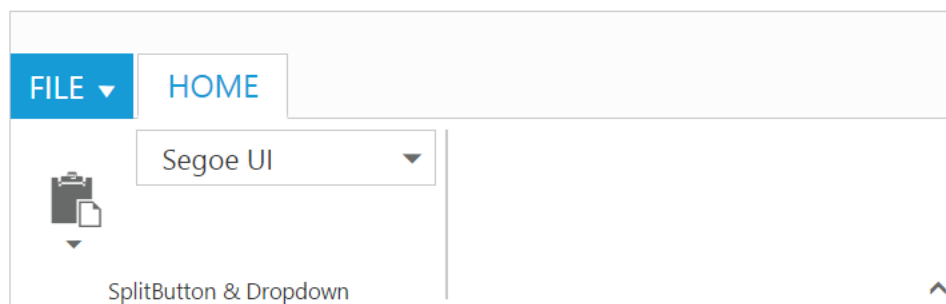
var fontFamily = ["Segoe UI", "Arial", "Times New Roman", "Tahoma",
"Helvetica"];
ReactDOM.render(
<div>
<ul id="ribbonmenu3">
<li><a>FILE</a>
<ul>
<li><a>New</a></li>
<li><a>Open</a></li>
<li><a>Save</a></li>
<li><a>Save As</a></li>
<li><a>Print</a></li>
</ul>
</li>
</ul>
<ul id="pasteSplit3">
<li><a>Paste</a></li>
</ul>
<EJ.Ribbon width="100%" applicationtab-type="menu" applicationtab-
menuitemid="ribbonmenu3">

```

```

<tabs>
<tab id="home" text="HOME">
<groups>
<group text="SplitButton & Dropdown" alignType="columns">
<content>
<content defaults-type="splitbutton" defaults-width={60} defaults-
height={70}>
<groups>
<group id="paste" text="Paste" customTooltip-prefixIcon="e-pastetip"
splitButtonSettings-contentType="imageonly" splitButtonSettings-
prefixIcon="e-icon e-ribbon e-ribbonpaste" splitButtonSettings-
targetID="pasteSplit3" splitButtonSettings-buttonMode="dropdown"
splitButtonSettings-arrowPosition="bottom" splitButtonSettings-
click="executeAction">
</group>
</groups>
</content>
<content defaults-type="dropdownlist" defaults-height={28}>
<groups>
<group id="fontFamily" dropdownSettings-dataSource={fontFamily}
dropdownSettings-text="Segoe UI" dropdownSettings-select="executeAction"
dropdownSettings-width={150}>
</group>
</groups>
</content>
</content>
</group>
</groups>
</tab>
</tabs>
</EJ.Ribbon>
</div>
document.getElementById('ribbon-default')
);

```



### Without using jsx Template

The Ribbon can be created from a HTML **DIV** element with the HTML **id** attribute set to it. Refer to the following code example.

### HTML

```

<body>
<div id="ribbon-default"></div>
<ul id="ribbonmenu3">

```

```

<li><a>FILE</a>
<ul>
<li><a>New</a></li>
<li><a>Open</a></li>
<li><a>Save</a></li>
<li><a>Save As</a></li>
<li><a>Print</a></li>
</ul>
</li>
</ul>
<ul id="pasteSplit3">
<li><a>Paste</a></li>
</ul>
</body>

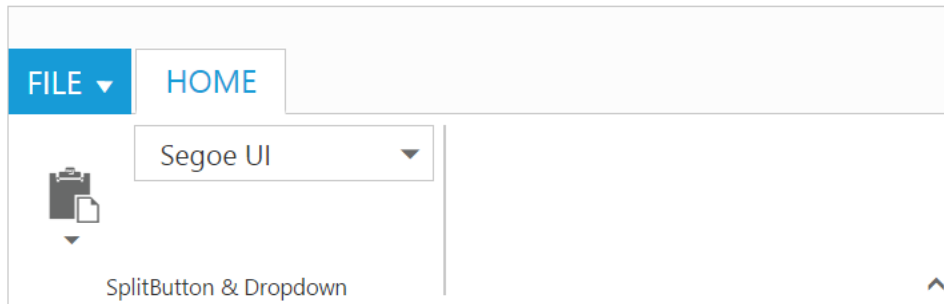
```

## HTML

```

var fontFamily = ["Segoe UI", "Arial", "Times New Roman", "Tahoma",
"Helvetica"];
React.createElement(EJ.Ribbon, {width: "100%", allowResizing: true,
"applicationTab-type": "menu", "applicationTab-menuItemID": "ribbonmenu3"},
React.createElement("tabs", null,
React.createElement("tab", {id: "home", text: "HOME"},
React.createElement("groups", null,
React.createElement("group", {text: "SplitButton & Dropdown", alignType:
"columns"},
React.createElement("content", null,
React.createElement("content", {"defaults-type": "splitbutton", "defaults-
width": 60, "defaults-height": 70},
React.createElement("groups", null,
React.createElement("group", {id: "paste", text: "Paste", "customTooltip-
prefixIcon": "e-pastetip", "splitButtonSettings-contentType": "imageonly",
"splitButtonSettings-prefixIcon": "e-icon e-ribbon e-ribbonpaste",
"splitButtonSettings-targetID": "pasteSplit3", "splitButtonSettings-
buttonMode": "dropdown", "splitButtonSettings-arrowPosition": "bottom",
"splitButtonSettings-click": "executeAction"}
)
)
),
React.createElement("content", {"defaults-type": "dropdownlist", "defaults-
height": 28},
React.createElement("groups", null,
React.createElement("group", {id: "fontFamily", "dropdownSettings-
dataSource": fontFamily, "dropdownSettings-text": "Segoe UI",
"dropdownSettings-select": "executeAction", "dropdownSettings-width": 150})
)
)
)
)
)
)
),
document.getElementById('ribbon-default')
);

```



## Application Tab

The Application Tab is used to represent a **Menu** that do some operations, such as File menu to create, open, and print documents. Application Tab classified by **type** property with the following:

- menu
- backstage

## Application Menu

The Application Menu is similar to traditional file menu options and Syncfusion **ejMenu** control is used internally to render this. To show Application Menu in Ribbon, set the **type** as **menu** and **menuSettings** to customize properties of **ejMenu**.

### \_Create Using Template\_

Set the UL element **id** to **menuItemID** property to create Application Menu and it will acts as template to render menu.

### HTML

```
<div id="ribbon-resize"></div>
<script src="app/ribbon/resize.js"></script>
<ul id="ribbonmenu1">
<li><a>FILE</a>
<ul>
<li><a>New</a></li>
<li><a>Open</a></li>
<li><a>Save</a></li>
<li><a>Save As</a></li>
<li><a>Print</a></li>
</ul>
</li>
</ul>
```

### HTML

```
"use strict";
ReactDOM.render(
  <EJ.Ribbon width="500px" applicationTab-type="menu" applicationTab-
  menuItemID="ribbonmenu1" applicationTab-menuSettings-openOnClick={false}>
    <tabs>
      <tab id="home" text="HOME">
        <groups>
          <group text="New" alignType="rows">
```



```

</group>
</groups>
</tab>
</tabs>
</EJ.Ribbon>,
document.getElementById('ribbon-resize')
);

```

![[/js/Ribbon/Application-Tabimages/Application-Tabimg1.png)

### Backstage Page

The Backstage page is where documents and related data of those can be managed, such as Create, Save and other information.

The Backstage page has a feature to add custom Control in left side of the page which contains menu items and the right side contains corresponding user controls.

You can set Application Tab `type` as `backstage` and set `id` , `text` to backstage items. Backstage `pages` can be added with required `itemType` and `contentID` as template id to render template into Backstage.

Separator between Backstage items can be enabled by setting `enableSeparator` as true. Width of backstage side header can be customized using `headerWidth`, If not set based on content given width will be considered.

To render the Ribbon with the Backstage page, refer to the following code snippet.

### HTML

```

<div id="ribbon-default"></div>
<script src="app/ribbon/default.js"></script>
<style>
.e-blank{
background-image: url("content/ej/themes/common-images/ribbon/blank.png");
}
.e-accuser{
background-image: url("content/ej/themes/common-images/ribbon/User.jpg");
}
.e-infopageicon {
background-repeat: no-repeat;
height: 150px;
width: 198px;
}
.e-newpageicon {
background-repeat: no-repeat;
height: 42px;
width: 42px;
}
.e-bspagestyle {
line-height: 0;
font-size: 30px;
}
.e-ribbon .e-ribbonbackstagepage .e-bsnewbtnstyle {
color: #212121;
background: #fdfdfd;
margin: 20px;
}

```

```

</style>
<div id="newCon">
<table>
<tr>
<td>
<button id="btn1" class="e-bsnewbtnstyle">Blank WorkBook</button></td>
</tr>
</table>
</div>
<div id="accountCon">
<div class="e-userDiv">
<span>User Information</span>
<div>
<div class="e-accuser e-newpageicon"></div>
<div class="e-userCon">
<div>user</div>
<div>xy@syncfusion.com</div>
</div>
</div>
</div>
</div>
<a href="#">Sign out</a>
</div>

```

## HTML

```

"use strict";
var backstage = {
type: ej.Ribbon.ApplicationTabType.Backstage,
backstageSettings: {
text: "FILE", height: 350, width: 1000, headerWidth: 120, pages: [
{id:"new",text:"New",contentID:"newCon"},
{id:"close",text:"Close",enableSeparator:true,itemType:ej.Ribbon.ItemType.Button},
{id:"account",text:"Office Account",contentID:"accountCon"}
]
}
};
function createControl(args) {
$("#btn1").ejButton({
size: "large",
height:200,
width:225,
contentType: "textandimage",
imagePosition: "imagetop",
prefixIcon: "e-blank e-infopageicon"
});
};
ReactDOM.render(
<div>
<EJ.Ribbon width="100%" allowResizing={true} applicationTab={backstage}
create={createControl}>
<tabs>
<tab id="home" text="HOME">
<groups>
<group text="New" alignType="rows">
</group>

```

```

</groups>
</tab>
</tabs>
</EJ.Ribbon>
</div>,
document.getElementById('ribbon-default')
);

```



**Note:** Height & width of backstage can be set using `height` and `width`, if these are not set, Ribbon's height & width will be considered.

You can add/remove/update backstage item to the ribbon control by using [addBackStageItem](#), [removeBackStageItem](#) and [updateBackStageItem](#) methods. Also you can show/hide the backstage page in ribbon control by using [showBackstage](#) and [hideBackstage](#) [https://help.syncfusion.com/api/js/ejribbon#methods:hidebackstage methods.

## Tab

Tab is a collection of control groups which enables you to organize related commands into single view. Tabs can be added to Ribbon using `tabs` property. `id` & `text` properties are used to set unique ID and header text to Tab. The manipulation of given text tab in the ribbon control can be done by using [addTab](#), [removeTab](#), [hideTab](#), [showTab](#) methods and [tabAdd](#), [tabCreate](#), [tabRemove](#), [tabClick](#) and [tabSelect](#) events.

## HTML

```

<div id="ribbon-resize"></div>
<div id="sendReceive">
Send/Receive All Folders
</div>
<script src="app/ribbon/resize.js"></script>
<ul id="ribbonmenu1">
<li><a>FILE</a>
<ul>
<li><a>New</a></li>
<li><a>Open</a></li>
<li><a>Save</a></li>
<li><a>Save As</a></li>
<li><a>Print</a></li>
</ul>
</li>
</ul>

```

## HTML

```

ReactDOM.render(
<EJ.Ribbon width="50%" applicationTab-type="menu" applicationTab-
menuItemID="ribbonmenu1" applicationTab-menuSettings-openOnClick={false}>
<tabs>
<tab id="home" text="HOME">
<groups>
<group text="New" alignType="rows">
<content>

```

```

<content defaults-type="button" defaults-width={60} defaults-height={60}
defaults-isBig={false}>
  <groups>
    <group id="new" text="New" toolTip="New" buttonSettings-
contentType="imageonly" buttonSettings-imagePosition="imagetop"
buttonSettings-prefixIcon="e-icon e-ribbon e-new" buttonSettings-
click="executeAction">
    </group>
  </groups>
</content>
</content>
</group>
</groups>
</tab>
<tab id="sendrec" text="Send/Receive">
  <groups>
    <group text="Send/Receive" type="custom" contentID="sendReceive">
    </group>
  </groups>
</tab>
</tabs>
</EJ.Ribbon>,
document.getElementById('ribbon-resize')
);

```



## Group

Group is a collection of logical content groups that are combined under related Tab. Each group can be defined using content groups or custom content.

### Adding Tab Groups

Group items can be added to Tabs by specifying text and corresponding content to be displayed. The content of group can be specified as either with content collection, contentID or customContent. You can add tab group dynamically in the ribbon control with given tab index, tab group object and group index position by using [addTabGroup](#) method.

### Adding Content

Add content to Group item which is based on type of content specified. The available types are button, splitButton, toggleButton, gallery, and dropDownList.

Groups and defaults settings can be added with the content. You can add group content dynamically in the ribbon control with given tab index, group index, content, content index and sub group index position by using [addTabGroupContent](#).

### \_Defaults\_

The [tabs.groups.content.defaults.height](#), [tabs.groups.content.defaults.width](#), [tabs.groups.content.defaults.type](#), [tabs.groups.content.defaults.isBig](#) property to the controls in the group can be specified commonly.

The height & width applicable to button, split button, dropdown list ,Toggle button controls and isBig applicable to only button controls ( button, split , toggle)

### Adding Content Groups

Controls such as button, split button, dropdown list, toggle button, gallery in the subgroup of the Ribbon tab can be rendered. All of these can be customized using its corresponding settings property such as `buttonSettings`, `dropdownSettings`, etc.

Custom controls or items (such as table, div etc.) can be added when the `type` set as `custom`. `defaults` control settings of group can be specified for an `individual group` instead of specifying them to groups collection commonly.

`Tooltip` and `Custom Tooltip` can be specified for each group controls.

### HTML

```
<div id="ribbon-default"></div>
<script src="app/ribbon/default.js"></script>
<ul id="ribbonmenu1">
<li><a>FILE</a>
<ul>
<li><a>New</a></li>
<li><a>Open</a></li>
<li><a>Save</a></li>
<li><a>Save As</a></li>
<li><a>Print</a></li>
</ul>
</li>
</ul>
```

### HTML

```
"use strict";
ReactDOM.render(
  <div>
    <EJ.Ribbon width="500px" applicationTab-type="menu" applicationTab-
    menuItemID="ribbonmenu1" applicationTab-menuSettings-openOnClick={false}>
    <tabs>
    <tab id="home" text="HOME">
    <groups>
    <group text="Clipboard" alignType="columns">
    <content>
    <content defaults-type="splitbutton" defaults-width={50} defaults-
    height={70}>
    <groups>
    <group id="paste" text="Paste" customTooltip-prefixIcon="e-pastetip"
    splitButtonSettings-contentType="imageonly" splitButtonSettings-
    prefixIcon="e-icon e-ribbon e-ribbonpaste" splitButtonSettings-
    targetID="pasteSplit1" splitButtonSettings-buttonMode="dropdown"
    splitButtonSettings-arrowPosition="right" splitButtonSettings-
    click="executeAction">
    </group>
    </groups>
    </content>
    </content>
    </group>
    <group text="Font" alignType="columns">
    <content>
```

```

<content defaults-type="button" defaults-width={75} defaults-
height={30}defaults-isBig={false}>
  <groups>
    <group id="cut" text="Cut Over" buttonSettings-prefixIcon="e-icon e-ribbon
e-ribboncut" buttonSettings-click="executeAction">
    </group>
    <group id="copy" text="Copy" buttonSettings-prefixIcon="e-icon e-ribbon e-
ribboncopy" buttonSettings-click="executeAction">
    </group>
  </groups>
</content>
</content>
</group>
</groups>
</tab>
</tabs>
</EJ.Ribbon>
</div>,
document.getElementById('ribbon-default')
);

```



#### \_Enable Separator\_

Separates the control from the next control in the group when group `alignType` is `row`. Set “true” to `enableSeparator`.

#### HTML

```

<div id="ribbon-default"></div>
<script src="app/ribbon/default.js"></script>
<ul id="ribbonmenu1">
  <li><a>FILE</a>
  <ul>
    <li><a>New</a></li>
    <li><a>Open</a></li>
    <li><a>Save</a></li>
    <li><a>Save As</a></li>
    <li><a>Print</a></li>
  </ul>
  </li>
</ul>

```

#### HTML

```

"use strict";
ReactDOM.render(
  <div>
    <EJ.Ribbon width="500px" applicationTab-type="menu" applicationTab-
menuItemID="ribbonmenu1" applicationTab-menuSettings-openOnClick={false}>
    <tabs>
      <tab id="home" text="HOME">
      <groups>
        <group text="New" alignType="rows">
        <content>

```

```

<content defaults-type="button" defaults-width={60} defaults-height={60}
defaults-isBig={false}>
  <groups>
    <group id="new" text="New" toolTip="New" enableSeparator="true"
buttonSettings-width="100">
    </group>
    <group id="font" text="Font" toolTip="font" buttonSettings-width="150">
    </group>
  </groups>
</content>
</content>
</group>
</groups>
</tab>
</tabs>
</EJ.Ribbon>
</div>,
document.getElementById('ribbon-default')
);

```



#### Adding Custom Content

Set group **type** as **custom** to add custom items such as div, table and custom controls. With type as custom, content can be added in two ways as specified below.

*HTML contents can be directly added into the groups as string content using **customContent** property*  
 Custom template id can be specified to render those specific custom template using **contentID** property

#### HTML

```

<div id="ribbon-default"></div>
<script src="app/ribbon/default.js"></script>
<ul id="ribbonmenu1">
<li><a>FILE</a>
<ul>
<li><a>New</a></li>
<li><a>Open</a></li>
<li><a>Save</a></li>
<li><a>Save As</a></li>
<li><a>Print</a></li>
</ul>
</li>
</ul>

```

#### HTML

```

ReactDOM.render (
  <div>
    <EJ.Ribbon width="500px" applicationTab-type="menu" applicationTab-
menuItemId="ribbonmenu1" applicationTab-menuSettings-openOnClick={false}>
    <tabs>
    <tab id="home" text="HOME">
    </group>
    <group text="New" type="custom" customContent="<button
id='customContent'>Using Custom Content</button>">

```

```

</group>
<group text="Data" type="custom" contentID="btn" tooltip="font"
buttonSettings-width="150">
</group>
</groups>
</tab>
</tabs>
</EJ.Ribbon>
</div>,
document.getElementById('ribbon-default')
);

```



### Group Expander

Set `enableGroupExpander` as true to show Group Expander for each group in Tab. These expanders can be customized using `groupExpand` event, such as to show popup dialog. To specify tooltip for the group expander of the group [tabs.groups.groupExpanderSettings](#) and

[tabs.groups.groupExpanderSettings.tooltip](#) can be used.

### HTML

```

<div id="ribbon-default"></div>
<script src="app/ribbon/default.js"></script>
<ul id="ribbonmenu1">
<li><a>FILE</a>
<ul>
<li><a>New</a></li>
<li><a>Open</a></li>
<li><a>Save</a></li>
<li><a>Save As</a></li>
<li><a>Print</a></li>
</ul>
</li>
</ul>

```

### HTML

```

ReactDOM.render (
<div>
<EJ.Ribbon width="500px" applicationTab-type="menu" applicationTab-
menuItemID="ribbonmenu1" applicationTab-menuSettings-openOnClick={false}>
<tabs>
<tab id="home" text="HOME">
<groups>
<group text="New" type="custom" enableGroupExpander="true" contentID="btn"
tooltip="font" buttonSettings-width="150">
</group>
</groups>
</tab>
</tabs>
</EJ.Ribbon>
</div>,
document.getElementById('ribbon-default')
);

```



```

```

## Controls Support

Button, Split Button, DropDownList, Toggle button, Gallery and Custom controls can be added to each groups. You can set **type** property in group to define the controls. Default **type** is **button**.

**Note:** 1. You can specify type either to **group's collection** or to each **group**.

2. For **type** property you can assign either string value ("splitbutton") or enum value (ej.Ribbon.type.splitButton).

## HTML

```
<div id="ribbon-resize"></div>
<script src="app/ribbon/resize.js"></script>
<ul id="ribbonmenu1">
<li><a>FILE</a>
<ul>
<li><a>New</a></li>
<li><a>Open</a></li>
<li><a>Save</a></li>
<li><a>Save As</a></li>
<li><a>Print</a></li>
</ul>
</li>
</ul>
<ul id="pasteSplit1">
<li><a>Paste</a></li>
</ul>
```

## HTML

```
"use strict";
var fontfamily = ["Segoe UI", "Arial", "Times New Roman", "Tahoma",
"Helvetica"];
var fontsize = ["1pt", "2pt", "3pt", "4pt", "5pt"];
ReactDOM.render(
<EJ.Ribbon width="100%" applicationTab-type="menu" applicationTab-
menuItemID="ribbonmenu1" applicationTab-menuSettings-openOnClick={false}>
<tabs>
<tab id="home" text="HOME">
<groups>
<group text="New" alignType="rows">
<content>
<content defaults-type="button" defaults-width={60} defaults-height={60}
defaults-isBig={false}>
<groups>
<group id="new" text="New" tooltip="New" buttonSettings-
contentType="imageonly" buttonSettings-imagePosition="imagetop"
buttonSettings-prefixIcon="e-icon e-ribbon e-new" buttonSettings-
click={executeAction}>
</group>
</groups>
</content>
</content>
```

```

</group>
<group text="Clipboard" alignType="columns">
<content>
<content defaults-type="splitbutton" defaults-width={60} defaults-
height={70}>
<groups>
<group id="paste" text="Paste" customTooltip-prefixIcon="e-pastetip"
splitButtonSettings-contentType="imageonly" splitButtonSettings-
prefixIcon="e-icon e-ribbon e-ribbonpaste" splitButtonSettings-
targetID="pasteSplit1" splitButtonSettings-buttonMode="dropdown"
splitButtonSettings-arrowPosition="bottom" splitButtonSettings-
click={executeAction}>
</group>
</groups>
</content>
</content>
</group>
<group text="Font" alignType="rows">
<content>
<content defaults-type="dropdownlist" defaults-height={28}>
<groups>
<group id="fontfamily" dropdownSettings-dataSource={fontfamily}
dropdownSettings-text="Segoe UI" dropdownSettings-select="executeAction"
dropdownSettings-width={150}>
</group>
<group id="fontsize" dropdownSettings-dataSource={fontsize}
dropdownSettings-text="1pt" dropdownSettings-select="executeAction"
dropdownSettings-width={65}>
</group>
</groups>
</content>
<content defaults-isBig={false}>
<groups>
<group id="bold" text="bold" type="togglebutton" toggleButtonSettings-
contentType="imageonly" toggleButtonSettings-defaultText="Bold"
toggleButtonSettings-activeText="Bold" toggleButtonSettings-
defaultPrefixIcon="e-icon e-ribbon e-resbold" toggleButtonSettings-
activePrefixIcon="e-icon e-ribbon e-resbold" toggleButtonSettings-
click={executeAction}></group>
<group id="italic" type="togglebutton" toggleButtonSettings-
contentType="imageonly" toggleButtonSettings-defaultText="Italic"
toggleButtonSettings-activeText="Italic" toggleButtonSettings-
defaultPrefixIcon="e-icon e-ribbon e-resitalic" toggleButtonSettings-
activePrefixIcon="e-icon e-ribbon e-resitalic" toggleButtonSettings-
click={executeAction}></group>
</groups>
</content>
</content>
</group>
</groups>
</tab>
</tabs>
</EJ.Ribbon>,
document.getElementById('ribbon-resize')
);

```

## Gallery

Galleries are used to display items that can be arranged in a grid-type layout. Items in the gallery can be customized as `button/menu` to display images, text, or both images and text. You can set `type` of group as `gallery`.

### Gallery Items

Gallery items are collection of the items to be included in the main gallery. You can set `text` and `toolTip` to gallery item which can also be customized with `buttonSettings`.

Number of `columns` to display in gallery for each row should be specified and the Number of columns in Expanded State (`expandedColumns`) can be customized, if not set, `columns` count will be set as default.

**Note:** The `itemHeight` and `itemWidth` for gallery item can be set, if not set default values will be used.

### HTML

```
<div id="ribbon-default"></div>
<script src="app/ribbon/default.js"></script>
<ul id="ribbonmenu1">
<li><a>FILE</a>
<ul>
<li><a>New</a></li>
<li><a>Open</a></li>
<li><a>Save</a></li>
<li><a>Save As</a></li>
<li><a>Print</a></li>
</ul>
</li>
</ul>
<style type="text/css">
.e-gallerycontent1 {
background-position: 0 -105px;
}
.e-gallerycontent2 {
background-position: -69px -105px;
}
.e-gallerycontent3 {
background-position: -136px -105px;
}
.e-gallerycontent4 {
background-position: 0 -53px;
}
.e-gbtnposition {
margin-top: 5px;
}
.e-gbtnimg {
background-image: url("content/ej/themes/common-
images/ribbon/homegallery.png");
background-repeat: no-repeat;
height: 64px;
width: 64px;
}
</style>
```

**HTML**

```

ReactDOM.render (
<div>
<EJ.Ribbon width="500px" applicationTab-type="menu" applicationTab-
menuItemID="ribbonmenu1" applicationTab-menuSettings-openOnClick={false}>
<tabs>
<tab id="home" text="HOME">
<groups>
<group text="Gallery" alignType="rows">
<content>
<content>
<groups>
<group id="Gallery" columns={2} itemHeight={54} itemWidth={68}
expandedColumns="3" type="gallery">
<galleryItems>
<galleryItem text="GalleryContent1" buttonSettings-contentType="imageonly"
buttonSettings-prefixIcon="e-icon e-gallerycontent1 e-gbtnimg"
buttonSettings-cssClass="e-gbtnposition"></galleryItem>
<galleryItem text="GalleryContent2" buttonSettings-contentType="imageonly"
buttonSettings-prefixIcon="e-icon e-gallerycontent2 e-gbtnimg"
buttonSettings-cssClass="e-gbtnposition"></galleryItem>
<galleryItem text="GalleryContent3" buttonSettings-contentType="imageonly"
buttonSettings-prefixIcon="e-icon e-gallerycontent3 e-gbtnimg"
buttonSettings-cssClass="e-gbtnposition"></galleryItem>
<galleryItem text="GalleryContent4" buttonSettings-contentType="imageonly"
buttonSettings-prefixIcon="e-icon e-gallerycontent4 e-gbtnimg"
buttonSettings-cssClass="e-gbtnposition"></galleryItem>
</galleryItems>
</group>
</groups>
</content>
</content>
</group>
</groups>
</tab>
</tabs>
</EJ.Ribbon>
</div>,
document.getElementById('ribbon-default')
);

```



Ribbon Gallery.



Gallery at Expanded State

### Custom Gallery Items

Custom gallery items are the additional items to be displayed at gallery expanded state. You can set `customItemType` as `button` or `menu`, Default is `button`.

You can also set `text` and `toolTip` to custom gallery item which can also be customized with `buttonSettings/menuSettings` based on the `customItemType` specified.

**HTML**

```

<div id="ribbon-default"></div>
<script src="app/ribbon/default.js"></script>
<ul id="ribbonmenu1">
<li><a>FILE</a>
<ul>
<li><a>New</a></li>
<li><a>Open</a></li>
<li><a>Save</a></li>
<li><a>Save As</a></li>
<li><a>Print</a></li>
</ul>
</li>
</ul>
<style type="text/css">
.e-gallerycontent1 {
background-position: 0 -105px;
}
.e-gallerycontent2 {
background-position: -69px -105px;
}
.e-gallerycontent3 {
background-position: -136px -105px;
}
.e-gallerycontent4 {
background-position: 0 -53px;
}
.e-gbtnposition {
margin-top: 5px;
}
.e-gbtnimg {
background-image: url("content/ej/themes/common-
images/ribbon/homegallery.png");
background-repeat: no-repeat;
height: 64px;
width: 64px;
}
</style>

```

**HTML**

```

ReactDOM.render (
<div>
<EJ.Ribbon width="500px" applicationTab-type="menu" applicationTab-
menuItemID="ribbonmenu1" applicationTab-menuSettings-openOnClick={false}>
<tabs>
<tab id="home" text="HOME">
<groups>
<group text="Gallery" alignType="rows">
<content>
<content>
<groups>
<group id="Gallery" columns={2} itemHeight={54} itemWidth={68}
expandedColumns="3" type="gallery">
<galleryItems>

```

```

<galleryItem text="GalleryContent1" buttonSettings-contentType="imageonly"
buttonSettings-prefixIcon="e-icon e-gallerycontent1 e-gbtnimg"
buttonSettings-cssClass="e-gbtnposition"></galleryItem>
<galleryItem text="GalleryContent2" buttonSettings-contentType="imageonly"
buttonSettings-prefixIcon="e-icon e-gallerycontent2 e-gbtnimg"
buttonSettings-cssClass="e-gbtnposition"></galleryItem>
<galleryItem text="GalleryContent3" buttonSettings-contentType="imageonly"
buttonSettings-prefixIcon="e-icon e-gallerycontent3 e-gbtnimg"
buttonSettings-cssClass="e-gbtnposition"></galleryItem>
<galleryItem text="GalleryContent4" buttonSettings-contentType="imageonly"
buttonSettings-prefixIcon="e-icon e-gallerycontent4 e-gbtnimg"
buttonSettings-cssClass="e-gbtnposition"></galleryItem>
</galleryItems>
<customGalleryItems>
<customGalleryItem customItemType="menu" menuId="extramenu" menuSettings-
openOnClick={false}></customGalleryItem>
<customGalleryItem text="Clear Formatting" toolTip="Clear Formatting"
customItemType="button" buttonSettings-cssClass="e-
extrabtnstyle"></customGalleryItem>
</customGalleryItems>
</group>
</groups>
</content>
</content>
</group>
</groups>
</tab>
</tabs>
</EJ.Ribbon>
</div>,
document.getElementById('ribbon-default')
);

```



## Resize

Ribbon control dynamically resizes to display possible number of controls in the optimal layout as the application window size changes.

As the window is narrowed, controls in the Ribbon will be combined as group button with dropdown arrow, in which controls can be expanded with dropdown arrow.

## Tablet Layout

Set `isResponsive` as true to enable responsive layout in Ribbon. If client width is above 420px or control content exceeds the page then, the ribbon will render in Tablet mode.

## HTML

```

<div id="ribbon-resize"></div>
<script src="app/ribbon/resize.js"></script>
<ul id="ribbonmenu1">
<li><a>FILE</a>
<ul>
<li><a>New</a></li>
<li><a>Open</a></li>
<li><a>Save</a></li>

```

```

<li><a>Save As</a></li>
<li><a>Print</a></li>
</ul>
</li>
</ul>

```

## HTML

```

ReactDOM.render(
  <EJ.Ribbon width="40%" isResponsive={true} applicationTab-type="menu"
  applicationTab-menuItemID="ribbonmenu1" applicationTab-menuSettings-
  openOnClick={false}>
    <tabs>
      <tab id="home" text="HOME">
        <groups>
          <group text="Clipboard" alignType="rows">
            <content>
              <content defaults-type="button" defaults-width={60} defaults-height={70}>
                <groups>
                  <group id="cut" text="Cut" buttonSettings-prefixIcon="e-icon e-ribbon e-
                  ribboncut" buttonSettings-click="executeAction">
                </group>
                  <group id="copy" text="Copy" buttonSettings-prefixIcon="e-icon e-ribbon e-
                  ribboncopy" buttonSettings-click="executeAction">
                </group>
                </groups>
              </content>
            </content>
          </group>
          <group text="Font" alignType="rows">
            <content>
              <content defaults-width={60} defaults-height={70} defaults-isBig={false}>
                <groups>
                  <group id="bold" text="bold" type="togglebutton" toggleButtonSettings-
                  defaultText="Bold" toggleButtonSettings-activeText="Bold"
                  toggleButtonSettings-defaultPrefixIcon="e-icon e-ribbon e-resbold"
                  toggleButtonSettings-activePrefixIcon="e-icon e-ribbon e-resbold"
                  toggleButtonSettings-click="executeAction"></group>
                  <group id="italic" type="togglebutton" toggleButtonSettings-
                  defaultText="Italic" toggleButtonSettings-activeText="Italic"
                  toggleButtonSettings-defaultPrefixIcon="e-icon e-ribbon e-resitalic"
                  toggleButtonSettings-activePrefixIcon="e-icon e-ribbon e-resitalic"
                  toggleButtonSettings-click="executeAction"></group>
                </groups>
              </content>
            </content>
          </group>
          <group text="Alignment" alignType="rows">
            <content>
              <content defaults-width={60} defaults-height={70} defaults-isBig={false}>
                <groups>
                  <group id="left" text="Left" type="togglebutton" toggleButtonSettings-
                  defaultText="Left" toggleButtonSettings-activeText="Left"
                  toggleButtonSettings-defaultPrefixIcon="e-icon e-ribbon e-resbold"
                  toggleButtonSettings-activePrefixIcon="e-icon e-ribbon e-resbold"
                  toggleButtonSettings-click="executeAction"></group>

```

```

<group id="right" text="Right" type="togglebutton" toggleButtonSettings-
defaultText="Right" toggleButtonSettings-activeText="Right"
toggleButtonSettings-defaultPrefixIcon="e-icon e-ribbon e-resitalic"
toggleButtonSettings-activePrefixIcon="e-icon e-ribbon e-resitalic"
toggleButtonSettings-click="executeAction"></group>
</groups>
</content>
</content>
</group>
</groups>
</tab>
</tabs>
</EJ.Ribbon>,
document.getElementById('ribbon-resize')
);

```



### Mobile Layout

If client width is less than 420px, the ribbon will render in mobile mode. In which, you can see that ribbon user interface is customized and redesigned for best view in small screens.

The customized features includes responsive tab & group rendering, backstage, gallery and button controls.

### Responsive Tab and group

Set `isResponsive` as true to enable responsive mode in Ribbon.

### HTML

```

<div id="ribbon-resize"></div>
<script src="app/ribbon/resize.js"></script>
<ul id="ribbonmenu1">
<li><a>FILE</a>
<ul>
<li><a>New</a></li>
<li><a>Open</a></li>
<li><a>Save</a></li>
<li><a>Save As</a></li>
<li><a>Print</a></li>
</ul>
</li>
</ul>

```

### HTML

```

"use strict";
ReactDOM.render(
<EJ.Ribbon width="100%" isResponsive={true} applicationTab-type="menu"
applicationTab-menuItemID="ribbonmenu1" applicationTab-menuSettings-
openOnClick={false}>
<tabs>
<tab id="home" text="HOME">
<groups>
<group text="Font" alignType="rows">
<content>

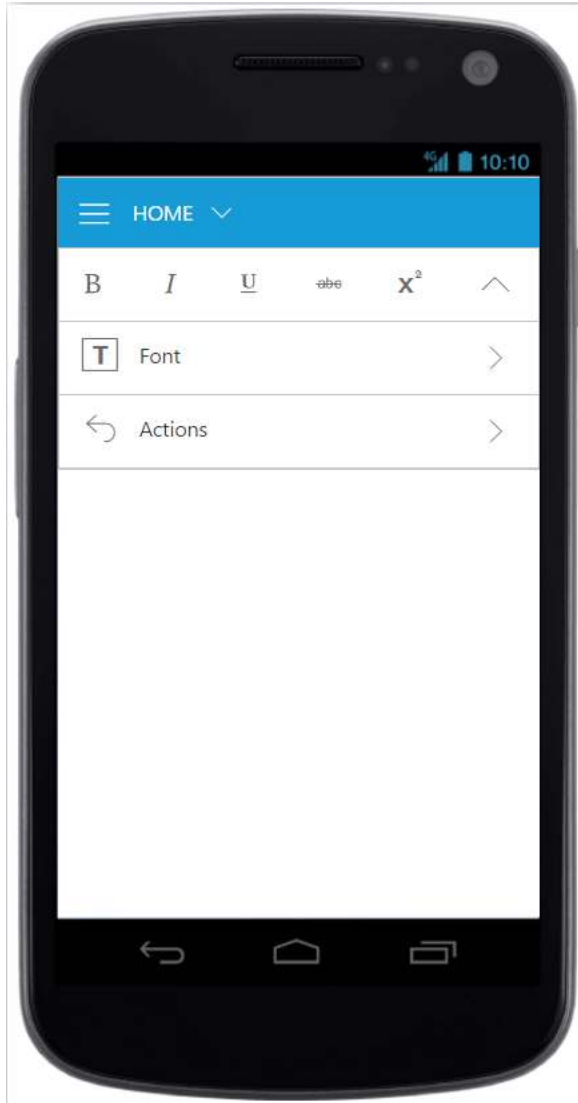
```



```

<content defaults-isBig={false}>
<groups>
<group id="bold" text="bold" isMobileOnly="true" type="togglebutton"
toggleButtonSettings-contentType="imageonly" toggleButtonSettings-
defaultText="Bold" toggleButtonSettings-activeText="Bold"
toggleButtonSettings-defaultPrefixIcon="e-icon e-ribbon e-resbold"
toggleButtonSettings-activePrefixIcon="e-icon e-ribbon e-resbold"
toggleButtonSettings-click="executeAction"></group>
<group id="italic" type="togglebutton" isMobileOnly="true"
toggleButtonSettings-contentType="imageonly" toggleButtonSettings-
defaultText="Italic" toggleButtonSettings-activeText="Italic"
toggleButtonSettings-defaultPrefixIcon="e-icon e-ribbon e-resitalic"
toggleButtonSettings-activePrefixIcon="e-icon e-ribbon e-resitalic"
toggleButtonSettings-click="executeAction"></group>
<group id="underline" isMobileOnly="true" text="underline"
type="togglebutton" toggleButtonSettings-contentType="imageonly"
toggleButtonSettings-defaultText="Underline" toggleButtonSettings-
activeText="Underline" toggleButtonSettings-defaultPrefixIcon="e-icon e-
ribbon e-resunderline" toggleButtonSettings-activePrefixIcon="e-icon e-
ribbon e-resunderline" toggleButtonSettings-click="executeAction"></group>
<group id="strikethrough" isMobileOnly="true" text="Strikethrough"
type="togglebutton" toggleButtonSettings-contentType="imageonly"
toggleButtonSettings-defaultText="Strikethrough" toggleButtonSettings-
activeText="Strikethrough" toggleButtonSettings-defaultPrefixIcon="e-icon e-
ribbon strikethrough" toggleButtonSettings-activePrefixIcon="e-icon e-ribbon
strikethrough" toggleButtonSettings-click="executeAction"></group>
<group id="superscript" text="Superscript" isMobileOnly="true"
buttonSettings-contentType="imageonly" buttonSettings-prefixIcon="e-icon e-
ribbon e-superscripticon" buttonSettings-click="executeAction"></group>
</groups>
</content>
</content>
</group>
</groups>
</tab>
</tabs>
</EJ.Ribbon>,
document.getElementById('ribbon-resize')
);

```



Ribbon Responsive with tab content

**Note:** To make the Ribbon control to react as responsive in mobile devices, it is necessary to refer the additional `ej.responsive.css` file in the application.

### Screen Tips

ScreenTip/Tooltip is used to reduce the controls related Help that are needed to the end user to do control related actions.

### HTML Tooltip

Standard `html tooltip` can be set using `tooltip` property of each group item.

### HTML

```
<div id="ribbon-resize"></div>
<script src="app/ribbon/resize.js"></script>
<ul id="ribbonmenu1">
<li><a>FILE</a>
<ul>
```

```

<li><a>New</a></li>
<li><a>Open</a></li>
<li><a>Save</a></li>
<li><a>Save As</a></li>
<li><a>Print</a></li>
</ul>
</li>
</ul>

```

## HTML

```

"use strict";
ReactDOM.render(
  <EJ.Ribbon width="40%" applicationTab-type="menu" applicationTab-
  menuItemID="ribbonmenu1" applicationTab-menuSettings-openOnClick={false}>
    <tabs>
      <tab id="home" text="HOME">
        <groups>
          <group text="Clipboard" alignType="rows">
            <content>
              <content defaults-type="button" defaults-width={60} defaults-height={70}>
                <groups>
                  <group id="cut" text="Cut" tooltip="Remove the selection and put it on
                  clipboard" buttonSettings-prefixIcon="e-icon e-ribbon e-ribboncut"
                  buttonSettings-click="executeAction">
                </group>
                  <group id="copy" text="Copy" tooltip="Put a copy of selection on clipboard"
                  buttonSettings-contentType="textandimage" buttonSettings-prefixIcon="e-icon
                  e-ribbon e-ribboncopy" buttonSettings-click="executeAction">
                </group>
                </groups>
              </content>
            </content>
          </group>
        </groups>
      </tab>
    </tabs>
  </EJ.Ribbon>,
  document.getElementById('ribbon-resize')
);

```



### Custom Tooltip

Custom Tooltip is used to set detailed help to the user about the controls. You can set **title**, **content** and **prefixIcon** class to customize the tooltip with icons.

### For Groups

Custom tooltip for each group controls can be specified. Such as to the controls button, split button, dropdown list etc.

## HTML

```

<div id="ribbon-resize"></div>
<script src="app/ribbon/resize.js"></script>

```

```

<ul id="ribbonmenu1">
<li><a>FILE</a>
<ul>
<li><a>New</a></li>
<li><a>Open</a></li>
<li><a>Save</a></li>
<li><a>Save As</a></li>
<li><a>Print</a></li>
</ul>
</li>
</ul>
<style type="text/css">
.e-pastetip {
background-image: url("content/ej/themes/common-images/ribbon/paste.png");
background-repeat: no-repeat;
height: 64px;
width: 64px;
}
</style>

```

## HTML

```

"use strict";
ReactDOM.render(
<EJ.Ribbon width="40%" applicationTab-type="menu" applicationTab-
menuItemID="ribbonmenu1" applicationTab-menuSettings-openOnClick={false}>
<tabs>
<tab id="home" text="HOME">
<groups>
<group text="Clipboard" alignType="rows">
<content>
<content defaults-type="button" defaults-width={60} defaults-height={70}>
<groups>
<group id="paste" text="Paste" customToolTip-title="Paste"customToolTip-
content="<h6>Paste the content.<br/><br/>Add content on the Clipboard to
your document.</h6>" customTooltip-prefixIcon="e-pastetip"
splitButtonSettings-contentType="imageonly" splitButtonSettings-
prefixIcon="e-icon e-ribbon e-ribbonpaste">
</group>
<group id="copy" text="Copy" tooltip="Put a copy of selection on clipboard"
buttonSettings-contentType="textandimage" buttonSettings-prefixIcon="e-icon
e-ribbon e-ribboncopy" buttonSettings-click="executeAction">
</group>
</groups>
</content>
</content>
</group>
</groups>
</tab>
</tabs>
</EJ.Ribbon>,
document.getElementById('ribbon-resize')
);

```



*For Gallery*

Custom tooltip for each gallery and custom gallery items button control can be specified.

**Note:** Custom gallery item menu is not supported to Custom tooltip.

**HTML**

```
<div id="ribbon-resize"></div>
<script src="app/ribbon/resize.js"></script>
<ul id="ribbonmenu1">
<li><a>FILE</a>
<ul>
<li><a>New</a></li>
<li><a>Open</a></li>
<li><a>Save</a></li>
<li><a>Save As</a></li>
<li><a>Print</a></li>
</ul>
</li>
</ul>
<style type="text/css">
.e-gallerycontent1 {
background-position: 0 -105px;
}
.e-gallerycontent2 {
background-position: -69px -105px;
}
.e-gallerycontent3 {
background-position: -136px -105px;
}
.e-gallerycontent4 {
background-position: 0 -53px;
}
.e-gbtnposition {
margin-top: 5px;
}
.e-gbtnimg {
background-image: url("content/ej/themes/common-
images/ribbon/homegallery.png");
background-repeat: no-repeat;
height: 64px;
width: 64px;
}
</style>
```

**HTML**

```
<"use strict";
ReactDOM.render(
<div>
<EJ.Ribbon width="500px" applicationTab-type="menu" applicationTab-
menuItemID="ribbonmenu1" applicationTab-menuSettings-openOnClick={false}>
<tabs>
<tab id="home" text="HOME">
<groups>
<group text="Gallery" alignType="rows">
```

```

<content>
<content>
<groups>
<group id="Gallery" columns={2} itemHeight={54} itemWidth={68}
expandedColumns="3" type="gallery">
<galleryItems>
<galleryItem text="Style 1"
customToolTip-title="Style 1" customToolTip-content="<I>Style 1 to customize
the table</I>"
buttonSettings-contentType="imageonly" buttonSettings-prefixIcon="e-icon e-
gallerycontent1 e-gbtnimg" buttonSettings-cssClass="e-
gbtnposition"></galleryItem>
<galleryItem text="Style 2"
customToolTip-title="Style 2" customToolTip-content="<I>Style 2 to customize
the table</I>"
buttonSettings-contentType="imageonly" buttonSettings-prefixIcon="e-icon e-
gallerycontent2 e-gbtnimg" buttonSettings-cssClass="e-
gbtnposition"></galleryItem>
<galleryItem text="Style 3"
customToolTip-title="Style 3" customToolTip-content="<I>Style 3 to customize
the table</I>"
buttonSettings-contentType="imageonly" buttonSettings-prefixIcon="e-icon e-
gallerycontent3 e-gbtnimg" buttonSettings-cssClass="e-
gbtnposition"></galleryItem>
<galleryItem text="GalleryContent4"
customToolTip-title="Style 4" customToolTip-content="<I>Style 4 to customize
the table</I>"
buttonSettings-contentType="imageonly" buttonSettings-prefixIcon="e-icon e-
gallerycontent4 e-gbtnimg" buttonSettings-cssClass="e-
gbtnposition"></galleryItem>
</galleryItems>
</group>
</groups>
</content>
</content>
</group>
</groups>
</tab>
</tabs>
</EJ.Ribbon>
</div>,
document.getElementById('ribbon-resize')
);

```



*For Expand Pin*

Specifies the **custom tooltip** for expand pin in the Ribbon.

### HTML

```

<div id="ribbon-resize"></div>
<script src="app/ribbon/resize.js"></script>
<ul id="ribbonmenu1">
<li><a>FILE</a>
</li>
</ul>

```

```

<li><a>New</a></li>
<li><a>Open</a></li>
<li><a>Save</a></li>
<li><a>Save As</a></li>
<li><a>Print</a></li>
</ul>
</li>
</ul>

```

## HTML

```

"use strict";
ReactDOM.render(
  <div>
    <EJ.Ribbon width="500px" expandPinSettings-customToolTip-title="Collapse the
    Ribbon"
    expandPinSettings-customToolTip-content="<h6>Click the icon to collapse the
    Ribbon.</h6>"
    applicationTab-type="menu" applicationTab-menuItemID="ribbonmenu1"
    applicationTab-menuSettings-openOnClick={false}>
    <tabs>
    <tab id="home" text="HOME">
    <groups>
    <group text="New" alignType="rows">
    <content>
    <content defaults-type="button" defaults-width={60} defaults-height={60}
    defaults-isBig={false}>
    <groups>
    <group id="new" text="New" toolTip="New" buttonSettings-
    contentType="imageonly" buttonSettings-imagePosition="imagetop"
    buttonSettings-prefixIcon="e-icon e-ribbon e-new" buttonSettings-
    click="executeAction">
    </group>
    </groups>
    </content>
    </content>
    </group>
    </groups>
    </tab>
    </tabs>
    </EJ.Ribbon>
  </div>,
  document.getElementById('ribbon-resize')
);

```

![(/js/Ribbon/Screen-Tipsimages/Screen-Tipsimg4.png)

*For Collapse Pin*

Specifies the **custom tooltip** for collapse pin in the Ribbon.

## HTML

```

< <div id="ribbon-resize"></div>
<script src="app/ribbon/resize.js"></script>
<ul id="ribbonmenu1">
<li><a>FILE</a>

```

```

<ul>
<li><a>New</a></li>
<li><a>Open</a></li>
<li><a>Save</a></li>
<li><a>Save As</a></li>
<li><a>Print</a></li>
</ul>
</li>
</ul>

```

## HTML

```

"use strict";
ReactDOM.render(
  <div>
    <EJ.Ribbon width="500px" collapsePinSettings-customToolTip-title="Pin the
    Ribbon"
    collapsePinSettings-customToolTip-content="<h6>Keep it open while you
    work</h6>"
    applicationTab-type="menu" applicationTab-menuItemID="ribbonmenu1"
    applicationTab-menuSettings-openOnClick={false}>
      <tabs>
        <tab id="home" text="HOME">
          <groups>
            <group text="New" alignType="rows">
              <content>
                <content defaults-type="button" defaults-width={60} defaults-height={60}
                defaults-isBig={false}>
                  <groups>
                    <group id="new" text="New" toolTip="New" buttonSettings-
                    contentType="imageonly" buttonSettings-imagePosition="imagetop"
                    buttonSettings-prefixIcon="e-icon e-ribbon e-new" buttonSettings-
                    click="executeAction">
                      </group>
                    </groups>
                  </content>
                </content>
              </group>
            </groups>
          </tab>
        </tabs>
      </EJ.Ribbon>
    </div>,
  document.getElementById('ribbon-resize')
);

```



*For GroupExpander*

Custom tooltip for each group expander can be specified.

## HTML

```

<div id="ribbon-resize"></div>
<script src="app/ribbon/resize.js"></script>
<ul id="ribbonmenu1">

```



```

<li><a>FILE</a>
<ul>
<li><a>New</a></li>
<li><a>Open</a></li>
<li><a>Save</a></li>
<li><a>Save As</a></li>
<li><a>Print</a></li>
</ul>
</li>
</ul>

```

## HTML

```

ReactDOM.render (
<div>
<EJ.Ribbon width="500px" applicationTab-type="menu" applicationTab-
menuItemID="ribbonmenu1" applicationTab-menuSettings-openOnClick={false}>
<tabs>
<tab id="home" text="HOME">
<groups>
<group text="New" enableGroupExpander="true" groupExpanderSettings-
customToolTip-title="Clipboard" groupExpanderSettings-customToolTip-
content="<h6>Show a popup for the Clipboard group.</h6>" alignType="rows">
<content>
<content defaults-type="button" defaults-width={60} defaults-height={60}
defaults-isBig={false}>
<groups>
<group id="new" text="New" toolTip="New" buttonSettings-
contentType="imageonly" buttonSettings-imagePosition="imagetop"
buttonSettings-prefixIcon="e-icon e-ribbon e-new" buttonSettings-
click="executeAction">
</group>
</groups>
</content>
</content>
</group>
</groups>
</tab>
</tabs>
</EJ.Ribbon>
</div>,
document.getElementById('ribbon-resize')
);

```



## Quick Access Toolbar

Quick Access Toolbar provides the shortcuts to the most commonly used commands by placing the controls at the Quick Access Toolbar section. It can be placed at the top or bottom of the Ribbon.

Set [showQAT](#) as true to enable Quick Access Toolbar in Ribbon. It supports the Button, Split Button, Toggle Button controls. The [quickAccessMode](#) is used to change the controls state in Quick Access Toolbar through options as `toolbar`, `menu` and `none`. Default value is `none` and QAT toolbar is created with specified controls added in Toolbar.

The **toolbar** option used to set controls visibility in Quick Access Toolbar. The **menu** option shows the controls in Quick Access Menu and does not show controls in Quick Access Toolbar.

Once the controls are visible in Toolbar , then controls state will be set as ticked in Quick Access Menu and vice versa.

The client side event for Quick Access Toolbar menu click is [gatMenuItemClick](#) and it will be triggered with QAT menu item click.

**More Commands** command provides with Quick Access Menu. This can be customized using [gatMenuItemClick](#) event, such as to show popup dialog.

### HTML

```
<div id="ribbon-quickaccesstoolbar"></div>
<script src="app/ribbon/quickaccesstoolbar.js"></script>
```

### HTML

```
"use strict";
ReactDOM.render (
  <div>
    <ul id="ribbonmenu3">
      <li><a>FILE</a>
      <ul>
        <li><a>New</a></li>
        <li><a>Open</a></li>
        <li><a>Save</a></li>
        <li><a>Save As</a></li>
        <li><a>Print</a></li>
      </ul>
    </li>
    </ul>
    <ul id="pasteSplit3">
      <li><a>Paste</a></li>
    </ul>
    <EJ.Ribbon width="100%" allowResizing={true} applicationTab-type="menu"
    applicationTab-menuItemID="ribbonmenu3" applicationTab-menuSettings-
    openOnClick={false} showQAT={true}>
      <tabs>
        <tab id="home" text="HOME">
          <groups>
            <group text="splitbutton" alignType="columns">
              <content>
                <content defaults-type="splitbutton" defaults-width={60} defaults-
                height={70}>
                  <groups>
                    <group id="paste" text="Paste" quickAccessMode="toolbar" customTooltip-
                    prefixIcon="e-pastetip" splitButtonSettings-contentType="imageonly"
                    splitButtonSettings-prefixIcon="e-icon e-ribbon e-ribbonpaste"
                    splitButtonSettings-targetID="pasteSplit3" splitButtonSettings-
                    buttonMode="dropdown" splitButtonSettings-arrowPosition="bottom"
                    splitButtonSettings-click="executeAction">
                  </group>
                </groups>
              </content>
            </group>
          </groups>
        </tab>
      </tabs>
    </EJ.Ribbon>
  </div>
```

```

</group>
<group text="button" alignType="rows">
<content>
<content defaults-isBig={false}>
<groups>
<group id="italic" type="togglebutton" quickAccessMode="toolbar"
toggleButtonSettings-contentType="imageonly" toggleButtonSettings-
defaultText="Italic" toggleButtonSettings-activeText="Italic"
toggleButtonSettings-defaultPrefixIcon="e-icon e-ribbon e-resitalic"
toggleButtonSettings-activePrefixIcon="e-icon e-ribbon e-resitalic"
toggleButtonSettings-click="executeAction"></group>
</groups>
</content>
</content>
</group>
<group text="togglebutton" alignType="rows">
<content>
<content defaults-isBig={false}>
<groups>
<group id="bold" text="bold" type="togglebutton" quickAccessMode="toolbar"
toggleButtonSettings-contentType="imageonly" toggleButtonSettings-
defaultText="Bold" toggleButtonSettings-activeText="Bold"
toggleButtonSettings-defaultPrefixIcon="e-icon e-ribbon e-resbold"
toggleButtonSettings-activePrefixIcon="e-icon e-ribbon e-resbold"
toggleButtonSettings-click="executeAction"></group>
</groups>
</content>
</content>
</group>
</groups>
</tab>
</tabs>
</EJ.Ribbon>
</div>,
document.getElementById('ribbon-quickaccesstoolbar')
);

```



## Globalization and Localization

### Localization

The localization support allows to customize the display of text within the Ribbon in a user-specific culture and locale. The Ribbon control can be localized in specific culture using the common API [locale](#) along with the collection of localized words defined for that culture using the `ej.Ribbon.Locale` [culture-code]. Please find the table with list of properties and its value in locale object.

Locale key words	Text
CustomizeQuickAccess	Customize Quick Access Toolbar
RemoveFromQuickAccessToolbar	Remove from Quick Access Toolbar
AddToQuickAccessToolbar	Add to Quick Access Toolbar

ShowAboveTheRibbon	Show Above the Ribbon
ShowBelowTheRibbon	Show Below the Ribbon
MoreCommands	More Commands...

**Note:** By default, the Ribbon control is localized in **en-US** culture.

For further information on – how to refer the required culture scripts into your application, refer [here](#).

### HTML

```
<div id="ribbon-resize"></div>
<script src="app/ribbon/resize.js"></script>
<ul id="ribbonmenu1">
<li><a>FILE</a>
<ul>
<li><a>New</a></li>
<li><a>Open</a></li>
<li><a>Save</a></li>
</ul>
</li>
</ul>
```

### HTML

```
"use strict";
ej.Ribbon.Locale["es-ES"] = {
CustomizeQuickAccess: "Agordu Rapida Aliro",
RemovefromQuickAccessToolbar: "Forigu de Rapida Aliro Ilobreto",
AddtoQuickAccessToolbar: "Aldoni al Rapida Aliro Ilobreto",
ShowAbovetheRibbon: "Montru Super la Ribbona",
ShowBelowtheRibbon: "Montru Sube la Ribbon",
MoreCommands: "pli Komando"
};
ReactDOM.render(
<div>
<EJ.Ribbon width="100%" locale="es-ES" applicationTab-type="menu"
applicationTab-menuItemID="ribbonmenu3" applicationTab-menuSettings-
openOnClick={false} showQAT={true}>
<tabs>
<tab id="home" text="HOME">
<groups>
<group text="Clipboard" alignType="columns">
<content>
<content defaults-type="splitbutton" defaults-width={60} defaults-
height={70}>
<groups>
<group id="paste" text="Paste" quickAccessMode="toolbar" customTooltip-
prefixIcon="e-pastetip" splitButtonSettings-contentType="imageonly"
splitButtonSettings-prefixIcon="e-icon e-ribbon e-ribbonpaste"
splitButtonSettings-targetID="pasteSplit3" splitButtonSettings-
buttonMode="dropdown" splitButtonSettings-arrowPosition="bottom"
splitButtonSettings-click="executeAction">
</group>
```

```

</groups>
</content>
</content>
</group>
</groups>
</tab>
</tabs>
</EJ.Ribbon>
</div>,
document.getElementById('ribbon-quickaccesstoolbar')
);

```



### Right to Left - RTL

By default, Ribbon render its content and layout from left to right. To customize Ribbon direction, you can change direction from LTR to RTL by using [enableRTL](#) as true.

### HTML

```

<div id="ribbon-resize"></div>
<script src="app/ribbon/resize.js"></script>
<ul id="ribbonmenu1">
<li><a>FILE</a>
<ul>
<li><a>New</a></li>
<li><a>Open</a></li>
<li><a>Save</a></li>
</ul>
</li>
</ul>

```

### HTML

```

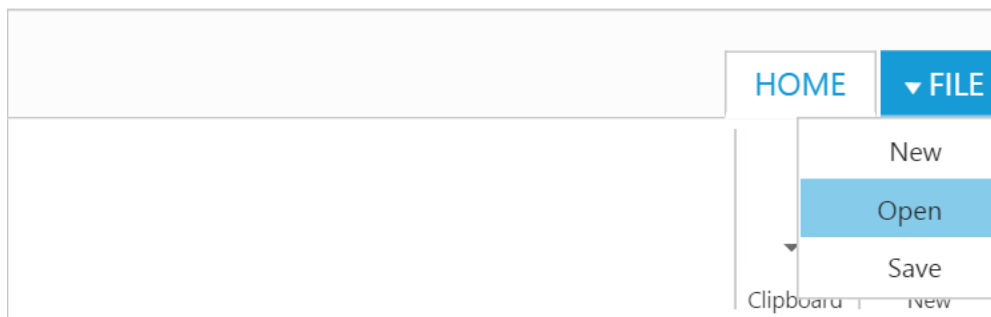
"use strict";
ReactDOM.render(
  <EJ.Ribbon width="500px" enableRTL={true} applicationTab-type="menu"
  applicationTab-menuItemID="ribbonmenu1" applicationTab-menuSettings-
  openOnClick={false}>
    <tabs>
    <tab id="home" text="HOME">
      <groups>
        <group text="New" alignType="rows">
          <content>

```

```

<content defaults-type="button" defaults-width={60} defaults-height={60}
defaults-isBig={false}>
<groups>
<group id="new" text="New" toolTip="New" buttonSettings-
contentType="imageonly" buttonSettings-imagePosition="imagetop"
buttonSettings-prefixIcon="e-icon e-ribbon e-new" buttonSettings-
click="executeAction">
</group>
</groups>
</content>
</content>
</group>
<group text="Clipboard" alignType="columns">
<content>
<content defaults-type="splitbutton" defaults-width={50} defaults-
height={70}>
<groups>
<group id="paste" text="Paste" customTooltip-prefixIcon="e-pastetip"
splitButtonSettings-contentType="imageonly" splitButtonSettings-
prefixIcon="e-icon e-ribbon e-ribbonpaste" splitButtonSettings-
targetID="pasteSplit1" splitButtonSettings-buttonMode="dropdown"
splitButtonSettings-arrowPosition="bottom" splitButtonSettings-
click="executeAction">
</group>
</groups>
</content>
</content>
</group>
</groups>
</tab>
</tabs>
</EJ.Ribbon>,
document.getElementById('ribbon-resize')
);

```



## RichTextEditor

### Overview

**Rich text editor** is a component that help you to display or edit the content including tables, hyperlinks, paragraphs, lists, video, and images. The editor supports file and folder management using FileExplorer component.

The following are some of the key features of the editor:

*Editing with formatting* *Toolbar* *Content Area with customization* *Image and File browser* *Links* *Tables* *Localization* *XHTML validation*

## Getting Started

This section helps to understand the getting started of RTE control with the step-by-step instruction.

### Create RTE Control in React JS

You can create a React application and add necessary scripts and styles with the help of the given [React Getting Started Documentation](#).

Create a JSX file for rendering RichTextEditor component using <EJ.RTE> syntax. Add required properties to it in <EJ.RTE> tag element

#### JS

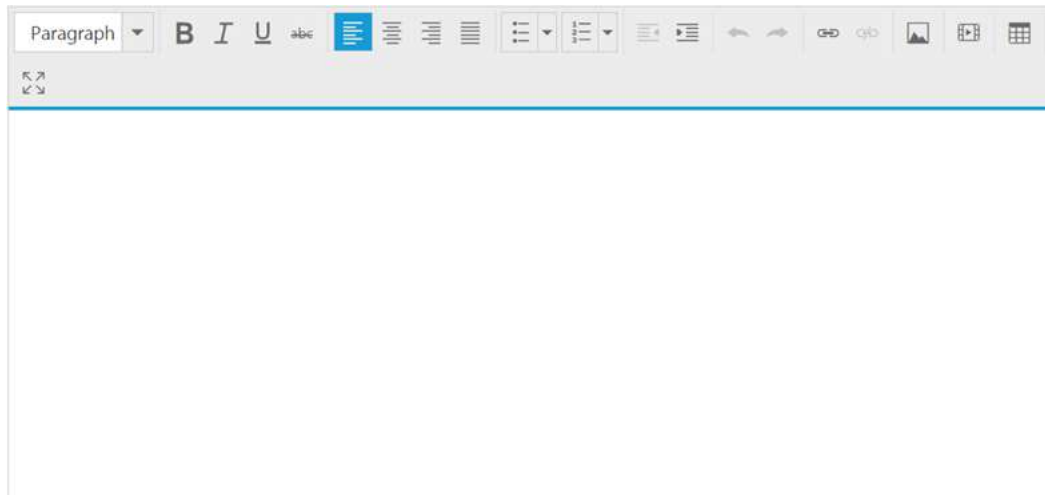
```
"use strict";
ReactDOM.render(
  <EJ.RTE width="100%" minWidth="150px" isResponsive={true}>
</EJ.RTE>,
  document.getElementById('rte-default')
);
```

Define an HTML element for adding RichTextEditor in the application and refer the JSX file.

#### HTML

```
<div id="rte-default"></div>
<script src="app/rte/default.js"></script>
```

The following screenshot displays a RTE widget.



### Toolbar-Configuration

You can configure a toolbar with the tools as your application requires.

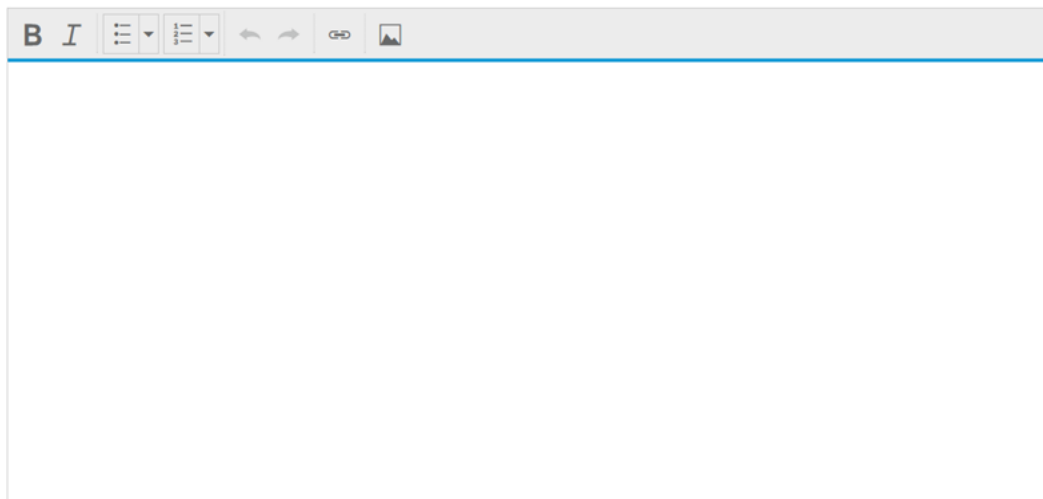
#### HTML

```
<div id="rte-default"></div>
<script src="app/rte/default.js"></script>
```

JS

```
"use strict";
var tool=["style", "lists", "doAction", "links", "images"];
var tools={style: ["bold", "italic"], lists: ["unorderedList",
"orderedList"],doAction: ["undo", "redo"],links: ["createLink"],images:
["image"] };
ReactDOM.render(
<EJ.RTE width="100%" minWidth="150px" isResponsive={true} toolsList={tool}
tools={tools}>
</EJ.RTE>,
document.getElementById('rte-default')
);
```

The following screenshot displays a RTE widget.



## Setting and Getting Content

You can set the content of the editor as follows.

HTML

```
<div id="rte-default"></div>
<script src="app/rte/default.js"></script>
```

JS

```
"use strict";
ReactDOM.render(
<EJ.RTE width="100%" minWidth="150px" isResponsive={true} >
&#60;p&#62;&#60;b&#62;Description:&#60;/b&#62;&#60;/p&#62;
&#60;p&#62;The Rich Text Editor (RTE) control is an easy to render in
client side. Customer easy to edit the contents and get the HTML content for
the displayed content. A rich text editor control provides users with a
toolbar
that helps them to apply rich text formats to the text entered in the text
area. &#60;/p&#62;
&#60;p&#62;&#60;b&#62;Functional
Specifications/Requirements:&#60;/b&#62;&#60;/p&#62;
&#60;ol&#62;&#60;li&#62;&#60;p&#62;Provide
```

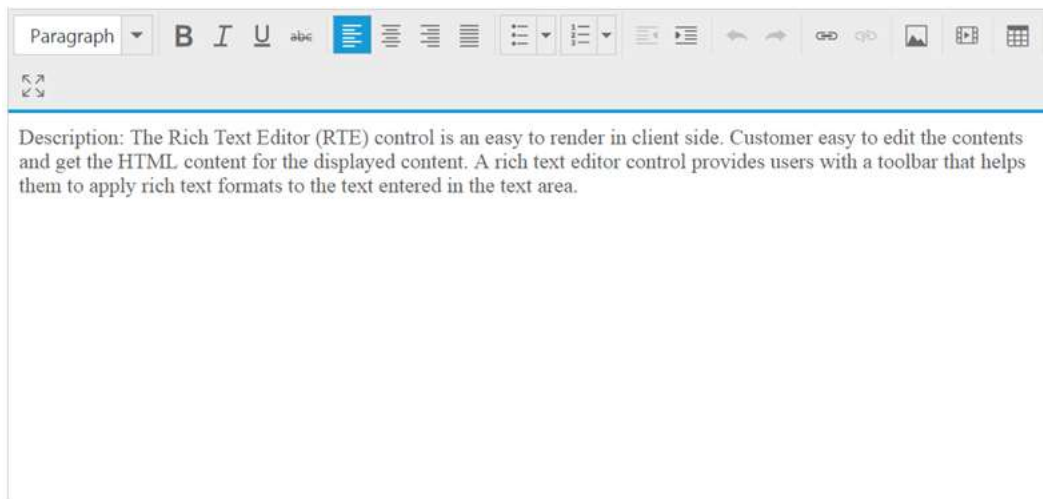


```

the tool bar support, it's also
customizable.<#60;/p><#60;/li><#60;li><#60;p>Options
to get the HTML elements with
styles.<#60;/p><#60;/li><#60;li><#60;p>Support
to insert image from a defined
path.<#60;/p><#60;/li><#60;li><#60;p>Footer
elements and styles(tag / Element information , Action button (Upload,
Cancel))<#60;/p><#60;/li><#60;li><#60;p>Re-size
the editor support. <#60;/p><#60;/li><#60;li><#60;p>Provide
efficient public methods and client side
events.<#60;/p><#60;/li><#60;li><#60;p>Keyboard
navigation support.<#60;/p><#60;/li><#60;/ol><#62;
</EJ.RTE>,
document.getElementById('rte-default')
);

```

The following screenshot displays a RTE widget.



You can also set the content of the editor using value property as follows.

## HTML

```
<div id="rte-default"></div>
<script src="app/rte/default.js"></script>
```

## JS

```
"use strict";
var RTEDefault = React.createClass({
  getInitialState: function()
  {
    return ({
      data: 'This is RTE Content'
    });
  },
  render: function () {
    return (
      <div id="rte default" >
```

```

<EJ.RTE width="100%" minWidth="150px" isResponsive={true}
value={this.state.data} >
</EJ.RTE>
</div>
);
}
});
ReactDOM.render(<RTEDefault />, document.getElementById('rte-default'));

```

The value that is set to the RTE is added in this.state. Hence, the value is accessible in componentDidMount()

## Rotator

### Overview

Our Essential ReactJS Rotator component displays a set of slides with images or images and content, or content with user-defined transition between them. It supports Data Binding, Thumbnail, Pager, dynamic number of slide move options and all custom animations. It supports all types of image formats (JPEG, GIF and so on).

### Key Features

**Data Binding:** Supports data binding with local and remote data.

**Image with Content:** Supports to render an image, content, or image with content.

**Customized slides:** Supports dynamic (one or more) number of slides and to move dynamic (one or more) number of slides at a time.

**Auto-play:** Supports auto-play mode for slide transition.

**Pager and Thumbnail:** Support to navigate between slides with thumbnail images or paging

### Getting Started

This section helps to get started with Essential ReactJS Rotator component.

#### Create a Rotator

Refer the common React Getting Started Documentation to create an application and add necessary scripts and styles for rendering our ReactJS components.

Create a JSX file and use <EJ.Rotator> syntax to render React Rotator component. Add required properties to <EJ.Rotator> tag element.

#### HTML

```

ReactDOM.render(
<EJ.Rotator id="sliderContent" slideWidth="600px" slideHeight={330}>
</EJ.Rotator>,
document.getElementById('rotator-default')
);

```

Define an HTML element for adding Rotator in the application and refer the JSX file created.

#### HTML

```

<div id="rotator-default"></div>
<script type="text/babel" src="sample.jsx">

```

This will render an empty Rotator component on executing.

### Configure data

To configure images for Rotator component, define data in an array and map corresponding fields to it.

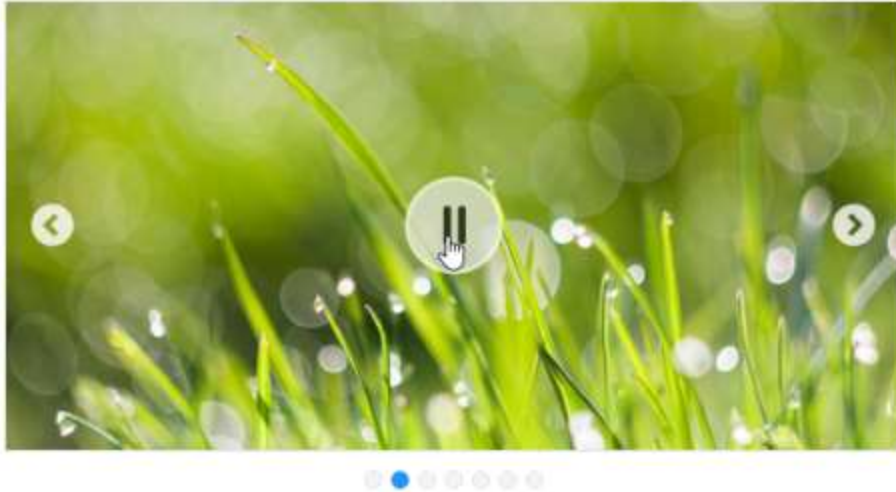
#### JAVASCRIPT

```
var dataList = [
  { text: "Colorful Night", url:
    "http://js.syncfusion.com/demos/web/content/images/rotator/night.jpg" },
  { text: "Technology", url:
    "http://js.syncfusion.com/demos/web/content/images/rotator/tablet.jpg" },
  { text: "Nature", url:
    "http://js.syncfusion.com/demos/web/content/images/rotator/nature.jpg" },
  { text: "Snow Fall", url:
    "http://js.syncfusion.com/demos/web/content/images/rotator/snowfall.jpg" },
  { text: "Credit Card", url:
    "http://js.syncfusion.com/demos/web/content/images/rotator/card.jpg" },
  { text: "Beautiful Bird", url:
    "http://js.syncfusion.com/demos/web/content/images/rotator/bird.jpg" },
  { text: "Amazing Sculptures", url:
    "http://js.syncfusion.com/demos/web/content/images/rotator/sculpture.jpg" }
];
var fieldset={text:"text",url:"url"};
```

In the JSX you have to assign the values for dataSource and fields as given below,

#### HTML

```
ReactDOM.render(
  <EJ.Rotator id="sliderContent" slideWidth="600px" slideHeight={330}
    fields={fieldset}
    isResponsive={true} dataSource={dataList} navigateSteps={1}
    orientation={ej.Orientation.Horizontal}
    showPager={true} enabled={true} showPlayButton={true} animationType="slide">
  </EJ.Rotator>,
  document.getElementById('rotator-default')
);
```



*Note: You can find the Rotator properties from the [API reference](#) document.*

## Schedule

### Overview

**Scheduler** is an event calendar that manages the list of various activities (events/appointments) in different available views (day, week, workweek, month and agenda) for various resources. It is mainly for tracking the user appointments and allows them to create a new or edit/delete the older ones. Almost all the features in JS Scheduler are applicable to Scheduler in ReactJS too.

### Key Features

Some of the key features of Scheduler are as follows,

- **DataSource** - Supports various client-side and remote data sources such as JSON, RESTful services, OData services, WCF services and much more.
- **Views** - 6 types of views are available namely Day, Week, WorkWeek, Month, Agenda and Custom View (user-specific date rendering).
- **Adaptive** - Scheduler UI layout adapts automatically according to the desktop/mobile mode (responsive support).
- **Resize & Drag** - Appointments can be resized and dragged anywhere within the Scheduler. External drag and drop of appointments are also applicable now.
- **Multiple Resources & Grouping** - Allows the Scheduler to categorize and display resources in a hierarchical structure either in a horizontal or vertical manner.
- **Orientation** - Two types of control orientation is supported namely - Vertical and Horizontal (Timeline View).
- **Categories** - 6 default types of Appointment categories along with user customization options are available to differentiate the appointment status.
- **Template** - Template customization provided for appointments, resource header, cells, date header, priority, tooltip, time scale and agenda view.
- **TimeZone & DST** - Supports observation of Daylight Saving Time in Scheduler for whichever time zone it is applicable.
- **Export & Print** - Supports exporting of single/all appointment(s) to an ICS file and also Prints all/specific appointment(s).

- **PDF Export** - Supports exporting of entire Scheduler into PDF format.
- **Appointment window Customization** - Entire appointment window can be customized with the user required fields.
- **Recurrence Editor** - Complete recurrence related options of Scheduler are collectively defined as a separate plug-in, which can be utilized directly within the customized appointment window.

### External and Internal Dependencies

The following list of external dependencies are mandatory in order to render any of the Syncfusion controls -

- [jQuery](#) - 1.7.1 and later versions
- [jsRender](#) - to render the templates

The required **ReactJS** script dependencies are as follows.

- react.min.js - <http://cdn.syncfusion.com/js/assets/external/react.min.js>
- react-dom.min.js - <http://cdn.syncfusion.com/js/assets/external/react-dom.min.js>
- browser.min.js - <http://cdnjs.cloudflare.com/ajax/libs/babel-core/5.8.34/browser.min.js>

The other required internal dependencies of Scheduler are tabulated below,

File	Description/Usage
ej.core.min.js	Must be referred always first before using all the JS controls.
ej.data.min.js	Used to handle data operation and should be used while binding data to JS controls.
ej.globalize.min.js	Must be referred to localize any of the JS control's text and content.
ej.schedule.min.js	Schedule core script file which includes schedule related scripts files such as ej.schedule.render.js, ej.schedule.resources.js and ej.schedule.horizontal.js
ej.scroller.js ej.touch.js ej.draggable.js ej.navigationdrawerbase.js ej.listviewbase.js ej.listview.js ej.recurrenceeditor.js	These files are referred for proper working of the sub-controls used within Scheduler.

ej.dropdownlist.js	
ej.radiobutton.js	
ej.dialog.js	
ej.button.js	
ej.autocomplete.js	
ej.datepicker.js	
ej.timepicker.js	
ej.checkbox.js	
ej.editor.js	
ej.menu.js	
ej.navigationdrawer.js	
ej.tooltip.js	
ej.web.react.min.js	Must be referred to render the Syncfusion widgets in React applications

**Note:** Scheduler uses one or more sub-controls, therefore refer the `ej.web.all.min.js` (which encapsulates all the `ej` controls and frameworks in a single file) in the application instead of referring all the above specified internal dependencies.

To get the real appearance of the Scheduler, the dependent CSS file `ej.web.all.min.css` (which includes styles of all the widgets) should also needs to be referred.

**Note:** Uncompressed version of library files are also available which is used for development or debugging purpose and can be generated from the custom script [here](#).

**Information:** `ej.web.react.min.js` file is additionally required to render the Scheduler in ReactJS which is available under the location, *(Installed location)\Syncfusion\Essential Studio\{{ site.releaseversion }}\JavaScript\assets\scripts\common*.

## Getting Started

An introduction to start with ReactJS can be referred from [here](#). To get start with the usage of Schedule control in ReactJS platform, create a React application layout and refer the following dependencies to it. Refer the individual script references required to render the Schedule control from [here](#).

### HTML

```
<head>
<meta charset="utf-8" />
```

```

<title>Getting Started - Schedule</title>
<!-- CSS reference -->
<link href="http://cdn.syncfusion.com/{{ site.releaseversion }}/js/web/flat-azure/ej.web.all.min.css" rel="stylesheet" />
<!-- Script references -->
<script src="http://cdn.syncfusion.com/js/assets/external/jquery-3.0.0.min.js"></script>
<script src="http://cdn.syncfusion.com/js/assets/external/jsrender.min.js"></script>
<script src="http://cdn.syncfusion.com/js/assets/external/react.min.js"></script>
<script src="http://cdn.syncfusion.com/js/assets/external/react-dom.min.js"></script>
<script src="https://cdnjs.cloudflare.com/ajax/libs/babel-core/5.8.34/browser.min.js"></script>
<script src="http://cdn.syncfusion.com/{{ site.releaseversion }}/js/web/ej.web.all.min.js"></script>
<script src="http://cdn.syncfusion.com/{{ site.releaseversion }}/js/common/ej.web.react.min.js"></script>
</head>

```

**Note:** Uncompressed version of the library files are also available which is used for development or debugging purpose and can be generated from the custom script [here](#).

### Control Initialization

Scheduler can be initialized in either of the 2 ways by making use of the jsx template or without it.

#### Using jsx Template

While making use of jsx template, we have to create both the HTML and jsx files. The .jsx file should be converted into .js file using [gulp](#) command and then needs to be added as a reference in the HTML page.

#### HTML file

##### HTML

```

<div id="schedule-default"></div>
<script src="scripts/default.js"></script>

```

#### JSX file

##### HTML

```

var dManager = [{
  Id: 1,
  Subject: "Bering Sea Gold",
  StartTime: new Date(2014, 4, 5, 5, 30),
  EndTime: new Date(2014, 4, 5, 7, 30),
  Description: "",
  AllDay: false,
  Recurrence: false
}];
ReactDOM.render(
  <EJ.Schedule id="Schedule1"
  width="100%"
  height="525px"
  currentDate={new Date(2014, 4, 5)}
  appointmentSettings-dataSource={dManager}>

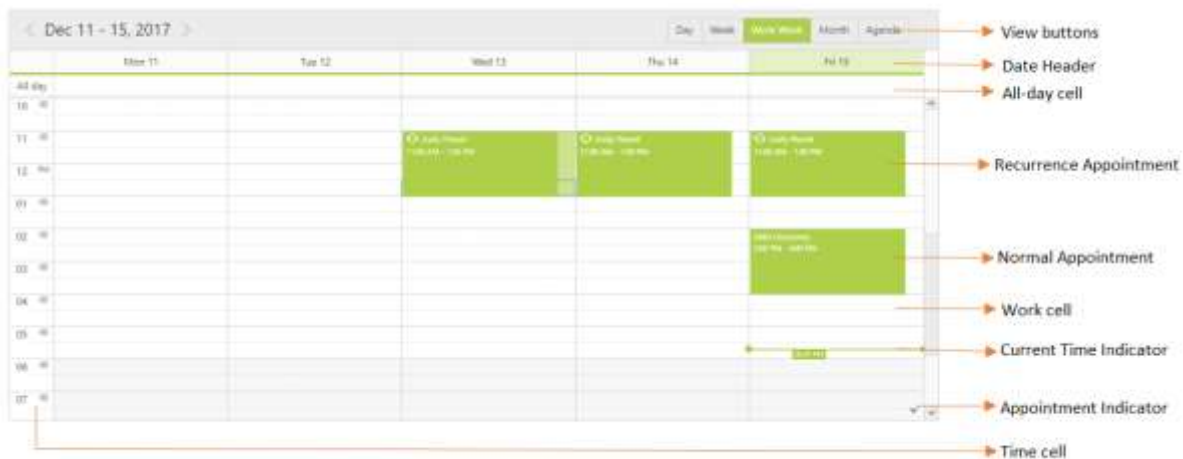
```

```

appointmentSettings-id="Id"
appointmentSettings-subject="Subject"
appointmentSettings-startTime="StartTime"
appointmentSettings-endTime="EndTime"
appointmentSettings-description="Description"
appointmentSettings-allDay="AllDay"
appointmentSettings-recurrence="Recurrence"
appointmentSettings-recurrenceRule="RecurrenceRule">
</EJ.Schedule>, document.getElementById('schedule-default')
);

```

**Note:** The above jsx template needs to be converted from .jsx to .js extension by using **gulp** NuGet package (refer [here](#)) and then it must be referred in the HTML page.



### Using without jsx Template

Create a Div element within the body section of the HTML document, where the Scheduler needs to be rendered.

#### HTML

```

<body>
<div id="schedule"></div>
</body>

```

Initialize the Schedule control by adding the following script code to the body section of the HTML document.

#### HTML

```

<body>
<div id="schedule"></div>
<script>
var dManager = [{
  Id: 1,
  Subject: "Bering Sea Gold",
  StartTime: new Date(2014, 4, 5, 5, 30),
  EndTime: new Date(2014, 4, 5, 7, 30),
  Description: "",

```



```

AllDay: false,
Recurrence: false
}];
ReactDOM.render(
  React.createElement(EJ.Schedule, {
    "id": "Schedule1",
    "width": "100%",
    "height": "525px",
    "currentDate": new Date(2014, 4, 5),
    "appointmentSettings-dataSource": dManager,
    "appointmentSettings-id": "Id",
    "appointmentSettings-subject": "Subject",
    "appointmentSettings-startTime": "StartTime",
    "appointmentSettings-endTime": "EndTime",
    "appointmentSettings-description": "Description",
    "appointmentSettings-allDay": "AllDay",
    "appointmentSettings-recurrence": "Recurrence",
    "appointmentSettings-recurrenceRule": "RecurrenceRule"
  }), document.getElementById('schedule')
);
</script>
</body>

```

### Working with timeZone Settings

Initially, the system timezone is preferred as the Schedule's default timezone. When some specific timezone is explicitly defined to the Schedule, it will be set to it.

Likewise, we can also set different timezones for the appointments. Usually, the system timezone is assigned as the appointment's default timezone and it will be positioned on the Scheduler based on the start/end time range and timezone assigned to it.

To set Scheduler timeZone and its appointment timeZone values, refer the below code example -

### HTML

```

<body>
<div id="schedule"></div>
<script>
var dManager = [{
  Id: 1,
  Subject: "Bering Sea Gold",
  StartTime: new Date(2014, 4, 5, 5, 30),
  StartTimeZone: "UTC +02:00",
  EndTime: new Date(2014, 4, 5, 7, 30),
  EndTimeZone: "UTC +02:00",
  Description: "",
  AllDay: false,
  Recurrence: false
}];
ReactDOM.render(
  React.createElement(EJ.Schedule, {
    "id": "Schedule1",
    "width": "100%",
    "height": "525px",
    "currentDate": new Date(2014, 4, 5),
    "timeZone": "UTC +05:30", // timeZone set to Scheduler

```

```

"appointmentSettings-dataSource": dManager,
"appointmentSettings-id": "Id",
"appointmentSettings-subject": "Subject",
"appointmentSettings-startTime": "StartTime",
"appointmentSettings-startTimeZone": "StartTimeZone",
"appointmentSettings-endTime": "EndTime",
"appointmentSettings-endTimeZone": "EndTimeZone",
"appointmentSettings-description": "Description",
"appointmentSettings-allDay": "AllDay",
"appointmentSettings-recurrence": "Recurrence",
"appointmentSettings-recurrenceRule": "RecurrenceRule"
}), document.getElementById('schedule')
);
</script>
</body>

```

## Data Binding

### Appointment Fields

The below listed names are the appointment fields which holds the appropriate column names from the dataSource.

Field name	Description
id	Binds the id field name for indexing and performing CRUD operation on the appointments. It's optional.
startTime	Binds the appointment start time field name which is mandatory and also its related validation rules.
startTimeZone	Binds the name of the start timezone field in the dataSource and also its related validation rules. If the startTimeZone field is not mentioned, then the appointment makes use of the Scheduler timeZone or System timeZone.
endTime	Binds the appointment end time field name which is mandatory and also its related validation rules.
endTimeZone	Binds the name of the end timezone field in the dataSource and also its related validation rules. If the endTimeZone field is not mentioned, then the appointment makes use of the Scheduler timeZone or System timeZone.
subject	Binds the appointment subject field name which holds the summary of the appointment and also its related validation rules.
location	Binds the name of the location field and also its related validation rules. It indicates the appointment location/occurrence place. This field needs to be bind to the Scheduler, when an API showLocationField is set to true.
description	Binds the appointment description field name and also its related validation rules.

allDay	Binds the name of the allDay field. It accepts the boolean value and indicates whether the appointment is an all-day appointment or not.
categorize	Binds the name of the categorize field and also its related validation rules. It indicates the category or status value (red categorize, green, yellow and so on).
priority	Binds the name of the priority field, its related validation rules and also indicates the priority (high, low, medium and none) of the appointments. This field should be bind to the Scheduler, when prioritySettings.enable is set to true.
resourceFields	Binds one or more fields in the resource collection. It maps the resource field names with the appointments, denoting to which resource the appointments actually belongs.
recurrence	Binds the name of the recurrence field. It accepts the boolean value and indicates whether the appointment is a recurrence appointment or not.
recurrenceRule	Binds the name of the recurrenceRule field. It holds the recurrence pattern associated with the appointments.
recurrenceId	Binds the recurrence Id field which acts as a parent id for Scheduler recurrence appointments.
recurrenceExDate	Binds the recurrence Exception field which accepts the recurrence Exception date values.

The below example depicts the appointment fields accepting the string type mapper fields,

#### HTML

```
var dManager = [{
  Id: 1,
  Subject: "Music Class",
  StartTime: new Date("2015/11/7 06:00 AM"),
  StartTimeZone: "UTC +05:30",
  EndTime: new Date("2015/11/7 07:00 AM"),
  EndTimeZone: "UTC +05:30",
  Description: "Never Give up on Obstacles",
  location: "US",
  AllDay: false,
  Recurrence: true,
  RecurrenceRule: "FREQ=WEEKLY;BYDAY=MO,TU;INTERVAL=1;COUNT=15",
  Categorize: "1",
  Priority: "medium",
  OwnerId: 3,
  RecurrenceId: 1,
  RecurrenceExDate: null
}];
var categorizeData = { enable: true };
var priorityData = { enable: true };
```

```

var groupData = { resources: ["Owners"] };
var ownerData = {
  dataSource: [
    { text: "Nancy", id: 1, color: "#f8a398" },
    { text: "Steven", id: 3, color: "#56ca85" },
    { text: "Michael", id: 5, color: "#51a0ed" }
  ],
  text: "text", id: "id", color: "color"
};
var Scheduler = React.createClass({
  render: function () {
    return (
      <EJ.Schedule id="Schedule1" width="100%" height="525px" currentDate={new
Date(2017, 5, 5)} showLocationField={true}
categorizeSettings={categorizeData} prioritySettings={priorityData}
group={groupData} resources={resourceData} appointmentSettings-
dataSource={dManager} appointmentSettings-id="Id" appointmentSettings-
subject="Subject" appointmentSettings-startTime="StartTime"
appointmentSettings-endTime="EndTime" appointmentSettings-
description="Description" appointmentSettings-allDay="AllDay"
appointmentSettings-recurrence="Recurrence" appointmentSettings-
recurrenceRule="RecurrenceRule" appointmentSettings-resourceFields=
"OwnerId">
      <resources>
        <resource allowMultiple={true} field="OwnerId" title="Owner" name="Owners"
resourceSettings={ownerData}></resource>
      </resources>
    </EJ.Schedule>
  );
}
});
ReactDOM.render(<Scheduler />, document.getElementById('schedule-default'));

```

### Appointment Field Validation

It is possible to validate the required fields of the appointment window from client-side before submitting it, by adding appropriate validation rules to each fields. The appointment fields have been extended to accept both String and object type values. Therefore, in order to perform validations, it is necessary to specify object values for the appointment fields.

Refer the appointment fields specified with validation rules from the following code example.

### HTML

```

var dManager =
ej.DataManager(window.Default).executeLocal(ej.Query().take(10));
var categorizeData = {
  enable: true,
  allowMultiple: false,
  dataSource: [
    { text: "Blue Category", id: 1, color: "#43b496", fontColor: "#ffffff" },
    { text: "Green Category", id: 2, color: "#7f993e", fontColor: "#ffffff" },
    { text: "Orange Category", id: 3, color: "#cc8638", fontColor: "#ffffff" },
    { text: "Purple Category", id: 4, color: "#ab54a0", fontColor: "#ffffff" },
    { text: "Red Category", id: 5, color: "#dd654e", fontColor: "#ffffff" },
    { text: "Yellow Category", id: 6, color: "#d0af2b", fontColor: "#ffffff" }
  ],

```

```

text: "text", id: "id", color: "color", fontColor: "fontColor"
};
var appointmentSettingsData = {
id: "Id",
dataSource: dManager,
subject: { field: "Subject", validationRules: { required: true } },
location: { field: "Location", validationRules: { required: true,
customRule: "/^[a-zA-Z0-9- ]*$/" } },
startTime: { field: "StartTime", validationRules: { required: true } },
endTime: { field: "EndTime", validationRules: { required: true } },
description: { field: "Description", validationRules: { required: true,
minlength: 5, maxlength: 500 } },
allDay: "AllDay",
recurrence: "Recurrence",
recurrenceRule: "RecurrenceRule",
categorize: { field: "Categorize", validationRules: { required: true,
messages: { required: "Categories are required." } } }
};
var Scheduler = React.createClass({
render: function () {
return (
<EJ.Schedule id="Schedule1" width="100%" height="525px" currentDate={new
Date(2017, 5, 5)} showLocationField={true}
categorizeSettings={categorizeData}
appointmentSettings={appointmentSettingsData}>
</EJ.Schedule>
);
}
});
ReactDOM.render(<Scheduler />, document.getElementById('schedule-default'));

```

### Binding to JSON Data Array

To bind the Scheduler events data as array of JSON objects, refer the below code example.

#### Example - Array of JSON Data Binding

##### HTML

```

var appointmentData = [
{
Id: 1,
Subject: "Music Class",
StartTime: new Date("2015/11/7 06:00 AM"),
EndTime: new Date("2015/11/7 07:00 AM")
}, {
Id: 2,
Subject: "School",
StartTime: new Date("2015/11/7 9:00 AM"),
EndTime: new Date("2015/11/7 02:30 PM")
}
];
var Scheduler = React.createClass({
render: function () {
return (
<EJ.Schedule id="Schedule1" width="100%" height="525px" currentDate={new
Date(2017, 5, 5)} appointmentSettings-dataSource={appointmentData}>

```

```

</EJ.Schedule>
);
}
});
ReactDOM.render(<Scheduler />, document.getElementById('schedule-default'));

```

### Binding Remote Data Service

The appointment data can be bound to the Scheduler through the [Odata](#) remote services, where the service URL is mapped with [Data manager](#) and then configured to the Schedule dataSource API.

#### HTML

```

var dataManager = ej.DataManager({
// referring data from remote service (url binding)
url: "http://mvc.syncfusion.com/OdataServices/Northwnd.svc/"
});
// query to fetch the records from the specified table "Events"
var queryManager = ej.Query().from("Events").take(10);
var Scheduler = React.createClass({
render: function () {
return (
<EJ.Schedule id="Schedule1" width="100%" height="525px" currentDate={new
Date(2017, 5, 5)} appointmentSettings-dataSource={dataManager}
appointmentSettings-query={queryManager}>
</EJ.Schedule>
);
}
});
ReactDOM.render(<Scheduler />, document.getElementById('schedule-default'));

```

### OData V4

The OData v4 is an improved version of OData protocols and the DataManager can also retrieve and consume appointment data from [OData v4](#) services.

#### HTML

```

var dataManager = ej.DataManager({
//OData v4 service
url: "http://services.odata.org/V4/Northwind/Northwind.svc/Orders/",
adaptor: new ej.ODataV4Adaptor()
});
var Scheduler = React.createClass({
render: function () {
return (
<EJ.Schedule id="Schedule1" width="100%" height="525px" currentDate={new
Date(2017, 5, 5)} appointmentSettings-dataSource={dataManager}
appointmentSettings-subject="ShipName" appointmentSettings-
startTime="OrderDate" appointmentSettings-endTime="RequiredDate"
appointmentSettings-description="ShipAddress">
</EJ.Schedule>
);
}
});
ReactDOM.render(<Scheduler />, document.getElementById('schedule-default'));

```

## WebAPI Binding

The Schedule appointment data can be bound through the Web API service and it is a programmatic interface to define the request and response messages system that is mostly exposed in **JSON** or **XML**.

### HTML

```
var dataManager = ej.DataManager({
// get the required appointments from Web API service
url: "http://mvc.syncfusion.com/OdataServices/api/ScheduleData/",
// enable cross domain
crossDomain: true
});
var Scheduler = React.createClass({
render: function () {
return (
<EJ.Schedule id="Schedule1" width="100%" height="525px" currentDate={new
Date(2017, 5, 5)} appointmentSettings-dataSource={dataManager}>
</EJ.Schedule>
);
}
});
ReactDOM.render(<Scheduler />, document.getElementById('schedule-default'));
```

The server-side code to retrieve the appointments are as follows.

### C#

```
// To retrieve the appointments from database and bind it to Scheduler
public IEnumerable<Event> GetData(String CurrentDate, String CurrentView,
String CurrentAction)
{
return new NORTHWNDEntities().Events.ToList();
}
```

## Data binding using OLEDB

The appointment data can also be bound to the Scheduler using OLEDB database as depicted below.

### HTML

```
var dataManager = ej.DataManager({
url: "Home/GetData", // This will trigger to bind the appointment data
initially to the Scheduler control
crudUrl: "Home/Batch", // This will trigger while performing CRUD operation
on the Scheduler appointments
adaptor: new ej.UrlAdaptor()
});
var Scheduler = React.createClass({
render: function () {
return (
<EJ.Schedule id="Schedule1" width="100%" height="525px" currentDate={new
Date(2017, 5, 5)} appointmentSettings-id="Id" appointmentSettings-
dataSource={dataManager} appointmentSettings-subject="Subject"
appointmentSettings-startTime="StartTime" appointmentSettings-
endtime="EndTime" appointmentSettings-startTimeZone="StartTimeZone"
appointmentSettings-endTimeZone="EndTimeZone" appointmentSettings-
description="Description" appointmentSettings-allDay="AllDay"

```

```

appointmentSettings-recurrence="Recurrence" appointmentSettings-
recurrenceRule="RecurrenceRule">
</EJ.Schedule>
);
}
});
ReactDOM.render(<Scheduler />, document.getElementById('schedule-default'));

```

The server-side controller code to retrieve and bind the appointment data to Scheduler are as follows. Also, define a class with all the required appointment fields as depicted in the below code example.

### C#

```

// Define a class with all appointment fields
public class ScheduleData
{
    public int Id { get; set; }
    public string Subject { get; set; }
    public DateTime StartTime { get; set; }
    public DateTime EndTime { get; set; }
    public Boolean AllDay { get; set; }
    public Boolean Recurrence { get; set; }
    public string RecurrenceRule { get; set; }
    public string StartTimeZone { get; set; }
    public string EndTimeZone { get; set; }
    public string Description { get; set; }
}
// To retrieve the appointments from database and bind it to Scheduler
public JsonResult GetData()
{
    // Mention your own dataSource to be used here.
    string strAccessConn = "Provider=Microsoft.Jet.OLEDB.4.0;Data
Source=|DataDirectory|/ScheduleDb.MDB";
    DataSet myDataSet = new DataSet();
    OleDbConnection myAccessConn = null;
    myAccessConn = new OleDbConnection(strAccessConn);
    OleDbCommand myAccessCommand = new OleDbCommand("SELECT * FROM
DefaultSchedule", myAccessConn);
    OleDbDataAdapter myDataAdapter = new OleDbDataAdapter(myAccessCommand);
    myAccessConn.Open();
    myDataAdapter.Fill(myDataSet, "DefaultSchedule");
    List<ScheduleData> datasource = new List<ScheduleData>();
    datasource = myDataSet.Tables[0].AsEnumerable().Select(dataRow => new
ScheduleData { Id = dataRow.Field<int>("Id"), Subject =
dataRow.Field<string>("Subject"), StartTime =
dataRow.Field<DateTime>("StartTime"), EndTime =
dataRow.Field<DateTime>("EndTime"), AllDay = dataRow.Field<bool>("AllDay"),
Recurrence = dataRow.Field<bool>("Recurrence"), RecurrenceRule =
dataRow.Field<string>("RecurrenceRule"), Description =
dataRow.Field<string>("Description"), StartTimeZone =
dataRow.Field<string>("StartTimeZone"), EndTimeZone =
dataRow.Field<string>("EndTimeZone") }).ToList();
    myAccessConn.Close();
    return Json(datasource, JsonRequestBehavior.AllowGet);
}

```



The control code to handle the CRUD operation are as follows.

### C#

```
public JsonResult Batch(EditParams param)
{
    // Mention your own dataSource to be used here.
    string strAccessConn = "Provider=Microsoft.Jet.OLEDB.4.0;Data
Source=|DataDirectory|/ScheduleDb.MDB";
    if (param.action == "insert" || (param.action == "batch" && param.added !=
null)) // this block of code will execute while inserting the appointments
    {
        var value = param.action == "insert" ? param.value : param.added[0];
        using (OleDbConnection myCon = new OleDbConnection(strAccessConn))
        {
            OleDbCommand cmd = new OleDbCommand();
            cmd.CommandType = CommandType.Text;
            cmd.CommandText = "INSERT INTO
DefaultSchedule (Subject, StartTime, EndTime, AllDay, Recurrence, RecurrenceRule, D
escription, StartTimeZone, EndTimeZone) VALUES
(@Subject, @StartTime, @EndTime, @AllDay, @Recurrence, @RecurrenceRule, @Descripti
on, @StartTimeZone, @EndTimeZone) ";
            if (string.IsNullOrEmpty(value.Subject))
            cmd.Parameters.AddWithValue("@Subject", DBNull.Value);
            else
            cmd.Parameters.AddWithValue("@Subject", value.Subject);
            cmd.Parameters.AddWithValue("@StartTime", value.StartTime);
            cmd.Parameters.AddWithValue("@EndTime", value.EndTime);
            cmd.Parameters.AddWithValue("@AllDay", value.AllDay);
            cmd.Parameters.AddWithValue("@Recurrence", value.Recurrence);
            if (string.IsNullOrEmpty(value.RecurrenceRule))
            cmd.Parameters.AddWithValue("@RecurrenceRule", DBNull.Value);
            else
            cmd.Parameters.AddWithValue("@RecurrenceRule", value.RecurrenceRule);
            if (string.IsNullOrEmpty(value.Description))
            cmd.Parameters.AddWithValue("@Description", DBNull.Value);
            else
            cmd.Parameters.AddWithValue("@Description", value.Description);
            if (string.IsNullOrEmpty(value.StartTimeZone))
            cmd.Parameters.AddWithValue("@StartTimeZone", DBNull.Value);
            else
            cmd.Parameters.AddWithValue("@StartTimeZone", value.StartTimeZone);
            if (string.IsNullOrEmpty(value.EndTimeZone))
            cmd.Parameters.AddWithValue("@EndTimeZone", DBNull.Value);
            else
            cmd.Parameters.AddWithValue("@EndTimeZone", value.EndTimeZone);
            cmd.Connection = myCon;
            myCon.Open();
            cmd.ExecuteNonQuery();
            myCon.Close();
        }
    }
    if (param.action == "remove" || param.deleted != null) // this block of
code will execute while removing the appointment
    {
        if (param.action == "remove")
        {

```

```

using (OleDbConnection myCon = new OleDbConnection(strAccessConn))
{
    myCon.Open();
    OleDbCommand cmd = new OleDbCommand("DELETE FROM DefaultSchedule WHERE Id = @Key", myCon);
    cmd.Parameters.AddWithValue("@Key", param.key);
    cmd.ExecuteNonQuery();
    myCon.Close();
}
}
else
{
    foreach (var apps in param.deleted)
    {
        using (OleDbConnection myCon = new OleDbConnection(strAccessConn))
        {
            myCon.Open();
            OleDbCommand cmd = new OleDbCommand("DELETE FROM DefaultSchedule WHERE Id = @Key", myCon);
            cmd.Parameters.AddWithValue("@Key", apps.Id);
            cmd.ExecuteNonQuery();
            myCon.Close();
        }
    }
}
if ((param.action == "batch" && param.changed != null) || param.action == "update") // this block of code will execute while updating the appointment
{
    var value = param.action == "update" ? param.value : param.changed[0];
    using (OleDbConnection myCon = new OleDbConnection(strAccessConn))
    {
        myCon.Open();
        OleDbCommand cmd = new OleDbCommand("UPDATE DefaultSchedule SET Subject=@Subject, StartTime=@StartTime, EndTime=@EndTime, AllDay=@AllDay, Recurrence=@Recurrence, RecurrenceRule=@RecurrenceRule, Description=@Description, StartTimeZone=@StartTimeZone, EndTimeZone=@EndTimeZone WHERE Id = @Key", myCon);
        if (string.IsNullOrEmpty(value.Subject))
            cmd.Parameters.AddWithValue("@Subject", DBNull.Value);
        else
            cmd.Parameters.AddWithValue("@Subject", value.Subject);
        cmd.Parameters.AddWithValue("@StartTime", value.StartTime);
        cmd.Parameters.AddWithValue("@EndTime", value.EndTime);
        cmd.Parameters.AddWithValue("@AllDay", value.AllDay);
        cmd.Parameters.AddWithValue("@Recurrence", value.Recurrence);
        if (string.IsNullOrEmpty(value.RecurrenceRule))
            cmd.Parameters.AddWithValue("@RecurrenceRule", DBNull.Value);
        else
            cmd.Parameters.AddWithValue("@RecurrenceRule", value.RecurrenceRule);
        if (string.IsNullOrEmpty(value.Description))
            cmd.Parameters.AddWithValue("@Description", DBNull.Value);
        else
            cmd.Parameters.AddWithValue("@Description", value.Description);
        if (string.IsNullOrEmpty(value.StartTimeZone))
            cmd.Parameters.AddWithValue("@StartTimeZone", DBNull.Value);
    }
}

```

```

else
cmd.Parameters.AddWithValue("@StartTimeZone", value.StartTimeZone);
if (string.IsNullOrEmpty(value.EndTimeZone))
cmd.Parameters.AddWithValue("@EndTimeZone", DBNull.Value);
else
cmd.Parameters.AddWithValue("@EndTimeZone", value.EndTimeZone);
cmd.Parameters.AddWithValue("@Key", value.Id);
cmd.ExecuteNonQuery();
myCon.Close();
}
}
OleDbConnection myAccessConn = new OleDbConnection(strAccessConn);
OleDbCommand myAccessCommand = new OleDbCommand("SELECT * FROM
DefaultSchedule", myAccessConn);
OleDbDataAdapter myDataAdapter = new OleDbDataAdapter(myAccessCommand);
DataSet myDataSet = new DataSet();
myAccessConn.Open();
myDataAdapter.Fill(myDataSet, "DefaultSchedule");
List<ScheduleData> datasource = new List<ScheduleData>();
datasource = myDataSet.Tables[0].AsEnumerable().Select(dataRow => new
ScheduleData { Id = dataRow.Field<int>("Id"), Subject =
dataRow.Field<string>("Subject"), StartTime =
dataRow.Field<DateTime>("StartTime"), EndTime =
dataRow.Field<DateTime>("EndTime"), AllDay = dataRow.Field<bool>("AllDay"),
Recurrence = dataRow.Field<bool>("Recurrence"), RecurrenceRule =
dataRow.Field<string>("RecurrenceRule"), Description =
dataRow.Field<string>("Description"), StartTimeZone =
dataRow.Field<string>("StartTimeZone"), EndTimeZone =
dataRow.Field<string>("EndTimeZone") }).ToList();
myAccessConn.Close();
return Json(datasource, JsonRequestBehavior.AllowGet);
}
// Class definition for EditParams to be used as parameter in the above Crud
method for receiving the object value in it.
public class EditParams
{
public string key { get; set; }
public string action { get; set; }
public List<ScheduleData> added { get; set; }
public List<ScheduleData> changed { get; set; }
public List<ScheduleData> deleted { get; set; }
public ScheduleData value { get; set; }
}

```

### ASP.NET Web Method Binding

The Schedule appointment data can retrieve data from ASP.NET Web methods by making use of the UrlAdaptor of ejDataManager.

### HTML

```

var dataManager = ej.DataManager({
url: "WebService1.aspx/GetData", // This will trigger to bind the
appointments data to schedule control
batchUrl: "WebService1.aspx/Crud", // This will trigger while saving the
appointment through detail window

```

```

insertUrl: "WebService1.asmx/add", // This will trigger while saving the
appointment through quick window
updateUrl: "WebService1.asmx/update", //This will trigger while saving the
resize or drag and drop the appointment
removeUrl: "WebService1.asmx/remove", // This will trigger to delete the
single appointment
adaptor: new ej.WebMethodAdaptor()
});
var Scheduler = React.createClass({
render: function () {
return (
<EJ.Schedule id="Schedule1" width="100%" height="525px" currentDate={new
Date(2017, 5, 5)} appointmentSettings-id="Id" appointmentSettings-
dataSource={dataManager}>
</EJ.Schedule>
);
}
});
ReactDOM.render(<Scheduler />, document.getElementById('schedule-default'));

```

The server-side code to handle the CRUD operations are as follows.

### C#

```

// To retrieve the appointments and bind it to Scheduler
[WebMethod]
[ScriptMethod(ResponseFormat = ResponseFormat.Json)]
public static object GetData(String CurrentView, String CurrentAction,
DateTime CurrentDate)
{
// ScheduleAppointmentsObjData is a user-defined class needs to be defined
with collection of scheduler appointments
ScheduleAppointmentsObjDatum obj = new ScheduleAppointmentsObjDatum();
IList<ScheduleAppointmentsObjData> appoint = obj.GetRecords().ToList();
return appoint;
}
public class ScheduleAppointmentsObjDatum
{
[DataObjectMethod(DataObjectMethodType.Select)]
public List<ScheduleAppointmentsObjData> GetRecords()
{
List<ScheduleAppointmentsObjData> list = new
List<ScheduleAppointmentsObjData>();
list.Add(new ScheduleAppointmentsObjData(100, "Bering Sea Gold", "Chennai",
"05/02/2014 09:00:00 AM", "05/02/2014 10:30:00 AM", "", "1", "", true, "",
"", "", "", false, "", "", "FREQ=DAILY;INTERVAL=2;COUNT=10"));
list.Add(new ScheduleAppointmentsObjData(101, "Bering Sea Gold", "mum",
"05/02/2014 04:00:00 AM", "05/02/2014 05:00:00 AM", "", "1", "", false, "",
"", "", "", false, "", "", ""));
list.Add(new ScheduleAppointmentsObjData(102, "Bering Sea Gold", "Trichy",
"05/02/2014 04:00:00 PM", "05/02/2014 05:30:00 PM", "", "1", "", false, "",
"", "", "", false, "", "", ""));
list.Add(new ScheduleAppointmentsObjData(103, "What Happened Next?",
"Chennai", "05/04/2014 03:00:00 AM", "05/04/2014 04:30:00 AM", "", "1", "",
false, "", "", "", false, "", "", ""));
}
}

```

```

list.Add(new ScheduleAppointmentsObjData(104, "Bering Sea Gold", "Trichy",
"05/04/2014 05:00:00 AM", "05/04/2014 05:40:00 AM", "", "1", "", false, "",
"", "", "", false, "", "", ""));
list.Add(new ScheduleAppointmentsObjData(105, "Daily Planet", "Chennai",
"05/03/2014 01:00:00 AM", "05/03/2014 02:00:00 AM", "", "1", "", false, "",
"", "", "", false, "", "", ""));
list.Add(new ScheduleAppointmentsObjData(106, "Alaska: The Last Frontier",
"Chennai", "05/03/2014 08:00:00 AM", "05/03/2014 09:00:00 AM", "", "1", "",
false, "", "", "", false, "", "", ""));
list.Add(new ScheduleAppointmentsObjData(107, "How It's Made", "Chennai",
"05/01/2014 06:00:00 AM", "05/01/2014 06:30:00 AM", "", "1", "", true, "",
"", "", "", false, "", "", "FREQ=WEEKLY;BYDAY=MO,TU;INTERVAL=1;COUNT=15"));
list.Add(new ScheduleAppointmentsObjData(108, "Dearest Catch", "Chennai",
"05/03/2014 04:00:00 PM", "05/03/2014 05:00:00 PM", "", "1", "", false, "",
"", "", "", false, "", "", ""));
list.Add(new ScheduleAppointmentsObjData(109, "MayDay", "Chennai",
"04/30/2014 06:30:00 AM", "04/30/2014 07:30:00 AM", "", "1", "", false, "",
"", "", "", false, "", "", ""));
list.Add(new ScheduleAppointmentsObjData(110, "MoonShiners", "Chennai",
"05/02/2014 02:00:00 AM", "05/02/2014 02:30:00 AM", "", "1", "", true, "",
"", "", "", false, "", "", "FREQ=DAILY;INTERVAL=1;COUNT=5"));
list.Add(new ScheduleAppointmentsObjData(111, "Close Encounters", "Chennai",
"04/30/2014 02:00:00 PM", "04/30/2014 03:00:00 PM", "", "1", "", true, "",
"", "", "", false, "", "", "FREQ=WEEKLY;BYDAY=MO,TH;INTERVAL=1;COUNT=5"));
list.Add(new ScheduleAppointmentsObjData(112, "Close Encounters", "Mumbai",
"04/30/2014 03:00:00 AM", "04/30/2014 03:30:00 AM", "", "1", "", true, "",
"", "", "", false, "", "", "FREQ=WEEKLY;BYDAY=WE;INTERVAL=1;COUNT=3"));
list.Add(new ScheduleAppointmentsObjData(113, "Highway Through Hell",
"Chennai", "05/01/2014 03:00:00 AM", "05/01/2014 07:00:00 AM", "", "1", "",
true, "", "", "", false, "", "", "FREQ=DAILY;INTERVAL=2;COUNT=10"));
list.Add(new ScheduleAppointmentsObjData(114, "Moon Shiners", "Chennai",
"05/02/2014 04:20:00 AM", "05/02/2014 05:50:00 AM", "", "1", "", false, "",
"", "", "", false, "", "", ""));
list.Add(new ScheduleAppointmentsObjData(115, "Cash Cab", "Chennai",
"04/30/2014 03:00:00 PM", "04/30/2014 04:30:00 PM", "", "1", "", true, "",
"", "", "", false, "", "", "FREQ=DAILY;INTERVAL=1;COUNT=5"));
return list;
}
}
[Serializable]
public class ScheduleAppointmentsObjData
{
private int _id;
private String _subject;
private String _location;
private String _startTime;
private String _endTime;
private String _description;
private String _owner;
private String _priority;
private Boolean _recurrence;
private String _recurrenceType;
private String _recurrenceTypeCount;
private String _remainderCategorize;
private String _customStyle;
private Boolean _allDay;
private String _recurrenceStartDate;

```

```

private String _recurrenceEndDate;
private String _recurrenceRule;
public ScheduleAppointmentsObjData()
{
}
public ScheduleAppointmentsObjData(int _id, string _subject, string
_location, string _startTime, string _endTime, string _description, string
_owner, string _priority, bool _recurrence, string _recurrenceType, string
_recurrenceTypeCount, string _remainderCategorize, string _customStyle, bool
_allDay, string _recurrenceStartDate, string _recurrenceEndDate, string
_recurrenceRule)
{
this._id = _id;
this._subject = _subject;
this._location = _location;
this._startTime = _startTime;
this._endTime = _endTime;
this._description = _description;
this._owner = _owner;
this._priority = _priority;
this._recurrence = _recurrence; ;
this._recurrenceType = _recurrenceType;
this._recurrenceTypeCount = _recurrenceTypeCount;
this._remainderCategorize = _remainderCategorize;
this._customStyle = _customStyle;
this._allDay = _allDay;
this._recurrenceStartDate = _recurrenceStartDate;
this._recurrenceEndDate = _recurrenceEndDate;
this._recurrenceRule = _recurrenceRule;
}
public int ID
{
get
{
return _id;
}
set
{
_id = value;
}
}
public string Subject
{
get
{
return _subject;
}
set
{
_subject = value;
}
}
public string Location
{
get
{
return _location;
}
}

```

```
}
set
{
    _location = value;
}
}
public string StartTime
{
    get
    {
        return _startTime;
    }
    set
    {
        _startTime = value;
    }
}
public string EndTime
{
    get
    {
        return _endTime;
    }
    set
    {
        _endTime = value;
    }
}
public string Description
{
    get
    {
        return _description;
    }
    set
    {
        _description = value;
    }
}
public string Owner
{
    get
    {
        return _owner;
    }
    set
    {
        _owner = value;
    }
}
public string Priority
{
    get
    {
        return _priority;
    }
    set
```

```
{
    _priority = value;
}
}
public Boolean Recurrence
{
    get
    {
        return _recurrence;
    }
    set
    {
        _recurrence = value;
    }
}
public string RecurrenceType
{
    get
    {
        return _recurrenceType;
    }
    set
    {
        _recurrenceType = value;
    }
}
public string RecurrenceTypeCount
{
    get
    {
        return _recurrenceTypeCount;
    }
    set
    {
        _recurrenceTypeCount = value;
    }
}
public string RemainderCategorize
{
    get
    {
        return _remainderCategorize;
    }
    set
    {
        _remainderCategorize = value;
    }
}
public string CustomStyle
{
    get
    {
        return _customStyle;
    }
    set
    {
        _customStyle = value;
    }
}
```



```
}
}
public Boolean AllDay
{
    get
    {
        return _allDay;
    }
    set
    {
        _allDay = value;
    }
}
public string RecurrenceStartDate
{
    get
    {
        return _recurrenceStartDate;
    }
    set
    {
        _recurrenceStartDate = value;
    }
}
public string RecurrenceEndDate
{
    get
    {
        return _recurrenceEndDate;
    }
    set
    {
        _recurrenceEndDate = value;
    }
}
public string RecurrenceRule
{
    get
    {
        return _recurrenceRule;
    }
    set
    {
        _recurrenceRule = value;
    }
}
}
// Method to retrieve appointments from the ScheduleAppointmentsObjDatum Class
public static IList<ScheduleAppointmentsObjData> GetAllRecords()
{
    ScheduleAppointmentsObjDatum obj = new ScheduleAppointmentsObjDatum();
    IList<ScheduleAppointmentsObjData> appoint = obj.GetRecords().ToList();
    return appoint;
}
// Method to convert appointment object of dictionary type into valid format.
```

```

public static ScheduleAppointmentsObjData GetObjectValue(Dictionary<string,
object> objValue)
{
    Dictionary<string, object> KeyVal = objValue;
    ScheduleAppointmentsObjData appointValue = new
    ScheduleAppointmentsObjData();
    foreach (KeyValuePair<string, object> keyValue in KeyVal)
    {
        if (keyValue.Key == "ID")
            appointValue.ID = Convert.ToInt32(keyValue.Value);
        else if (keyValue.Key == "Subject")
            appointValue.Subject = Convert.ToString(keyValue.Value);
        else if (keyValue.Key == "Location")
            appointValue.Location = Convert.ToString(keyValue.Value);
        else if (keyValue.Key == "StartTime")
            appointValue.StartTime =
            Convert.ToDateTime(keyValue.Value).ToString("MM '/' dd '/' yyyy hh:mm:ss tt");
        else if (keyValue.Key == "EndTime")
            appointValue.EndTime =
            Convert.ToDateTime(keyValue.Value).ToString("MM '/' dd '/' yyyy hh:mm:ss tt");
        else if (keyValue.Key == "Description")
            appointValue.Description = Convert.ToString(keyValue.Value);
        else if (keyValue.Key == "AllDay")
            appointValue.AllDay = Convert.ToBoolean(keyValue.Value);
        else if (keyValue.Key == "Recurrence")
            appointValue.Recurrence = Convert.ToBoolean(keyValue.Value);
        else if (keyValue.Key == "RecurrenceRule")
            appointValue.RecurrenceRule = Convert.ToString(keyValue.Value);
    }
    return appointValue;
}

// Triggers while creating appointment through quick window.
[WebMethod]
[ScriptMethod(ResponseFormat = ResponseFormat.Json)]
public static void add(object value)
{
    ScheduleAppointmentsObjData appointValue = GetObjectValue(value as
    Dictionary<string, object>);
    GetAllRecords().Insert(0, appointValue);
}

// Triggers while editing the appointments.
[WebMethod]
[ScriptMethod(ResponseFormat = ResponseFormat.Json)]
public static void update(object value)
{
    ScheduleAppointmentsObjData obj = GetObjectValue(value as Dictionary<string,
    object>);
    var filterData = GetAllRecords().ToList().Where(c => c.ID ==
    Convert.ToInt32(obj.ID));
    if (filterData.Count() > 0)
    {
        ScheduleAppointmentsObjData appoint = GetAllRecords().Single(A => A.ID ==
        Convert.ToInt32(obj.ID));
        appoint.StartTime = obj.StartTime;
        appoint.EndTime = obj.EndTime;
        appoint.Subject = obj.Subject;
        appoint.Recurrence = obj.Recurrence;
    }
}

```

```

appoint.AllDay = obj.AllDay;
appoint.RecurrenceRule = obj.RecurrenceRule;
}
}
// Triggers on deleting an appointment.
[WebMethod]
[ScriptMethod(ResponseFormat = ResponseFormat.Json)]
public static void remove(int key)
{
    ScheduleAppointmentsObjData removeApp = GetAllRecords().Where(c => c.ID ==
key).FirstOrDefault();
    if (removeApp != null)
        GetAllRecords().Remove(removeApp);
}
// Triggers for all CRUD actions on Scheduler appointments.
[WebMethod]
[ScriptMethod(ResponseFormat = ResponseFormat.Json)]
public static void Crud(List<object> added, List<object> changed,
List<object> deleted)
{
    if (added != null && added.Count > 0)
    {
        ScheduleAppointmentsObjData appointValue = GetObjectValue(added[0] as
Dictionary<string, object>);
        GetAllRecords().Insert(0, appointValue);
    }
    if (changed != null && changed.Count > 0)
    {
        ScheduleAppointmentsObjData value = GetObjectValue(changed[0] as
Dictionary<string, object>);
        var filterData = GetAllRecords().ToList().Where(c => c.ID ==
Convert.ToInt32(value.ID));
        if (filterData.Count() > 0)
        {
            ScheduleAppointmentsObjData appoint = GetAllRecords().Where(A => A.ID ==
Convert.ToInt32(value.ID)).FirstOrDefault();
            appoint.StartTime = value.StartTime;
            appoint.EndTime = value.EndTime;
            appoint.Subject = value.Subject;
            appoint.Recurrence = value.Recurrence;
            appoint.AllDay = value.AllDay;
            appoint.RecurrenceRule = value.RecurrenceRule;
        }
    }
    if (deleted != null && deleted.Count > 0)
    {
        foreach (var delete in deleted)
        {
            ScheduleAppointmentsObjData value = GetObjectValue(delete as
Dictionary<string, object>);
            ScheduleAppointmentsObjData removeApp = GetAllRecords().Where(c => c.ID ==
value.ID).FirstOrDefault();
            if (removeApp != null)
                GetAllRecords().Remove(removeApp);
        }
    }
}

```

### MVC Controller Action Binding

The Schedule appointment data can retrieve data from MVC controller. This can be achieved by using the [UrlAdaptor](#) of [ej.DataManager](#).

### HTML

```
var dataManager = ej.DataManager({
  url: "Home/GetData", // This will trigger to bind the appointments data to
  schedule control
  batchUrl: "Home/Crud", // This will trigger while saving the appointment
  through detail window
  insertUrl: "Home/add", // This will trigger while saving the appointment
  through quick window
  updateUrl: "Home/update", //This will trigger while saving the resize or
  drag and drop the appointment
  removeUrl: "Home/remove", // This will trigger to delete the single
  appointment
  adaptor: new ej.UrlAdaptor()
});
var Scheduler = React.createClass({
  render: function () {
    return (
      <EJ.Schedule id="Schedule1" width="100%" height="525px" currentDate={new
      Date(2017, 5, 5)} appointmentSettings-id="Id" appointmentSettings-
      dataSource={dataManager}>
      </EJ.Schedule>
    );
  }
});
ReactDOM.render(<Scheduler />, document.getElementById('schedule-default'));
```

The server-side controller code to handle the CRUD operations are as follows.

### C#

```
// To initially bind the appointments with Scheduler
public JsonResult GetData()
{
  // ScheduleDataDataContext is a LINQ-to-SQL data class name that is defined
  in the .dbml file to access the tables from the database
  IEnumerable data = new ScheduleDataDataContext().Appointments.Take(100);
  return Json(data, JsonRequestBehavior.AllowGet);
}
// Triggers while saving a new appointment through quick window
public JsonResult add(Appointment value)
{
  ScheduleDataDataContext db = new ScheduleDataDataContext();
  int intMax = db.Appointments.ToList().Count > 0 ?
  db.Appointments.ToList().Max(p => p.Id) : 1;
  Appointment appoint = new Appointment()
  {
    Id = intMax + 1,
    StartTime = value.StartTime,
    EndTime = value.EndTime,
    Subject = value.Subject,
```

```

Description = value.Description,
OwnerId = value.OwnerId,
Recurrence = value.Recurrence,
AllDay = value.AllDay,
RecurrenceRule = value.RecurrenceRule
};
db.Appointments.InsertOnSubmit(appoint);
db.SubmitChanges();
IEnumerable data = new ScheduleDataDataContext().Appointments.Take(100);
return Json(data, JsonRequestBehavior.AllowGet);
}
// Triggers while editing/dragging/resizing the existing appointment
public JsonResult update(Appointment value)
{
    ScheduleDataDataContext db = new ScheduleDataDataContext();
    var filterData = db.Appointments.Where(c => c.Id ==
    Convert.ToInt32(value.Id));
    Appointment appoint = db.Appointments.Single(A => A.Id ==
    Convert.ToInt32(value.Id));
    if (filterData.Count() > 0)
    {
        DateTime startTime = Convert.ToDateTime(value.StartTime);
        DateTime endTime = Convert.ToDateTime(value.EndTime);
        appoint.StartTime = startTime;
        appoint.EndTime = endTime;
        appoint.Subject = value.Subject;
        appoint.Description = value.Description;
        appoint.OwnerId = value.OwnerId;
        appoint.Recurrence = Convert.ToByte(value.Recurrence);
        appoint.AllDay = value.AllDay;
        appoint.RecurrenceRule = value.RecurrenceRule;
    }
    db.SubmitChanges();
    IEnumerable data = new ScheduleDataDataContext().Appointments.Take(100);
    return Json(data, JsonRequestBehavior.AllowGet);
}
// Triggers when an appointment is deleted
public JsonResult remove(string key)
{
    ScheduleDataDataContext db = new ScheduleDataDataContext();
    Appointment app = db.Appointments.Where(c => c.Id ==
    Convert.ToInt32(key)).FirstOrDefault();
    if (app != null) db.Appointments.DeleteOnSubmit(app);
    db.SubmitChanges();
    IEnumerable data = new ScheduleDataDataContext().Appointments.Take(100);
    return Json(data, JsonRequestBehavior.AllowGet);
}
// Triggers for any of the Scheduler CRUD operation
public JsonResult Crud(EditParams param)
{
    ScheduleDataDataContext db = new ScheduleDataDataContext();
    if (param.action == "insert" || (param.action == "batch" && param.added !=
    null)) // this block of code will execute while inserting the appointments
    {
        var value = param.action == "insert" ? param.value : param.added[0];
        int intMax = db.Appointments.ToList().Count > 0 ?
        db.Appointments.ToList().Max(p => p.Id) : 1;

```

```

DateTime startTime = Convert.ToDateTime(value.StartTime);
DateTime endTime = Convert.ToDateTime(value.EndTime);
Appointment appoint = new Appointment()
{
    Id = intMax + 1,
    StartTime = startTime,
    EndTime = endTime,
    Subject = value.Subject,
    Description = value.Description,
    OwnerId = value.OwnerId,
    Recurrence = value.Recurrence,
    AllDay = value.AllDay,
    RecurrenceRule = value.RecurrenceRule
};
db.Appointments.InsertOnSubmit(appoint);
db.SubmitChanges();
}
else if (param.action == "remove") // this block of code will execute while
removing the appointment
{
    Appointment app = db.Appointments.Where(c => c.Id ==
Convert.ToInt32(param.key)).FirstOrDefault();
    if (app != null) db.Appointments.DeleteOnSubmit(app);
    db.SubmitChanges();
}
else if ((param.action == "batch" && param.changed != null) || param.action
== "update") // this block of code will execute while updating the
appointment
{
    var value = param.action == "update" ? param.value : param.changed[0];
    var filterData = db.Appointments.Where(c => c.Id ==
Convert.ToInt32(value.Id));
    if (filterData.Count() > 0)
    {
        DateTime startTime = Convert.ToDateTime(value.StartTime);
        DateTime endTime = Convert.ToDateTime(value.EndTime);
        Appointment appoint = db.Appointments.Single(A => A.Id ==
Convert.ToInt32(value.Id));
        appoint.StartTime = startTime.ToUniversalTime();
        appoint.EndTime = endTime.ToUniversalTime();
        appoint.Subject = value.Subject;
        appoint.Description = value.Description;
        appoint.OwnerId = value.OwnerId;
        appoint.Recurrence = Convert.ToByte(value.Recurrence);
        appoint.AllDay = value.AllDay;
        appoint.RecurrenceRule = value.RecurrenceRule;
    }
    db.SubmitChanges();
}
IEnumerable data = new ScheduleDataContext().Appointments.Take(500);
return Json(data, JsonRequestBehavior.AllowGet);
}
// Class definition for EditParams to be used as parameter in the above Crud
method for receiving the object value in it.
public class EditParams
{
    public string key { get; set; }
}

```

```

public string action { get; set; }
public List<Appointment> added { get; set; }
public List<Appointment> changed { get; set; }
public Appointment value { get; set; }
}

```

### Loading Data on Demand

Load on demand feature allows the Scheduler to retrieve only the filtered appointment data (for the current Scheduler date range) from the service/database during **loading time**, and that too only for the current Scheduler view. There are 3 parameters made available on the server-side namely **CurrentDate**, **CurrentView** and **CurrentAction** through which only the necessary appointments are retrieved from the database and then assigned to the Scheduler dataSource. With this kind of Scheduler action, consuming only lesser data will reduce the usage of network bandwidth size and loading time.

The **enableLoadOnDemand** property is used to enable or disable the load on demand functionality of the schedule.

### HTML

```

var dataManager = ej.DataManager({
// get the required appointments from service
url: "http://mvc.syncfusion.com/OdataServices/api/ScheduleData/",
crossDomain: true
});
var Scheduler = React.createClass({
render: function () {
return (
<EJ.Schedule id="Schedule1" width="100%" height="525px"
enableLoadOnDemand={true} currentDate={new Date(2017, 5, 5)}
appointmentSettings-id="Id" appointmentSettings-dataSource={dataManager}>
</EJ.Schedule>
);
}
});
ReactDOM.render(<Scheduler />, document.getElementById('schedule-default'));

```

The server-side code to handle the load on demand is as follows.

### C#

```

// retrieve the appointments based on the current date.
public IEnumerable<Event> GetData(String CurrentDate, String CurrentView,
String CurrentAction)
{
var dateString = Regex.Match(CurrentDate.ToString(),
@"^(\\w+\\b.*?){4}").ToString();
string format = "ddd MMM dd yyyy";
DateTime dateTimeValue;
try
{
dateTimeValue = DateTime.ParseExact(dateString, format,
CultureInfo.InvariantCulture);
}
catch (FormatException)
{
}
}

```

```

var dateSplit = CurrentDate.Split(' '); //For IE<=10 Fri Mar 20 11:00:22 UTC+0530 2015
if (dateSplit[2].Length == 1) dateSplit[2] = string.Concat("0", dateSplit[2]);
dateString = string.Concat(dateSplit[0], ' ', dateSplit[1], ' ', dateSplit[2], ' ', dateSplit[dateSplit.Length - 1]);
dateTimeValue = DateTime.ParseExact(dateString, format, CultureInfo.InvariantCulture);
}
// AppointmentDeposit is a user-defined class within which the FilterAppointment method is defined.
AppointmentDeposit rep = new AppointmentDeposit();
var data = rep.FilterAppointment(dateTimeValue, CurrentAction, CurrentView);
return data;
}
// Method to filter the appointments based on the date range
public List<Event> FilterAppointment(DateTime CurrentDate, String CurrentAction, String CurrentView)
{
    DateTime CurrDate = Convert.ToDateTime(CurrentDate);
    DateTime StartDate = FirstWeekDate(CurrDate.Date);
    DateTime EndDate = FirstWeekDate(CurrDate.Date);
    List<Event> appointmentList = new NORTHWNDEntities().Events.ToList();
    switch (CurrentView)
    {
        case "day":
            StartDate = CurrentDate;
            EndDate = CurrentDate;
            break;
        case "week":
            EndDate = EndDate.AddDays(7);
            break;
        case "workweek":
            EndDate = EndDate.AddDays(5);
            break;
        case "month":
            StartDate = CurrDate.Date.AddDays(-CurrDate.Day + 1);
            EndDate = StartDate.AddMonths(1);
            break;
    }
    appointmentList = new NORTHWNDEntities().Events.ToList().Where(app =>
        ((Convert.ToDateTime(app.StartTime).Date >=
        Convert.ToDateTime(StartDate.Date)) &&
        (Convert.ToDateTime(app.EndTime).Date <=
        Convert.ToDateTime(EndDate.Date)))).ToList();
    return appointmentList;
}
internal static DateTime FirstWeekDate(DateTime CurrentDate)
{
    try
    {
        DateTime FirstDayOfWeek = CurrentDate;
        DayOfWeek WeekDay = FirstDayOfWeek.DayOfWeek;
        switch (WeekDay)
        {
            case DayOfWeek.Sunday:
                break;

```



```

case DayOfWeek.Monday:
FirstDayOfWeek = FirstDayOfWeek.AddDays(-1);
break;
case DayOfWeek.Tuesday:
FirstDayOfWeek = FirstDayOfWeek.AddDays(-2);
break;
case DayOfWeek.Wednesday:
FirstDayOfWeek = FirstDayOfWeek.AddDays(-3);
break;
case DayOfWeek.Thursday:
FirstDayOfWeek = FirstDayOfWeek.AddDays(-4);
break;
case DayOfWeek.Friday:
FirstDayOfWeek = FirstDayOfWeek.AddDays(-5);
break;
case DayOfWeek.Saturday:
FirstDayOfWeek = FirstDayOfWeek.AddDays(-6);
break;
}
return (FirstDayOfWeek);
}
catch
{
return DateTime.Now;
}
}

```

## Views

The number of days and its associated appointments are usually grouped together in Scheduler to organize different views. The available view options in Scheduler are as follows,

- Day
- Week
- Workweek
- Month
- Custom View
- Agenda
- Timeline View

Usually these view options are displayed as a toolbar in the date-header section of the Schedule control. The items within the views toolbar can be added/removed based on the value passed to the [views](#) property.

By default, the Schedule control's active view is **Week** view. Also, it is possible to change the active view of the Scheduler by setting [currentView](#) option with the required view name as depicted below.

## HTML

```

var Scheduler = React.createClass({
render: function () {
return (
<EJ.Schedule id="Schedule1" width="100%" height="525px" currentDate={new
Date(2017, 5, 5)} views={["Day", "WorkWeek"]}
currentView={ej.Schedule.CurrentView.Workweek}>

```

```

</EJ.Schedule>
);
}
});
ReactDOM.render(<Scheduler />, document.getElementById('schedule-default'));

```

**Note:** The **currentView** property accepts both the **string** and **ej.Schedule.CurrentView** enum value.

### Day

It represents a single day Scheduler view (single date display) with all its related appointments.

### HTML

```

var dataManager = [{
  Id: 1,
  Subject: "Music Class",
  StartTime: new Date("2015/11/7 06:00 AM"),
  EndTime: new Date("2015/11/7 07:00 AM")
}, {
  Id: 2,
  Subject: "School",
  StartTime: new Date("2015/11/7 9:00 AM"),
  EndTime: new Date("2015/11/7 02:30 PM")
}];
var Scheduler = React.createClass({
  render: function () {
    return (
      <EJ.Schedule id="Schedule1" width="100%" height="525px" currentDate={new
      Date(2017, 5, 5)} currentView={ej.Schedule.CurrentView.Day}
      appointmentSettings-dataSource={dataManager}>
      </EJ.Schedule>
    );
  }
});
ReactDOM.render(<Scheduler />, document.getElementById('schedule-default'));

```

### Week

It's a view displaying a count of 7 days (from Sunday to Saturday) with all its related appointments. The first day of the week can be changed using the [firstDayOfWeek](#) API which accepts either the **integer** (Sunday=0, Monday=1, Tuesday=2, etc) or **string** ("Sunday", "Monday", etc) or **ej.Schedule.DayOfWeek** enum type value. The default value of this **firstDayOfWeek** depends on the current culture (language) used in the Scheduler.

### HTML

```

var dataManager = [{
  Id: 1,
  Subject: "Music Class",
  StartTime: new Date("2015/11/7 06:00 AM"),
  EndTime: new Date("2015/11/7 07:00 AM")
}, {
  Id: 2,
  Subject: "School",
  StartTime: new Date("2015/11/7 9:00 AM"),
  EndTime: new Date("2015/11/7 02:30 PM")
}];

```

```

    });
    var Scheduler = React.createClass({
    render: function () {
    return (
    <EJ.Schedule id="Schedule1" width="100%" height="525px" currentDate={new
    Date(2017, 5, 5)} currentView={ej.Schedule.CurrentView.Week}
    firstDayOfWeek="monday" appointmentSettings-dataSource={dataManager}>
    </EJ.Schedule>
    );
    }
    });
    ReactDOM.render(<Scheduler />, document.getElementById('schedule-default'));

```

### Work Week

Work week view displays the working days of the week (count of 5 days) and its associated appointments. It is also possible to customize the days to be displayed in the work week view using [workWeek](#) API which accepts the string array such as ["Monday", "Tuesday", "Wednesday", "Thursday" and "Friday"]. By default, it renders from Monday to Friday (5 days).

### HTML

```

var dataManager = [{
    Id: 1,
    Subject: "Music Class",
    StartTime: new Date("2015/11/7 06:00 AM"),
    EndTime: new Date("2015/11/7 07:00 AM")
}, {
    Id: 2,
    Subject: "School",
    StartTime: new Date("2015/11/7 9:00 AM"),
    EndTime: new Date("2015/11/7 02:30 PM")
}];
var Scheduler = React.createClass({
    render: function () {
    return (
    <EJ.Schedule id="Schedule1" width="100%" height="525px" currentDate={new
    Date(2017, 5, 5)} currentView={ej.Schedule.CurrentView.WorkWeek}
    workWeek=[["Monday", "Tuesday", "Thursday", "Friday", "Saturday"]]
    appointmentSettings-dataSource={dataManager}>
    </EJ.Schedule>
    );
    }
    });
    ReactDOM.render(<Scheduler />, document.getElementById('schedule-default'));

```

### Month

Month view displays the entire days of a particular month and all its related appointments. An alternative way to navigate to a particular date in a day view directly from Month view, clicking on the appropriate month cell date header will do so. If the week date range column is clicked, it will navigate to the corresponding week view.

The next and previous month date cells in the Month view can be shown/hidden on the Scheduler using [showNextPrevMonth](#) property by setting it to *false*.

For example – To set the Month view as current view in Scheduler and to hide the other month days in it, refer the below code example.

### HTML

```
var dataManager = [{
  Id: 1,
  Subject: "Music Class",
  StartTime: new Date("2015/11/7 06:00 AM"),
  EndTime: new Date("2015/11/7 07:00 AM")
}, {
  Id: 2,
  Subject: "School",
  StartTime: new Date("2015/11/7 9:00 AM"),
  EndTime: new Date("2015/11/7 02:30 PM")
}];
var Scheduler = React.createClass({
  render: function () {
    return (
      <EJ.Schedule id="Schedule1" width="100%" height="525px" currentDate={new
      Date(2017, 5, 5)} currentView={ej.Schedule.CurrentView.Month}
      showNextPrevMonth={false} appointmentSettings-dataSource={dataManager}>
      </EJ.Schedule>
    );
  }
});
ReactDOM.render(<Scheduler />, document.getElementById('schedule-default'));
```

**Note:** An appointment directly created in Month view will be considered as an all-day appointment.

### Custom

The Scheduler can be displayed with the user-specified date ranges, such as 4 days or any specific date ranges instead of default view options, by making use of the [renderDates](#) property. This property includes two sub properties namely **start** and **end**, which accepts the date object or date value in string format to specify the date range.

To display the custom view option in the toolbar-like view options in the scheduler header area, add the **CustomView** value to the views property array collection as shown below.

### HTML

```
var dataManager = [{
  Id: 1,
  Subject: "Music Class",
  StartTime: new Date("2015/11/7 06:00 AM"),
  EndTime: new Date("2015/11/7 07:00 AM")
}, {
  Id: 2,
  Subject: "School",
  StartTime: new Date("2015/11/7 9:00 AM"),
  EndTime: new Date("2015/11/7 02:30 PM")
}];
var dateCollection = {
  start: new Date(2015, 11, 6), // Render start date
  end: new Date(2015, 11, 9) // Render end date
};
```

```
var Scheduler = React.createClass({
  render: function () {
    return (
      <EJ.Schedule id="Schedule1" width="100%" height="525px" currentDate={new
      Date(2017, 5, 5)} views=["Day", "Week", "WorkWeek", "Month", "CustomView"]}
      currentView={ej.Schedule.CurrentView.CustomView}
      renderDates={dateCollection} appointmentSettings-dataSource={dataManager}>
      </EJ.Schedule>
    );
  }
});
ReactDOM.render(<Scheduler />, document.getElementById('schedule-default'));
```

When the date difference between the provided start and end date is greater than 7, then the month-like view will get displayed in Vertical Scheduler mode - whereas with the date difference less than 7 days displays the Scheduler with exact count of the specified days.

**Note:** When the `currentDate` property of Scheduler is set with a date, that lies beyond the specified custom date range - then the Scheduler navigates to the current date with the mentioned date differences.

### Agenda

This View option lists out the appointments in a grid-like view for the next 7 days by default from the current date. The count of the number of appointments to be listed in this view can be customized using the [agendaViewSettings.daysInAgenda](#).

### HTML

```
var dataManager = [{
  Id: 1,
  Subject: "Music Class",
  StartTime: new Date("2015/11/7 06:00 AM"),
  EndTime: new Date("2015/11/7 07:00 AM")
}, {
  Id: 2,
  Subject: "School",
  StartTime: new Date("2015/11/7 9:00 AM"),
  EndTime: new Date("2015/11/7 02:30 PM")
}];
var Scheduler = React.createClass({
  render: function () {
    return (
      <EJ.Schedule id="Schedule1" width="100%" height="525px" currentDate={new
      Date(2017, 5, 5)} currentView={ej.Schedule.CurrentView.Agenda}
      agendaViewSettings-daysInAgenda={5} appointmentSettings-
      dataSource={dataManager}>
      </EJ.Schedule>
    );
  }
});
ReactDOM.render(<Scheduler />, document.getElementById('schedule-default'));
```

**Note:** In Agenda view, the templates can be applied for the date and time columns which can be referred [here](#). Also, the template passed through the [appointmentTemplateID](#) will get applied to the event column in Agenda view.

#### Restriction on View Navigation

It is possible to restrict the users to display only the specific list of views in the Schedule header section and also not to navigate to other views that are not listed.

**For example,** if the views property is set only with **Month** view – then the Schedule header section displays only the Month option in the view toolbar and also other additional available actions like navigating to day/week view on clicking the month header dates and week date-range is stopped.

#### HTML

```
var dataManager = [{
  Id: 1,
  Subject: "Music Class",
  StartTime: new Date("2015/11/7 06:00 AM"),
  EndTime: new Date("2015/11/7 07:00 AM")
}, {
  Id: 2,
  Subject: "School",
  StartTime: new Date("2015/11/7 9:00 AM"),
  EndTime: new Date("2015/11/7 02:30 PM")
}];
var Scheduler = React.createClass({
  render: function () {
    return (
      <EJ.Schedule id="Schedule1" width="100%" height="525px" currentDate={new
      Date(2017, 5, 5)} views=["Month"] appointmentSettings-
      dataSource={dataManager}>
      </EJ.Schedule>
    );
  }
});
ReactDOM.render(<Scheduler />, document.getElementById('schedule-default'));
```

**Note:** Even though Week view is the active view in Scheduler by default, if it is not listed in the views collection – then the first listed option in the views collection will be taken as current view of the Scheduler.

#### Timeline View

Timeline view displays the day, time and its associated events horizontally arranged from left to right. By default, Scheduler renders in vertical mode and it can be changed to the timeline mode using [orientation](#) property which accepts both the [string](#) and [ej.Schedule.Orientation](#) enum value.

All the applicable features in Vertical mode works similar with Timeline mode (Horizontal) and only the visualization of the layout changes based on the orientation.

#### HTML

```
var dataManager = [{
  Id: 1,
  Subject: "Music Class",
  StartTime: new Date("2015/11/7 06:00 AM"),
```

```

EndTime: new Date("2015/11/7 07:00 AM")
}, {
Id: 2,
Subject: "School",
StartTime: new Date("2015/11/7 9:00 AM"),
EndTime: new Date("2015/11/7 02:30 PM")
}];
var Scheduler = React.createClass({
render: function () {
return (
<EJ.Schedule id="Schedule1" width="100%" height="525px" currentDate={new
Date(2017, 5, 5)} orientation={ej.Schedule.Orientation.Horizontal}
appointmentSettings-dataSource={dataManager}>
</EJ.Schedule>
);
}
});
ReactDOM.render(<Scheduler />, document.getElementById('schedule-default'));

```

## Working with Appointments

An appointment represents a certain time interval in a schedule cell depicting a plan made for the specified time interval.

### Appointment Types

The types of appointments available within Scheduler can be categorized as follows

#### Normal

Represents an appointment that is created for a certain time interval in one or more number of days. If the normal appointment is created for more than 24 hours, then those longer appointments will be rendered on the all-day row.

**Note:** If the normal appointment is to be created for two days (say from November 25, 2015 – 11.00 PM to November 26, 2015 2.00 AM) but less than 24 hour time interval, then the appointment is split into two partitions and will be displayed appropriately on both the days.

#### All-Day

Represents an appointment that is created for an entire day such as holiday events. It renders separately in an All-day row, a separate place for all-day appointments. In Timeline (horizontal) view, all-day appointment renders in the usual work cells, as no all-day cells are present in that view.

**Note:** An all-day row is normally visible on the Scheduler, as the [showAllDayRow](#) property is set to true by default.

#### Recurrence

Represents an appointment that is created for a certain time interval that occurs repeatedly in a daily, weekly, monthly, yearly or every weekday basis at the same time interval based on the recurrence rule. The other available options and validations that can be performed on recurrence appointments can be referred from [here](#).

### CRUD operation

Appointments play a dynamic role within the Schedule control with which the users mostly interact. You can manipulate (add/edit/delete/drag/resize) the required appointments that reveals one of the main purpose of the Schedule control.

### Add/Edit Appointments

The appointments can be added/edited in the Scheduler using any one of the following ways,

- Quick window
- Inline creation/editing
- Default appointment window
- [Context menu](#)
- Through programmatically

### Quick Window

The Quick window usually pops out while single clicking on the Scheduler cells or appointments. It requires the user to enter the Subject to proceed with the appointment creation. It also includes an **Edit Appointment** option displayed at the bottom left corner – on selection which opens up the normal appointment window.

On single clicking the scheduler appointments, the pop-up that shows up contains the appointment information along with the other options that are listed below,

- Edit Appointment
- Edit Series (only for the recurrence appointments)
- Delete icon

The quick window option can be enabled/disabled by using [showQuickWindow](#) API, whereas its default value is set to **true**.

### HTML

```

var dataManager = [{
  Id: 1,
  Subject: "Music Class",
  StartTime: new Date("2015/11/7 06:00 AM"),
  EndTime: new Date("2015/11/7 07:00 AM")
}, {
  Id: 2,
  Subject: "School",
  StartTime: new Date("2015/11/7 9:00 AM"),
  EndTime: new Date("2015/11/7 02:30 PM")
}];
var Scheduler = React.createClass({
  render: function () {
    return (
      <EJ.Schedule id="Schedule1" width="100%" height="525px" currentDate={new
      Date(2017, 5, 5)} showQuickWindow={false} appointmentSettings-
      dataSource={dataManager}>
      </EJ.Schedule>
    );
  }
});
ReactDOM.render(<Scheduler />, document.getElementById('schedule-default'));

```

**Note:** Select multiple cells either using mouse or keyboard access keys (shift+arrow keys) and press enter key, so that the quick window opens up for the selected date/time range.



Another way to disable the quick window option at dynamic time can be achieved through the [cellClick](#) and [appointmentClick](#) events. The below code example shows the way to disable the quick appointment window only while clicking on the cells, but displays for appointments.

### HTML

```
var dataManager = [{
  Id: 1,
  Subject: "Music Class",
  StartTime: new Date("2015/11/7 06:00 AM"),
  EndTime: new Date("2015/11/7 07:00 AM")
}, {
  Id: 2,
  Subject: "School",
  StartTime: new Date("2015/11/7 9:00 AM"),
  EndTime: new Date("2015/11/7 02:30 PM")
}];
var Scheduler = React.createClass({
  onCellClick: function(args) {
    args.cancel = true; // Prevents the display of quick window on clicking the cells.
  },
  render: function () {
    return (
      <EJ.Schedule id="Schedule1" width="100%" height="525px" currentDate={new Date(2017, 5, 5)} showQuickWindow={true} appointmentSettings-
      dataSource={dataManager} cellClick={this.onCellClick}>
    </EJ.Schedule>
    );
  }
});
ReactDOM.render(<Scheduler />, document.getElementById('schedule-default'));
```

### Inline Appointment Creation/Editing

Another easier way, for adding or editing the appointment's subject alone can be achieved using inline Add/Edit support. It allows the user to add and edit the appointments inline.

To get familiar with inline Add mode, single click on any of the Scheduler cells or press **enter** key on the selected cells. When the inline adding mode is ON, a text box will get created within the clicked Scheduler cells with a blinking cursor in it, requiring the user to enter the subject of an appointment. Once the subject is entered, the appointment will be saved on pressing the **enter** key.

To enable inline edit mode, single click on any of the existing appointment's subject, so that the user can edit the subject of that appointment. The edited subject of that appointment is then updated on pressing the **enter** key.

The inline option can be enabled/disabled on Scheduler by using the [allowInline](#) API, whereas its default value is set to **false**.

### HTML

```
var dataManager = [{
  Id: 1,
  Subject: "Music Class",
  StartTime: new Date("2015/11/7 06:00 AM"),
  EndTime: new Date("2015/11/7 07:00 AM")
}];
```

```

}, {
  Id: 2,
  Subject: "School",
  StartTime: new Date("2015/11/7 9:00 AM"),
  EndTime: new Date("2015/11/7 02:30 PM")
}];
var Scheduler = React.createClass({
  render: function () {
    return (
      <EJ.Schedule id="Schedule1" width="100%" height="525px" currentDate={new
      Date(2017, 5, 5)} allowInline={true} appointmentSettings-
      dataSource={dataManager}>
      </EJ.Schedule>
    );
  }
});
ReactDOM.render(<Scheduler />, document.getElementById('schedule-default'));

```

### Enabling Inline Edit alone

It is possible to disable the inline appointment creation and enabling only the editing mode of inline by making use of the `cellClick` event. The below code example shows the way to disable the inline appointment creation while clicking on the cells, but appointments can be edited while clicking on it's subject.

### HTML

```

var dataManager = [{
  Id: 1,
  Subject: "Music Class",
  StartTime: new Date("2015/11/7 06:00 AM"),
  EndTime: new Date("2015/11/7 07:00 AM")
}, {
  Id: 2,
  Subject: "School",
  StartTime: new Date("2015/11/7 9:00 AM"),
  EndTime: new Date("2015/11/7 02:30 PM")
}];
var Scheduler = React.createClass({
  onCellClick: function(args) {
    args.cancel = true; // Prevents inline appointment creation on clicking the
    cells.
  },
  render: function () {
    return (
      <EJ.Schedule id="Schedule1" width="100%" height="525px" currentDate={new
      Date(2017, 5, 5)} showQuickWindow={true} appointmentSettings-
      dataSource={dataManager} cellClick={this.onCellClick}>
      </EJ.Schedule>
    );
  }
});
ReactDOM.render(<Scheduler />, document.getElementById('schedule-default'));

```

### Default Appointment Window

The default appointment window is available with options like

- Subject
- Start and End Time
- All-Day
- TimeZone (for both Start and End time)
- Repeat options
- Description

The other additional options available are listed below for which the appropriate API's are needed to be configured to display these options on the appointment window.

- Location ([showLocationField](#))
- Priority ([prioritySettings](#))
- Categorize ([categorizeSettings](#))
- [Resources](#)

The appointments can be created by double-clicking the Scheduler cells across the required time slots, which makes the Create Appointment window to pop-up. The start and end time fields will get automatically populated, according to the time-slot selection. Clicking on the done button in an appointment window will create the appointment for the selected time cells.

**Note:** Select multiple cells both using mouse or keyboard access keys (shift+arrow keys) and press Alt+N key, so that the default appointment window opens up for the selected date/time range with the Start and End time fields automatically filled in.

To prevent the display of default appointment window on double clicking the Scheduler cells, either the [appointmentWindowOpen](#) or [cellDoubleClick](#) event can be used, within which the **args.cancel** needs to be set to true. This behavior is depicted in the below code example.

### HTML

```
var dataManager = [{
  Id: 1,
  Subject: "Music Class",
  StartTime: new Date("2015/11/7 06:00 AM"),
  EndTime: new Date("2015/11/7 07:00 AM")
}, {
  Id: 2,
  Subject: "School",
  StartTime: new Date("2015/11/7 9:00 AM"),
  EndTime: new Date("2015/11/7 02:30 PM")
}];
var Scheduler = React.createClass({
  onAppointmentWindowOpen: function(args) {
    args.cancel = true; // prevents the display of default appointment window
  },
  render: function () {
    return (
      <EJ.Schedule id="Schedule1" width="100%" height="525px" currentDate={new
Date(2017, 5, 5)} showQuickWindow={true} appointmentSettings-
```

```

dataSource={dataManager}
appointmentWindowOpen={this.onAppointmentWindowOpen}>
</EJ.Schedule>
);
}
});
ReactDOM.render(<Scheduler />, document.getElementById('schedule-default'));

```

### Through Programmatically

You can add/edit the appointments dynamically through the public method [saveAppointment](#). It accepts the JSON Object data (either a new or updated appointment object) as its argument.

### HTML

```

var dataManager = [{
  Id: 1,
  Subject: "Music Class",
  StartTime: new Date("2015/11/7 06:00 AM"),
  EndTime: new Date("2015/11/7 07:00 AM")
}, {
  Id: 2,
  Subject: "School",
  StartTime: new Date("2015/11/7 9:00 AM"),
  EndTime: new Date("2015/11/7 02:30 PM")
}];
var Scheduler = React.createClass({
  render: function () {
    return (
      <EJ.Schedule id="Schedule1" width="100%" height="525px" currentDate={new
      Date(2017, 5, 5)} appointmentSettings-dataSource={dataManager}>
      </EJ.Schedule>
    );
  }
});
var Button = React.createClass({
  addAppointment: function() {
    //appointment details
    var appointment = {
      Subject: "Gym",
      StartTime: new Date("2015/11/7 03:30 AM"),
      EndTime: new Date("2015/11/7 04:30 AM")
    };
    //create the schedule object
    var schObj = $("#Schedule1").data("ejSchedule");
    //pass the JSON object in the public method
    schObj.saveAppointment(appointment);
  },
  render: function () {
    return (
      <EJ.Button id="btnAdd" text="Add" click={this.addAppointment}></EJ.Button>
    );
  }
});
ReactDOM.render(<Button />, document.getElementById('button-default'));
ReactDOM.render(<Scheduler />, document.getElementById('schedule-default'));

```

### Delete Appointments

The appointments can be deleted in either of the following ways,

- Selecting an appointment and clicking the delete icon in the quick appointment window.
- Hovering the mouse over appointments and clicking on Inline delete option which is enabled by default for all the appointments.
- Selecting an appointment and pressing `Delete` key.
- Through Programmatically.

A pop-up with a confirmation message will get displayed before deleting an appointment, which can be either switched on/off using the API `showDeleteConfirmationDialog`. Also, the confirmation text in that pop-up can be customized as mentioned [here](#).

**For example**, to localize only the delete confirmation message in the delete window -

#### HTML

```
var dataManager = [{
  Id: 101,
  Subject: "Talk with Nature",
  StartTime: new Date(2015, 11, 5, 10, 00),
  EndTime: new Date(2015, 11, 5, 11, 00)
}];
var deleteCustomizationMessage = {
  // customize the confirmation message
  DeleteConfirmation: "Do you want to delete this Event?",
  // customize the delete window title
  MouseOverDeleteTitle: "Delete Event",
  // customize the recurrence delete window title
  RecurrenceDeleteTitle: "Delete Repeat Event"
};
// Extend only the required changes to the original locale collection
$.extend(ej.Schedule.Locale["en-US"], deleteCustomizationMessage);
var Scheduler = React.createClass({
  render: function () {
    return (
      <EJ.Schedule id="Schedule1" width="100%" height="525px" currentDate={new
      Date(2017, 5, 5)} appointmentSettings-dataSource={dataManager}>
      </EJ.Schedule>
    );
  }
});
ReactDOM.render(<Scheduler />, document.getElementById('schedule-default'));
```

**Note:** All these CRUD operations on appointments (add/edit/delete) can also be done through the default [context menu](#) options **Add Appointment**, **Edit Appointment** and **Delete Appointment** which is available, when context menu settings is enabled within Scheduler.

#### Through Programmatically

You can delete the appointments dynamically using the method [deleteAppointment](#), which accepts the Guid of the appointment or complete appointment data as its argument. The Guid is availed as one of the appointment element's attribute.

### Example 1 - Using GUID

The below code example depicts the way to delete the appointments using GUID programmatically by calling the **deleteAppointment** function within the [appointmentClick](#) event, which triggers whenever the user clicks on an appointment.

#### HTML

```
var dataManager = [{
  Id: 1,
  Subject: "Music Class",
  StartTime: new Date("2015/11/7 06:00 AM"),
  EndTime: new Date("2015/11/7 07:00 AM")
}, {
  Id: 2,
  Subject: "School",
  StartTime: new Date("2015/11/7 9:00 AM"),
  EndTime: new Date("2015/11/7 02:30 PM")
}];
var Scheduler = React.createClass({
  onAppointmentClick: function(args) {
    var schObj = $("#Schedule1").data("ejSchedule");
    schObj.deleteAppointment(args.appointment.Guid);
    // $(".e-appointment").eq(0).attr("guid") --> To get the guid attribute
    value of an element directly.
  },
  render: function () {
    return (
      <EJ.Schedule id="Schedule1" width="100%" height="525px" currentDate={new
      Date(2017, 5, 5)} appointmentSettings-dataSource={dataManager}
      appointmentClick={this.onAppointmentClick}>
    </EJ.Schedule>
    );
  }
});
ReactDOM.render(<Scheduler />, document.getElementById('schedule-default'));
```

### Example 2 - Using Appointment object

The below code example depicts the way to delete the appointments using appointment data programmatically by calling the **deleteAppointment** function within the [appointmentClick](#) event, which triggers whenever the user clicks on an appointment.

#### HTML

```
var dataManager = [{
  Id: 1,
  Subject: "Music Class",
  StartTime: new Date("2015/11/7 06:00 AM"),
  EndTime: new Date("2015/11/7 07:00 AM")
}, {
  Id: 2,
  Subject: "School",
  StartTime: new Date("2015/11/7 9:00 AM"),
  EndTime: new Date("2015/11/7 02:30 PM")
}];
var Scheduler = React.createClass({
  onAppointmentClick: function(args) {
```

```

var schObj = $("#schedule").data("ejSchedule");
schObj.deleteAppointment(args.appointment);
},
render: function () {
return (
<EJ.Schedule id="Schedule1" width="100%" height="525px" currentDate={new
Date(2017, 5, 5)} appointmentSettings-dataSource={dataManager}
appointmentClick={this.onAppointmentClick}>
</EJ.Schedule>
);
}
});
ReactDOM.render(<Scheduler />, document.getElementById('schedule-default'));

```

### Handling Appointment Actions

It is possible to define some specific actions to take place before the CRUD operation occurs on the Scheduler appointments through the following available client-side events,

- [beforeAppointmentCreate](#)
- [beforeAppointmentChange](#)
- [beforeAppointmentRemove](#)

**beforeAppointmentCreate** – Triggers before saving a new appointment.

**beforeAppointmentChange** – Triggers when an appointment is edited and before it is being updated to the dataSource.

**beforeAppointmentRemove** – Triggers before deleting an existing appointment.

To stop the save, edit and delete actions on the Scheduler appointments, following code example can be used.

### HTML

```

var dataManager = [{
Id: 1,
Subject: "Music Class",
StartTime: new Date("2015/11/7 06:00 AM"),
EndTime: new Date("2015/11/7 07:00 AM")
}, {
Id: 2,
Subject: "School",
StartTime: new Date("2015/11/7 9:00 AM"),
EndTime: new Date("2015/11/7 02:30 PM")
}];
var Scheduler = React.createClass({
onAppointmentSave: function(args) {
args.cancel = true; // cancels the save action on appointments.
},
onAppointmentEdit: function(args) {
args.cancel = true; // cancels the edit action on appointments.
},
onAppointmentDelete: function(args) {
args.cancel = true; // cancels the delete action on appointments.
},

```

```
render: function () {
  return (
    <EJ.Schedule id="Schedule1" width="100%" height="525px" currentDate={new
    Date(2017, 5, 5)} appointmentSettings-dataSource={dataManager}
    beforeAppointmentCreate={this.onAppointmentSave}
    beforeAppointmentChange={this.onAppointmentEdit}
    beforeAppointmentRemove={this.onAppointmentDelete}>
    </EJ.Schedule>
  );
}
});
ReactDOM.render(<Scheduler />, document.getElementById('schedule-default'));
```

### Read Only

An interaction with the appointments of the Scheduler can be enabled/disabled through the [readOnly](#) property. When the `readOnly` property is set to `true`, it is not possible to do any actions on the appointments, but you can navigate between the schedule dates, views and can also be able to see the appointment details in the quick window. By default, this property is set to `false`.

### HTML

```
var dataManager = [{
  Id: 1,
  Subject: "Music Class",
  StartTime: new Date("2015/11/7 06:00 AM"),
  EndTime: new Date("2015/11/7 07:00 AM")
}, {
  Id: 2,
  Subject: "School",
  StartTime: new Date("2015/11/7 9:00 AM"),
  EndTime: new Date("2015/11/7 02:30 PM")
}];
var Scheduler = React.createClass({
  render: function () {
    return (
      <EJ.Schedule id="Schedule1" width="100%" height="525px" currentDate={new
      Date(2017, 5, 5)} readOnly={true} appointmentSettings-
      dataSource={dataManager}>
      </EJ.Schedule>
    );
  }
});
ReactDOM.render(<Scheduler />, document.getElementById('schedule-default'));
```

**Note:** When the `readOnly` property is set to `true` – double clicking the cells will open the appointment window filled with appointment details, which can be allowed to view but cannot be edited or saved.

### Drag and Drop

The appointment time can be modified through the drag and drop behavior, by dragging and dropping it to the new location, so that the start time and end time of the appointment gets changed automatically. We can enable/disable the drag and drop functionality through the `allowDragAndDrop` property. By default, it is set to `true`.

### HTML



```

var dataManager = [{
  Id: 1,
  Subject: "Music Class",
  StartTime: new Date("2015/11/7 06:00 AM"),
  EndTime: new Date("2015/11/7 07:00 AM")
}, {
  Id: 2,
  Subject: "School",
  StartTime: new Date("2015/11/7 9:00 AM"),
  EndTime: new Date("2015/11/7 02:30 PM")
}];
var Scheduler = React.createClass({
  render: function () {
    return (
      <EJ.Schedule id="Schedule1" width="100%" height="525px" currentDate={new
      Date(2017, 5, 5)} allowDragAndDrop={false} appointmentSettings-
      dataSource={dataManager}>
    </EJ.Schedule>
    );
  }
});
ReactDOM.render(<Scheduler />, document.getElementById('schedule-default'));

```

### Handling Drag Actions Dynamically

The drag and drop functionality can be handled with the following three events,

- [dragStart](#)
- [drag](#)
- [dragStop](#)

**dragStart** – Triggers when the appointments are started to drag from its source location.

**drag** – Triggers when the appointments are being dragged over.

**dragStop** – Triggers when the appointments are dropped on a destined location.

The following code example shows how to cancel the dragging functionality with the help of one of these events.

### HTML

```

var dataManager = [{
  Id: 1,
  Subject: "Music Class",
  StartTime: new Date("2015/11/7 06:00 AM"),
  EndTime: new Date("2015/11/7 07:00 AM")
}, {
  Id: 2,
  Subject: "School",
  StartTime: new Date("2015/11/7 9:00 AM"),
  EndTime: new Date("2015/11/7 02:30 PM")
}];
var Scheduler = React.createClass({
  onDragStop: function(args) {
    args.cancel = true; // cancels the drag action on appointments.
  },

```

```

render: function () {
  return (
    <EJ.Schedule id="Schedule1" width="100%" height="525px" currentDate={new
    Date(2017, 5, 5)} dragStop={this.onDragStop} appointmentSettings-
    dataSource={dataManager}>
    </EJ.Schedule>
  );
}
});
ReactDOM.render(<Scheduler />, document.getElementById('schedule-default'));

```

### External Drag and Drop

It is possible to drag and drop the external items to and fro the Scheduler control. This action is handled through the property [appointmentDragArea](#), which specifies the draggable area name stating whether the appointments can be dragged outside of the control or within it.

The following code example lets you drag and drop the external items from the tree view control to the Scheduler.

### HTML

```

<!-- Treeview List -->
<div class="col-md-2">
  <span class=""><b>Tutorials </b> </span>
  <ul id="treeViewDrag">
    <li class="expanded">
      HTML
      <ul>
        <li>Introduction</li>
        <li>Editors</li>
        <li>Styles</li>
        <li>Formatting</li>
        <li>Tables</li>
      </ul>
    </li>
  </ul>
</div>
<!--Container for ejScheduler widget-->
<div id="schedule-default"></div>
<div id="customWindow" style="display: none">
  <form id="custom">
    <table width="100%" cellpadding="5">
      <tbody>
        <tr>
          <td>Subject:</td>
          <td colspan="2">
            <input id="subject" type="text" value="" name="Subject" style="width: 100%"
            readonly />
          </td>
        </tr>
        <tr>
          <td>Description:</td>
          <td colspan="2">
            <textarea id="customDescription" name="Description" rows="3" cols="50"
            style="width: 100%; resize: vertical"></textarea>
          </td>
        </tr>
      </tbody>
    </table>
  </form>

```

```

</tr>
<tr>
<td>StartTime:</td>
<td>
<input id="StartTime" type="text" value="" name="StartTime" />
</td>
</tr>
<tr>
<td>EndTime:</td>
<td>
<input id="EndTime" type="text" value="" name="EndTime" />
</td>
</tr>
<tr>
<td>Resource:</td>
<td colspan="2">
<input id="resource" type="text" value="" name="Resource" style="width:
100%" readonly />
</td>
</tr>
<tr style="display: none">
<td>ownerId:</td>
<td colspan="2">
<input id="ownerId" type="text" name="ownerId" />
</td>
</tr>
</tbody>
</table>
</form>
<div>
<button type="submit" id="buttonCancel" style="float:right;margin-
right:20px;margin-bottom:10px;">Cancel</button>
<button type="submit" id="buttonSubmit" style="float:right;margin-
right:20px;margin-bottom:10px;">Save</button>
</div>
</div>
<script src="app/schedule/scheduler.js"></script>

```

## HTML

```

var dataManager =
ej.DataManager(window.HorizontalResourcesTutorials).executeLocal(ej.Query().
take(10));
var groupData = { resources: ["Owners"] };
var ownerData = {
dataSource: [
{ text: "Nancy", id: 1, groupId: 1, color: "#f8a398" },
{ text: "Steven", id: 3, groupId: 2, color: "#56ca85" },
{ text: "Michael", id: 5, groupId: 1, color: "#51a0ed" },
{ text: "Milan", id: 13, groupId: 3, color: "#99ff99" },
{ text: "Paul", id: 15, groupId: 3, color: "#cc99ff" }
],
text: "text", id: "id", groupId: "groupId", color: "color"
};
var Scheduler = React.createClass({
render: function () {

```

```

return (
<EJ.Schedule id="Schedule1" width="100%" height="525px" currentDate={new
Date(2017, 5, 5)} orientation={ej.Schedule.Orientation.Horizontal}
views={["Day", "Week", "WorkWeek", "Month"]}
currentView={ej.Schedule.CurrentView.Workweek} group={groupData}
appointmentSettings-dataSource={dataManager}>
<resources>
<resource allowMultiple={true} field="OwnerId" title="Owner" name="Owners"
resourceSettings={ownerData}></resource>
</resources>
</EJ.Schedule>
);
}
});
var TreeView = React.createClass({
onNodeDragStart: function(args) {
if (args.targetElementData.parentId == "") return false;
},
onNodeDropped: function() {
if ($(e.target).parents(".e-schedule").length != 0) {
var scheduleObj = $("#Schedule1").data("ejSchedule");
var result = scheduleObj.getSlotByElement($(e.target));
// set value to custom appointment window fields
$("#subject").val(e.droppedElementData.text);
$("#customdescription").val(e.droppedElementData.text);
$("#StartTime").ejDateTimePicker({ value: new Date(result.startTime) });
$("#EndTime").ejDateTimePicker({ value: new Date(result.endTime) });
$("#resource").val(result.resources.text);
$("#ownerId").val(result.resources.id);
$("#customWindow").ejDialog("open");
}
},
render: function () {
return (
<EJ.TreeView id="treeViewDrag" width="170px" allowDragAndDrop={true}
allowDropChild={false} allowDropSibling={false}
allowDragAndDropAcrossControl={true} nodeDragStart={this.onNodeDragStart}
nodeDropped={this.onNodeDropped}>
</EJ.TreeView>
);
}
});
var Button1 = React.createClass({
onClick: function(args) {
var obj = {};
var formElement = $("#customWindow").find("#custom").get(0);
for (var index = 0; index < formElement.length; index++) {
var columnName = formElement[index].name, $element = $(formElement[index]);
if (columnName != undefined) {
if (columnName == "Subject") var value = formElement[index].value;
if (columnName == "Description") value = formElement[index].value;
if (columnName == "StartTime") value = new Date(formElement[index].value);
if (columnName == "EndTime") value = new Date(formElement[index].value);
if (columnName == "ownerId") value = parseInt(formElement[index].value);
if (columnName != "Resource") obj[columnName] = value;
}
}
}
});

```

```

$("#customWindow").ejDialog("close");
var object = $("#Schedule1").data("ejSchedule");
object.saveAppointment(obj);
},
render: function () {
return (
<EJ.Button id="Submit" width="85px" click={this.onClick}></EJ.Button>
);
}
});
var Button2 = React.createClass({
onClick: function(args) {
$("#customWindow").ejDialog("close");
},
render: function () {
return (
<EJ.Button id="Cancel" width="85px" click={this.onClick}></EJ.Button>
);
}
});
var DateTimePicker1 = React.createClass({
render: function () {
return (
<EJ.DateTimePicker id="StartTime" width="150px"></EJ.DateTimePicker>
);
}
});
var DateTimePicker2 = React.createClass({
render: function () {
return (
<EJ.DateTimePicker id="EndTime" width="150px"></EJ.DateTimePicker>
);
}
});
var Dialog = React.createClass({
clearFields: function(args) {
$("#customId").val("");
$("#subject").val("");
$("#customdescription").val("");
$("#allday").prop("checked", false);
$("#recurrence").prop("checked", false);
document.getElementById("rType").selectedIndex = "0";
$("#tr.recurrence").css("display", "none");
$("#StartTime,#EndTime").ejDateTimePicker({ enabled: true });
},
render: function () {
return (
<EJ.Dialog id="customdialog" width="600px" height="auto" showOnInit={false}
enableModal={true} title="Create Appointment" enableResize={false}
allowKeyboardNavigation={false} close={this.clearFields}>
</EJ.Dialog>
);
}
});
ReactDOM.render(<Scheduler />, document.getElementById('schedule-default'));
ReactDOM.render(<TreeView />, document.getElementById('treeViewDrag'));
ReactDOM.render(<Button1 />, document.getElementById('buttonSubmit'));

```

```
ReactDOM.render(<Button2 />, document.getElementById('buttonCancel'));
ReactDOM.render(<DateTimePicker1 />, document.getElementById('StartTime'));
ReactDOM.render(<DateTimePicker2 />, document.getElementById('EndTime'));
ReactDOM.render(<Dialog />, document.getElementById('customWindow'));
```

### Resize

Resizing an appointment is another way to change its start and end time. Mouse hover on the appointments, so that the resizing handlers gets displayed on either sides of the appointment which allows resizing. The resizing functionality can be enabled/disabled by setting the [enableAppointmentResize](#) property. By default it is set to `true`.

### HTML

```
var dataManager = [{
  Id: 1,
  Subject: "Music Class",
  StartTime: new Date("2015/11/7 09:00 AM"),
  EndTime: new Date("2015/11/7 10:00 AM")
}, {
  Id: 2,
  Subject: "School",
  StartTime: new Date("2015/11/7 02:00 PM"),
  EndTime: new Date("2015/11/7 06:30 PM")
}];
var Scheduler = React.createClass({
  render: function () {
    return (
      <EJ.Schedule id="Schedule1" width="100%" height="525px" currentDate={new
      Date(2017, 5, 5)} enableAppointmentResize={false} appointmentSettings-
      dataSource={dataManager}>
      </EJ.Schedule>
    );
  }
});
ReactDOM.render(<Scheduler />, document.getElementById('schedule-default'));
```

### Handling Resize Actions Dynamically

The appointment resizing functionality can be handled through the following three events,

- [resizeStart](#)
- [resize](#)
- [resizeStop](#)

**resizeStart** – Triggers when the appointments are started resizing from its original time.

**resize** – Triggers when the appointment resizing is in progress.

**resizeStop** – Triggers when the appointment resizing is done.

The following code example shows how to cancel the resizing functionality with the help of one of these events.

### HTML

```

var dataManager = [{
  Id: 1,
  Subject: "Music Class",
  StartTime: new Date("2015/11/7 09:00 AM"),
  EndTime: new Date("2015/11/7 10:00 AM")
}, {
  Id: 2,
  Subject: "School",
  StartTime: new Date("2015/11/7 02:00 PM"),
  EndTime: new Date("2015/11/7 06:30 PM")
}];
var Scheduler = React.createClass({
  onResizeStart: function(args) {
    args.cancel = true; // cancels the resize action on appointments.
  },
  render: function () {
    return (
      <EJ.Schedule id="Schedule1" width="100%" height="525px" currentDate={new
      Date(2017, 5, 5)} resizeStart={this.onResizeStart} appointmentSettings-
      dataSource={dataManager}>
      </EJ.Schedule>
    );
  }
});
ReactDOM.render(<Scheduler />, document.getElementById('schedule-default'));

```

### Categorization

It allows to differentiate the appointments with various categorize options and individual colors. You can also denote the status of the appointments using this categorize option and can specify your own user-defined category collection. It is also possible to select multiple categorize for a single appointment.

#### Categorize Settings

The [categorizeSettings](#) holds the below categorize related properties such as,

- [enable](#) - It accepts true or false value, denoting whether to enable/disable the categorize option. Its default value is `false`.
- [allowMultiple](#) – It enables or disables the multiple selection of categories for each appointments in the appointment window as well as in the context menu. Its default value is `false`.
- [dataSource](#) – Binds the categorize dataSource collection. This property should be assigned with the JSON data array collection or instance of [ej.DataManger](#).

We have below 6 default values for Categorize dataSource collection.

#### JAVASCRIPT

```

categorizeSettings: {
  dataSource: [{
    text: "Blue Category",
    id: 1,
    color: "#43b496",
    fontColor: "#ffffff"
  }, {
    text: "Green Category",
    id: 2,

```

```

color: "#7f993e",
fontColor: "#ffffff"
}, {
text: "Orange Category",
id: 3,
color: "#cc8638",
fontColor: "#ffffff"
}, {
text: "Purple Category",
id: 4,
color: "#ab54a0",
fontColor: "#ffffff"
}, {
text: "Red Category",
id: 5,
color: "#dd654e",
fontColor: "#ffffff"
}, {
text: "Yellow Category",
id: 6,
color: "#d0af2b",
fontColor: "#ffffff"
}]
}

```

The below categorize fields holds the appropriate column names from the dataSource -

Field name	Description
id	It holds the binding name for id field in the categorize dataSource
text	It holds the binding name for text field in the categorize dataSource
color	It holds the binding name for color field in the categorize dataSource.
fontColor	It holds the binding name for fontColor field in the categorize dataSource. This font color applies for the appointment.

### HTML

```

var dataManager = [{
  Id: 1,
  Subject: "Music Class",
  StartTime: new Date("2015/11/7 09:00 AM"),
  EndTime: new Date("2015/11/7 10:00 AM"),
  categorize: "3",
}, {
  Id: 2,
  Subject: "School",
  StartTime: new Date("2015/11/7 02:00 PM"),
  EndTime: new Date("2015/11/7 06:30 PM"),
  categorize: "1,2,6" // Multiple categorize id passing
}

```



```

    }];
    var categorizeData = {
    enable: true, //Enable the categorize options
    allowMultiple: true, //enable multiple selection options
    //data source collection binding
    dataSource: [
    { text: "Blue Category", id: 1, color: "#43b496", fontColor: "#ffffff" },
    { text: "Green Category", id: 2, color: "#7f993e", fontColor: "#ffffff" },
    { text: "Orange Category", id: 3, color: "#cc8638", fontColor: "#ffffff" },
    { text: "Purple Category", id: 4, color: "#ab54a0", fontColor: "#ffffff" },
    { text: "Red Category", id: 5, color: "#dd654e", fontColor: "#ffffff" },
    { text: "Yellow Category", id: 6, color: "#d0af2b", fontColor: "#ffffff" }
    ],
    text: "text", //text mapper field
    id: "id", //id mapper field
    color: "color", //categorize color mapper field
    fontColor: "fontColor" //categorize appointment font color mapper field
    };
    var Scheduler = React.createClass({
    render: function () {
    return (
    <EJ.Schedule id="Schedule1" width="100%" height="525px" currentDate={new
    Date(2017, 5, 5)} categorizeSettings={categorizeData} appointmentSettings-
    dataSource={dataManager}>
    </EJ.Schedule>
    );
    }
    });
    ReactDOM.render(<Scheduler />, document.getElementById('schedule-default'));

```

### Priority

This option prioritize the appointments based on its importance and it can be differentiated with each individual icons/images. By default, there are some specific set of default priority collection and you can also customize it with your own priority collection.

#### Priority Settings

The [prioritySettings](#) holds the below priority related properties such as,

- [enable](#) - It accepts true or false value, denoting whether to enable/disable the priority option. Its default value is **false**.
- [template](#) – Customize the priority icon/images using template options.
- [dataSource](#) – binds the priority dataSource collection. This property should be assigned with the JSON data array collection or instance of [ej.DataManger](#).

We have below 4 default values for priority dataSource collection.

#### JAVASCRIPT

```

prioritySettings: {
dataSource: [{
text: "None",
value: "none"
}, {
text: "High",

```

```

value: "high"
}, {
text: "Medium",
value: "medium"
}, {
text: "Low",
value: "low"
}]
}

```

The below priority fields holds the appropriate column names from the dataSource,

Field name	Description
text	It holds the binding name for text field in the priority dataSource
value	It holds the binding name for value field in the priority dataSource.

### HTML

```

var dataManager = [{
  Id: 1,
  Subject: "Music Class",
  StartTime: new Date("2015/11/7 09:00 AM"),
  EndTime: new Date("2015/11/7 10:00 AM"),
  priority: "low", //pass the priority value
}, {
  Id: 2,
  Subject: "School",
  StartTime: new Date("2015/11/7 02:00 PM"),
  EndTime: new Date("2015/11/7 06:30 PM"),
  priority: "high" //pass the priority value
}];
var priorityData = {
  enable: true, //enable the priority option
  //Configure the priority data source
  dataSource: [
    { text: "None", value: "none" },
    { text: "High", value: "high" },
    { text: "Low", value: "low" }
  ],
  text: "text", //mapper field for text
  value: "value" //mapper field for value
};
var Scheduler = React.createClass({
  render: function () {
    return (
      <EJ.Schedule id="Schedule1" width="100%" height="525px" currentDate={new
      Date(2017, 5, 5)} prioritySettings={priorityData} appointmentSettings-
      dataSource={dataManager}>
      </EJ.Schedule>
    );
  }
}

```

```
});
ReactDOM.render(<Scheduler />, document.getElementById('schedule-default'));
```

## Search or Filter Appointments

### Appointment Search

The public method [searchAppointments](#) is used to search the appointments in the Scheduler dataSource. It contains the below four arguments such as search string, search field, filter operator and ignore case.

**searchString** - It is used to search the given word/sentence within the appointment data.

**fields** - It is the field with which the search operation takes place. It's an optional argument.

**filterOperator** - It denotes the filter type like **contains**, **greaterthan** or **lessthan**. It's an optional argument.

**ignoreCase** - It is a boolean value to set the search string as case sensitive or not. It's an optional argument.

### HTML

```
<input id="btnSearch" class="searchApp" type="button" value="Search" />
<div id="grid1"></div>
<!--Container for ejScheduler widget-->
<div id="Schedule1"></div>
```

### HTML

```
var dataManager =
ej.DataManager(window.Default).executeLocal(ej.Query().take(10));
var Scheduler = React.createClass({
render: function () {
return (
<EJ.Schedule id="Schedule1" width="100%" height="525px" currentDate={new
Date(2017, 5, 5)} prioritySettings={priorityData} appointmentSettings-
dataSource={dataManager}>
</EJ.Schedule>
);
}
});
var Button = React.createClass({
onClick: function(args) {
var _searchString = $("#txtSearch").val();
var schObj = $("#Schedule1").data("ejSchedule");
// method to retrieve the appointment based on search string
var result = schObj.searchAppointments(_searchString);
if (!ej.isNullOrUndefined(result) && result.length != 0 && _searchString !=
"") {
$("#grid1").show();
$("#grid1").data("ejGrid");
$("#grid1").ejGrid("destroy");
$("#grid1").ejGrid({
allowScrolling: true,
dataSource: result,
allowPaging: true
```

```

});
}
},
render: function () {
return (
<EJ.Button id="btnClick" text="Search" click={this.onClick}>
</EJ.Button>
);
}
});
ReactDOM.render(<Scheduler />, document.getElementById('schedule-default'));
ReactDOM.render(<Button />, document.getElementById('btnSearch'));

```

### Appointment Filters

The appointments can be filtered or shortlisted based on the simple or complex conditions with four available properties such as **field**, **operator**, **value** and **predicate** which is passed to the public method [filterAppointments](#).

**field** - It is the field, with which the search operation takes place. It's an optional argument.

**operator** – It is generally used to specify the [filter](#) type.

**value** – It is the filter keyword based on which the records are filtered.

**predicate** – To add more than one conditional query, need to use **and**, **or** [predicates](#).

### HTML

```

<input id="btnSearch" class="searchApp" type="button" value="Filter" />
<div id="grid1"></div>
<!--Container for ejScheduler widget-->
<div id="Schedule1"></div>

```

### HTML

```

var dataManager =
ej.DataManager(window.Default).executeLocal(ej.Query().take(10));
var Scheduler = React.createClass({
render: function () {
return (
<EJ.Schedule id="Schedule1" width="100%" height="525px" currentDate={new
Date(2017, 5, 5)} prioritySettings={priorityData} appointmentSettings-
dataSource={dataManager}>
</EJ.Schedule>
);
}
});
var Button = React.createClass({
onClick: function(args) {
// Add the filter data as like in the below format
var filter = [{
field: "Subject", // field configure
operator: "contains",
value: "Music",
predicate: "or" // predicate
}, {

```

```

field: "Subject", // field configure
operator: "contains",
value: "School",
predicate: "or" // predicate
}];
var schObj = $("#Schedule1").data("ejSchedule");
// Method to get the Filtered appointment
var result = schObj.filterAppointments(filter);
if (!ej.isNullOrUndefined(result) && result.length != 0) {
$("#grid1").show();
$("#grid1").data("ejGrid");
$("#grid1").ejGrid("destroy");
$("#grid1").ejGrid({
dataSource: result,
allowPaging: true
});
}
},
render: function () {
return (
<EJ.Button id="btnClick" text="Search" click={this.onClick}>
</EJ.Button>
);
}
});
ReactDOM.render(<Scheduler />, document.getElementById('schedule-default'));
ReactDOM.render(<Button />, document.getElementById('btnSearch'));

```

### Recurrence Options

There are scenarios where you require the same appointments to be repeatedly created for multiple days on daily, weekly, monthly, and yearly or on every weekday basis.

In appointment data collection, **recurrence** and **recurrenceRule** are dependent fields. While creating or binding the recurrence appointment, the **recurrence** field is set to **true** and **recurrenceRule** contains recurrence pattern in string format.

#### Recurrence Rule

The recurrence appointments are created based on the recurrence rule. The RecurrenceRule is a string value that contains the details of the recurrence appointments like

- repeat type - daily/weekly/monthly/yearly/every weekday
- how many times it needs to be repeated
- the interval duration
- the time period to render the appointment, etc.,

It has the following properties based on which the recurrence appointments are rendered in the Schedule control with its respective time period.

S.No	Property	Purpose
1	FREQ	Maintains the Repeat type value of the appointment.

		<p>(Example: Daily, Weekly, Monthly, Yearly, Every week day)</p> <p>Example: FREQ=DAILY;INTERVAL=1</p>
2	INTERVAL	<p>Maintains the interval value of the appointments.</p> <p>For example, when you create the daily appointment at an interval of 2, the appointments are rendered on the days Monday, Wednesday and Friday. (Creates an appointment on all days by leaving the interval of one day gap)</p> <p>Example: FREQ=DAILY;INTERVAL=2</p>
3	COUNT	<p>It holds the appointment's count value.</p> <p>For example, When the recurrence appointment count value is 10, which means that 10 instances of appointments are created in the recurrence series.</p> <p>Example: FREQ=DAILY;INTERVAL=1;COUNT=10</p>
4	UNTIL	<p>This property is used to store the recurrence end date value.</p> <p>For example, when you set the end date of appointment as 11/30/2015, the UNTIL property holds the end date value denoting when the recurrence actually ends.</p> <p>Example: FREQ=DAILY;INTERVAL=1;UNTIL=11/30/2015</p>
5	BYDAY	<p>It holds the DAY values, representing on which the appointments actually renders.</p> <p>For example, Create the weekly appointment, and select the day(s) from the day options (Monday/Tuesday/Wednesday/Thursday/Friday/Saturday/Sunday). When Monday is selected, the first two letters of the selected day "MO" is saved in the BYDAY property. When multiple days are selected, the values are separated by commas.</p> <p>Example: FREQ=WEEKLY;INTERVAL=1;BYDAY=MO,WE;COUNT=10</p>
6	BYMONTHDAY	<p>This property is used to store the date value of the Month, while creating the Month recurrence appointment.</p> <p>For example, when you create a Monthly recurrence appointment for every 3rd day of the month, then BYMONTHDAY holds the value 3 and creates the appointment on 3rd day of every month.</p> <p>Example: FREQ=MONTHLY;BYMONTHDAY=3;INTERVAL=1;COUNT=10</p>
7	BYMONTH	<p>This property is used to store the index value of the selected Month while creating the yearly appointments.</p>

		<p>For example, when you create the yearly appointment on June month, the index value of June month 6 will get stored in the BYMONTH field. The appointment is created on every 6th month of a year.</p> <p>Example: <code>FREQ=YEARLY;BYMONTHDAY=16;BYMONTH=6;INTERVAL=1;COUNT=10</code></p>
8	BYSETPOS	<p>This property is used to store the index value of the week.</p> <p>For example, when you create the monthly appointment in second week of a month, the index value of the second week (2) is stored in BYSETPOS.</p> <p>Example: <code>FREQ=MONTHLY;BYDAY=MO;BYSETPOS=2;UNTIL=8/11/2015</code></p>
9	WKST	<p>This property is used to store the start day value of a week.</p> <p>For example, when you render the workweek the "WKST" value is Monday.</p>
10	EXDATE	<p>EXDATE is used to hold the modified appointment date details (date value) in the recurrence appointment series.</p> <p>For example, when you change the recurrence appointment instance under the date "6/19/2015", then this date is added to the recurrence rule EXDATE field. "EXDATE" is also used to differentiate the edited occurrence of the recurrence series for some internal process while doing the "Edit Series or Delete series" actions.</p> <p>Example:  <code>FREQ=WEEKLY;BYDAY=MO,TU,WE,TH,FR;COUNT=10;EXDATE=6/18/2015,6/20/2015;RECUREDITID=1651</code></p>
11	RECUREDITID	<p>This property contains the Parent Id value of the edited appointment. It is used to track the edited appointment occurrence with its parent recurrence appointment series.</p> <p>For example, when you edit the particular occurrence of the recurrence appointment series, the "RECUREDITID" is added to that edited appointment depicting its parent Id.</p> <p>Example:  <code>FREQ=WEEKLY;BYDAY=MO,TU,WE,TH,FR;COUNT=10;EXDATE=6/18/2015,6/20/2015;RECUREDITID=1651</code></p>

To know more about other possible combinations of above specified recurrence rule properties, refer [here](#).

#### **HTML**

```

var dataManager = [{
  Id: 1,
  Subject: "Music Class",
  StartTime: new Date("2015/11/7 06:00 AM"),
  EndTime: new Date("2015/11/7 07:00 AM")
}, {
  Id: 2,
  Subject: "School",
  StartTime: new Date("2015/11/7 9:00 AM"),
  EndTime: new Date("2015/11/7 02:30 PM"),
  Recurrence: true, //enable recurrence options
  RecurrenceRule: "FREQ=DAILY;INTERVAL=1;COUNT=5" //Recurrence rule.
}];
var Scheduler = React.createClass({
  render: function () {
    return (
      <EJ.Schedule id="Schedule1" width="100%" height="525px" currentDate={new
      Date(2017, 5, 5)} appointmentSettings-dataSource={dataManager}
      appointmentSettings-recurrence="Recurrence" appointmentSettings-
      recurrenceRule="RecurrenceRule">
      </EJ.Schedule>
    );
  }
});
ReactDOM.render(<Scheduler />, document.getElementById('schedule-default'));

```

### Recurrence Validation

The default recurrence validation has been included for recurrence appointments similar to the one available in outlook. The validation occurs during the recurrence appointment creation, drag and drop or resizing of the recurrence appointments and also if any single occurrence changes. The validation can be disabled by setting the [enableRecurrenceValidation](#) property to false.

### HTML

```

var dataManager = [{
  Id: 1,
  Subject: "Music Class",
  StartTime: new Date("2015/11/7 06:00 AM"),
  EndTime: new Date("2015/11/7 07:00 AM")
}, {
  Id: 2,
  Subject: "School",
  StartTime: new Date("2015/11/7 9:00 AM"),
  EndTime: new Date("2015/11/7 02:30 PM"),
  Recurrence: true, //enable recurrence options
  RecurrenceRule: "FREQ=DAILY;INTERVAL=1;COUNT=5" //Recurrence rule.
}];
var Scheduler = React.createClass({
  render: function () {
    return (
      <EJ.Schedule id="Schedule1" width="100%" height="525px" currentDate={new
      Date(2017, 5, 5)} enableRecurrenceValidation={false} appointmentSettings-
      dataSource={dataManager} appointmentSettings-recurrence="Recurrence"
      appointmentSettings-recurrenceRule="RecurrenceRule">
      </EJ.Schedule>
    );
  }
});

```



```

}
});
ReactDOM.render(<Scheduler />, document.getElementById('schedule-default'));

```

**Note:** You can parse the **RecurrenceRule** of an appointment from the server-side by making use of a new generic utility class **RecurrenceHelper**. Refer this [KB document](#).

#### *Recurrence Edit and delete options*

The recurring appointments can be edited or deleted in three ways as below:

- Single occurrence
- Following appointments
- Entire series

#### *Single occurrence*

When an option "Only this Appointment" is selected, a single occurrence of the recurrence appointment alone will be edited or deleted.

#### *Following appointments*

When an option "Following Appointments" is selected, all the following events of the recurrence appointment from the current instance will be edited or deleted. The previous instances of the recurrence appointment before this current instances will be retained as it is on the Scheduler.

#### *Entire series*

The entire recurrence series will be edited / deleted, on selecting this option.

#### *Reminder*

Reminder option notifies all the appointments before some specific time. By default, it notifies before 5 minutes. Each and every appointment triggers the [reminder](#) event and can utilize this event for other user actions like mailing particular event to someone or to do any kind of manipulations with the reminder appointments and so on. The `reminderSettings` includes the following 2 properties namely,

- **enable** - To enable the reminder settings of the Schedule control, set the **enable** property as **true** within the [reminderSettings](#) option.
- **alertBefore** - Accepts the integer value to denote the time, before how long the reminder should be notified to the user.

#### **HTML**

```

var dataManager = [{
  Id: 1,
  Subject: "Music Class",
  StartTime: new Date(new Date().setMinutes(new Date().getMinutes() + 11)), //
  new Date("2015/11/7 06:00 AM"),
  EndTime: new Date(new Date().setHours(new Date().getHours() + 1)) // new
  Date("2015/11/7 07:00 AM")
}, {
  Id: 2,
  Subject: "School",
  StartTime: new Date(new Date().setHours(new Date().getHours() + 2)),
  EndTime: new Date(new Date().setHours(new Date().getHours() + 4))
}];

```

```

var reminderData = {
  enable: true, //enable the reminder options
  alertBefore: 10 //notify before 10 minutes
};
var Scheduler = React.createClass({
  reminderCustom: function(args) {
    alert("Reminder Appointment");
  },
  render: function () {
    return (
      <EJ.Schedule id="Schedule1" width="100%" height="525px" currentDate={new
      Date(2017, 5, 5)} reminderSettings={reminderData} appointmentSettings-
      dataSource={dataManager} reminder={this.reminderCustom}>
      </EJ.Schedule>
    );
  }
});
ReactDOM.render(<Scheduler />, document.getElementById('schedule-default'));

```

**Note:** Whenever the reminder setting is enabled in the Scheduler with some specific value (in minutes) assigned to the **alertBefore** property, the **reminder** event gets triggered before this specified value. It includes the reminder appointment's entire information within its arguments.

### Block Time Intervals

It allows to block the particular timeslots in Schedule. When specific timeslots are blocked, the appointments that lies in that range can either be made read-only or else can be allowed to interact with the users based on the value assigned to the **isBlockAppointment** property.

### Blockout Settings

The **blockoutSettings** holds the below block intervals related properties such as,

- **enable** - It accepts true or false value, denoting whether to enable/disable the block intervals option. It's default value is **false**.
- **templateId** - It applies the template design to block the intervals.
- **dataSource** - Binds the block intervals dataSource collection. This property should be assigned either with the JSON data array collection or instance of **ej.DataManger**.

The below blockout fields holds the appropriate column names from the dataSource -

Field name	Description
id	It holds the binding name for id field in the blockout dataSource
subject	It holds the binding name for subject field in the blockout dataSource
startTime	It holds the binding name for startTime field in the blockout dataSource.
endTime	It holds the binding name for endTime field in the blockout dataSource.

isBlockAppointment	It holds the binding name for isBlockAppointment field in the blockout dataSource.
isAllDay	It holds the binding name for isAllDay field in the blockout dataSource.
resourceId	It holds the binding name for resourceId field in the blockout dataSource.
customStyle	It holds the binding name for customStyle field in the blockout dataSource.

**HTML**

```

var dataManager = [{
  BlockId: 101,
  BlockStartTime: new Date(2014, 4, 5, 10, 00),
  BlockEndTime: new Date(2014, 4, 5, 11, 00),
  BlockSubject: "Service",
  IsBlockAppointment: true
}, {
  BlockId: 102,
  BlockStartTime: new Date(2014, 4, 4, 12, 00),
  BlockEndTime: new Date(2014, 4, 4, 13, 00),
  BlockSubject: "Maintenance",
  IsBlockAppointment: true
}];
var blockoutData = {
  enable: true, //Enable the blockout options
  dataSource: dataManager, //data source collection binding
  id: "BlockId", //id mapper field
  startTime: "BlockStartTime", //start time mapper field
  endTime: "BlockEndTime", //end time mapper field
  subject: "BlockSubject", //subject mapper field
  isBlockAppointment: "IsBlockAppointment" //block appointment mapper field
};
var Scheduler = React.createClass({
  render: function () {
    return (
      <EJ.Schedule id="Schedule1" width="100%" height="525px" currentDate={new
      Date(2017, 5, 5)} blockoutSettings={blockoutData}>
      </EJ.Schedule>
    );
  }
});
ReactDOM.render(<Scheduler />, document.getElementById('schedule-default'));

```

*Blocking Appointments*

The Appointments that lies within the blocked time range can be restricted to perform CRUD operations in it and can be made read-only. This can be achieved by setting [isBlockAppointment](#) property to true.

**HTML**

```

var dataManager = [{
  BlockId: 101,
  BlockStartTime: new Date(2014, 4, 5, 10, 00),

```

```

BlockEndTime: new Date(2014, 4, 5, 11, 00),
BlockSubject: "Service",
IsBlockAppointment: true
}];
var blockoutData = {
enable: true, //Enable the blockout options
dataSource: dataManager, //data source collection binding
id: "BlockId",
startTime: "BlockStartTime",
endTime: "BlockEndTime",
subject: "BlockSubject",
isBlockAppointment: "IsBlockAppointment" //Bind the block appointment field
in datasource
};
var Scheduler = React.createClass({
render: function () {
return (
<EJ.Schedule id="Schedule1" width="100%" height="525px" currentDate={new
Date(2017, 5, 5)} blockoutSettings={blockoutData}>
</EJ.Schedule>
);
}
});
ReactDOM.render(<Scheduler />, document.getElementById('schedule-default'));

```

#### Customizing block time intervals

The [blockoutSettings](#) holds the below properties to customize the block intervals such as,

- [templateId](#) - Template design that applies on the block intervals.
- [customStyle](#) - The custom CSS that applies on the blocked intervals.

#### HTML

```

<!--Container for ejScheduler widget-->
<div id="schedule-default"></div>
<!--Template to apply block intervals-->
<script id="blockTemplate" type="text/x-jsrender">
<div style="height:100%">
<div>{:BlockSubject}</div>
</div>
</script>

```

#### HTML

```

var dataManager = [{
BlockId: 101,
BlockStartTime: new Date(2014, 4, 5, 10, 00),
BlockEndTime: new Date(2014, 4, 5, 11, 00),
BlockSubject: "Service",
IsBlockAppointment: true
}];
var blockoutData = {
enable: true,
templateId: "#blockTemplate",

```

```

dataSource: dataManager,
id: "BlockId",
startTime: "BlockStartTime",
endTime: "BlockEndTime",
subject: "BlockSubject",
isBlockAppointment: "IsBlockAppointment"
});
var Scheduler = React.createClass({
render: function () {
return (
<EJ.Schedule id="Schedule1" width="100%" height="525px" currentDate={new
Date(2017, 5, 5)} blockoutSettings={blockoutData}>
</EJ.Schedule>
);
}
});
ReactDOM.render(<Scheduler />, document.getElementById('schedule-default'));

```

### *Blocking time interval based on resources*

- [resourceId](#) - property used within the `blockoutSettings` which accepts the resource id's can be used to apply the block intervals based on the resources.

### **HTML**

```

var groupData = { resources: ["Owners"] };
var resourcesData = [{
field: "ownerId",
title: "Owner",
name: "Owners",
allowMultiple: true,
resourceSettings: {
dataSource: [
{ text: "Nancy", id: 1, color: "#f8a398" },
{ text: "Steven", id: 2, color: "#56ca85" }
],
text: "text", id: "id", color: "color"
}
}];
var appointData = {
dataSource: [{
EventId: 100,
EventSubject: "Research on Sky Miracles",
EventStartTime: new Date(2014, 4, 2, 9, 00),
EventEndTime: new Date(2014, 4, 2, 10, 30),
ownerId: 2
}],
id: "EventId",
startTime: "EventStartTime",
endTime: "EventEndTime",
subject: "EventSubject",
resourceFields: "ownerId"
};
var blockoutData = {
enable: true,

```

```

dataSource: [{
  BlockId: 101,
  BlockStartTime: new Date(2014, 4, 1, 10, 00),
  BlockEndTime: new Date(2014, 4, 1, 11, 00),
  BlockSubject: "Travel",
  IsBlockAppointment: true,
  BlockResId: 2
}],
id: "BlockId",
startTime: "BlockStartTime",
endTime: "BlockEndTime",
subject: "BlockSubject",
isBlockAppointment: "IsBlockAppointment",
resourceId: "BlockResId"
};
var Scheduler = React.createClass({
  render: function () {
    return (
      <EJ.Schedule id="Schedule1" width="100%" height="525px" currentDate={new
      Date(2017, 5, 5)} group={groupData} resources={resourcesData}
      appointmentSettings={appointData} blockoutSettings={blockoutData}>
    </EJ.Schedule>
    );
  }
});
ReactDOM.render(<Scheduler />, document.getElementById('schedule-default'));

```

## Context Menu

Scheduler provides default context menu options for both appointments as well as work cells. It also allows to define additional custom context menu options.

The options that are available under [contextMenuSettings](#) are as follows,

- **enable** - Enables/disables the context menu option in Scheduler.
- **menuItems** - Contains the sub-menu collections related to both the appointment and cells.

## Default Menu Options

The menu items contains two separate collection based on the appointment and cells.

### Appointment

The appointment collection includes the following options.

Open Appointment (default)
Delete Appointment (default)
Print Appointment
Categorize

### Cells

The default options available within the cell collection includes -

New Appointment
New Recurring Appointment
Today
Go to date
Settings (View, TimeMode, Work Hours)

The following code snippet shows how to enable the context menu settings in Scheduler and to make use of it with default available options.

### HTML

```
var contextMenuData = {
  enable: true,
  menuItems: {
    appointment: [
      { id: "open", text: "Open Appointment" },
      { id: "delete", text: "Delete Appointment" }
    ],
    cells: [
      { id: "new", text: "New Appointment" },
      { id: "recurrence", text: "New Recurring Appointment" },
      { id: "today", text: "Today" },
      { id: "goToDate", text: "Go to date" },
      { id: "settings", text: "Settings" },
      { id: "view", text: "View", parentId: "settings" },
      { id: "timeMode", text: "TimeMode", parentId: "settings" },
      { id: "view_Day", text: "Day", parentId: "view" },
      { id: "view_Week", text: "Week", parentId: "view" },
      { id: "view_Workweek", text: "Workweek", parentId: "view" },
      { id: "view_Month", text: "Month", parentId: "view" },
      { id: "timeMode_Hour12", text: "12 Hours", parentId: "timeMode" },
      { id: "timeMode_Hour24", text: "24 Hours", parentId: "timeMode" },
      { id: "workhours", text: "Work Hours", parentId: "settings" }
    ]
  }
};

var Scheduler = React.createClass({
  render: function () {
    return (
      <EJ.Schedule id="Schedule1" width="100%" height="525px" currentDate={new
      Date(2017, 5, 5)} contextMenuSettings={contextMenuData}>
      </EJ.Schedule>
    );
  }
});

ReactDOM.render(<Scheduler />, document.getElementById('schedule-default'));
```

**Note:** In agenda view, only the appointment menu items shows up in the context menu options. For default menu items, the id must be defined the same as mentioned in the above code example – as we have processed the menus based on this id within our source.

### Custom Menu Options

Apart from the default available options, it is also possible to add custom menu options to the context-menu of both the appointment and cell collection.

The following code example depicts how **to add the custom menu items** to the appointment and cell collection of the context menu settings.

#### HTML

```
var appointData = [{
  Id: 100,
  Subject: "Research on Sky Miracles",
  StartTime: new Date(2015, 11, 2, 9, 00),
  EndTime: new Date(2015, 11, 2, 10, 30)
}];
var contextMenuData = {
  enable: true,
  menuItems: {
    appointment: [
      { id: "open", text: "Open Appointment" },
      { id: "delete", text: "Delete Appointment" },
      { id: "option1", text: "User Option 1" }
    ],
    cells: [
      { id: "celloption1", text: "Custom Option 1" }
    ]
  }
};
var Scheduler = React.createClass({
  render: function () {
    return (
      <EJ.Schedule id="Schedule1" width="100%" height="525px" currentDate={new
      Date(2017, 5, 5)} contextMenuSettings={contextMenuData} appointmentSettings-
      dataSource={appointData}>
    </EJ.Schedule>
  );
}
});
ReactDOM.render(<Scheduler />, document.getElementById('schedule-default'));
```

**Note:** The **id** given for the custom menu items **must be unique** in both the appointment and cell collection.

### Handling Menu Actions

To define specific actions for a click made on the custom menu items, the client-side event [menuItemClick](#) can be used as depicted in the below code example.

#### HTML

```
var appointData = [{
  Id: 100,
  Subject: "Research on Sky Miracles",
  StartTime: new Date(2015, 11, 2, 9, 00),
  EndTime: new Date(2015, 11, 2, 10, 30)
}];
var contextMenuData = {
```



```

enable: true,
menuItems: {
  appointment: [
    { id: "open", text: "Open Appointment" },
    { id: "delete", text: "Delete Appointment" },
    { id: "option1", text: "User Option 1" }
  ],
  cells: [
    { id: "celloption1", text: "Custom Option 1" }
  ]
}
};
var Scheduler = React.createClass({
  onMenuItemClick: function() {
    //args.events contains information of the clicked menu item.
    if (args.events.ID == "option1")
      alert("Custom menu clicked");
  },
  render: function () {
    return (
      <EJ.Schedule id="Schedule1" width="100%" height="525px" currentDate={new
      Date(2017, 5, 5)} contextMenuSettings={contextMenuData} appointmentSettings-
      dataSource={appointData} menuItemClick={this.onMenuItemClick}>
    </EJ.Schedule>
    );
  }
});
ReactDOM.render(<Scheduler />, document.getElementById('schedule-default'));

```

Also, it is possible to predict the target on which the right click is made, either on the cells or appointments with the use of the event [beforeContextMenuOpen](#). The below code example shows how to block the display of context menu, when right clicked on the cells by setting **args.cancel** as **true**.

### HTML

```

var appointData = [{
  Id: 100,
  Subject: "Research on Sky Miracles",
  StartTime: new Date(2015, 11, 2, 9, 00),
  EndTime: new Date(2015, 11, 2, 10, 30)
}];
var contextMenuData = {
  enable: true,
  menuItems: {
    appointment: [
      { id: "open", text: "Open Appointment" },
      { id: "delete", text: "Delete Appointment" },
      { id: "option1", text: "User Option 1" }
    ]
  }
};
var Scheduler = React.createClass({
  onBeforeContextMenuOpen: function(args) {
    //args.events.target - target information to depict either cell/appointment
    if ($(args.events.target).hasClass("e-workcells") ||
    $(args.events.target).hasClass("e-monthcells"))

```

```

args.cancel = true;
},
render: function () {
return (
<EJ.Schedule id="Schedule1" width="100%" height="525px" currentDate={new
Date(2017, 5, 5)} beforeContextMenuOpen={this.onBeforeContextMenuOpen}
appointmentSettings-dataSource={appointData}>
</EJ.Schedule>
);
}
});
ReactDOM.render(<Scheduler />, document.getElementById('schedule-default'));

```

### Adding Categorize Option

To include the default categorize option within the context menu, it is necessary to enable the [categorizeSettings](#) property as shown in the below code example.

### HTML

```

var appointData = [{
Id: 100,
Subject: "Research on Sky Miracles",
StartTime: new Date(2015, 11, 2, 9, 00),
EndTime: new Date(2015, 11, 2, 10, 30)
}];
var contextMenuData = {
enable: true,
menuItems: {
appointment: [
{ id: "open", text: "Open Appointment" },
{ id: "delete", text: "Delete Appointment" },
{ id: "categorize", text: "Categorize" }
],
}
};
var Scheduler = React.createClass({
onBeforeContextMenuOpen: function(args) {
//args.events.target - target information to depict either cell/appointment
if ($(args.events.target).hasClass("e-workcells") ||
$(args.events.target).hasClass("e-monthcells"))
args.cancel = true;
},
render: function () {
return (
<EJ.Schedule id="Schedule1" width="100%" height="525px" currentDate={new
Date(2017, 5, 5)} contextMenuSettings={contextMenuData} categorizeSettings-
enable={true} appointmentSettings-dataSource={appointData}>
</EJ.Schedule>
);
}
});
ReactDOM.render(<Scheduler />, document.getElementById('schedule-default'));

```

**Note:** The **categorize** option must be added only to the **appointment** collection, which displays on right clicking the appointments.

*Binding remote data*

It is also possible to bind the categorize datasource using remote data. The below code example shows how to bind the remote dataSource to the `categorizeSettings`.

**HTML**

```
var appointData = [{
  Id: 100,
  Subject: "Research on Sky Miracles",
  StartTime: new Date(2015, 11, 2, 9, 00),
  EndTime: new Date(2015, 11, 2, 10, 30),
  Categorize: "1"
}];
var dataManager = ej.DataManager({
  url: "Home/GetCategorizeData",
  adaptor: new ej.UrlAdaptor()
});
var categorizeSettingsData = {
  enable: true,
  allowMultiple: true,
  dataSource: dataManager,
  id: "Id",
  fontColor: "FontColor",
  color: "Color",
  text: "Text"
};
var Scheduler = React.createClass({
  onBeforeContextMenuOpen: function(args) {
    //args.events.target - target information to depict either cell/appointment
    if ($(args.events.target).hasClass("e-workcells") ||
      $(args.events.target).hasClass("e-monthcells"))
      args.cancel = true;
  },
  render: function () {
    return (
      <EJ.Schedule id="Schedule1" width="100%" height="525px" currentDate={new
      Date(2017, 5, 5)} categorizeSettings={categorizeSettingsData}
      appointmentSettings-dataSource={appointData} appointmentSettings-
      categorize="Categorize">
      </EJ.Schedule>
    );
  }
});
ReactDOM.render(<Scheduler />, document.getElementById('schedule-default'));
```

The server-side controller code to fetch and pass the categorize data from the database to the Schedule are as follows.

**C#**

```
public JsonResult GetCategorizeData()
{
  // ScheduleDataDataContext is a LINQ-to-SQL data class name that is defined
  in the .dbml file to access the tables from the database
  IEnumerable data = new ScheduleDataDataContext().CategoryData;
  return Json(data, JsonRequestBehavior.AllowGet);
}
```

```
}

```

## Resources

The Scheduler provides **Resources** support, with which the single Scheduler is shared by multiple resources simultaneously. Each resource in the Scheduler is arranged in a column/row wise, with individual spacing to display all its respective appointments on a single page. It also supports the grouping of resources, thus enabling the categorization of resources in a hierarchical structure and shows it either in expandable groups (**Horizontal view**) or else vertical hierarchy one after the other (**Vertical view**).

One or more resources can be assigned to the Scheduler appointments by making selection of the resource options available in the appointment window.

### Fields of Resources

The default options available within the [resources](#) collection are as follows,

*name (\*\*String\*\*)*

A unique resource name which is used for differentiating various resource objects while grouping it in levels.

*title (\*\*String\*\*)*

It holds the title name of the resource field to be displayed on the Scheduler appointment window.

*field (\*\*String\*\*)*

It holds the name of the resource field to be bound to the Scheduler appointments which contains the resource Id.

*allowMultiple (\*\*Boolean\*\*)*

When set to true, allows multiple selection of resource names, thus creating multiple instances of same appointment for the selected resources.

*resourceSettings (\*\*Object\*\*)*

It holds the field names of the resources dataSource to be bound to the Scheduler.

The following are the resource fields which must be defined within the **resourceSettings** that holds the appropriate column names from the dataSource.

Field name	Description
text	Binds the text field name in the dataSource to the resourceSettings text. These text gets listed out in the resources field of the appointment window. Itâ€™s mandatory.
id	Binds the id field name in the dataSource to the resourceSettings id. Itâ€™s mandatory.
groupId	Binds the groupId field name in the dataSource to the resourceSettings groupId. This field is not necessary for a resource object (resource data) defined as first level

	within the resources collection.
color	Binds the color field name in the dataSource to the resourceSettings color. It is optional.
appointmentClass	Binds the appointmentClass field name in the dataSource. It applies the custom CSS class name to the appointments based on the resources.
start	Binds the starting work hour field name in the dataSource. It's optional, but when provided with some numeric value will set the starting work hour for specific resources.
end	Binds the end work hour field name in the dataSource. It's optional, but when provided with some numeric value will set the end work hour for specific resources.
workWeek	Binds the resources working days field name in the dataSource. It's optional, and accept array of strings which is nothing but only the week day names. When provided with some values (array of day names), only those days will render for the specific resources.

**Example:** To set the resources options using all the above specified fields,

#### HTML

```
var dManager = [{
  Id: 100,
  Subject: "Research on Sky Miracles",
  StartTime: new Date(2015, 04, 05, 9, 00),
  EndTime: new Date(2015, 04, 05, 10, 30),
  OwnerId: 3,
  RoomId: 2
}];
var groupData = { resources: ["Rooms, Owners"] };
var roomData = {
  dataSource: [
    { text: "ROOM 1", id: 1, groupId: 1, color: "#cb6bb2" },
    { text: "ROOM 2", id: 2, groupId: 1, color: "#56ca85" }
```

```

],
text: "text", id: "id", groupId: "groupId", color: "color"
};
var ownerData = {
dataSource: [
{ text: "Nancy", id: 1, groupId: 1, color: "#ffaa00", on: 10, off: 18 },
{ text: "Steven", id: 3, groupId: 2, color: "#f8a398", on: 6, off: 10 },
{ text: "Michael", id: 5, groupId: 1, color: "#7499e1", on: 11, off: 15 }
],
text: "text", id: "id", groupId: "groupId", color: "color", start: "on",
end: "off"
};
var Scheduler = React.createClass({
render: function () {
return (
<EJ.Schedule id="Schedule1" width="100%" height="525px" currentDate={new
Date(2017, 5, 5)} group={groupData} appointmentSettings-
dataSource={dManager} appointmentSettings-id="Id" appointmentSettings-
subject="Subject" appointmentSettings-startTime="StartTime"
appointmentSettings-endTime="EndTime" appointmentSettings-
description="Description" appointmentSettings-allDay="AllDay"
appointmentSettings-recurrence="Recurrence" appointmentSettings-
recurrenceRule="RecurrenceRule" appointmentSettings-resourceFields=
"RoomId,OwnerId">
<resources>
<resource allowMultiple={false} field="RoomId" title="Room" name="Rooms"
resourceSettings={roomData}></resource>
<resource allowMultiple={true} field="OwnerId" title="Owner" name="Owners"
resourceSettings={ownerData}></resource>
</resources>
</EJ.Schedule>
);
}
});
ReactDOM.render(<Scheduler />, document.getElementById('schedule-default'));

```

**Note:** The resource object defined at **first level** within the **resources** collection doesn't make use of the **groupId** field, as there is no previous levels applicable to map.

### Data Binding

The resource data can be bound to the Schedule control through the **resourceSettings** options available within the **resources** property. The data-binding can be done either using JSON object collection or DataManager ([ej.DataManager](#)) instance which contains the resources related data.

### JSON Data

**Example:** To set the resource data with array of **JSON** object collection

### HTML

```

var dManager = [{
Id: 100,
Subject: "Research on Sky Miracles",
StartTime: new Date(2015, 04, 05, 9, 00),
EndTime: new Date(2015, 04, 05, 10, 30),
OwnerId: 2
}, {

```

```

Id: 101,
Subject: "Research on Clouds",
StartTime: new Date(2015, 04, 07, 7, 00),
EndTime: new Date(2015, 04, 07, 10, 30),
OwnerId: 1
});
var groupData = { resources: ["Owners"] };
var ownerData = {
dataSource: [
{ text: "Nancy", id: 1, color: "#f8a398" },
{ text: "Steven", id: 2, color: "#56ca85" }
],
text: "text", id: "id", color: "color"
};
var Scheduler = React.createClass({
render: function () {
return (
<EJ.Schedule id="Schedule1" width="100%" height="525px" currentDate={new
Date(2017, 5, 5)} group={groupData} appointmentSettings-
dataSource={dManager} appointmentSettings-id="Id" appointmentSettings-
subject="Subject" appointmentSettings-startTime="StartTime"
appointmentSettings-endTime="EndTime" appointmentSettings-
description="Description" appointmentSettings-allDay="AllDay"
appointmentSettings-recurrence="Recurrence" appointmentSettings-
recurrenceRule="RecurrenceRule" appointmentSettings-resourceFields=
"OwnerId">
<resources>
<resource allowMultiple={true} field="OwnerId" title="Owner" name="Owners"
resourceSettings={ownerData}></resource>
</resources>
</EJ.Schedule>
);
}
});
ReactDOM.render(<Scheduler />, document.getElementById('schedule-default'));

```

### Remote Data

**Example:** To set the resource data through the instance of **ejDataManager**

### HTML

```

var dManager = [{
Id: 100,
Subject: "Research on Sky Miracles",
StartTime: new Date(2015, 04, 05, 9, 00),
EndTime: new Date(2015, 04, 05, 10, 30),
OwnerId: 2
}, {
Id: 101,
Subject: "Research on Clouds",
StartTime: new Date(2015, 04, 07, 7, 00),
EndTime: new Date(2015, 04, 07, 10, 30),
OwnerId: 1
}
];
var dataManager = ej.DataManager({
// referring data from remote service (url binding)

```

```

url: "http://mvc.syncfusion.com/OdataServices/Northwnd.svc"
});
// query to fetch the records from the specified table "Events"
var queryResource = ej.Query().select("CategoryID",
"CategoryName").from("Categories").take(3);
var groupData = { resources: ["Owners"] };
var ownerData = {
dataSource: dataManager,
text: "CategoryName",
id: "CategoryID",
query: queryResource
};
var Scheduler = React.createClass({
render: function () {
return (
<EJ.Schedule id="Schedule1" width="100%" height="525px" currentDate={new
Date(2017, 5, 5)} group={groupData} appointmentSettings-
dataSource={dManager} appointmentSettings-id="Id" appointmentSettings-
subject="Subject" appointmentSettings-startTime="StartTime"
appointmentSettings-endTime="EndTime" appointmentSettings-
description="Description" appointmentSettings-allDay="AllDay"
appointmentSettings-recurrence="Recurrence" appointmentSettings-
recurrenceRule="RecurrenceRule" appointmentSettings-resourceFields=
"OwnerId">
<resources>
<resource allowMultiple={true} field="OwnerId" title="Owner" name="Owners"
resourceSettings={ownerData}></resource>
</resources>
</EJ.Schedule>
);
}
});
ReactDOM.render(<Scheduler />, document.getElementById('schedule-default'));

```

### Multiple Resources (Without Grouping)

It is possible to display the Scheduler in default look without visually showcasing all the resources on it, but it allow the user to assign the required resources to the appointments through the appointment window resource options.

The appointments belonging to all the resources will be displayed on the Scheduler which will be differentiated based on the resource color assigned in the **resourceSettings** (depicting to which resource that particular appointment belongs).

**Example:** To display default Scheduler with multiple resource options in the appointment window,

### HTML

```

var dManager = [{
Id: 100,
Subject: "Research on Sky Miracles",
StartTime: new Date(2015, 04, 05, 9, 00),
EndTime: new Date(2015, 04, 05, 10, 30),
OwnerId: 2
}, {
Id: 101,
Subject: "Research on Clouds",

```



```

StartTime: new Date(2015, 04, 07, 6, 00),
EndTime: new Date(2015, 04, 07, 9, 30),
OwnerId: 1
}];
var ownerData = {
dataSource: [
{ text: "Nancy", id: 1, color: "#f8a398" },
{ text: "Steven", id: 2, color: "#56ca85" }
],
text: "text", id: "id", color: "color"
};
var Scheduler = React.createClass({
render: function () {
return (
<EJ.Schedule id="Schedule1" width="100%" height="525px" currentDate={new
Date(2017, 5, 5)} appointmentSettings-dataSource={dManager}
appointmentSettings-id="Id" appointmentSettings-subject="Subject"
appointmentSettings-startTime="StartTime" appointmentSettings-
endTime="EndTime" appointmentSettings-description="Description"
appointmentSettings-allDay="AllDay" appointmentSettings-
recurrence="Recurrence" appointmentSettings-recurrenceRule="RecurrenceRule"
appointmentSettings-resourceFields= "OwnerId">
<resources>
<resource allowMultiple={true} field="OwnerId" title="Owner" name="Owners"
resourceSettings={ownerData}></resource>
</resources>
</EJ.Schedule>
);
}
});
ReactDOM.render(<Scheduler />, document.getElementById('schedule-default'));

```

**Note:** Setting **allowMultiple** to **true** in the above code snippet allows the user to select multiple resources in the appointment window and also creates multiple copies of the same appointment in the Scheduler for each resources while saving.

### Grouping

Scheduler supports both single and multiple levels of resource grouping that can be customized either in horizontal or vertical Scheduler views. In Vertical view - the levels are displayed in a tree structure one after the other, but in horizontal view – the levels are grouped in a vertically expandable/collapsible structure.

#### Single-Level

This type of grouping allows the Scheduler to display all the resources at a single level simultaneously. The appointments will make use of the **color** defined for the first resource instance as its background color.

**Example:** To display the Scheduler with single level resource grouping options,

### HTML

```

var dManager = [{
Id: 100,
Subject: "Research on Sky Miracles",
StartTime: new Date(2015, 04, 05, 9, 00),

```

```

EndTime: new Date(2015, 04, 05, 10, 30),
OwnerId: 2
}, {
Id: 101,
Subject: "Discovery of Exoplanets",
StartTime: new Date(2015, 04, 07, 6, 00),
EndTime: new Date(2015, 04, 07, 9, 30),
OwnerId: 1
}];
var groupData = { resources: ["Owners"] };
var ownerData = {
dataSource: [
{ text: "Nancy", id: 1, color: "#f8a398" },
{ text: "Steven", id: 2, color: "#56ca85" }
],
text: "text", id: "id", color: "color"
};
var Scheduler = React.createClass({
render: function () {
return (
<EJ.Schedule id="Schedule1" width="100%" height="525px" currentDate={new
Date(2017, 5, 5)} group={groupData} appointmentSettings-
dataSource={dManager} appointmentSettings-id="Id" appointmentSettings-
subject="Subject" appointmentSettings-startTime="StartTime"
appointmentSettings-endTime="EndTime" appointmentSettings-
description="Description" appointmentSettings-allDay="AllDay"
appointmentSettings-recurrence="Recurrence" appointmentSettings-
recurrenceRule="RecurrenceRule" appointmentSettings-resourceFields=
"OwnerId">
<resources>
<resource allowMultiple={true} field="OwnerId" title="Owner" name="Owners"
resourceSettings={ownerData}></resource>
</resources>
</EJ.Schedule>
);
}
});
ReactDOM.render(<Scheduler />, document.getElementById('schedule-default'));

```

**Note:** The **name** field mentioned in the **resource** object needs to be specified within the **group** property in order to enable the grouping option in Scheduler.

### Multi-Level

This type of grouping displays the resources in the Scheduler at multiple levels with a set of resources grouped under each parent. The appointments will make use of the **color** defined for the first/top level resource instance as its background color.

**Example:** To display the Scheduler with multiple level resource grouping options,

### HTML

```

var dManager = [{
Id: 100,
Subject: "Research on Sky Miracles",
StartTime: new Date(2015, 04, 05, 9, 00),
EndTime: new Date(2015, 04, 05, 10, 30),

```

```

OwnerId: 3,
RoomId: 2
}];
var groupData = { resources: ["Rooms, Owners"] };
var roomData = {
dataSource: [
{ text: "ROOM 1", id: 1, groupId: 1, color: "#cb6bb2" },
{ text: "ROOM 2", id: 2, groupId: 1, color: "#56ca85" }
],
text: "text", id: "id", groupId: "groupId", color: "color"
};
var ownerData = {
dataSource: [
{ text: "Nancy", id: 1, groupId: 1, color: "#ffaa00", on: 10, off: 18 },
{ text: "Steven", id: 3, groupId: 2, color: "#f8a398", on: 6, off: 10 },
{ text: "Michael", id: 5, groupId: 1, color: "#7499e1", on: 11, off: 15 }
],
text: "text", id: "id", groupId: "groupId", color: "color", start: "on",
end: "off"
};
var Scheduler = React.createClass({
render: function () {
return (
<EJ.Schedule id="Schedule1" width="100%" height="525px" currentDate={new
Date(2017, 5, 5)} group={groupData} appointmentSettings-
dataSource={dManager} appointmentSettings-id="Id" appointmentSettings-
subject="Subject" appointmentSettings-startTime="StartTime"
appointmentSettings-endTime="EndTime" appointmentSettings-
description="Description" appointmentSettings-allDay="AllDay"
appointmentSettings-recurrence="Recurrence" appointmentSettings-
recurrenceRule="RecurrenceRule" appointmentSettings-resourceFields=
"RoomId,OwnerId">
<resources>
<resource allowMultiple={false} field="RoomId" title="Room" name="Rooms"
resourceSettings={roomData}></resource>
<resource allowMultiple={true} field="OwnerId" title="Owner" name="Owners"
resourceSettings={ownerData}></resource>
</resources>
</EJ.Schedule>
);
}
});
ReactDOM.render(<Scheduler />, document.getElementById('schedule-default'));

```

**Note:** Here, the appointments will make use of the **color** defined for the Owners resource instance as its background color.

### Different Working days and Hours for Resources

It is possible to assign different workdays and workhours for each resources present within the Scheduler. The process of assigning different working days for every individual resources is applicable only for the vertical Scheduler mode and not for timeline view, whereas the customization of workhours for each resources is applicable on both the Scheduler orientation. The custom workdays and workhours needs to be defined within the `resourceSettings` property using the following 3 sub-properties available within it.

- [start](#) is used to define the work start hour for each individual resources.
- [end](#) is used to define the work end hour for each individual resources.
- [workWeek](#) is used to define different working days for each individual resources.

**Example:** To display the Scheduler with each individual resources having different workhours and workdays, the code example is depicted below.

### HTML

```
var dManager = [{
  Id: 100,
  Subject: "Research on Sky Miracles",
  StartTime: new Date(2015, 04, 05, 9, 00),
  EndTime: new Date(2015, 04, 05, 10, 30),
  OwnerId: 3,
  RoomId: 2
}];
var groupData = { resources: ["Rooms, Owners"] };
var roomData = {
  dataSource: [
    { text: "ROOM 1", id: 1, groupId: 1, color: "#cb6bb2" },
    { text: "ROOM 2", id: 2, groupId: 1, color: "#56ca85" }
  ],
  text: "text", id: "id", groupId: "groupId", color: "color"
};
var ownerData = {
  dataSource: [
    { text: "Nancy", id: 1, groupId: 1, color: "#ffaa00", on: 10, off: 18,
      customDays: ["monday", "wednesday", "friday"] },
    { text: "Steven", id: 3, groupId: 2, color: "#f8a398", on: 6, off: 10,
      customDays: ["tuesday", "thursday"] },
    { text: "Michael", id: 5, groupId: 1, color: "#7499e1", on: 11, off: 15,
      customDays: ["sunday", "tuesday", "thursday", "saturday"] }
  ],
  text: "text", id: "id", groupId: "groupId", color: "color", start: "on",
  end: "off", workWeek: "customDays"
};
var Scheduler = React.createClass({
  render: function () {
    return (
      <EJ.Schedule id="Schedule1" width="100%" height="525px" currentDate={new
Date(2017, 5, 5)} group={groupData} appointmentSettings-
dataSource={dManager} appointmentSettings-id="Id" appointmentSettings-
subject="Subject" appointmentSettings-startTime="StartTime"
appointmentSettings-endTime="EndTime" appointmentSettings-
description="Description" appointmentSettings-allDay="AllDay"
appointmentSettings-recurrence="Recurrence" appointmentSettings-
recurrenceRule="RecurrenceRule" appointmentSettings-resourceFields=
"RoomId,OwnerId">
        <resources>
          <resource allowMultiple={false} field="RoomId" title="Room" name="Rooms"
resourceSettings={roomData}></resource>
          <resource allowMultiple={true} field="OwnerId" title="Owner" name="Owners"
resourceSettings={ownerData}></resource>
        </resources>
      </EJ.Schedule>
```

```
);
}
});
ReactDOM.render(<Scheduler />, document.getElementById('schedule-default'));
```

## Customization

The Scheduler can be customized in various aspects like -

- Setting different Start/end hour limits
- Highlighting the working hours
- Setting different [date format](#)
- Specifying minimum and maximum date ranges
- Customize the entire appointment window with the user required fields
- Setting different time Slot duration
- Complete Scheduler customization using queryCellInfo event
- Setting different [first day of week](#)

## Hour Customization

It includes customization of displaying Scheduler with specific Start/End hours and also defining the working hour time range which is differentiated as business hours.

### Schedule Display Hours

It denotes the start and end hour time limits to be displayed on the Scheduler. To set this time limit, two properties namely [startHour](#) and [endHour](#) can be used.

- **startHour** - The hour from which the Scheduler time display actually starts.
- **endHour** - The hour on which the Scheduler time display should end.

The following code example renders the scheduler from 7.00 AM to 6.00 PM.

## HTML

```
var appointData = [{
  Id: 101,
  Subject: "Talk with Nature",
  StartTime: new Date(2015, 11, 5, 10, 00),
  EndTime: new Date(2015, 11, 5, 11, 00)
}];
var Scheduler = React.createClass({
  render: function () {
    return (
      <EJ.Schedule id="Schedule1" width="100%" height="525px" startHour={7}
        endHour={18} currentDate={new Date(2017, 5, 5)} appointmentSettings-
        dataSource={appointData}>
      </EJ.Schedule>
    );
  }
});
ReactDOM.render(<Scheduler />, document.getElementById('schedule-default'));
```

### Working Hours

Working hours indicates the work hour limit within the Scheduler, which is highlighted visually with white colored work cells. To enable the highlighting of work hours on the Scheduler, set the **highlight** option available within the workHours property to **true**. By default, it is set to true. [workHour](#) is a object property which contains the below specified options,

- [highlight](#) – enables/disables the highlighting of work hours.
- [start](#) - sets the start time of the working/business hour in a day.
- [end](#) - sets the end time limit of the working/business hour in a day.

### HTML

```
var appointData = [{
  Id: 101,
  Subject: "Talk with Nature",
  StartTime: new Date(2015, 11, 5, 10, 00),
  EndTime: new Date(2015, 11, 5, 11, 00)
}];
var workHourData = {
  highlight: true,
  start: 8,
  end: 16
};
var Scheduler = React.createClass({
  render: function () {
    return (
      <EJ.Schedule id="Schedule1" width="100%" height="525px"
        workHours={workHourData} currentDate={new Date(2017, 5, 5)}
        appointmentSettings-dataSource={appointData}>
      </EJ.Schedule>
    );
  }
});
ReactDOM.render(<Scheduler />, document.getElementById('schedule-default'));
```

**Note:** By default, work hour **start** is set to **9** and **end** is set to **18**. Also, the Scheduler cells automatically scrolls up or down based on the starting work hour, to make the user to view that particular time initially.

### TimeScale

The [TimeScale](#) allows the user to set the required time slot duration for the work cells that displays on the Scheduler. It provides option to customize both the major and minor slots using template option. It includes the below properties such as,

- [enable](#) - It accepts true or false value, denoting whether to show or hide the time slots. Its default value is **true**.
- [majorSlot](#) – Specifies the major time slot duration.
- [minorSlotCount](#) – Specifies the value, based on which the minor time slots are divided into appropriate count.
- [TimeScale templates](#) - 2 template options available for customizing timeScales namely [minorSlotTemplateId](#) and [majorSlotTemplateId](#).

The majorSlot and minorSlot can be set on the Scheduler with the following code example.

### HTML

```
var appointData = [{
  Id: 100,
  Subject: "Wild Discovery",
  StartTime: new Date(2015, 11, 2, 9, 00),
  EndTime: new Date(2015, 11, 2, 10, 30),
  Location: "CHINA"
}];
var timeScaleData = {
  enable: true,
  majorSlot: 60,
  minorSlotCount: 6
};
var Scheduler = React.createClass({
  render: function () {
    return (
      <EJ.Schedule id="Schedule1" width="100%" height="525px"
        timeScale={timeScaleData} currentDate={new Date(2017, 5, 5)}
        appointmentSettings-dataSource={appointData}>
      </EJ.Schedule>
    );
  }
});
ReactDOM.render(<Scheduler />, document.getElementById('schedule-default'));
```

### Hide Weekend days

The Scheduler can be customized to display only the working days, thus hiding the weekend days from it. The working days render based on the values given in the [workWeek](#) property. The days that are not mentioned in the `workWeek` collection is considered to be the weekend days and it can be hidden from the Scheduler by setting `false` to the [showWeekend](#) property.

The following code example renders the Scheduler by hiding the weekend days.

### HTML

```
var appointData = [{
  Id: 100,
  Subject: "Wild Discovery",
  StartTime: new Date(2015, 11, 2, 9, 00),
  EndTime: new Date(2015, 11, 2, 10, 30),
  Location: "CHINA"
}];
var Scheduler = React.createClass({
  render: function () {
    return (
      <EJ.Schedule id="Schedule1" width="100%" height="525px" showWeekend={false}
        currentDate={new Date(2017, 5, 5)} appointmentSettings-
        dataSource={appointData}>
      </EJ.Schedule>
    );
  }
});
ReactDOM.render(<Scheduler />, document.getElementById('schedule-default'));
```

### Date Customization

The date in the Scheduler can be customized by setting specific minimum and maximum date ranges and also defining various date formats to it.

#### Current Date

The Current date indicates the date with which the Scheduler loads initially and based on which the appropriate date range displays in the week/workweek/month/agenda views. To set the current date to the Scheduler – use the following code example,

#### HTML

```
var appointData = [{
  Id: 101,
  Subject: "Talk with Nature",
  StartTime: new Date(2015, 11, 5, 10, 00),
  EndTime: new Date(2015, 11, 5, 11, 00)
}];
var Scheduler = React.createClass({
  render: function () {
    return (
      <EJ.Schedule id="Schedule1" width="100%" height="525px" currentDate={new
      Date(2017, 5, 5)} appointmentSettings-dataSource={appointData}>
      </EJ.Schedule>
    );
  }
});
ReactDOM.render(<Scheduler />, document.getElementById('schedule-default'));
```

**Note:** By default, the System current date will be taken as Scheduler's current date.

#### MinDate and MaxDate

Providing the [minDate](#) and [maxDate](#) property with some date values, allows the Scheduler to set the minimum and maximum date range. The Scheduler date that lies beyond these minimum and maximum date range will be in a disabled state, so that the date navigation is blocked beyond these specified date range. Also, the appointments that lies beyond these date ranges will not be displayed on the Scheduler.

The following code example shows how to set the minDate and maxDate properties of the Scheduler.

#### HTML

```
var appointData = [{
  Id: 101,
  Subject: "Talk with Nature",
  StartTime: new Date(2015, 11, 5, 10, 00),
  EndTime: new Date(2015, 11, 5, 11, 00)
}];
var Scheduler = React.createClass({
  render: function () {
    return (
      <EJ.Schedule id="Schedule1" width="100%" height="525px" minDate={new
      Date(2015, 11, 4)} maxDate={new Date(2015, 11, 8)} appointmentSettings-
      dataSource={appointData}>
      </EJ.Schedule>
    );
  }
});
```



```

}
});
ReactDOM.render(<Scheduler />, document.getElementById('schedule-default'));

```

**Note:** The **maxDate** value provided should always be greater than that of **minDate** value.

### Appointment Window Customization

It is possible to use the custom appointment window option to design it with the user-required extra fields apart from the other default available fields. To make use of the customized appointment window, it is necessary to use the [appointmentWindowOpen](#) event within which the display of default appointment window is prevented.

The following code example lets you create the custom appointment window (using recurrence editor feature) with a single extra field for defining the appointment type.

### HTML

```

<!--Container for ejScheduler widget-->
<div id="schedule-default"></div>
<div id="customWindow" style="display: none">
<div id="appWindow">
<form id="custom">
<table width="100%" cellpadding="5">
<tbody>
<tr style="display: none">
<td>Id:</td>
<td colspan="2">
<input id="customId" type="text" name="Id" />
</td>
</tr>
<tr>
<td>Subject:</td>
<td colspan="2">
<input id="subject" type="text" value="" name="Subject" onfocus="temp()"
style="width: 100%" />
</td>
</tr>
<tr>
<td>Description:</td>
<td colspan="2">
<textarea id="customdescription" name="Description" rows="3" cols="50"
style="width: 100%; resize: vertical"></textarea>
</td>
</tr>
<tr>
<td>StartTime:</td>
<td>
<input id="StartTime" type="text" value="" name="StartTime" />
</td>
</tr>
<tr>
<td>EndTime:</td>
<td>
<input id="EndTime" type="text" value="" name="EndTime" />
</td>
</tr>

```

```

<tr>
<td>Appointment Type:</td>
<td><input type="text" id="AppointmentType" /></td>
</tr>
<tr>
<td colspan="3">
<div class="customcheck">AllDay:</div>
<div class="customcheck">
<input id="allDay" type="checkbox" name="AllDay" onchange="allDayCheck()" />
</div>
<div class="customcheck">Recurrence:</div>
<div>
<input id="recurrence" type="checkbox" name="Recurrence"
onchange="recurCheck()" />
</div>
</td>
</tr>
<tr id="summaryTr" style="display:none;">
<td colspan="3">
<div class="recSummary">Summary:</div>
<div>
<label id="recSummary" name="Summary"></label>
</div>
</td>
</tr>
<tr id="editor" style="display:none;">
<td colspan="3">
<div><a id="recedit" onclick="recurrenceRule()">Edit</a></div>
</td>
</tr>
</tbody>
</table>
</form>
<div>
<button type="submit" onclick="cancel()" id="buttonCancel"
style="float:right;margin-right:20px;margin-bottom:10px;">Cancel</button>
<button type="submit" onclick="save()" id="buttonSubmit"
style="float:right;margin-right:20px;margin-bottom:10px;">Submit</button>
</div>
</div>
<div id="recWindow" style="display: none">
<div id="recurrenceEditor"></div>
<br />
<div>
<button type="submit" id="recCancel">Cancel</button>
<button type="submit" id="recSubmit">Submit</button>
</div>
</div>
</div>
<script src="app/schedule/scheduler.js"></script>

```

The styles to be applied for the controls within the custom appointment window are as follows.

#### HTML

```
<style type="text/css">
```

```
.customcheck {
float: left;
margin-right: 10px;
}
.error {
background-color: #FF8A8A;
}
#custom table td {
padding: 5px;
}
</style>
```

## HTML

```
var Scheduler = React.createClass({
onAppointmentWindowOpen: function(args) {
args.cancel = true; // prevents the display of default appointment window
var schObj = $("#Schedule1").data("ejSchedule");
// When double clicked on the Scheduler cells, fills the StartTime and
EndTime fields appropriately
$("#StartTime").ejDateTimePicker({ value: args.startTime });
$("#EndTime").ejDateTimePicker({ value: args.endTime });
$("#recWindow").css("display", "none");
$("#appWindow").css("display", "");
if (!ej.isNullOrUndefined(args.target)) {
// When double clicked on the Scheduler cells, if the target is all day or
month cells - only then enable check mark on the all day checkbox
if ($(args.target.currentTarget).hasClass("e-alldaycells") ||
(args.startTime.getHours() == 0 && args.endTime.getHours() == 23))
$("#allDay").prop("checked", true);
else
args.model.currentView == "month" ? $("#allDay").prop("checked", true) :
$("#allDay").prop("checked", false);
// If the target is allDay or month cells - disable the StartTime and
EndTime fields
$("#StartTime,#EndTime").ejDateTimePicker({
enabled: ($(args.target.currentTarget).hasClass("e-alldaycells") ||
(args.startTime.getHours() == 0 && args.endTime.getHours() == 23) ||
$(args.target.currentTarget).hasClass("e-monthcells") ||
args.model.currentView == "month") ? false : true
});
}
// If double clicked on the appointments, fill the custom appointment window
fields with appropriate values.
if (!ej.isNullOrUndefined(args.appointment)) {
$("#customId").val(args.appointment.Id);
$("#subject").val(args.appointment.Subject);
$("#StartTime").ejDateTimePicker({ value: new
Date(args.appointment.StartTime) });
$("#EndTime").ejDateTimePicker({ value: new Date(args.appointment.EndTime)
});
// Fills the Appointment type dropdown with its value
var value = args.appointment.AppointmentType;
$("#AppointmentType").ejDropDownList("clearText");
$("#AppointmentType").ejDropDownList({
text: value,
```

```

value: value
});
$("#allDay").prop("checked", args.appointment.AllDay);
$("#recurrence").ejCheckBox({ checked: args.appointment.Recurrence });
if (args.appointment.Recurrence) {
$("#editor").css("display", "");
$("#recSummary").html(args.appointment.RecurrenceRule);
$("#summaryTr").css("display", "");
recObj._recRule = args.appointment.RecurrenceRule; // app recurrence rule is
stored in Recurrence editor object
recObj.recurrenceRuleSplit(args.appointment.RecurrenceRule,
args.appointment.recurrenceExDate); //splitting the recurrence rule
recObj.showRecurrenceSummary(args.appointment.Id); // updating the
recurrence rule in Recurrence editor
}
}
$("#customWindow").ejDialog("open");
},
render: function () {
return (
<EJ.Schedule id="Schedule1" width="100%" height="525px" currentDate={new
Date(2017, 5, 5)} appointmentSettings-dataSource={dataManager}
appointmentWindowOpen="onAppointmentWindowOpen">
</EJ.Schedule>
);
}
});
var Button1 = React.createClass({
onClick: function(args) {
// checks if the subject value is not left blank before saving it.
if ($.trim($("#subject").val()) == "") {
$("#subject").addClass("error");
return false;
}
var obj = {}, temp = {}, rType;
var formElement = $("#customWindow").find("#custom").get(0);
// looping through the custom form elements to get each value and form a
JSON object
for (var index = 0; index < formElement.length; index++) {
var columnName = formElement[index].name, $element = $(formElement[index]);
if (columnName != undefined) {
if (columnName == "")
columnName = formElement[index].id.replace(this._id, "");
if (columnName != "" && obj[columnName] == null) {
var value = formElement[index].value;
if (columnName == "Id" && value != "")
value = parseInt(value);
if ($element.hasClass("e-datetimepicker"))
value = new Date(value);
if (formElement[index].type == "checkbox")
value = formElement[index].checked;
obj[columnName] = value;
}
}
}
obj["RecurrenceRule"] = (obj.Recurrence) ? recurRule : null;
var appTypeObj = $("#AppointmentType").data("ejDropDownList");

```

```

obj["AppointmentType"] = appTypeObj.getSelectedValue();
$("#customWindow").ejDialog("close");
var object = $("#Schedule1").data("ejSchedule");
object.saveAppointment(obj);
},
render: function () {
return (
<EJ.Button id="Submit" width="85px" click={this.onClick}></EJ.Button>
);
}
});
var Button2 = React.createClass({
onClick: function(args) {
$("#customWindow").ejDialog("close");
},
render: function () {
return (
<EJ.Button id="Cancel" width="85px" click={this.onClick}></EJ.Button>
);
}
});
var DateTimePicker1 = React.createClass({
render: function () {
return (
<EJ.DateTimePicker id="StartTime" width="150px"></EJ.DateTimePicker>
);
}
});
var DateTimePicker2 = React.createClass({
render: function () {
return (
<EJ.DateTimePicker id="EndTime" width="150px"></EJ.DateTimePicker>
);
}
});
var Dialog = React.createClass({
clearFields: function(args) {
$("#customId").val("");
$("#subject").val("");
$("#customdescription").val("");
$("#allday").prop("checked", false);
$("#recurrence").prop("checked", false);
document.getElementById("rType").selectedIndex = "0";
$("#tr.recurrence").css("display", "none");
$("#StartTime,#EndTime").ejDateTimePicker({ enabled: true });
},
render: function () {
return (
<EJ.Dialog id="customdialog" width="600px" height="auto" showOnInit={false}
enableModal={true} title="Create Appointment" enableResize={false}
allowKeyboardNavigation={false} close={this.clearFields}>
</EJ.Dialog>
);
}
});
var CheckBox = React.createClass({
recurCheck: function(args) {

```

```

},
render: function () {
return (
<EJ.CheckBox id="recurCheck" change={this.recurCheck}>
</EJ.CheckBox>
);
}
});
var Button3 = React.createClass({
render: function () {
return (
<EJ.Button id="recSubmit" width="85px"
click="onRecurrenceClick"></EJ.Button>
);
}
});
var Button4 = React.createClass({
render: function () {
return (
<EJ.Button id="recCancel" width="85px"
click="onRecurrenceClick"></EJ.Button>
);
}
});
ReactDOM.render(<Scheduler />, document.getElementById('schedule-default'));
ReactDOM.render(<Button1 />, document.getElementById('buttonSubmit'));
ReactDOM.render(<Button2 />, document.getElementById('buttonCancel'));
ReactDOM.render(<DateTimePicker1 />, document.getElementById('StartTime'));
ReactDOM.render(<DateTimePicker2 />, document.getElementById('EndTime'));
ReactDOM.render(<Dialog />, document.getElementById('customWindow'));
ReactDOM.render(<CheckBox />, document.getElementById('recurrence'));
ReactDOM.render(<Button3 />, document.getElementById('recSubmit'));
ReactDOM.render(<Button4 />, document.getElementById('recCancel'));

```

On clicking the **Submit** button within the Custom Appointment window, the following function gets executed – which will validate the appointment fields and then save it appropriately.

### HTML

```

<script type="text/javascript">
function onRecurrenceClick(args) {
if ($(args.e.currentTarget).attr("id") == "recSubmit") {
recObj = $("#recurrenceEditor").ejRecurrenceEditor('instance');
recObj.closeRecurPublic();
recurRule = recObj._recurRule;
$("#recSummary").html(recurRule);
}
else
if (($ (args.e.currentTarget).attr("id") == "recCancel")) {
if ($("#recSummary").html() == "") {
$("#editor").css("display", "none");
$("#recurrence").ejCheckBox({ checked: false });
}
else
$("#recurrence").ejCheckBox({ checked: true });
}
}

```

```

$("#recWindow").css("display", "none");
$("#appWindow").css("display", "");
if ($("#recSummary").html() != "")
$("#summaryTr").css("display", "");
}
// This function executes when the Edit anchor tag in the edit appointment
window is clicked.
function recurrenceRule() {
$("#recWindow").css("display", "");
$("#appWindow").css("display", "none");
}
// This function executes when the recurrence checkbox is checked in the
custom appointment window
function recurCheck(args) {
if (args.isInteraction) {
if ($("#recurrence").get(0).checked == true) {
$("#recWindow").css("display", "");
$("#appWindow").css("display", "none");
$("#editor").css("display", "");
}
else {
$("#recWindow").css("display", "none");
$("#editor").css("display", "none");
$("#recSummary").html("");
$("#summaryTr").css("display", "none");
}
}
}
// This function executes when the All-day checkbox is checked in the custom
appointment window
function allDayCheck() {
if ($("#allDay").prop("checked")) {
var a = ($("#StartTime").data("ejDateTimePicker").model.value;
a.setHours(0, 0, 0);
var b = ($("#EndTime").data("ejDateTimePicker").model.value;
b.setHours(23, 59, 0);
$("#StartTime").ejDateTimePicker({ value: new Date(a), enabled: false });
$("#EndTime").ejDateTimePicker({ value: new Date(b), enabled: false });
} else {
$("#StartTime").ejDateTimePicker({ enabled: true });
$("#EndTime").ejDateTimePicker({ enabled: true });
}
}
// This function executes when the subject text field is currently being
focused
function temp() {
$("#subject").removeClass("error");
}
// This function executes when the cancel button in the custom appointment
window is pressed.
function cancel() {
recObj = ($("#recurrenceEditor").ejRecurrenceEditor('instance');
clearFields();
$("#customWindow").ejDialog("close");
}
</script>

```

### Scheduler Customization using queryCellInfo

It is possible to format and customize almost every child elements of scheduler such as work cells, header cells, time cells and so on using [queryCellInfo](#) event.

The following code snippet shows how to customize the appointment and work cells based on the query cell info event.

#### HTML

```
var appointData = [{
  Id: 100,
  Subject: "Research on Sky Miracles",
  StartTime: new Date(2015, 11, 2, 9, 00),
  EndTime: new Date(2015, 11, 2, 10, 30)
}];
var Scheduler = React.createClass({
  checkInfo: function(args) {
    switch (args.requestType) {
      case "workcells":
        args.element.css("background-color", "#ffe9cc");
        break;
      case "monthcells":
        args.element.css("background-color", "#faa41a");
        args.element.css("border-color", "#faa41a");
        break;
    }
  },
  render: function () {
    return (
      <EJ.Schedule id="Schedule1" queryCellInfo={this.checkInfo}
        appointmentSettings-dataSource={appointData}>
      </EJ.Schedule>
    );
  }
});
ReactDOM.render(<Scheduler />, document.getElementById('schedule-default'));
```

The Scheduler elements are listed below which can be formatted through this event. The names are listed in the format with which it can be accessed or used within the requestType argument of the event.

Request Type	Description
appointment	Depicts the appointment element within the Scheduler.
agendacells	Depicts the Agenda Cell element within the Scheduler.
alldaycells	Depicts the AllDay cell element within the Scheduler.
headercells	Depicts the header cell element within the Scheduler.
resourceheadercells	Depicts the resource header cell element within the Scheduler.
leftheadcells	Depicts the left empty space on header cell element within the Scheduler.



Request Type	Description
leftindentcells	Depicts the left empty space on date cell element within the Scheduler.
timecells	Depicts the left side time panel cell element within the Scheduler.
headerdate	Depicts the header date cell element within the Scheduler.
emptytd	Depicts the empty space above the vertical scroller within the Scheduler.
resourcegroupheader	Depicts the header group cell in horizontal orientation in the Scheduler.
monthcells	Depicts the month cell element within the Scheduler.
workcells	Depicts the work cell element within the Scheduler.

## Navigation

Navigation in Scheduler can be classified based on Scheduler views, date and also the appointments in it.

### View Navigation

By default, all the available [view options](#) except the Custom View are available at the top right corner of the Schedule header, which can be traverse through continuously as and when needed.

Clicking on the particular date header in the Week/Work Week/Month/Custom View will navigate to the day view automatically. Also, clicking on the week header ranges displayed at the left side in the month view will navigate to the Week view. These particular actions can take place only if the Week and Day view options are present in the [views](#) Collection.

### Handling View navigation actions

Usually, the [navigation](#) event gets triggered whenever the views/dates are being navigated. To block the navigation to day and week views from month view, the **navigation** event can be used in the following way.

### HTML

```
var appointData = [{
  Id: 100,
  Subject: "Research on Sky Miracles",
  StartTime: new Date(2015, 11, 2, 9, 00),
  EndTime: new Date(2015, 11, 2, 10, 30)
}];
var Scheduler = React.createClass({
  onNavigation: function(args) {
    //args.target.currentTarget - target element which is clicked.
    var target = $(args.target.currentTarget);
    if (args.requestType == "viewNavigate" && (target.hasClass("e-headercells")
    || target.hasClass("e-monthheader") || target.hasClass("e-timecells")))
    args.cancel = true;
  },
  render: function () {
    return (
      <EJ.Schedule id="Schedule1" navigation={this.onNavigation}
      appointmentSettings-dataSource={appointData}>
    </EJ.Schedule>
```

```

    });
    ReactDOM.render(<Scheduler />, document.getElementById('schedule-default'));

```

**Note:** Based on the navigation, the appointments that lies between the particular date ranges of the current view are fetched and rendered in the Scheduler.

### Date Navigation

The Scheduler date can be navigated on two aspects either in a continuous or random manner. On pressing the previous and next navigation arrow icons in the Scheduler header will move the scheduler one date back and forth respectively.

Another way of navigating through date is by making use of the built-in calendar available within the Scheduler, which pops out when the header date range is clicked. Selecting any date in the calendar will make the Scheduler to move to that particular date appropriately.

### Handling date navigation actions

To handle the date navigation actions, the [navigation](#) event can be used. For example, to block the date navigation, follow the below code example.

### HTML

```

var appointData = [{
  Id: 100,
  Subject: "Research on Sky Miracles",
  StartTime: new Date(2015, 11, 2, 9, 00),
  EndTime: new Date(2015, 11, 2, 10, 30)
}];
var Scheduler = React.createClass({
  onNavigation: function(args) {
    //args.target - target element which is clicked.
    //args.currentDate - current date of the Scheduler.
    //args.requestType - Specifies the navigation type.
    if (args.requestType == "dateNavigate")
      args.cancel = true;
  },
  render: function () {
    return (
      <EJ.Schedule id="Schedule1" navigation={this.onNavigation}
        appointmentSettings-dataSource={appointData}>
      </EJ.Schedule>
    );
  }
});
ReactDOM.render(<Scheduler />, document.getElementById('schedule-default'));

```

### Appointment Navigation

The Appointment navigation bars (labeled **Previous/Next Appointment**) are rendered parallel to each other on the left and right centric corners of the Schedule control. It is controlled by an API [showAppointmentNavigator](#) which is set to true by default. When it is set to false, these bars will not be displayed on the Scheduler.

Whenever the previous/Next Appointment bars are clicked, it navigates the Scheduler to the corresponding closest date where the appointments are available. If no appointments are available beyond the current date, then these appointment bars will be in a disabled state.

The following code example shows the way to define the **showAppointmentNavigator** property for Scheduler.

### HTML

```
var appointData = [{
  Id: 100,
  Subject: "Research on Sky Miracles",
  StartTime: new Date(2015, 11, 2, 9, 00),
  EndTime: new Date(2015, 11, 2, 10, 30)
}];
var Scheduler = React.createClass({
  render: function () {
    return (
      <EJ.Schedule id="Schedule1" showAppointmentNavigator={true}
        appointmentSettings-dataSource={appointData}>
      </EJ.Schedule>
    );
  }
});
ReactDOM.render(<Scheduler />, document.getElementById('schedule-default'));
```

### Template

Template is applicable to all the below specified elements of the Scheduler,

- Appointments
- Cells
- Resource header
- Date header
- Priority field
- TimeScale
- Date and time columns in Agenda view
- Tooltip

### Appointment Template

The template design that applies on for the Scheduler appointments. The field names that are mapped from the `dataSource` to the appropriate field properties within the [appointmentSettings](#) can be accessed within the template.

Apart from the `dataSource` field names, the template can also access the current view of the Scheduler using the name **View** – which can contain either of the following values in lowercase.

- day
- week
- workweek
- agenda
- month
- custom view

It is controlled by an API named [appointmentTemplateId](#) which accepts the id value of the template design block preceded by a symbol #.

Usually, the appointments are displayed with its **Subject** and **Start/End time** on the Scheduler. If suppose, the subject needs to be accompanied with location text, it can be done with the following code example.

### HTML

```
<!--Container for ejScheduler widget-->
<div id="schedule-default"></div>
<script id="appTemplate" type="text/x-jsrender">
{{"{{"}}if View !== "agenda"{{{}}}
<div style="height:100%; background-color:orange; margin-left: 5px;">
<div style="margin-left: 2px;">{{"{{"}}:Subject{{{}}}</div>
<div style="margin-left: 2px;">{{"{{"}}:Location{{{}}}</div>
</div>
{{"{{"}}else{{{}}}
<div>{{"{{"}}:Subject{{{}}}, {{"{{"}}:Location{{{}}}</div>
{{"{{"}}/if{{{}}}}
</script>
```

### HTML

```
var appointData = [{
  Id: 100,
  Subject: "Wild Discovery",
  StartTime: new Date(2015, 11, 2, 9, 00),
  EndTime: new Date(2015, 11, 2, 10, 30),
  Location: "CHINA"
}];
var Scheduler = React.createClass({
  render: function () {
    return (
      <EJ.Schedule id="Schedule1" appointmentTemplateId="#appTemplate"
        appointmentSettings-dataSource={appointData}>
      </EJ.Schedule>
    );
  }
});
ReactDOM.render(<Scheduler />, document.getElementById('schedule-default'));
```

### Cell Templates

The template design that applies on the Scheduler elements such as all-day cells, work cells and month cells which allows the customization to be done based on the date, view, resources and timescale. The cells can be customized to add images, colors, and other elements etc and can also access the current view of the Scheduler using the name **view**.

**All-day cells** - An API named [allDayCellsTemplateId](#) can be used to customize the all-day cells, which accepts the id of the template design block preceded with a symbol #.

**Work cells and Month cells** - An API named [workCellsTemplateId](#) can be used to customize the work cells in all the views, which accepts the id of the template design block preceded by a symbol #.

The cells can be customized with the following code example.

**HTML**

```

<!--Container for ejScheduler widget-->
<div id="schedule-default"></div>
<!-- Template for All-day cells -->
<script id="allDayTemplate" type="text/x-jsrender">
<div class="e-icon e-scheduleallday" style="opacity:0.5"></div>
<span style="opacity:0.5">AllDay</span>
</script>
<!-- Template for Workcells and Monthcells -->
<script id="workTemplate" type="text/x-jsrender">
{{"{{"}}if resource.classname == 'e-parentnode'{{}}}}
{{"{{"}}:resource.text{{}}}}
{{"{{"}}else{{}}}}
{{"{{"}}if date.getDay() == 0 || date.getDay() == 6{{}}}}
<div style="background-color:lightblue">Weekend</div>
{{"{{"}}else{{}}}}
{{"{{"}}if view == 'month' && resource.text == 'Party Hall-A' &&
date.getDay() == 5{{}}}}
<div style="background-color:burlywood">Meeting</div>
{{"{{"}}else resource.text != 'Party Hall-B' && date.getDate() == 15{{}}}}
<div style="background-color:thistle">Holiday</div>
{{"{{"}}else view != 'month' && resource.text == 'Party Hall-A' &&
date.getDay() == 5 && date.getHours() == 10{{}}}}
<div style="background-color:burlywood">Meeting</div>
{{"{{"}}else view == 'month' && resource.text == 'Party Hall-B' &&
date.getDay() == 5{{}}}}
<div style="background-color:lightblue">Conf.</div>
{{"{{"}}else resource.text == 'Party Hall-B' && date.getDate() == 16{{}}}}
<div style="background-color:darkkhaki">HappyDay</div>
{{"{{"}}else view != 'month' && resource.text == 'Party Hall-B' &&
date.getDay() == 5 && date.getHours() == 12{{}}}}
<div style="background-color:goldenrod">Conf.</div>
{{"{{"}}else date.getDate() == 10 && date.getMonth() == 11{{}}}}
<div style="background-color:palegreen">Day Special</div>
{{"{{"}}else date.getDate() == 25 && date.getMonth() == 11{{}}}}
<div style="background-color:sandybrown">Christmas</div>
{{"{{"}}}/if{{}}}}
{{"{{"}}}/if{{}}}}
{{"{{"}}}/if{{}}}}
</script>

```

**HTML**

```

var appointData = [{
  Id: 100,
  Subject: "Wild Discovery",
  StartTime: new Date(2015, 11, 2, 9, 00),
  EndTime: new Date(2015, 11, 2, 10, 30),
  Location: "CHINA"
}];
var Scheduler = React.createClass({
  render: function () {
    return (

```

```

<EJ.Schedule id="Schedule1" allDayCellsTemplateId="#allDayTemplate"
workCellsTemplateId="#workTemplate" appointmentSettings-
dataSource={appointData}>
</EJ.Schedule>
);
}
});
ReactDOM.render(<Scheduler />, document.getElementById('schedule-default'));

```

### Date Header Template

The template design that applies on for the date header part of the Scheduler. An API named [dateHeaderTemplateId](#) can be used to customize the date header which accepts the id value of the template design block preceded by a symbol #. The template can also access the current view of the Scheduler in using the name **view**.

The Date header can be customized with the following code example.

### HTML

```

<!--Container for ejScheduler widget-->
<div id="schedule-default"></div>
<!-- Template for Dateheader -->
<script id="dateTemplate" type="text/x-jsrender">
<div>{{"{{" }}:~dTemplate(date) {{}}}}</div>
</script>

```

### HTML

```

var appointData = [{
  Id: 100,
  Subject: "Wild Discovery",
  StartTime: new Date(2015, 11, 2, 9, 00),
  EndTime: new Date(2015, 11, 2, 10, 30),
  Location: "CHINA"
}];
var Scheduler = React.createClass({
  render: function () {
    return (
      <EJ.Schedule id="Schedule1" dateHeaderTemplateId="#dateTemplate"
appointmentSettings-dataSource={appointData}>
</EJ.Schedule>
    );
  }
});
ReactDOM.render(<Scheduler />, document.getElementById('schedule-default'));

```

### JAVASCRIPT

```

function _dateFormat(date) {
  var dFormat = ej.format(new Date(date), "dd/MM");
  return dFormat;
}
$.views.helpers({ dTemplate: _dateFormat });

```

### Resource Header Template

The template structure that applies on the resource headers of the Scheduler. By default, only the resource names will be displayed on the resource header bar. Also, the way of rendering resource headers on the Scheduler is comparatively different for both the vertical and horizontal scheduler views.

The field names that are mapped from the dataSource to the appropriate field properties within the **resourceSettings** can be accessed within the resource header template.

#### Resource Header Template in Vertical View

To customize the resource header with some additional images or other customizations in **vertical Scheduler View** – refer the below code example.

#### HTML

```
<!--Container for ejScheduler widget-->
<div id="schedule-default"></div>
<script id="resTemplate" type="text/x-jsrender">
<div style="height:100%">
<div style="width:15px;height:15px;margin-left:275px;margin-
top:2px;float:left;background:{{"{{"}}:ResourceColor{{}}}};"></div>
<div style="float:left;margin-left:5px;">{{"{{"}}:ResourceText{{}}}}</div>
</div>
</script>
```

#### HTML

```
var appointData = [{
  Id: 100,
  Subject: "Wild Discovery",
  StartTime: new Date(2015, 11, 2, 9, 00),
  EndTime: new Date(2015, 11, 2, 10, 30),
  RoomId: 2
}];
var groupData = { resources: ["Rooms"] };
var roomData = {
  dataSource: [
    { ResourceText: "ROOM1", id: 1, ResourceColor: "orange" },
    { ResourceText: "ROOM2", id: 2, ResourceColor: "#56ca85" }
  ],
  text: "ResourceText", id: "id", color: "ResourceColor"
};
var Scheduler = React.createClass({
  render: function () {
    return (
      <EJ.Schedule id="Schedule1" group={groupData}
        resourceHeaderTemplateId="#resTemplate" appointmentSettings-
        dataSource={appointData} appointmentSettings-resourceFields="RoomId">
        <resources>
          <resource allowMultiple={false} field="RoomId" title="Room" name="Rooms"
            resourceSettings={roomData}></resource>
        </resources>
      </EJ.Schedule>
    );
  }
});
ReactDOM.render(<Scheduler />, document.getElementById('schedule-default'));
```

### Resource Header Template in Horizontal View

To perform the above specified same customization in **horizontal Scheduler view**, the template structure varies a little bit as depicted below.

#### HTML

```
<!--Container for ejScheduler widget-->
<div id="schedule-default"></div>
<script id="resTemplate" type="text/x-jsrender">
<div style="height:100%">
<div style="width:15px;height:15px;margin-right:5px;margin-
top:2px;float:left;background:{{"{{":ResColor{}}}};"></div>
<div>{{"{{":ResText{}}}}</div>
</div>
</script>
```

#### HTML

```
var appointData = [{
  Id: 100,
  Subject: "Wild Discovery",
  StartTime: new Date(2015, 11, 2, 9, 00),
  EndTime: new Date(2015, 11, 2, 10, 30),
  RoomId: 2
}];
var groupData = { resources: ["Rooms"] };
var roomData = {
  dataSource: [
    { ResourceText: "ROOM1", id: 1, ResourceColor: "orange" },
    { ResourceText: "ROOM2", id: 2, ResourceColor: "#56ca85" }
  ],
  text: "ResourceText", id: "id", color: "ResourceColor"
};
var Scheduler = React.createClass({
  render: function () {
    return (
      <EJ.Schedule id="Schedule1" orientation="horizontal" group={groupData}
        resourceHeaderTemplateId="#resTemplate" appointmentSettings-
        dataSource={appointData} appointmentSettings-resourceFields="RoomId">
        <resources>
        <resource allowMultiple={false} field="RoomId" title="Room" name="Rooms"
          resourceSettings={roomData}></resource>
        </resources>
      </EJ.Schedule>
    );
  }
});
ReactDOM.render(<Scheduler />, document.getElementById('schedule-default'));
```

**Note:** In horizontal Scheduler, the header template makes use of an additional field namely **classname** which holds either **e-parentnode** or **e-childnode** value. The field **classname** can be used in the application scenario to check for the parent or child header node. You can apply the different template customization accordingly based on it.



## TimeScale Templates

The [TimeScale](#) is also availed with template options to allow customization. It includes the following 2 properties for customization -

- [majorSlotTemplateId](#) - Accepts the id value of the template design block preceded by a symbol #, which gets applied for the major time slots.
- [minorSlotTemplateId](#) - Accepts the id value of the template design block preceded by a symbol #, which gets applied for the minor time slots.

The template customization for major and minor timeslots can be referred from the following code example.

### HTML

```
<!--Container for ejScheduler widget-->
<div id="schedule-default"></div>
<!-- Template for Majorslot -->
<script id="majorTemplate" type="text/x-jsrender">
<div>{{"{"}}:~major(date){{}}}</div>
</script>
<!-- Template for Minorslot -->
<script id="minorTemplate" type="text/x-jsrender">
<div>{{"{"}}:~minor(date){{}}}</div>
</script>
```

### HTML

```
var appointData = [{
  Id: 100,
  Subject: "Wild Discovery",
  StartTime: new Date(2015, 11, 2, 9, 00),
  EndTime: new Date(2015, 11, 2, 10, 30),
  RoomId: 2
}];
var timeScaleData = {
  enable: true,
  majorSlot: 60,
  majorSlotTemplateId: "#majorTemplate",
  minorSlotCount: 6,
  minorSlotTemplateId: "#minorTemplate"
};
var Scheduler = React.createClass({
  render: function () {
    return (
      <EJ.Schedule id="Schedule1" orientation="horizontal" group={groupData}
        timeScale={timeScaleData} appointmentSettings-dataSource={appointData}>
        </EJ.Schedule>
    );
  }
});
ReactDOM.render(<Scheduler />, document.getElementById('schedule-default'));
```

### JAVASCRIPT

```
<script type="text/javascript">
function _majorFormat(date) {
var dFormat = ej.format(new Date(date), "hh:mm:ss");
return dFormat;
}
function _minorFormat(date) {
var dFormat = ej.format(new Date(date), "hh:mm:ss");
return dFormat;
}
$.views.helpers({
major: _majorFormat,
minor: _minorFormat
});
</script>
```

### Priority Settings Template

The template design which can be applied to the content of the priority field in the appointment window. By default, the appropriate icons are displayed for each priority options such as **None**, **High**, **Medium** and **Low**.

When template is applied for the [prioritySettings](#), these default icons will be replaced by the custom icons or styles defined newly. The following code example depicts the way to enable the priority settings and to define the new custom styles to replace the default icons in the Priority field.

#### HTML

```
<!--Container for ejScheduler widget-->
<div id="schedule-default"></div>
<style type="text/css">
.critical,
.ultra-critical,
.none {
height: 13px;
width: 13px;
float: left;
margin-right: 4px;
background-repeat: no-repeat;
background-size: 60px;
padding: 1px;
margin-top: 2px;
}
.critical {
background-color: orange;
background-position: -13px;
}
.ultra-critical {
background-color: #56ca85;
background-position: -59px;
}
</style>
```

#### HTML

```
var appointData = [{
Id: 100,
```

```

Subject: "Wild Discovery",
StartTime: new Date(2015, 11, 2, 9, 00),
EndTime: new Date(2015, 11, 2, 10, 30),
Location: "CHINA",
Priority: "critical"
});
var priorityData = {
enable: true,
template: "<div class='${value}'></div>",
dataSource: [
{ text: "None", id: 1, value: "none" },
{ text: "Critical", id: 2, value: "critical" },
{ text: "Ultra Critical", id: 3, value: "ultra-critical" }
]
};
var Scheduler = React.createClass({
render: function () {
return (
<EJ.Schedule id="Schedule1" orientation="horizontal" group={groupData}
prioritySettings={priorityData} appointmentSettings-dataSource={appointData}
appointmentSettings-priority="Priority">
</EJ.Schedule>
);
}
});
ReactDOM.render(<Scheduler />, document.getElementById('schedule-default'));

```

The custom style class names defined for the priority template should be same as that of the values defined for each priority option within the `dataSource`, so that it applies properly.

**Note:** Additionally, the priority field within the [appointmentSettings](#) should be defined with appropriate `dataSource` field name. When an appointment is assigned with a priority value, the custom style/icon defined for that priority option will get applied over that appointment.

### Tooltip Template

The tooltip can be applied with the customized template design. Currently the tooltip support is provided only for the appointments and the default tooltip displays the Subject and duration on hovering across the appointments.

By making use of template feature with tooltip, all the field names that are mapped from the `dataSource` to the appropriate field properties within the **appointmentSettings** can be accessed.

To define the template option for tooltip, the [tooltipSettings](#) must be enabled first. The following code example depicts the way to add the tooltip template.

### HTML

```

<!--Container for ejScheduler widget-->
<div id="schedule-default"></div>
<script id="tooltipTemplate" type="text/x-jsrender">
<div style="width:145px">
<div style="padding-top:3px;">
<div style="float:left; font:13px Segoe UI; font-
weight:bold;">Subject&#160;&#160;&#160;</div>
<div style="padding-top:2px; font:12px Segoe UI
SemiBold;">{{{"{}}":Subject{{}}}}</div>

```

```

</div>
<div style="padding-top:3px">
<div style="float:left; font:13px Segoe UI; font-
weight:bold;">Location:&#160;</div>
<div style="padding-top:2px; font:12px Segoe UI
SemiBold;">{{{"{":Location{}}}}</div>
</div>
</div>
</script>

```

## HTML

```

var appointData = [{
  Id: 100,
  Subject: "Wild Discovery",
  StartTime: new Date(2015, 11, 2, 9, 00),
  EndTime: new Date(2015, 11, 2, 10, 30),
  Location: "CHINA"
}];
var tooltipData = {
  enable: true,
  templateId: "#tooltipTemplate"
};
var Scheduler = React.createClass({
  render: function () {
    return (
      <EJ.Schedule id="Schedule1" orientation="horizontal"
      tooltipSettings={tooltipData} appointmentSettings-dataSource={appointData}>
      </EJ.Schedule>
    );
  }
});
ReactDOM.render(<Scheduler />, document.getElementById('schedule-default'));

```

## Agenda View Templates

Agenda View provides two separate templates – one for date column and another for time column. These templates allows the customization of the content of both the date and time columns. Apart from this, the event column can also be customized through the existing API named [appointmentTemplateId](#).

The following code snippet shows how to customize the content of the date, time and event column.

## HTML

```

<!--Container for ejScheduler widget-->
<div id="schedule-default"></div>
// Template for date column
<script id="dateTemplate" type="text/x-jsrender">
<div style="height:100%">
<div>
<div>{{{"{":~dateDisplay(StartTime){}}}}</div>
</div>
</div>
</script>
// Template for time column
<script id="timeTemplate" type="text/x-jsrender">

```

```

<div style="height:100%">
<div>
<div>{{{"{}}":~timeDisplay(StartTime){{}}}}</div>
</div>
</div>
</script>
// Template for appointment which applies for event column in agenda view.
<script id="appTemplate" type="text/x-jsrender">
{{{"{}}"}if View !== "agenda"{{{}}}
<div style="height:100%; background-color:orange; margin-left: 5px;">
<div style="margin-left: 2px;">{{{"{}}":Subject{{{}}}}</div>
<div style="margin-left: 2px;">{{{"{}}":Location{{{}}}}</div>
</div>
{{{"{}}"}else{{{}}}
<div>{{{"{}}":Subject{{{}}}}, {{{"{}}":Location{{{}}}}</div>
{{{"{}}"}/if{{{}}}
</script>
<script type="text/javascript">
function _getDate(date) {
var dateCol = new Date(date);
return dateCol.toString();
}
function _getTime(date) {
var time = new Date(date);
return time.toLocaleTimeString();
}
$.views.helpers({
dateDisplay: _getDate,
timeDisplay: _getTime
});
</script>

```

## HTML

```

var appointData = [{
Id: 100,
Subject: "Wild Discovery",
StartTime: new Date(2015, 11, 2, 9, 00),
EndTime: new Date(2015, 11, 2, 10, 30),
Location: "CHINA"
}];
var tooltipData = {
enable: true,
templateId: "#tooltipTemplate"
};
var agendaViewData = {
dateColumnTemplateId: "#dateTemplate",
timeColumnTemplateId: "#timeTemplate"
};
var Scheduler = React.createClass({
render: function () {
return (
<EJ.Schedule id="Schedule1" orientation="horizontal"
appointmentTemplateId={tooltipData} agendaViewSettings={agendaViewData}
appointmentSettings-dataSource={appointData}>
</EJ.Schedule>

```

```
);
}
});
ReactDOM.render(<Scheduler />, document.getElementById('schedule-default'));
```

## Globalization and Localization

### Globalization

The Scheduler control is built with default **globalization** support as it format the dates according to the user's locale automatically and processes it internally without any need for manual conversions. This kind of default handling of Scheduler dates is achieved through the built-in **ej.globalize** library which globalize the date, day and month names accordingly.

### Localization

Scheduler also comes with default localization support which allows it to customize the display of text within the Scheduler in a user-specific culture and locale. The Schedule control can be localized in specific culture using the common API [locale](#) along with the collection of localized words defined for that culture using the `ej.Schedule.Locale [culture-code]`.

**Note:** By default, the Schedule control is localized in **en-US** culture.

To localize Scheduler into any particular culture, it is necessary to refer the culture-specific script files in your application after the reference of **ej.web.all.min.js** file, which are available under the following location.

*<Installed location>\Syncfusion\Essential Studio\{{ site.releaseversion }}\JavaScript\assets\scripts\i18n*

The following code example shows how to localize the Schedule control in **fr-FR** culture.

### HTML

```
ej.Schedule.Locale["fr-FR"] = {
  ReminderWindowTitle: "Fenêtre de rappel",
  CreateAppointmentTitle: "créer un rendez-",
  RecurrenceEditTitle: "Modifier répétition nomination",
  RecurrenceEditMessage: "Comment voulez-vous changer le cas dans la série?",
  RecurrenceEditOnly: "Seulement cette nomination",
  RecurrenceEditSeries: "La série entière",
  PreviousAppointment: "Nomination précédente",
  NextAppointment: "prochain rendez-vous",
  AppointmentSubject: "sujet",
  StartTime: "Heure de début",
  EndTime: "Heure de fin",
  AllDay: "toute la journée",
  Today: "aujourd'hui",
  Recurrence: "répétition",
  Done: "Terminé",
  Cancel: "annuler",
  Ok: "Ok",
  RepeatBy: "Répétez par",
  RepeatEvery: "répéter chaque",
  RepeatOn: "répéter l'opération sur",
  StartsOn: "démontre sur",
  Ends: "extrémités",
  Summary: "résumé",
  Daily: "quotidien",
```

```
Weekly: "hebdomadaire",
Monthly: "mensuel",
Yearly: "annuel",
Every: "tous",
EveryWeekDay: "chaque jour de la semaine",
Never: "jamais",
After: "après",
Occurrence: "apparition",
On: "sur",
Edit: "Modifier",
RecurrenceDay: "Jour (s)",
RecurrenceWeek: "Semaine (s)",
RecurrenceMonth: "Mois (s)",
RecurrenceYear: "Année (s)",
The: "la",
OfEvery: "de chaque",
First: "première",
Second: "deuxième",
Third: "troisième",
Fourth: "quatrième",
Last: "dernier",
WeekDay: "jour de la semaine",
WeekEndDay: "Jour de week-end",
Subject: "sujet",
Categorize: "Catégories",
DueIn: "En raison",
DismissAll: "rejeter tout",
Dismiss: "rejeter",
OpenItem: "Ouvrir l'élément",
Snooze: "répétition",
Day: "jour",
Week: "semaine",
WorkWeek: "Semaine de travail",
Month: "mois",
AddEvent: "Ajouter événement",
CustomView: "Vue personnalisée",
Agenda: "ordre du jour",
Detailed: "détaillé",
EventBeginsin: "Nomination commence dans",
Editevent: "Modifier nomination",
Editseries: "Modifier série",
Times: "fois",
Until: "jusqu'à",
Eventwas: "rendez-vous était",
Hours: "hrs",
Minutes: "minutes",
Overdue: "en retard",
Days: "jour (s)",
Event: "Sujet",
Select: "sélectionner",
Previous: "prev",
Next: "suivant",
Close: "proche",
Delete: "effacer",
Date: "date",
Showin: "montrer en",
Gotodate: "Aller à la date",
```

```

Resources: "RESSOURCES",
RecurrenceDeleteTitle: "Supprimer répétition rendez-",
Location: "emplacement",
Priority: "priorité",
RecurrenceAlert: "alerte",
WrongPattern: "Le modèle de récurrence est pas valable",
CreateError: "La durée de la nomination doit être plus courte que la façon
dont elle se produit fréquemment. Raccourcir la durée ou changer le modèle
de récurrence dans la boîte de dialogue Récurrence de rendez.",
DragResizeError: "Impossible de replanifier une occurrence du rendez-vous
récurrent. si elle saute sur une occurrence ultérieure du même rendez-
vous.",
StartEndError: "L'heure de fin doit être supérieur à l'heure de début",
MouseOverDeleteTitle: "supprimer un rendez-",
DeleteConfirmation: "Êtes-vous sûr de vouloir supprimer ce rendez-vous?",
Time: "Temps"
};
var appointData = [{
Id: 100,
Subject: "Wild Discovery",
StartTime: new Date(2015, 11, 2, 9, 00),
EndTime: new Date(2015, 11, 2, 10, 30),
Location: "CHINA"
}];
var Scheduler = React.createClass({
render: function () {
return (
<EJ.Schedule id="Schedule1" currentDate={new Date(2015, 11, 2)} locale="fr-
FR" appointmentSettings-dataSource={appointData}>
</EJ.Schedule>
);
}
});
ReactDOM.render(<Scheduler />, document.getElementById('schedule-default'));

```

**Note:** Refer the `ej.culture.fr-FR.min.js` file in your HTML application and also define the **locale** property for the Schedule control with the appropriate **culture-code** [**fr-FR**].

For further information on – how to refer the required culture scripts into your application, refer [here](#).

### *Localizing Specific Words*

To customize or localize only some specific words in the default `ej.Schedule.Locale["en-US"]` collection, the words to be localized/customized can be defined in a separate variable and then extended to the original collection as depicted in the following code example.

### **HTML**

```

var customizationMessage = {
// customize the appointment window title
CreateAppointmentTitle: "Create Event",
// customize the view options text in the Schedule header
Day: "1 Day",
Week: "7 Days",
WorkWeek: "5 Days",
Month: "Month"
};

```



```
// Extend only the required changes to the original locale collection
$.extend(ej.Schedule.Locale["en-US"], customizationMessage);
var appointData = [{
  Id: 100,
  Subject: "Wild Discovery",
  StartTime: new Date(2015, 11, 2, 9, 00),
  EndTime: new Date(2015, 11, 2, 10, 30),
  Location: "CHINA"
}];
var Scheduler = React.createClass({
  render: function () {
    return (
      <EJ.Schedule id="Schedule1" currentDate={new Date(2015, 11, 2)}
        appointmentSettings-dataSource={appointData}>
      </EJ.Schedule>
    );
  }
});
ReactDOM.render(<Scheduler />, document.getElementById('schedule-default'));
```

### Time Zone

The Scheduler makes use of the System time zone by default. If it needs to follow some other user-specific time zone, then the API [timeZone](#) can be used. Also, the Scheduler can be set to observe the Daylight Saving Time (DST) with its **isDST** property which is set to **false** by default.

When [isDST](#) property is set to **true**, the Scheduler internally processes the time difference values (for the Start and end time of the appointments) related to the Scheduler time zone that observes daylight savings time.

The following code example shows the way to set the specific time zone value with the daylight savings time observed in the Scheduler.

### HTML

```
var appointData = [{
  Id: 100,
  Subject: "Wild Discovery",
  StartTime: new Date(2015, 11, 2, 9, 00),
  EndTime: new Date(2015, 11, 2, 10, 30),
  Location: "CHINA"
}];
var Scheduler = React.createClass({
  render: function () {
    return (
      <EJ.Schedule id="Schedule1" currentDate={new Date(2015, 11, 2)}
        timeZone="UTC +05:30" isDST={true} appointmentSettings-
        dataSource={appointData}>
      </EJ.Schedule>
    );
  }
});
ReactDOM.render(<Scheduler />, document.getElementById('schedule-default'));
```

### Setting different TimeZone for Scheduler Appointments

Apart from the default action of applying specific timezone to the entire Scheduler, it is also possible to set different time zone values for each appointments through the properties **startTimeZone** and **endTimeZone** which can be defined as separate fields within the appointment dataSource. When these properties are not explicitly defined for appointments, the appointments Start and End time will be processed based on the Scheduler time zone.

**Note:** The **isDST** property closely relies on the appointment fields like [StartTimeZone](#) and [EndTimeZone](#), for appropriate time difference calculations. If these two fields are not defined for appointments, then **isDST** depends on the System **timeZone** value.

The following code snippet shows how to define isDST and the time zones for specific appointments.

#### HTML

```
var appointData = [{
  Id: 100,
  Subject: "Wild Discovery",
  StartTime: new Date(2015, 11, 2, 9, 00),
  EndTime: new Date(2015, 11, 2, 10, 30),
  Location: "CHINA",
  StartTimeZone: "UTC +02:00",
  EndTimeZone: "UTC +02:00"
}];
var Scheduler = React.createClass({
  render: function () {
    return (
      <EJ.Schedule id="Schedule1" currentDate={new Date(2015, 11, 2)} isDST={true}
        appointmentSettings-dataSource={appointData}>
      </EJ.Schedule>
    );
  }
});
ReactDOM.render(<Scheduler />, document.getElementById('schedule-default'));
```

### Customizing the TimeZone Collection

It is also possible to define or customize the default time zone collection of the Scheduler, by using the [timeZoneCollection](#) API as follows.

#### HTML

```
var appointData = [{
  Id: 100,
  Subject: "Wild Discovery",
  StartTime: new Date(2015, 11, 2, 9, 00),
  EndTime: new Date(2015, 11, 2, 10, 30),
  Location: "CHINA",
  StartTimeZone: "UTC +02:00",
  EndTimeZone: "UTC +02:00"
}];
var timeZoneCollectionData = {
  dataSource: [
    { text: "UTC -04:00", id: "10", value: "UTC -04:00" },
    { text: "UTC -03:30", id: "11", value: "UTC -03:30" },
    { text: "UTC -03:00", id: "12", value: "UTC -03:00" },
    { text: "UTC -02:00", id: "13", value: "UTC -02:00" },
  ]
};
```

```

{ text: "UTC -01:00", id: "14", value: "UTC -01:00" },
{ text: "UTC +00:00", id: "15", value: "UTC +00:00" },
{ text: "UTC +01:00", id: "16", value: "UTC +01:00" },
{ text: "UTC +02:00", id: "17", value: "UTC +02:00" },
{ text: "UTC +03:00", id: "18", value: "UTC +03:00" },
{ text: "UTC +03:30", id: "19", value: "UTC +03:30" },
{ text: "UTC +04:00", id: "20", value: "UTC +04:00" },
{ text: "UTC +04:30", id: "21", value: "UTC +04:30" },
{ text: "UTC +05:00", id: "22", value: "UTC +05:00" }
],
text: "text", id: "id", value: "value"
};
var Scheduler = React.createClass({
render: function () {
return (
<EJ.Schedule id="Schedule1" currentDate={new Date(2015, 11, 2)}
timeZoneCollection={timeZoneCollectionData} appointmentSettings-
dataSource={appointData}>
</EJ.Schedule>
);
}
});
ReactDOM.render(<Scheduler />, document.getElementById('schedule-default'));

```

**Note:** The values defined within the **timeZoneCollection** dataSource are usually the only options displayed within the start and end time zone dropdown fields of the appointment window.

### Time Mode

The time mode of the Scheduler can be either **12** or **24 hours** format which is based on the [locale](#) set to the Scheduler. Since the default locale value of the Scheduler is **en-US**, therefore the time mode will be set to **12 hours** format (by default) automatically based on the culture.

The user can also set specific time mode for the Scheduler using [timeMode](#) property which accepts either **String** or **enum** value. It accepts the following **enum** values,

- `ej.Schedule.TimeMode.Hour12`
- `ej.Schedule.TimeMode.Hour24`

The following code snippet shows the way to set specific **24 hour format** time mode for the Scheduler.

### HTML

```

var appointData = [{
Id: 100,
Subject: "Wild Discovery",
StartTime: new Date(2015, 11, 2, 9, 00),
EndTime: new Date(2015, 11, 2, 10, 30),
Location: "CHINA"
}];
var Scheduler = React.createClass({
render: function () {
return (
<EJ.Schedule id="Schedule1" currentDate={new Date(2015, 11, 2)}
timeMode={ej.Schedule.TimeMode.Hour24} appointmentSettings-
dataSource={appointData}>

```

```

</EJ.Schedule>
);
}
});
ReactDOM.render(<Scheduler />, document.getElementById('schedule-default'));

```

**Note:** If the **timeMode** property is not set with specific value, then the value will be taken based on the locale assigned for the Scheduler.

### Date Format

Scheduler can be used with all valid date formats. The default date format used in Scheduler is “MM/dd/yyyy”.

If the [dateFormat](#) property is not specified particularly, then it will be taken based on the locale that is assigned to the Scheduler. The default locale applied on the Scheduler is “en-US”, which makes it to follow the “MM/dd/yyyy” pattern by default.

### HTML

```

var appointData = [{
  Id: 100,
  Subject: "Wild Discovery",
  StartTime: new Date(2015, 11, 2, 9, 00),
  EndTime: new Date(2015, 11, 2, 10, 30),
  Location: "CHINA"
}];
var Scheduler = React.createClass({
  render: function () {
    return (
      <EJ.Schedule id="Schedule1" currentDate={new Date(2015, 11, 2)}
        dateFormat="yyyy/MM/dd" appointmentSettings-dataSource={appointData}>
      </EJ.Schedule>
    );
  }
});
ReactDOM.render(<Scheduler />, document.getElementById('schedule-default'));

```

### First Day of Week

The [firstDayOfWeek](#) property allows to set any of the week days as start of the week/workweek/month view in Scheduler. It accepts either the integer (Sunday=0, Monday=1, Tuesday=2, etc) or string (“Sunday”, “Monday”, etc) or `ej.Schedule.DayOfWeek` enum type value. The default value of this `firstDayOfWeek` depends on the current culture (language) assigned to the Scheduler.

To set different first day of week in Scheduler,

### HTML

```

var appointData = [{
  Id: 1,
  Subject: "Music Class",
  StartTime: new Date("2015/11/7 06:00 AM"),
  EndTime: new Date("2015/11/7 07:00 AM")
}, {
  Id: 2,
  Subject: "School",

```

```

StartTime: new Date("2015/11/7 9:00 AM"),
EndTime: new Date("2015/11/7 02:30 PM")
}];
var Scheduler = React.createClass({
render: function () {
return (
<EJ.Schedule id="Schedule1" currentDate={new Date(2015, 11, 2)}
firstDayOfWeek={ej.Schedule.FirstDayOfWeek.Tuesday} appointmentSettings-
dataSource={appointData}>
</EJ.Schedule>
);
}
});
ReactDOM.render(<Scheduler />, document.getElementById('schedule-default'));

```

**Note:** The sub-control datepicker which is used within Scheduler for start/end time fields in appointment window and for date navigation purposes will also follow the same first day of week.

### Keyboard Navigation

The shortcut keys for accessing the sub-elements of the scheduler and other Scheduler actions are tabulated in the following table.

Keys	Functionality description
arrow keys	To traverse through the Scheduler cells.
Shift+arrow keys	Multiple cell selection
Enter	Pressing enter key, after a single/multiple scheduler cell selection will make the quick appointment window to pop-up.  Also, when the focus is being moved on to the fields like checkbox or buttons of the appointment window, pressing enter will select that particular action.
Esc	Closes any of the popup that displays on the Schedule control.
Alt+N	Opens the Create Appointment window
Ctrl+E	Opens the Edit Appointment window
Alt+C	Opens the calendar popup within the Scheduler.
Ctrl+left arrow	Previous date navigation
Ctrl+right arrow	Next date navigation
Alt++	Forward traversing of view items in the toolbar
Alt+-	Reverse traversing of view items in the toolbar
Space	When the previous/next navigation icons are currently being focused, pressing space navigates through the corresponding dates.

	Also, when the focus is being moved on to the fields like checkbox or buttons of the appointment window, pressing enter will select that particular action.
Del	Deletes the currently selected appointment.
Tab	Traversing through the appointments in a forward direction.
Shift+tab	Traversal of appointments in a reverse order.

**Note:** By default [allowKeyboardNavigation](#) property is set to **true**, which allows the Scheduler to be accessed through the above specified keys.

### Setting Dimension

The dimension normally refers to the height and width of the element. With Scheduler, it is possible to set 2 kinds of dimensions.

- Scheduler dimension (Scheduler control height and width)
- Cell dimension (Scheduler cells height and width)

### Scheduler Dimension

The [height](#) and [width](#) properties can be defined to set the outer dimension of the Scheduler control.

#### HTML

```
var dManager = [{
  Id: 100,
  Subject: "Research on Sky Miracles",
  StartTime: new Date(2015, 04, 05, 9, 00),
  EndTime: new Date(2015, 04, 05, 10, 30),
  OwnerId: 3,
  RoomId: 2
}];
var Scheduler = React.createClass({
  render: function () {
    return (
      <EJ.Schedule id="Schedule1" width="70%" height="500px" currentDate={new
      Date(2017, 5, 5)} appointmentSettings-dataSource={dManager}>
    </EJ.Schedule>
    );
  }
});
ReactDOM.render(<Scheduler />, document.getElementById('schedule-default'));
```

**Note:** The height and width properties accepts both **pixel** and **percentage** values.

### Scheduler Cell Dimensions

#### Cell Height

The [cellHeight](#) property allows the Scheduler to set the height of the cells in pixels. The appointment height in vertical mode changes accordingly as per the cell size within which it renders.

#### HTML

```
var dManager = [{
```

```

Id: 100,
Subject: "Research on Sky Miracles",
StartTime: new Date(2015, 04, 05, 9, 00),
EndTime: new Date(2015, 04, 05, 10, 30)
}];
var Scheduler = React.createClass({
render: function () {
return (
<EJ.Schedule id="Schedule1" cellHeight="40px" currentDate={new Date(2017, 5, 5)} appointmentSettings-dataSource={dManager}>
</EJ.Schedule>
);
}
});
ReactDOM.render(<Scheduler />, document.getElementById('schedule-default'));

```

**Note:** In **desktop** mode, the default height value of the cells is set to **20px**, whereas for **mobile** mode – the Scheduler cells are rendered with **40px** by default.

#### Cell Auto-Height

The height of the cells specifically in timeline view can be made to adjust automatically based on its exceeding appointment count. It is controlled by an API named [showOverflowButton](#) which accepts true or false value, denoting whether to enable/disable the cell auto-height adjusting option. To enable this option, set the value of `showOverflowButton` as `false` whereas its default value is `true`.

In **Vertical** view, the same functionality is made applicable only in the **Month View** whereas in **Horizontal** mode, it is applicable in all the views.

#### HTML

```

var dManager = [{
Id: 100,
Subject: "Wild Discovery",
StartTime: new Date(2015, 11, 2, 9, 00),
EndTime: new Date(2015, 11, 2, 10, 30),
Location: "CHINA"
}];
var Scheduler = React.createClass({
render: function () {
return (
<EJ.Schedule id="Schedule1" width="100%" height="525px"
showOverflowButton={false} currentView={ej.Schedule.CurrentView.Month}
currentDate={new Date(2017, 5, 5)} appointmentSettings-dataSource={dManager}>
</EJ.Schedule>
);
}
});
ReactDOM.render(<Scheduler />, document.getElementById('schedule-default'));

```

#### Cell Width

The [cellWidth](#) property allows the Scheduler to set the width of the cells in pixels. The appointment width adjusts based on the cell width of the Scheduler.

#### HTML

```
var dManager = [{
  Id: 100,
  Subject: "Research on Sky Miracles",
  StartTime: new Date(2015, 11, 2, 9, 00),
  EndTime: new Date(2015, 11, 2, 10, 30)
}];
var Scheduler = React.createClass({
  render: function () {
    return (
      <EJ.Schedule id="Schedule1" cellWidth="97px" showOverflowButton={false}
        currentView={ej.Schedule.CurrentView.Month} currentDate={new Date(2017, 5,
        5)} appointmentSetings-dataSource={dManager}>
      </EJ.Schedule>
    );
  }
});
ReactDOM.render(<Scheduler />, document.getElementById('schedule-default'));
```

**Note:** When the **cellHeight** and **cellWidth** properties are set with some specific pixel values, the cell size does not adapt to the responsive behavior of the Scheduler while resizing it in desktop/mobile mode.

## Responsiveness

Scheduler is provided with default **responsive** support, so that it adjust or auto-resize it's UI content based on the window or the container size on which it is placed.

### Auto-Resizing Scheduler

By default, setting 100% width to the Scheduler makes it adaptable to the window or its parent container, as and when the browser is resized. This will not allow the scheduler to adapt height-wise.

## HTML

```
var dManager = [{
  Id: 100,
  Subject: "Wild Discovery",
  StartTime: new Date(2015, 11, 2, 9, 00),
  EndTime: new Date(2015, 11, 2, 10, 30),
  Location: "CHINA"
}];
var Scheduler = React.createClass({
  render: function () {
    return (
      <EJ.Schedule id="Schedule1" width="100%" currentDate={new Date(2017, 5, 5)}
        appointmentSetings-dataSource={dManager}>
      </EJ.Schedule>
    );
  }
});
ReactDOM.render(<Scheduler />, document.getElementById('schedule-default'));
```

To auto-resize the Scheduler height – set the **height** property of the Scheduler to **100%** and also set some specific height (either percentage or pixel values) to its parent container (or) to the HTML and body tag elements as shown below.

## HTML



```

<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml" style="height:100%">
<head>
<title>My first HTML page</title>
<!-- required CSS REFERENCES -->
<!-- required SCRIPT REFERENCES -->
</head>
<body style="height:100%">
<!--Container for ejScheduler widget-->
<div id="schedule-default"></div>
</body>
</html>

```

## HTML

```

var dManager = [{
  Id: 100,
  Subject: "Wild Discovery",
  StartTime: new Date(2015, 11, 2, 9, 00),
  EndTime: new Date(2015, 11, 2, 10, 30),
  Location: "CHINA"
}];
var Scheduler = React.createClass({
  render: function () {
    return (
      <EJ.Schedule id="Schedule1" height="100%" currentDate={new Date(2017, 5, 5)}
        appointmentSettings-dataSource={dManager}>
      </EJ.Schedule>
    );
  }
});
ReactDOM.render(<Scheduler />, document.getElementById('schedule-default'));

```

**Note:** When the Scheduler width is set to have **less than 600px**, then the View selection options will be displayed in a **navigation drawer**.

Also, whenever the Scheduler resizes - the width of the work cells adjusts automatically (if no **cellWidth** property is specified with pixel values), but the height of the cells are kept to remain as same as 20px until **cellHeight** property is set explicitly with some pixel values.

### Scheduler in Mobile/Tablets

The Scheduler layout adapts automatically in the mobile and tablet devices with the appropriate UI changes, as the [isResponsive](#) property is set to **true** by default. In case, if the user wants to display the normal scheduler in mobile devices without any adaptive enhancements, then the **isResponsive** property can be set to false.

Some of the default changes done for adaptive Scheduler to render in mobile devices are as follows,

- Animation effect introduced between view/date navigation.
- Cell Height increased
- The header date display changes (displayed next to the navigation icons).
- View options displayed in Navigation drawer
- Time cell width decreased

- Quick window display blocked on cell/appointment single click.
- Appointment resizing action blocked.
- Multiple cell selection blocked.
- Delete appointment option made available within the appointment window.
- Week header display in month view hidden.
- With Multiple resources - only one resource is viewable initially on the screen and to further look onto other resources, user need to swipe the screen.

**Note:** To make the Scheduler control to react as responsive in mobile/tablet devices, it is necessary to refer the additional [ej.responsive.css](http://cdn.syncfusion.com/{{ site.releaseversion }}/js/web/responsive-css/ej.responsive.css) file in the application.

The following code snippet depicts the Scheduler code with responsive set to true.

### HTML

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml" style="height:100%">
<head>
<title>My first HTML page</title>
<link href="http://cdn.syncfusion.com/{{ site.releaseversion }}/js/web/flat-azure/ej.web.all.min.css" rel="stylesheet" />
<link href=" http://cdn.syncfusion.com/{{ site.releaseversion
}}/js/web/responsive-css/ej.responsive.css" rel="stylesheet" />
<!-- Other required SCRIPT REFERENCES -->
</head>
<body style="height:100%">
<!--Container for ejScheduler widget-->
<div id="schedule-default"></div>
</body>
</html>
```

### HTML

```
var dManager = [{
  Id: 100,
  Subject: "Wild Discovery",
  StartTime: new Date(2015, 11, 2, 9, 00),
  EndTime: new Date(2015, 11, 2, 10, 30),
  Location: "CHINA"
}];
var Scheduler = React.createClass({
  render: function () {
    return (
      <EJ.Schedule id="Schedule1" isResponsive={true} currentDate={new Date(2017, 5, 5)} appointmentSetings-dataSource={dManager}>
      </EJ.Schedule>
    );
  }
});
ReactDOM.render(<Scheduler />, document.getElementById('schedule-default'));
```

## Persistence

State persistence allows the Scheduler to retain the current model value in the browser cookies for state maintenance. This action is handled through the property [enablePersistence](#) which is set to false by default.

When it is set to **true**, some of the Schedule control model values will be retained even after refreshing the page which are listed below.

currentView	timeMode
firstDayOfWeek	dateFormat
isDST	timeZone
timeScale	startHour
endHour	workHours
Height	Width
cellHeight	cellWidth
currentDate	minDate
maxDate	renderDates
orientation	views
workWeek	agendaViewSettings.daysInAgenda
enableLoadOnDemand	showLocationField
showAllDayRow	isResponsive
enableRecurrenceValidation	showOverflowButton
allowDragAndDrop	showDeleteConfirmationDialog
showNextPrevMonth	appointmentDragArea

The Schedule properties that are not retained while maintaining state persistence are included within the **ignoreOnPersist** list, which makes it not to persist by default.

## Export and Print

### Export Appointments to ICS file

The Appointments can be exported as a whole collection or else a single appointment alone can be exported from the Scheduler. By default, the appointments are exported to .ics format which can then be imported and used in any of the other external calendars.

### Single Appointment Exporting

A single appointment can be exported by making use of its Id. You can achieve this functionality by making use of an [exportSchedule](#) method by passing the id of an appointment to export as one of its parameter.

It can also be achieved in another way by enabling the context menu settings and then adding a custom menu option for export appointment functionality. When right clicked on an appointment, and **export Appointment** option is chosen, the exporting functionality can be handled through the [menuItemClick](#) event, within which the **exportSchedule** method should be defined with the following parameters,

- Action name (to be called in the server-side)
- Server Event (optional)
- Appointment Id (mandatory for single Appointment exporting)

The following code example shows the way to export single appointment from the Scheduler.

### HTML

```
var dManager = [{
  Id: 100,
  Subject: "Wild Discovery",
  StartTime: new Date(2015, 11, 2, 9, 00),
  EndTime: new Date(2015, 11, 2, 10, 30),
  Location: "CHINA"
}];
var contextMenuData = {
  enable: true,
  menuItemClick: {
    appointment: [
      { id: "open", text: "Open Appointment" },
      { id: "delete", text: "Delete Appointment" },
      { id: "export", text: "Export Appointment" }
    ]
  }
};
var Scheduler = React.createClass({
  onMenuItemClick: function(args) {
    if (args.events.ID == "export") {
      var obj = $("#Schedule1").data("ejSchedule");
      // exportSchedule() method will send a post to the server-side to call a
      // specified action.
      obj.exportSchedule("ExportToICS", null, args.targetInfo.Id);
    }
  },
  render: function () {
    return (
      <EJ.Schedule id="Schedule1" currentDate={new Date(2017, 5, 5)}
        contextMenuSettings={contextMenuData} menuItemClick={this.onMenuItemClick}
        appointmentSettings-dataSource={dManager}>
      </EJ.Schedule>
    );
  }
});
ReactDOM.render(<Scheduler />, document.getElementById('schedule-default'));
```

**Note:** The Id value of the appointment passed in the above code example can be retrieved in the server-side action through `Request.Form["AppointmentId"]`, as the id passed from the script is stored as hidden value in the input field of the form under this name internally.

### Exporting all Appointments

To export the entire Scheduler appointments, the same [exportSchedule](#) method can be used without passing the id value to its parameter list. To achieve this, keep an individual button to export, and when it is clicked - the Scheduler can be allowed to export all the appointments.

The following code example depicts the way to export all the Scheduler appointments as a whole.

#### HTML

```
<!--Container for ejScheduler widget-->
<div id="schedule-default"></div>
<!-- Button div for Export Option-->
<button id="exportButton">Export</button>
```

#### HTML

```
var dManager = [{
  Id: 100,
  Subject: "Wild Discovery",
  StartTime: new Date(2015, 11, 2, 9, 00),
  EndTime: new Date(2015, 11, 2, 10, 30),
  Location: "CHINA"
}];
var Scheduler = React.createClass({
  render: function () {
    return (
      <EJ.Schedule id="Schedule1" currentDate={new Date(2017, 5, 5)} width="100%"
        height="525px" appointmentSettings-dataSource={dManager}>
      </EJ.Schedule>
    );
  }
});
var Button = React.createClass({
  onClick: function(args) {
    var obj = $("#Schedule1").data("ejScheduler");
    // Calls the server-side action ExportToICS
    obj.exportSchedule("ExportToICS", null, null);
  },
  render: function () {
    return (
      <EJ.Button id="btnExport" width="70px" height="30px" click={this.onClick}>
      </EJ.Button>
    );
  }
});
ReactDOM.render(<Button />, document.getElementById('exportButton'));
ReactDOM.render(<Scheduler />, document.getElementById('schedule-default'));
```

The server-side action **ExportToICS** contains the following code example to export the Scheduler appointments.

#### C#

```
public void ExportToICS(FormCollection form)
{
  IEnumerable data;
```

```

string JSONModel = Request.Form["ScheduleModel"];
var model = JsonConvert.DeserializeObject<Dictionary<string,
object>>>(JSONModel);
var Id = Request.Form["AppointmentId"];
if(Id != null)
data = db.DefaultSchedules.Where(app => app.Id.ToString() ==
Id.ToString()).ToList(); // To export single appointment
else
data = db.DefaultSchedules.ToList(); // To export all the appointments
ScheduleExport obj = new ScheduleExport(model, data);
}

```

### PDF Export

Scheduler supports exporting it along with all its appointments in PDF format, for which the same [exportSchedule](#) method can be used without passing any id value to its parameter list. To achieve this, keep an individual button to export and when it is clicked, the Scheduler with appointments can be allowed to export as PDF.

The following code example depicts the way to export the Scheduler with appointments in PDF format.

### HTML

```

<!--Container for ejScheduler widget-->
<div id="Schedule1"></div>
<!-- Button div for Export Option-->
<button id="btnExport">Export</button>

```

### HTML

```

var dManager = [{
Id: 100,
Subject: "Wild Discovery",
StartTime: new Date(2015, 11, 2, 9, 00),
EndTime: new Date(2015, 11, 2, 10, 30),
Location: "CHINA"
}];
var Scheduler = React.createClass({
render: function () {
return (
<EJ.Schedule id="Schedule1" currentDate={new Date(2017, 5, 5)} width="100%"
height="525px" appointmentSettings-dataSource={dManager}>
</EJ.Schedule>
);
}
});
var Button = React.createClass({
onClick: function(args) {
var obj = $("#Schedule1").data("ejSchedule");
obj.exportSchedule("http://js.syncfusion.com/ejservices/api/Schedule/PdfExport", null, null);
args.cancel = true;
},
render: function () {
return (
<EJ.Button id="btnExport" width="70px" height="30px" click={this.onClick}>

```

```

</EJ.Button>
);
}
});
ReactDOM.render(<Button />, document.getElementById('exportButton'));
ReactDOM.render(<Scheduler />, document.getElementById('schedule-default'));

```

The server-side action **PDFExport** contains the following code example to export the Scheduler.

### C#

```

public ActionResult PDFExport(string scheduleModel)
{
    PdfExport exp = new PdfExport();
    JavaScriptSerializer serializer = new JavaScriptSerializer();
    SchedulePDFExport convert = new SchedulePDFExport();
    // Here calling the public method to convert the model values to
    // ScheduleProperties type from string type
    ScheduleProperties scheduleObject =
    convert.ScheduleSerializeModel(scheduleModel);
    // Here converting the schedule appointments data into enumerable object by
    // deserializing
    IEnumerable scheduleAppointments =
    (IEnumerable) serializer.Deserialize(Request.Form["ScheduleProcesedApps"],
    typeof(IEnumerable));
    // Here passing the theme values from enum collection
    PdfDocument document = exp.Export(scheduleObject, scheduleAppointments,
    ExportTheme.FlatSaffron, Request.Form["locale"]);
    document.Save("Schedule.pdf", HttpContext.ApplicationInstance.Response,
    HttpStatusCode.Save);
    return RedirectToAction("PDFExportController");
}

```

**Note:** To pass the theme value as a string instead of the enum value, refer to the following code example of passing the string type theme value.

### C#

```

PdfDocument document = exp.Export(scheduleObject, scheduleAppointments,
"flat-saffron", Request.Form["locale"]);

```

### Setting Page Options

It is possible to set/change the PDF page settings such as size, margins (left, right, top and bottom), transition, and rotate angle and header/footer values by passing the page settings and page document template object values into the export method.

### Page Settings

You can set/apply the following page property values to the PDF file before exporting it with the Scheduler by using the **PdfPageSettings** object.

- a) Page Size
- b) Orientation
- c) Margins ( Left, Right, Top, Bottom)

d) Transition ( Direction, Dimension, Duration, Motion, PageDuration, Scale, Style)

e) Rotate ( RotateAngle0, RotateAngle90, RotateAngle180, RotateAngle270)

f) Height

g) Width

To apply the above page setting values, you need to create an object for the PdfPageSettings class. Then, you can assign the values to the available properties within it such as Size = PdfPageSize.A3, Orientation = PdfPageOrientation.Landscape and so on. Once done, pass this object into the export method which will set these page settings values to the PDF file before exporting it. Refer to the following code example (server-side) to set/change the page settings values.

### C#

```
public ActionResult PDFExport(string scheduleModel)
{
    JavaScriptSerializer serializer = new JavaScriptSerializer();
    SchedulePDFExport convert= new SchedulePDFExport();
    ScheduleProperties scheduleObject =
    convert.ScheduleSerializeModel(scheduleModel); // Here calling the public
    method to convert the model values to ScheduleProperties type from string
    type
    IEnumerable scheduleAppointments =
    (IEnumerable)serializer.Deserialize(Request.Form["ScheduleProcesedApps"],
    typeof(IEnumerable)); // Here deserialize the appointments to enumerable
    object
    PdfExport exp = new PdfExport();
    // Here the 50f is the default value for the margins. If you are not
    assigning any separate values like Margins.Left = 15 means 50f value will be
    setting to all the Margin properties.
    PdfPageSettings pageSettings = new PdfPageSettings(50f);
    // Here you can assign the PdfPageSize enum value (ex: A3, A4)
    pageSettings.Size = PdfPageSize.A3;
    // Here Landscape value assigned to the orientation. You can set either
    Landscape/Portrait for this Orientation property
    pageSettings.Orientation = PdfPageOrientation.Landscape;
    // Here assigned different values for the margins
    pageSettings.Margins.Left = 15;
    pageSettings.Margins.Right = 15;
    pageSettings.Margins.Top = 20;
    pageSettings.Margins.Bottom = 20;
    // Here assigned the Transition Direction as BottomToTop. You can also set
    any of the Transition values to this property
    pageSettings.Transition.Direction= PdfTransitionDirection.BottomToTop;
    // Here assigned the Rotate Angle as RotateAngle180. You can also set any of
    the Rotate values to this property
    pageSettings.Rotate = PdfPageRotateAngle.RotateAngle180;
    // Here passing the page settings object value into the export method.
    PdfDocument document = exp.Export(scheduleObject, scheduleAppointments,
    ExportTheme.FlatSaffron, Request.Form["locale"], pageSettings);
    document.Save("Schedule.pdf", HttpContext.ApplicationInstance.Response,
    HttpReadType.Save);
    return RedirectToAction("PDFExportController");
}
```



## Header/Footer Settings

You can also set the header and footer options to the PDF page before it is being exported, by passing the `PDFDocumentTemplate` object values as mentioned in the following code example.

**C#**

```
public ActionResult PDFExport(string scheduleModel)
{
    JavaScriptSerializer serializer = new JavaScriptSerializer();
    SchedulePDFExport convert = new SchedulePDFExport();
    ScheduleProperties scheduleObject =
    convert.ScheduleSerializeModel(scheduleModel); // Here calling the public
    method to convert the model values to ScheduleProperties type from string
    type
    IEnumerable scheduleAppointments =
    (IEnumerable)serializer.Deserialize(Request.Form["ScheduleProcesedApps"],
    typeof(IEnumerable)); // Here deserialize the appointments to enumerable
    object
    PdfExport exp = new PdfExport();
    PdfDocument document = new PdfDocument();
    PdfPage pdfPage = document.Pages.Add();
    //Create a header and draw the string.
    // Here the bounds value is used to store the position/axis value, where the
    header and footer content to be displayed
    RectangleF bounds = new RectangleF(0, 0,
    document.Pages[0].GetClientSize().Width, 50);
    // Creating object for the PdfPageTemplateElement
    PdfPageTemplateElement header = new PdfPageTemplateElement(bounds);
    // Here setting the font style and color to display the header/footer
    content
    PdfSolidBrush brush = new PdfSolidBrush(Color.Gray);
    PdfFont font = new PdfStandardFont(PdfFontFamily.Helvetica, 6,
    PdfFontStyle.Bold);
    PdfStringFormat format = new PdfStringFormat();
    format.Alignment = PdfTextAlignment.Center;
    format.LineAlignment = PdfVerticalAlignment.Middle;
    header.Graphics.DrawString("Schedule", font, brush, new RectangleF(0, 18,
    header.Width, header.Height), format);
    //Create a Page template that can be used as footer (here creating template
    to display the page number).
    PdfPageTemplateElement footer = new PdfPageTemplateElement(bounds);
    //Create page number field.
    PdfPageNumberField pageNumber = new PdfPageNumberField(font, brush);
    //Create page count field.
    PdfPageCountField count = new PdfPageCountField(font, brush);
    //Add the fields in composite fields.
    PdfCompositeField compositeField = new PdfCompositeField(font, brush, "Page
    {0} of {1}", pageNumber, count);
    compositeField.Bounds = footer.Bounds;
    //Draw the composite field in footer.
    compositeField.Draw(footer.Graphics, new PointF(470, 40));
    PdfDocumentTemplate headerFooterTemplate = new PdfDocumentTemplate();
    //Here assigning the header template value to the PdfDocumentTemplate
    Object.
    headerFooterTemplate.Top = header;
    //Here assigning the footer template value to the PdfDocumentTemplate
    Object.
}
```

```
headerFooterTemplate.Bottom = footer;
//Here passing the PdfDocumentTemplate Object value into the export method.
document = exp.Export(scheduleObject, scheduleAppointments,
ExportTheme.FlatSaffron, Request.Form["locale"], headerFooterTemplate);
document.Save("Schedule.pdf", HttpContext.ApplicationInstance.Response,
HttpReadType.Save);
return RedirectToAction("PDFExportController");
}
```

**Note:** The header and footer values can be passed to the PDF file only through the template option. In the above mentioned code example, the template has been written with the text “Schedule” to display it as the header text and the page number (ex: Page 1 of 2) has been displayed in the footer part.

It is also possible to pass both the `PageSettings` and header/footer template objects in the export method altogether to apply the page settings as well as the header and footer options to the PDF file which can be referred from the following code example,

### C#

```
public ActionResult PDFExport(string scheduleModel)
{
//Here add the code example of both the page settings and header/footer
values settings.
//Then pass both PdfPageSettings and PdfDocumentTemplate object into the
export method.
document = exp.Export(scheduleObject, scheduleAppointments,
ExportTheme.FlatSaffron, Request.Form["locale"], pageSettings,
headerFooterTemplate);
document.Save("Schedule.pdf", HttpContext.ApplicationInstance.Response,
HttpReadType.Save);
return RedirectToAction("PDFExportController");
}
```

### Excel Export

Scheduler supports exporting all or simply the current view appointments in an Excel format, for which the [exportToExcel](#) method is used. To showcase this functionality, here we have kept an individual button to export and therefore when it is clicked, the appointments are allowed to export to a Excel format file.

In [exportToExcel](#) method, we can pass three arguments as follows:

- Action name (to be called in the server-side)
- Server Event (Optional)
- Appointment export type (Either to include or exclude the occurrence appointments)

The appointments with additional fields can also be exported based on the number of new fields added to specific class. For this, we should create a class with specific fields to be exported and based on that class, object can be deserialized and passed for exporting.

The following code example depicts the way to export the appointments in an Excel format.

### HTML

```
<!--Container for ejScheduler widget-->
```

```
<div id="Schedule1"></div>
<!-- Button div for Export Option-->
<button id="btnExport">Export</button>
```

## HTML

```
var dManager = [{
  Id: 100,
  Subject: "Wild Discovery",
  StartTime: new Date(2015, 11, 2, 9, 00),
  EndTime: new Date(2015, 11, 2, 10, 30),
  Location: "CHINA"
}];
var Scheduler = React.createClass({
  render: function () {
    return (
      <EJ.Schedule id="Schedule1" currentDate={new Date(2017, 5, 5)} width="100%"
        height="525px" appointmentSettings-dataSource={dManager}>
      </EJ.Schedule>
    );
  }
});
var Button = React.createClass({
  onClick: function(args) {
    var obj = $("#Schedule1").data("ejSchedule");
    obj.exportToExcel("http://js.syncfusion.com/ejservices/api/Schedule/XlsExport", null, false);
    args.cancel = true;
  },
  render: function () {
    return (
      <EJ.Button id="btnExport" width="70px" height="30px" click={this.onClick}>
      </EJ.Button>
    );
  }
});
ReactDOM.render(<Button />, document.getElementById('exportButton'));
ReactDOM.render(<Scheduler />, document.getElementById('schedule-default'));
```

The server-side action **XlsExport** contains the following code example to export the Scheduler.

## C#

```
public ActionResult XlsExport()
{
  List<AppointmentData> scheduleAppointments =
    (List<AppointmentData>) JsonConvert.DeserializeObject(Request.Form["ScheduleAppointment"],
    typeof(List<AppointmentData>));
  ExcelExport xlExport = new ExcelExport();
  return xlExport.Export(scheduleAppointments, ExcelVersion.Excel2013);
}
public class AppointmentData
{
  public int Id { get; set; }
  public string Subject { get; set; }
  public string StartTime { get; set; }
}
```

```

public string EndTime { get; set; }
public bool AllDay { get; set; }
public bool Recurrence { get; set; }
public string RecurrenceRule { get; set; }
}

```

In Export method, the appointments are mandatory to be passed as a first argument. In addition, we can also pass the specific filename (on which the exported file should be saved in the machine) and excel version as optional parameters, to export the file on specified filename and version.

### Print

Two types of Print options are available – either to print the entire Scheduler layout including all the appointments in it or else to print any particular appointment alone. The public method [print](#) is used to print the entire Scheduler.

#### Print the Scheduler

The following code example shows the way to print the entire Scheduler, by keeping an extra button in a page – on which clicking the button will print the entire Scheduler.

### HTML

```

<!--Container for ejScheduler widget-->
<div id="Schedule1"></div>
<!--Button div for Print Option-->
<button id="exportButton">Print</button>

```

### HTML

```

var dManager = [{
  Id: 100,
  Subject: "Wild Discovery",
  StartTime: new Date(2015, 11, 2, 9, 00),
  EndTime: new Date(2015, 11, 2, 10, 30),
  Location: "CHINA"
}];
var Scheduler = React.createClass({
  render: function () {
    return (
      <EJ.Schedule id="Schedule1" currentDate={new Date(2017, 5, 5)} width="100%"
        height="525px" appointmentSettings-dataSource={dManager}>
      </EJ.Schedule>
    );
  }
});
var Button = React.createClass({
  onClick: function(args) {
    var obj = $("#Schedule1").data("ejScheduler");
    obj.print();
  },
  render: function () {
    return (
      <EJ.Button id="btnExport" width="70px" height="30px" click={this.onClick}>
      </EJ.Button>
    );
  }
});

```

```
});
ReactDOM.render(<Button />, document.getElementById('exportButton'));
ReactDOM.render(<Scheduler />, document.getElementById('schedule-default'));
```

### Printing Specific Appointments

To print a particular appointment, either the default context menu **print** option can be used or else the [print](#) method can be used by passing the id of the appointment to be printed as its argument.

### Using Context Menu

Here, the print action is handled internally by default, so that when an appointment is right clicked – choosing print option from the context menu that pops-out will automatically print that particular appointment.

**Note:** To handle this functionality, the context menu settings needs to be enabled with print option added to the appointment items.

The following code example depicts the way to print a particular appointment.

### HTML

```
var dManager = [{
  Id: 100,
  Subject: "Wild Discovery",
  StartTime: new Date(2015, 11, 2, 9, 00),
  EndTime: new Date(2015, 11, 2, 10, 30),
  Location: "CHINA"
}];
var contextMenuData = {
  enable: true,
  menuItems: {
    appointment: [
      { id: "open", text: "Open Appointment" },
      { id: "delete", text: "Delete Appointment" },
      { id: "print", text: "Print Appointment" }
    ]
  }
};
var Scheduler = React.createClass({
  render: function () {
    return (
      <EJ.Schedule id="Schedule1" currentDate={new Date(2017, 5, 5)}
        contextMenuSettings={contextMenuData} appointmentSettings-
        dataSource={dManager}>
      </EJ.Schedule>
    );
  }
});
ReactDOM.render(<Scheduler />, document.getElementById('schedule-default'));
```

### Using print() method

The public method **print** needs to be used here by passing the appointment object as its argument, that needs to be printed.

For example, here the below code example depicts the way to print the particular appointment programmatically by calling the **print** function within the [appointmentClick](#) event, which triggers whenever the user clicks on an appointment.

### HTML

```
var dManager = [{
  Id: 100,
  Subject: "Wild Discovery",
  StartTime: new Date(2015, 11, 2, 9, 00),
  EndTime: new Date(2015, 11, 2, 10, 30),
  Location: "CHINA"
}];
var Scheduler = React.createClass({
  onAppointmentClick: function(args) {
    var schObj = $("#Schedule1").data("ejSchedule");
    schObj.print(args.appointment);
  },
  render: function () {
    return (
      <EJ.Schedule id="Schedule1" currentDate={new Date(2017, 5, 5)}
        appointmentClick={this.onAppointmentClick} appointmentSettings-
        dataSource={dManager}>
      </EJ.Schedule>
    );
  }
});
ReactDOM.render(<Scheduler />, document.getElementById('schedule-default'));
```

### Import Appointments

To import appointments into the Scheduler, server-side method `renderingImportAppointments` needs to be used, which returns the list of appointments retrieved from the specified file path.

Refer the following code example to import the appointments into Scheduler.

### HTML

```
var dManager = [{
  Id: 100,
  Subject: "Wild Discovery",
  StartTime: new Date(2015, 11, 2, 9, 00),
  EndTime: new Date(2015, 11, 2, 10, 30),
  Location: "CHINA"
}];
var Scheduler = React.createClass({
  render: function () {
    return (
      <EJ.Schedule id="Schedule1" currentDate={new Date(2017, 5, 5)}
        appointmentSettings-dataSource={dManager}>
      </EJ.Schedule>
    );
  }
});
var Button = React.createClass({
  onScheduleImport: function(args) {
```

```

var dataManger = ej.DataManager({ url: '@Url.Action("ScheduleImportData",
"Schedule")', crossDomain: true });
$("#Schedule1").ejSchedule({
appointmentSettings: {
dataSource: dataManger
}
});
},
render: function () {
return (
<EJ.Button id="btnExport" width="70px" height="30px"
click={this.onScheduleImport}>
</EJ.Button>
);
}
});
ReactDOM.render(<Button />, document.getElementById('exportButton'));
ReactDOM.render(<Scheduler />, document.getElementById('schedule-default'));

```

The server-side action `ScheduleImportData` contains the following code example to import the Scheduler appointments.

### C#

```

public JsonResult ScheduleImportData() {
var destinationPath = @"FilePath\iCalender.ics";
ScheduleImport importApps = new ScheduleImport();
var app = importApps.renderingImportAppointments(destinationPath);
int intMax = app.Max(a => a.Id);
List<ScheduleAppointmentsObjData> result = new
List<ScheduleAppointmentsObjData>();
for (var i = 0; i < app.Count; i++) {
app[i].Id = intMax + 1;
ScheduleAppointmentsObjData row = new ScheduleAppointmentsObjData(app[i].Id,
app[i].Subject, app[i].Location, app[i].StartTime, app[i].EndTime,
app[i].Description, null, null, app[i].Recurrence, null, null,
app[i].AppointmentCategorize, null, app[i].AllDay, null, null,
app[i].RecurrenceRules, null, null);
result.Add(row);
intMax = app[i].Id;
}
return Json(result, JsonRequestBehavior.AllowGet);
}

```

## Miscellaneous

### Time Indicator

The current system time can be depicted in a scheduler with an indication of a piece of text displaying the time over a horizontal line across today's date (Vertical Scheduler mode). In Horizontal mode, the time indicator is displayed as a small vertical line within the header time cells (depicting current time).

It is enabled by default in Scheduler and to hide it, [showCurrentTimeIndicator](#) property needs to be set as **false** as shown below.

### HTML

```

var dManager = [{
  Id: 100,
  Subject: "Research on Sky Miracles",
  StartTime: new Date(2015, 11, 2, 9, 00),
  EndTime: new Date(2015, 11, 2, 10, 30)
}];
var Scheduler = React.createClass({
  render: function () {
    return (
      <EJ.Schedule id="Schedule1" currentDate={new Date(2017, 5, 5)}
        showCurrentTimeIndicator={false} appointmentSetings-dataSource={dManager}>
      </EJ.Schedule>
    );
  }
});
ReactDOM.render(<Scheduler />, document.getElementById('schedule-default'));

```

**Note:** When Scheduler rendered with multiple resources, multiple time indicators are displayed in Vertical mode – whereas single indicator displays for horizontal mode.

#### Show/Hide All-Day Row

The All-day row cell is a single row region displayed at the top of the work cells area to hold the all-day appointments together. The Scheduler displays it by default and if it needs to be hidden, the [showAllDayRow](#) property can be set to **false** as shown below.

#### HTML

```

var dManager = [{
  Id: 100,
  Subject: "Research on Sky Miracles",
  StartTime: new Date(2015, 11, 2, 9, 00),
  EndTime: new Date(2015, 11, 2, 10, 30)
}];
var Scheduler = React.createClass({
  render: function () {
    return (
      <EJ.Schedule id="Schedule1" currentDate={new Date(2017, 5, 5)}
        showAllDayRow={false} appointmentSetings-dataSource={dManager}>
      </EJ.Schedule>
    );
  }
});
ReactDOM.render(<Scheduler />, document.getElementById('schedule-default'));

```

**Note:** The All-day row expands vertically, whenever more number of appointments are populated in one or more days.

#### Show/Hide Header bar

The header bar is the top most region of the Scheduler which includes the date navigation icons and view navigation options – and also shown on the Scheduler by default. To hide the header bar, set the [showHeaderBar](#) property to **false** as shown below.

#### HTML

```

var dManager = [{

```



```

Id: 100,
Subject: "Research on Sky Miracles",
StartTime: new Date(2015, 11, 2, 9, 00),
EndTime: new Date(2015, 11, 2, 10, 30)
}];
var Scheduler = React.createClass({
render: function () {
return (
<EJ.Schedule id="Schedule1" currentDate={new Date(2017, 5, 5)}
showHeaderBar={false} appointmentSetings-dataSource={dManager}>
</EJ.Schedule>
);
}
});
ReactDOM.render(<Scheduler />, document.getElementById('schedule-default'));

```

### Show/Hide TimeScale

The TimeScale is the left most region of the Scheduler in vertical mode and the same is displayed at the top position in horizontal mode. It depicts the time interval either in a 12 or 24 hour mode. By default, the timeScale is displayed in the Scheduler and in order to hide it – set the [showTimeScale](#) option to **false** as shown below.

#### HTML

```

var dManager = [{
Id: 100,
Subject: "Research on Sky Miracles",
StartTime: new Date(2015, 11, 2, 9, 00),
EndTime: new Date(2015, 11, 2, 10, 30)
}];
var Scheduler = React.createClass({
render: function () {
return (
<EJ.Schedule id="Schedule1" currentDate={new Date(2017, 5, 5)}
showTimeScale={false} appointmentSetings-dataSource={dManager}>
</EJ.Schedule>
);
}
});
ReactDOM.render(<Scheduler />, document.getElementById('schedule-default'));

```

### Show/Hide Location Field

The Location field in the default appointment window can be displayed or hidden from it using the [showLocationField](#) property. By default, this property is set to false. To display the location option in the appointment window, then set **showLocationField** to **true** and also bind the appropriate field to the location property within the appointmentSettings as shown below.

#### HTML

```

var dManager = [{
Id: 100,
Subject: "Research on Sky Miracles",
StartTime: new Date(2015, 11, 2, 9, 00),
EndTime: new Date(2015, 11, 2, 10, 30),

```

```

EventLocation: "RDU"
}];
var Scheduler = React.createClass({
  render: function () {
    return (
      <EJ.Schedule id="Schedule1" currentDate={new Date(2017, 5, 5)}
        showLocationField={true} appointmentSettings-dataSource={dManager}
        appointmentSettings-location="EventLocation">
      </EJ.Schedule>
    );
  }
});
ReactDOM.render(<Scheduler />, document.getElementById('schedule-default'));

```

## Recurrence Editor

The **Recurrence Editor** includes the entire recurrence related information in a separate portable manner which can be either utilized as a separate widget or else can be embed within the appointment window of Scheduler to enable recurrence options within it. The recurrence rule can be easily generated based on the frequency selected. The customizations like changing the labels of the recurrence editor is also possible to achieve through its properties. The frequencies available are Never, Daily, Weekly, Monthly, Yearly and Every weekday.

### Getting Started

To render the Recurrence Editor control as a separate widget, the following list of external dependencies are needed,

- [jQuery](#) - 1.7.1 and later versions
- [jsRender](#) - to render the templates
- [jQuery.easing](#) - to support animation effects in the components

The other required internal dependencies are tabulated below,

File	Description/Usage
ej.core.min.js	Must be referred always first before using all the JS controls.
ej.globalize.min.js	Must be referred to localize any of the JS control's text and content.
ej.recurrenceeditor.min.js	Schedule core file
ej.scroller.min.js	These files are referred for proper working of the sub-controls used within RecurrenceEditor.
ej.datepicker.min.js	
ej.checkbox.min.js	
ej.editor.min.js	
ej.dropdownlist.min.js	

ej.radiobutton.min.js	
-----------------------	--

**Note:** Recurrence Editor uses one or more sub-controls, therefore refer the `ej.web.all.min.js` (which encapsulates all the `ej` controls and frameworks in a single file) in the application instead of referring all the above specified internal dependencies.

To get the real appearance of the Recurrence Editor, the dependent CSS file `ej.web.all.min.css` (which includes styles of all the widgets) should also needs to be referred.

### Control Initialization

Create a div element within the body section of the HTML document, where the Recurrence Editor needs to be rendered.

#### HTML

```
<body>
<div id="RecurrenceEditor"></div>
</body>
```

Initialize the Recurrence Editor control, by adding the following script code to the body section of the HTML document.

#### HTML

```
<body>
<!-- div for RecurrenceEditor creation -->
<div id="RecurrenceEditor"></div>
</body>
```

#### HTML

```
var Recurrence = React.createClass({
  render: function () {
    return (
      <EJ.RecurrenceEditor id="editor"></EJ.RecurrenceEditor>
    );
  }
});
ReactDOM.render(<Recurrence />,
  document.getElementById('RecurrenceEditor'));
```

### Generating Recurrence Rule

The Recurrence Editor can be used to generate the recurrence rule as a string, based on the repeat options selected.

The following code example depicts the way to generate the recurrence rule.

#### HTML

```
var Recurrence = React.createClass({
  onCreate: function(args) {
    this.element.find("#recurrencetype_wrapper").css("width", "33%");
```

```

$("#RecurrenceEditor").after($("#donerecur1"));
},
render: function () {
return (
<EJ.RecurrenceEditor id="editor" selectedRecurrenceType={0}
create={this.onCreate}></EJ.RecurrenceEditor>
);
}
});
var Button = React.createClass({
closeRecurrence: function(args) {
var obj = $("#RecurrenceEditor").data("ejRecurrenceEditor");
alert(obj.getRecurrenceRule());
},
render: function() {
return (
<EJ.Button id="donerecur1" width="155px" text="Recurrence String"
height="35px" showRoundedCorner={true}
click={this.closeRecurrence}></EJ.Button>
);
}
});
ReactDOM.render(<Recurrence />,
document.getElementById('RecurrenceEditor'));

```

## How To

### Validate the Custom Appointment Window Fields

The client-side validation of the fields present within the custom appointment window can be handled before submitting it, by adding appropriate validation classes to each field.

Refer the steps [here](#) and create a sample for Custom Appointment window, before proceeding with the following validations.

In the custom appointment window sample, create an additional CSS class **validation** as mentioned below to add it to the appropriate fields, if the validation of such fields fails.

### HTML

```

<style>
.validation {
border-color: red;
}
</style>

```

The sample contains the fields like Subject, Description, StartTime and EndTime which needs to be validated at client-side. To do so, add the required validation code within the script section as follows.

### HTML

```

<script type="text/javascript">
// To Validate the Subject field.
$("#subject").focusout(function() {
if ($.trim($("#subject").val()) == "") {
$("#subject").addClass("validation");
return false;
}
}

```

```

}
})
// To Validate the Description field.
$("#customDescription").focusout(function() {
if ($.trim($("#customDescription").val()) == "") {
$("#customDescription").addClass("validation");
return false;
}
})
// To Validate the Time duration of the appointments
$("#EndTime").focusout(function() {
if (new Date($("#EndTime").val()).getDate() >= new
Date($("#StartTime").val()).getDate()) {
if (new Date($("#StartTime").val()).getTime() >= new
Date($("#EndTime").val()).getTime())
alert("EndTime value is lesser than the StartTime value");
}
})
</script>

```

Now, after adding the above validations – whenever the fields within the custom appointment window are skipped without filling any information, it will be notified to the users appropriately.

### Highlight Different Work Hours for Each Resources

By default, the work hours of the Scheduler is highlighted based on the start and end values provided within the [workHours](#) object. It remains same for all the resources, when the Scheduler is rendered with multiple resources. To customize this behavior so as to highlight different work hours range for each of the resources, the following workaround can be utilized by making use of the Scheduler events **create** and [actionComplete](#).

Initially, set the **highlight** as false for the **workHours**, so as to disable the highlighting of default work hour range.

### HTML

```

var dManager = [{
Id: 100,
Subject: "Research on Sky Miracles",
StartTime: new Date(2015, 04, 05, 9, 00),
EndTime: new Date(2015, 04, 05, 10, 30),
OwnerId: 3,
RoomId: 2
}];
var groupData = { resources: ["Rooms, Owners"] };
var roomData = {
dataSource: [
{ text: "ROOM 1", id: 1, groupId: 1, color: "#cb6bb2" },
{ text: "ROOM 2", id: 2, groupId: 1, color: "#56ca85" }
],
text: "text", id: "id", groupId: "groupId", color: "color"
};
var ownerData = {
dataSource: [
{ text: "Nancy", id: 1, groupId: 1, color: "#ffaa00", on: 10, off: 18 },
{ text: "Steven", id: 3, groupId: 2, color: "#f8a398", on: 6, off: 10 },
{ text: "Michael", id: 5, groupId: 1, color: "#7499e1", on: 11, off: 15 }
]
}

```

```

],
text: "text", id: "id", groupId: "groupId", color: "color", start: "on",
end: "off"
});
var Scheduler = React.createClass({
render: function () {
return (
<EJ.Schedule id="Schedule1" width="100%" height="525px" currentDate={new
Date(2017, 5, 5)} group={groupData} appointmentSettings-
dataSource={dManager} appointmentSettings-id="Id" appointmentSettings-
subject="Subject" appointmentSettings-startTime="StartTime"
appointmentSettings-endTime="EndTime" appointmentSettings-
description="Description" appointmentSettings-allDay="AllDay"
appointmentSettings-recurrence="Recurrence" appointmentSettings-
recurrenceRule="RecurrenceRule" appointmentSettings-resourceFields=
"RoomId,OwnerId">
<resources>
<resource allowMultiple={false} field="RoomId" title="Room" name="Rooms"
resourceSettings={roomData}></resource>
<resource allowMultiple={true} field="OwnerId" title="Owner" name="Owners"
resourceSettings={ownerData}></resource>
</resources>
</EJ.Schedule>
);
}
});
ReactDOM.render(<Scheduler />, document.getElementById('schedule-default'));

```

### Display Scheduler with Appointments Filtered by Subject

It is possible to display the Scheduler with appointments, which is filtered based on the Subject. For example, if we keep a text box and start typing a character – the list of appointments whose subject matches the typed character alone will be shown on the Scheduler. The following code example depicts the way to achieve this.

#### HTML

```

<!--textbox for entering search character-->
<input id='txtSearch' type='text' onkeyup='searchKeyUp()' />
<!--Container for ejScheduler widget-->
<div id="schedule-default"></div>
<script type="text/javascript">
// This function executes when a character is entered in the textbox
function searchKeyUp() {
var searchString = $("#txtSearch").val();
var schObj = $("#Schedule1").data("ejScheduler");
var result = schObj.searchAppointments(searchString);
schObj.option("appointmentSettings", { dataSource: [] });
if (!ej.isNullOrUndefined(result) && result.length != 0 && searchString !=
"")
schObj.option("appointmentSettings", { dataSource: result });
else
schObj.option("appointmentSettings", { dataSource: window.Default });
}
</script>

```

**HTML**

```
window.Default = [{
  Id: 101,
  Subject: "Bering Sea Gold",
  StartTime: new Date(2015, 11, 5, 10, 00),
  EndTime: new Date(2015, 11, 5, 11, 00),
  Description: "",
  AllDay: false,
  Recurrence: false,
  Categorize: "1,3"
}, {
  Id: 102,
  Subject: "Bering Sea Gold",
  StartTime: new Date(2015, 11, 2, 16, 00),
  EndTime: new Date(2015, 11, 2, 17, 30),
  Description: "",
  AllDay: false,
  Recurrence: false,
  Categorize: "2,5"
}, {
  Id: 104,
  Subject: "What Happened Next?",
  StartTime: new Date(2015, 11, 5, 12, 30),
  EndTime: new Date(2015, 11, 5, 15, 00),
  Description: "",
  AllDay: false,
  Recurrence: false,
  Categorize: "4,1"
}, {
  Id: 105,
  Subject: "Daily Planet",
  StartTime: new Date(2015, 11, 3, 01, 00),
  EndTime: new Date(2015, 11, 3, 02, 00),
  Description: "",
  AllDay: false,
  Recurrence: false,
  Categorize: "1,3,6"
}, {
  Id: 107,
  Subject: "How It's Made",
  StartTime: new Date(2015, 11, 1, 06, 00),
  EndTime: new Date(2015, 11, 1, 07, 30),
  Description: "",
  AllDay: false,
  Recurrence: true,
  RecurrenceRule: "FREQ=WEEKLY;BYDAY=MO,TU;INTERVAL=1;COUNT=15",
  Categorize: "2,3,6"
}, {
  Id: 108,
  Subject: "Deadest Catch",
  StartTime: new Date(2015, 11, 3, 16, 00),
  EndTime: new Date(2015, 11, 3, 17, 00),
  Description: "",
  AllDay: false,
  Recurrence: false,
  Categorize: "2,4,6,1"
```

```

}, {
  Id: 109,
  Subject: "MayDay",
  StartTime: new Date(2015, 3, 30, 06, 30),
  EndTime: new Date(2015, 3, 30, 07, 30),
  Description: "",
  AllDay: false,
  Recurrence: false,
  Categorize: "5,3"
}, {
  Id: 115,
  Subject: "Cash Cab",
  StartTime: new Date(2015, 3, 30, 15, 00),
  EndTime: new Date(2015, 3, 30, 16, 30),
  Description: "",
  AllDay: false,
  Recurrence: true,
  RecurrenceRule: "FREQ=DAILY;INTERVAL=1;COUNT=5",
  Categorize: "1,3"
}];
var Scheduler = React.createClass({
  render: function () {
    return (
      <EJ.Schedule id="Schedule1" currentDate={new Date(2017, 5, 5)}
        showCurrentTimeIndicator={false} appointmentSetings-dataSource={dManager}>
      </EJ.Schedule>
    );
  }
});
ReactDOM.render(<Scheduler />, document.getElementById('schedule-default'));

```

### Customize the Default Appointment Window

Apart from the custom appointment window, it is possible to customize the default appointment window by adding/removing the required number of fields into it. This can be achieved through the **appointmentWindowOpen** event of the scheduler.

The following code example depicts the way to achieve the customization of default appointment window.

#### HTML

```

var dManager = [{
  Id: 100,
  Subject: "Wild Discovery",
  StartTime: new Date(2015, 11, 2, 9, 00),
  EndTime: new Date(2015, 11, 2, 10, 30),
  Location: "CHINA",
  AppointmentType: "Tentative",
  Status: "90%"
}];
var Scheduler = React.createClass({
  onAppointmentWindowOpen: function(args) {
    // to add custom element in default appointment window
    if (this._appointmentAddWindow.find(".custom-fields").length == 0) {
      var customDesign = "<tr class='custom-fields'><td class='e-textlabel'>Event
        Type</td><td><input class='app-type' type='text' /></td><td class='e-

```



```

textlabel'>Event Status </td><td><input class='status'
type='text' /></td></tr>";
$(customDesign).insertAfter(this._appointmentAddWindow.find("." + this._id +
"appointmentArrow"));
}
if (!ej.isNullOrUndefined(args.appointment)) {
// if double clicked on the appointments, retrieve the custom field values
from the appointment object and fills it in the appropriate fields.
this._appointmentAddWindow.find(".app-
type").val(args.appointment.AppointmentType);
this._appointmentAddWindow.find(".status").val(args.appointment.Status);
} else {
// if double clicked on the cells, clears the field values.
this._appointmentAddWindow.find(".app-type").val("");
this._appointmentAddWindow.find(".status").val("");
}
},
render: function () {
return (
<EJ.Schedule id="Schedule1" currentDate={new Date(2017, 5, 5)}
showCurrentTimeIndicator={false}
appointmentWindowOpen={this.onAppointmentWindowOpen} appointmentSettings-
dataSource={dManager}>
</EJ.Schedule>
);
}
});
ReactDOM.render(<Scheduler />, document.getElementById('schedule-default'));

```

### Synchronize the Schedule with Outlook

Schedule appointments can be synchronized with Outlook and vice versa, by making use of the Microsoft Outlook 12/15 Object library.

The following code example depicts the way to synchronize the Schedule with Outlook.

#### HTML

```

var dataManager = ej.DataManager({
url: '@Url.Action("GetApp", "Home")',
crudUrl: '@Url.Action("Batch", "Home")',
adaptor: new ej.UrlAdaptor(),
crossDomain: true
});
var appointData = {
dataSource: dataManager,
id: "Id",
subject: "Subject",
startTime: "StartTime",
endTime: "EndTime",
startTimeZone: "StartTimeZone",
endTimeZone: "EndTimeZone",
allDay: "AllDay",
recurrence: "Recurrence",
recurrenceRule: "RecurrenceRule"
};
var Scheduler = React.createClass({

```

```

render: function () {
    return (
        <EJ.Schedule id="Schedule1" currentDate={new Date(2017, 5, 5)}
        appointmentSettings={appointData}>
        </EJ.Schedule>
    );
}
});
ReactDOM.render(<Scheduler />, document.getElementById('schedule-default'));

```

Schedule control is not directly compatible with the Outlook calendar object, as it returns the COM object. Therefore, in order to bind the schedule control with those data retrieved from outlook, then it is necessary to convert the COM object into the schedule acceptable object format. To do so add the following code example in your code behind.

### C#

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;
using System.ComponentModel;
using System.Configuration;
using System.Data;
using System.Data.SqlClient;
using Microsoft.Office.Interop.Outlook; // required Microsoft Assembly
using System.Web.Script.Serialization;
using System.Web.UI;
using System.Web.UI.WebControls;
using ScheduleCRUDJS.Models;
using System.Collections;
namespace ScheduleCRUDJS.Controllers
{
    public class HomeController : Controller
    {
        ScheduleDataDataContext db = new ScheduleDataDataContext();
        public ActionResult Index()
        {
            return View();
        }
        [HttpPost]
        public JsonResult GetApp() // function to display the outlook appointments
        in Schedule initially
        {
            IEnumerable data = GetData();
            return Json(data, JsonRequestBehavior.AllowGet);
        }
        public List<MultipleResource> GetData() // function to return the outlook
        appointments
        {
            Microsoft.Office.Interop.Outlook.Application oApp = new
            Microsoft.Office.Interop.Outlook.Application();
            Microsoft.Office.Interop.Outlook.NameSpace mapNamespace =
            oApp.GetNamespace("MAPI");

```

```

Microsoft.Office.Interop.Outlook.MAPIFolder CalendarFolder =
mapNamespace.GetDefaultFolder(Microsoft.Office.Interop.Outlook.OlDefaultFold
ers.olFolderCalendar);
Microsoft.Office.Interop.Outlook.Items outlookCalendarItems =
CalendarFolder.Items;
List<Microsoft.Office.Interop.Outlook.AppointmentItem> appointmentList = new
List<Microsoft.Office.Interop.Outlook.AppointmentItem>();
foreach (Microsoft.Office.Interop.Outlook.AppointmentItem item in
outlookCalendarItems)
{
appointmentList.Add(item);
}
List<MultipleResource> newApp = new List<MultipleResource>();
// converting COM object into IEnumerable object
for (var i = 0; i < appointmentList.Count; i++)
{
MultipleResource app = new MultipleResource();
app.Id = i;
app.Subject = appointmentList[i].Subject;
app.AllDay = appointmentList[i].AllDayEvent;
app.StartTime = Convert.ToDateTime(appointmentList[i].Start.ToString());
string endTime = appointmentList[i].End.ToString();
DateTime appEndDate = Convert.ToDateTime(endTime);
if (endTime.Contains("12:00:00 AM") && app.AllDay == true)
app.EndTime = appEndDate.AddMinutes(-1);
else
app.EndTime = appEndDate;
app.Recurrence = false;
app.RecurrenceRule = null;
newApp.Add(app);
}
return newApp;
}
public JsonResult Batch(EditParams param)
{
if (param.action == "insert" || (param.action == "batch" && param.added !=
null)) // this block of code will execute while inserting the
appointments
{
var value = param.action == "insert" ? param.value : param.added[0];
int intMax = db.MultipleResources.ToList().Count > 0 ?
db.MultipleResources.ToList().Max(p => p.Id) : 1;
DateTime startTime = Convert.ToDateTime(value.StartTime);
DateTime endTime = Convert.ToDateTime(value.EndTime);
MultipleResource appoint = new MultipleResource()
{
Id = intMax + 1,
StartTime = startTime,
EndTime = endTime,
StartTimeZone = value.StartTimeZone,
EndTimeZone = value.EndTimeZone,
Subject = value.Subject,
OwnerId = value.OwnerId,
AllDay = value.AllDay,
Recurrence = value.Recurrence,
RecurrenceRule = value.RecurrenceRule
};
};

```

```

db.MultipleResources.InsertOnSubmit(appoint);
db.SubmitChanges();
Microsoft.Office.Interop.Outlook.Application outlookApp = new
Microsoft.Office.Interop.Outlook.Application();
Microsoft.Office.Interop.Outlook.AppointmentItem newAppointment = null;
newAppointment =
(Microsoft.Office.Interop.Outlook.AppointmentItem)outlookApp.CreateItem(Micr
osoft.Office.Interop.Outlook.OlItemType.olAppointmentItem);
newAppointment.Start =
Convert.ToDateTime(value.StartTime).ToUniversalTime();
newAppointment.End = Convert.ToDateTime(value.EndTime).ToUniversalTime();
newAppointment.Subject = value.Subject.ToString();
newAppointment.Save();
}
IEnumerable data = new
ScheduleDataDataContext().MultipleResources.Take(200);
return Json(data, JsonRequestBehavior.AllowGet);
}
}
}

```

**Note:** In order to achieve the above scenario, need to refer the Microsoft assembly in your application (Microsoft Outlook 12/15 Object library [Microsoft.Office.Interop.Outlook] and refer it in the controller page as shown above).

## Scroller

### Overview

The **Scroller** control has a sliding document whose position corresponds to a value. The document has text, HTML content or images. You can also customize the Scroller control by resizing the scrolling bar and changing the theme.

### Key Features

- **Height and Width** - Set the height and width of the scroll panel.
- **Button Size** - Customize the width and height of the buttons (UP, DOWN, RIGHT and LEFT).
- **Scroller Size** - Customize the scrollbar width and height.
- **Step Increment** - Set the number of pixels to be moved on pressing the ARROW key.
- **RTL Support** - Sets the alignment of the horizontal Scroller to the right, the reading order to right-to-left and the layout of the control to flow from right to left.
- **Theme** - Essential JavaScript controls consists of 12 built in themes ( 6 – flat and 6 – gradient effects), and also supports custom skins to set user-defined themes.

### Getting Started

This section helps to get started of the Scroller component in a ReactJS application

#### Create a Scroller

Refer the common ReactJS Getting Started Documentation to create an application and add necessary scripts and styles for rendering our ReactJS components.

Create a JSX file and use <EJ.Scroller> syntax to render ReactJS Scroller component. Add required properties to <EJ.Scroller> tag element.

**HTML**

```

ReactDOM.render(
  <EJ.Scroller id="scrollcontent" height={300} width="100%" >
    <div>
      <div className="sampleContent">
        <h3 style={{fontSize: 20 + "px"}}>MVC</h3>
        <div>
          <p>
            Model-view-controller (MVC) is a software architecture pattern which
            separates the
            representation of information from the user's interaction with it.
            The model consists of application data, business rules, logic, and
            functions. A view can be any
            output representation of data, such as a chart or a diagram. Multiple views
            of the same data
            are possible, such as a bar chart for management and a tabular view for
            accountants.
            The controller mediates input, converting it to commands for the model or
            view. The central
            ideas behind MVC are code reusability and in addition to dividing the
            application into three
            kinds of components, the MVC design defines the interactions between them.
          </p>
          <ul>
            <li>
              <b>A controller </b>can send commands to its associated view to change the
              view's presentation of the model (e.g., by scrolling through a document).
              It can also send commands to the model to update the model's state (e.g.,
              editing a document).
            </li>
            <li>
              <b>A model</b> notifies its associated views and controllers when there has
              been a change in its state. This notification allows the views to produce
              updated output, and the controllers to change the available set of commands.
              A passive implementation of MVC omits these notifications, because the
              application does not require them or the software platform does not support
              them.
            </li>
            <li>
              <b>A view</b> requests from the model the information that it needs to
              generate an output representation to the user.
            </li>
          </ul>
        </div>
      </div>
    </div>
  </EJ.Scroller>,
  document.getElementById('scrollbar')
);

```

Define an HTML element for adding Scroller in the application and refer the JSX file created.

**HTML**

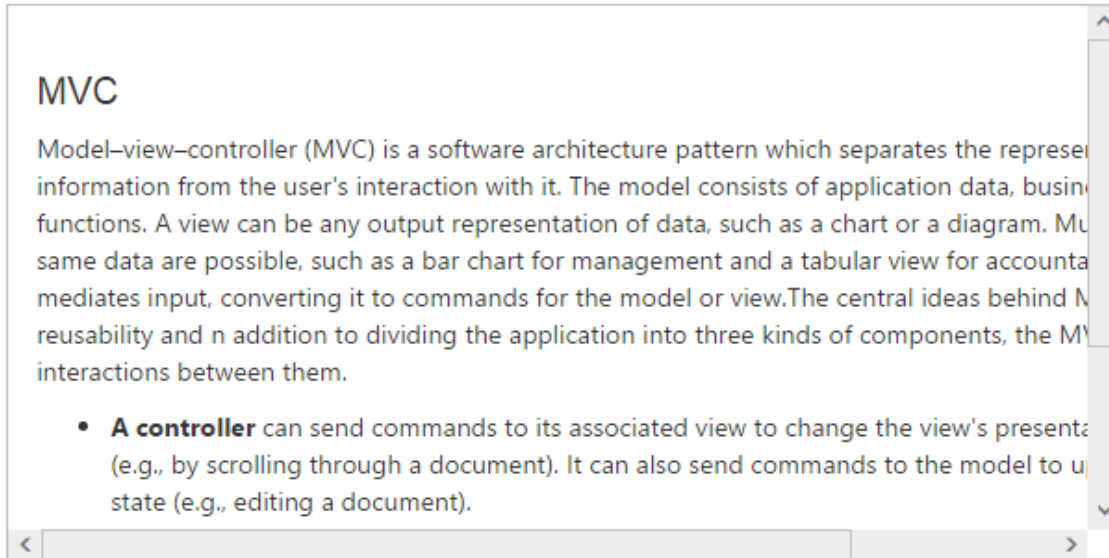
```

<div id="scrollbar"></div>
<script type="text/babel" src="sample.jsx">

```

This will render scroller component on executing.

Run the above code to render the following output,



*Note: You can find the Scroller properties from the [API reference](#) document.*

## Signature

### Overview

The Essential ReactJS Signature is a Plugin that is used to capture or drawing the smooth signatures. It captures user's signature as vector outlines of strokes. Using Signature we can customize the background, stroke width and stroke color and also convert captured signature to an image format. It is also provided with Undo, Redo & Clear options.

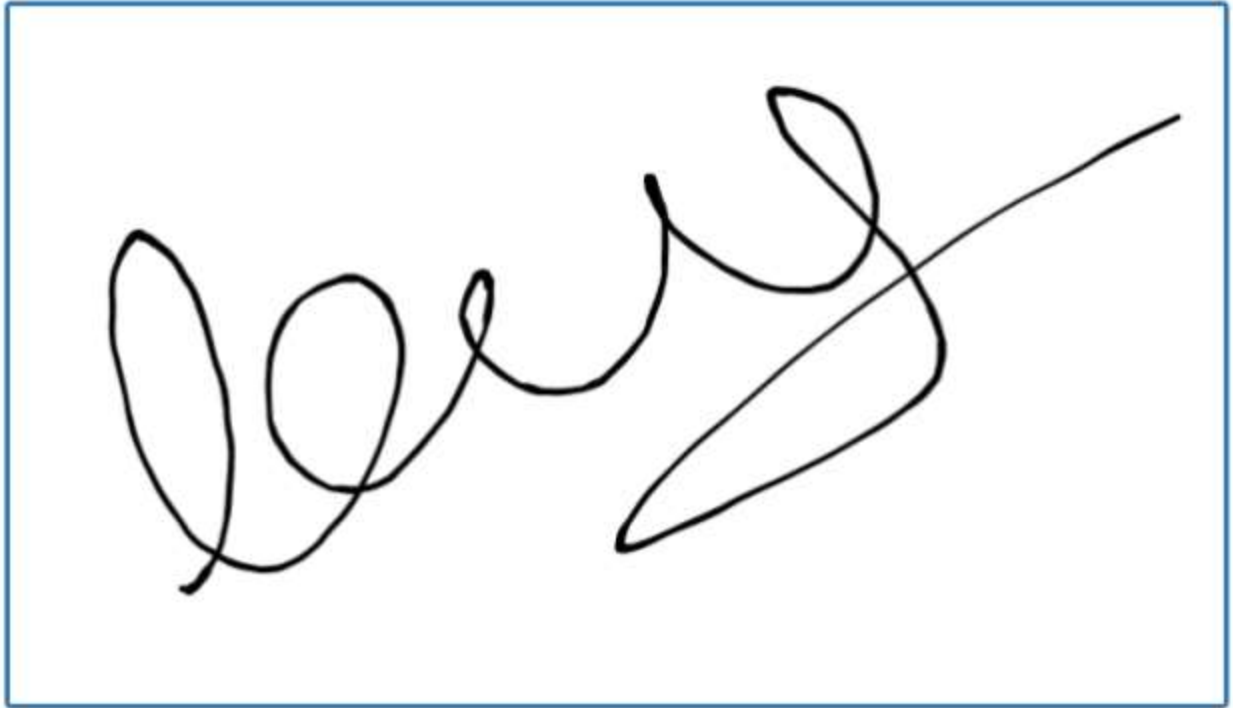
### Key Features

- **Height and Width** - Set the height and width of the signature canvas.
- **Stroke Color and Background Color** - Customize the stroke color and background color for the canvas
- **Background Image** - Sets the background image for the signature canvas.
- **Stroke Width** - Sets the stroke's minimum and maximum width for the drawing stroke.
- **Undo and Redo** – Undo and Redo the strokes in the signature, if wrongly drawn

### Getting Started

The ReactJS Signature simplifies creation of a signature capture in a browser, allowing a user to draw a signature using mouse or touch.

This section explains briefly about how to create a **Signature** component in your application with ReactJS by the following step-by-step instructions. The following screenshot demonstrates the functionality of Signature component.



Create a Signature component in ReactJS

You can create a React application and add necessary scripts and styles with the help of the given [React Getting Started Documentation](#).

Create a JSX file for rendering AutoComplete component using <EJ.Signature> syntax. Add required properties to it in <EJ. Signature > tag element

#### **HTML**

```
"use strict";
ReactDOM.render(
  <EJ.Signature id="mySignature" height={400}>
</EJ.Signature>,
  document.getElementById('signature-default')
);
```

Define an HTML element for adding AutoComplete in the application and refer the JSX file.

#### **HTML**

```
<div id="signature-default"></div>
<script type="text/babel" src="signature.jsx"></script>
```

Run the above code to render the following output.



### Adjusting Signature Size

You can customize the width and height of the Signature by width and height properties. These properties completely depend on signature canvas. The width and height are adjusted within the signature canvas.

The following code example is used to render the Signature component with customized width and height.

#### **HTML**

```
"use strict";
ReactDOM.render(
  <EJ.Signature id="mySignature" isResponsive="true" height={300}
  width={200}>
    </EJ.Signature>
  ,
  document.getElementById('signature-default')
);
```

The following screenshot illustrates signature with customized width and height.





## SplitButton

### Overview

The **SplitButton** allows you to perform an action by clicking the button and choosing extra options from the dropdown button. The **SplitButton** can also display both text and images. The drop down button option is given to the right most corner of the button.

### Key Features

- **Trendy Look** : Rich appearance with theme Support
- **RTL** : Supports for right to left alignment
- **Text and Image** : Supports both text and image as **SplitButton** content
- **Built-in Icon** : Supports the built-in icon libraries
- **Easy Customization** : The customization of **SplitButton** control to any form is made simple

### Getting Started

This section explains briefly about how to create a SplitButton in your application with ReactJS.

#### Create a SplitButton

Refer the common ReactJS getting started documentation to create an application and add necessary scripts and styles for rendering our ReactJS components.

Create a JSX file and use `<EJ.SplitButton>` syntax to render ReactJS SplitButton component. Add required properties to `<EJ.SplitButton>` tag element.

### HTML

```
ReactDOM.render(  
  <EJ.SplitButton id="spltbutton21" targetID="U121" >login</EJ.SplitButton>  
  <ul id="U121">  
    <li><span>User</span></li>  
    <li><span>Guest</span></li>  
    <li><span>Admin</span></li>  
  </ul>  
)
```

```

</ul>
document.getElementById('button-splitbutton')
);

```

Define an HTML element for adding SplitButton in the application and refer the JSX file created.

### HTML

```

<div id="button-splitbutton"></div>
<script src="app/button/splitbutton.js"></script>

```

This will render SplitButton component on executing.

### Configure Properties

In the JSX, need to declare the SplitButton properties. Refer to the following code,.

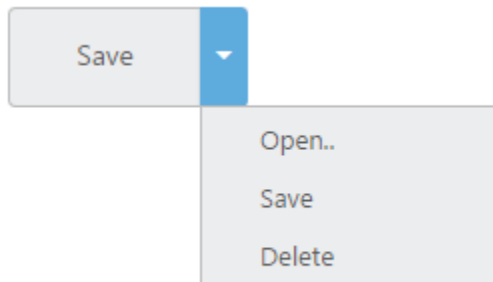
### HTML

```

ReactDOM.render (
  <EJ.SplitButton id="spltbutton21" showRoundedCorner={true} targetID="U121"
  >login</EJ.SplitButton>
  <ul id="U121">
    <li><span>User</span></li>
    <li><span>Guest</span></li>
    <li><span>Admin</span></li>
  </ul>
  document.getElementById('button-splitbutton')
);

```

Run the above code to render the following output,



*Note: You can find the SplitButton properties from the [API reference](#) document.*

## Splitter

### Overview

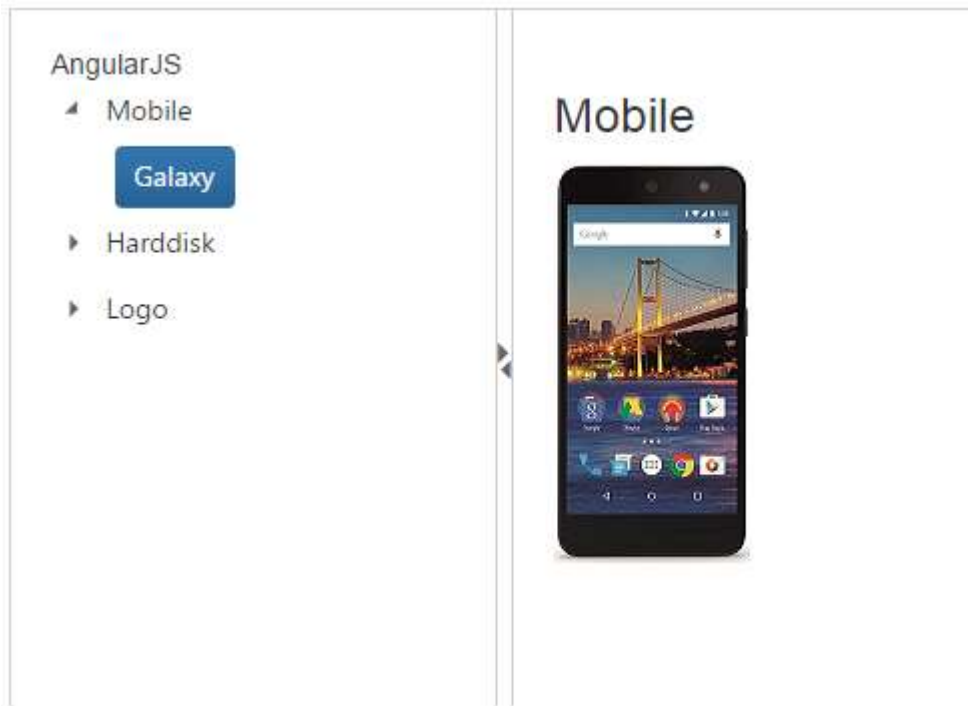
Our Essential ReactJS Splitter component enables you to divide a Web page into distinct areas by inserting resizable panes. You can create any number of Splitter panes and place them inside the Splitter component. The split bars are inserted automatically in between the adjacent panes.

## Key Features

- **Expand or Collapse Panes:** Support for expanding or collapsing panes by using the Expand or Collapse arrows.
- **Animation:** Supports for animating panes while expanding or collapsing the panes, and also setting the animation speed.
- **Window Resizing:** Supports for resizing the Splitter while resizing the window.
- **RTL:** Supports for right to left alignment of Splitter panes.

## Getting Started

This section helps to get started of the Splitter component in a React application.



### Create a Splitter

Refer the common React JS [Getting Started Documentation](#) to create an application and add necessary scripts and styles for rendering our React JS components.

Create a JSX file for rendering Splitter component using `<EJ.Splitter>` syntax. Add required properties to it in `<EJ.Splitter>` tag element.

### HTML

```
"use strict";
ReactDOM.render(
  <EJ.Splitter id="default_outerSplitter" width= "100%" height="100%"
  isResponsive={true} orientation={ej.Orientation.Vertical}>
</EJ.Splitter>,
  document.getElementById('splitter-default')
);
```

Define an HTML element for adding Rotator in the application and refer the JSX file created.

## HTML

```
<div id="splitter-default"></div>
<script type="text/babel" src="sample.jsx">
```

This will render an empty Splitter component on executing.

*Note: You can find the Splitter properties from the [API reference](#) document*

### Configure Splitter Panes

To configure properties for Splitter component, define properties in the JSX.

## HTML

```
<EJ.Splitter id="integration_outterSplitter" width= "100%" height="100%"
isResponsive={true}>
</EJ.Splitter>
```

Configure the Splitter panes with images. Save the images in the corresponding location.

## HTML

```
<div id="samplecontrol" className="integrationContainer">
<EJ.Splitter id="integration_outterSplitter" width= "100%" height="100%"
isResponsive={true}>
<div>
<div className="cont">
<h3 className="h3">React JS</h3>
</div>
</div>
<div>
<div className="cont">
<div className="_content">Select any product from the tree to show the
description.</div>
<div className="mobile spe">
<h3>Mobile</h3>

</div>
<div className="harddisk spe">
<h3>Harddisk</h3>

</div>
<div className="logo spe">
<h3>Logo</h3>

</div>
</div>
</div>
</EJ.Splitter>
</div>
```

### Configure Tree View

For adding Treeview component, you have to use <EJ.TreeView> syntax to corresponding element. You need to use “nodeSelect” event handler to perform any action while selection the node in Tree view.

Add the following code example in JSX to configure Tree View.

**HTML**

```

<EJ.TreeView id="treeView" className="visibleHide"
nodeSelect={this.treeClicked}>
<li>Mobile
<ul>
<li id="mobile" className="_child">Galaxy</li>
</ul>
</li>
<li>Harddisk
<ul>
<li id="harddisk" className="_child">Segate </li>
</ul>
</li>
<li>Logo
<ul>
<li id="logo" className="_child">Amazon</li>
</ul>
</li>
</EJ.TreeView>

```

**Set Actions**

Add the nodeSelect event in JSX to set the action to view the images for “li” content of the Tree view.

**JAVASCRIPT**

```

treeClicked: function(e) {
if (e.currentElement.hasClass('_child')) {
var content = $('.' + e.currentElement[0].id).html();
$('_content').html(content);
}
}

```

You can render Splitter with Tree View using React JS, Please refer the below code in JSX.

**HTML**

```
ReactDOM.render(document.getElementById(splitter-default'));
```

## Sparkline

### Getting Started

This section explains you the steps required to populate the Sparkline with data, tooltips and type for Sparkline. This section covers only the minimal features that you need to know to get started with the Sparkline.

### Adding Script Reference

Create an **HTML** page and add the scripts references in the order mentioned in the following code example.

- [jQuery](#) 1.10.2 and later versions

The required ReactJS script dependencies as follows. And you can also refer [React](#) to know more about react js.

- react.min.js - <http://cdn.syncfusion.com/js/assets/external/react.min.js>
- react-dom.min.js - <http://cdn.syncfusion.com/js/assets/external/react-dom.min.js>
- ej.web.react.min.js - <http://cdn.syncfusion.com/{{ site.releaseversion }}/js/common/ej.web.react.min.js>

To get started, you can use the `ej.web.all.min.js` file that encapsulates all the `ej` controls and frameworks in one single file.

## HTML

```
<!DOCTYPE html>
<html>
<head>
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<meta name="description" content="Essential Studio for React JS">
<meta name="author" content="Syncfusion">
<title>Getting Started for Ribbon React JS</title>
<!-- Essential Studio for JavaScript theme reference -->
<link href="http://cdn.syncfusion.com/{{ site.releaseversion }}/js/web/flat-
azure/ej.web.all.min.css" rel="stylesheet" />
<!-- Essential Studio for JavaScript script references -->
<script src="http://cdn.syncfusion.com/js/assets/external/jquery-
3.0.0.min.js"></script>
<script
src="http://cdn.syncfusion.com/js/assets/external/react.min.js"></script>
<script src="http://cdn.syncfusion.com/js/assets/external/react-
dom.min.js"></script>
<script src="http://cdn.syncfusion.com/{{ site.releaseversion
}}/js/web/ej.web.all.min.js"></script>
<script src="http://cdn.syncfusion.com/{{ site.releaseversion
}}/js/common/ej.web.react.min.js"></script>
<!-- Add your custom scripts here -->
</head>
<body>
</body>
</html>
```

**Note:** 1. In production, we highly recommend you to use our [custom script generator](#) to create custom script file with required controls and its dependencies only. Also to reduce the file size further please use [GZip compression](#) in your server.

2. For themes, you can use the `ej.web.all.min.css` CDN link from the code snippet given. To add the themes in your application, please refer to [this link](#).

## Control Initialization

Control can be initialized in two ways.

- Using jsx Template
- Without using jsx Template

### Using jsx Template

By using the jsx template, we can create the html file and jsx file. The .jsx file can be convert to .js file and it can be referred in html page.

#### Create Sparkline

You can easily create the Sparkline widget by following the below steps.

1.Create a

tag.

#### HTML

```
<!DOCTYPE html>
<html>
<body>
<div id="sparkline-default" ></div>
<script src="app/sparkline/default.js"></script>
</body>
</html>
```

2.Initialize the Sparkline by using the `EJ.Sparkline` tag

#### JAVASCRIPT

```
"use strict";
ReactDOM.render(
  <div className="default">
    <EJ.Sparkline id="sparkline1"></EJ.Sparkline>,
  </div>,
  document.getElementById('sparkline-default')
);
```

Now, the Sparkline is rendered with some auto-generated random values and with default Line type Sparkline.



#### Simple Sparkline

The Sparkline is rendered to it's default size. You can also customize the Sparkline dimension either by setting the width and height of the container element as in the above code example or by using the **size** of the Sparkline.

#### Populate Sparkline with data

Now, this section explains how to plot JSON data to the Sparkline. First, let us prepare a sample JSON data with each object containing following fields – employeeId and sales.

```
var sparklineData = [
  { employeeId: 1, sales: 25 },
  { employeeId: 2, sales: 28 },
```

```
{ employeeId: 3, sales: 34 },  
{ employeeId: 4, sales: 32 },  
{ employeeId: 5, sales: 40 },  
{ employeeId: 6, sales: 32 },  
{ employeeId: 7, sales: 35 },  
{ employeeId: 8, sales: 55 },  
{ employeeId: 9, sales: 38 },  
{ employeeId: 10, sales: 30 }];
```

Now, add the `dataSource` to the Sparkline and provide the field name to get the values from the `dataSource` in `xName` and `yName` options.

### JAVASCRIPT

```
<script type="text/babel">  
<!DOCTYPE html>  
<html>  
<body>  
<script type="text/babel">  
ReactDOM.render(  
<div className="default">  
<EJ.Sparkline id="sparkline1" dataSource={sparklineData} xName="employeeId"  
yName="sales"></EJ.Sparkline>,  
</div>,  
document.getElementById('sparkline-default')  
>);  
</script>  
</body>  
</html>
```

### Sparkline Type

To change the sparkline types, following example code illustrates this.

### JAVASCRIPT

```
<script type="text/babel">  
<!DOCTYPE html>  
<html>  
<body>  
<script type="text/babel">  
ReactDOM.render(  
<div className="default">  
<EJ.Sparkline id="sparkline1" type="column"></EJ.Sparkline>,  
</div>,  
document.getElementById('sparkline-default')  
>);  
</script>  
</body>  
</html>
```



### Enable Tooltip

To enable the Sparkline tooltip we can see the current point values.

The following code example illustrates enable tooltip in sparkline,

#### JAVASCRIPT

```
<script type="text/babel">
var tooltip = {
  visible: true,
};
<!DOCTYPE html>
<html>
<body>
<script type="text/babel">
ReactDOM.render(
<div className="default">
<EJ.Sparkline id="sparkline1" tooltip={tooltip}></EJ.Sparkline>,
</div>,
document.getElementById('sparkline-default')
);
</script>
</body>
</html>
```



### Without using jsx Template

The Sparkline can be created from a HTML DIV element with the HTML id attribute set to it. Refer to the following code example.

#### HTML

```
<div id="sparkline-default"></div>
```

#### JAVASCRIPT

```
<script type="text/babel">
var tooltip = {
  visible: true,
};
var dataSource1= [25, 28, 34, 32, 40, 32, 35, 55, 38, 30];
ReactDOM.render(
React.createElement(EJ.Sparkline, {id: "sparkline1",
dataSource: dataSource1,
tooltip: tooltip,
type: "line",
}
),
document.getElementById('sparkline- default')
);
</script>
```

Run the above code example and you will see the following output.



## Working with Data

### Local Data

1. You can bind the data to the Sparkline by using [dataSource](#) property and then you need to map the **X** and **Y** value with **xName** and **yName** properties respectively.

### JAVASCRIPT

```
"use strict";
var sparklinedata = [
  { employeeId: 1, sales: 25 },
  { employeeId: 2, sales: 28 },
  { employeeId: 3, sales: 34 },
  { employeeId: 4, sales: 32 },
  { employeeId: 5, sales: 40 },
  { employeeId: 6, sales: 32 },
  { employeeId: 7, sales: 35 },
  { employeeId: 8, sales: 55 },
  { employeeId: 9, sales: 38 },
  { employeeId: 10, sales: 30 }];
ReactDOM.render(
  <EJ.Sparkline id="sparkline1" dataSource = {sparklinedata} xName =
    "employeeId"
    yName = "sales"></EJ.Sparkline>,
  document.getElementById('sparkline')
);
```

;

2. You can also bind an array of data to Sparkline by using [dataSource](#) property.

### JAVASCRIPT

```
"use strict";
ReactDOM.render(
  <EJ.Sparkline id="sparkline1" dataSource = [ 2, 6, -1, 1, 12, 5, -2, 7, -3,
    5, 8, 10]
  ></EJ.Sparkline>,
  document.getElementById('sparkline')
);
```

;

### Sparkline Dimensions

You can set the size directly on the Sparkline or to the container of the sparkline. When you do not specify the size, it takes 30px as the height and 50px as its width, by default.

### Set size for the container

You can customize the Sparkline dimension by setting the width and height for the container element.

#### HTML

```
<body>
<div id="container" style="width:820px;height:500px;"></div>
<script type="text/javascript" language="javascript ">
</script>
</body>
```



### Set size in pixels

You can also set the Sparkline dimension by using the [size](#) property of the Sparkline.

#### HTML

```
"use strict";
var size = { width: '60', height: '40' };
ReactDOM.render(
  <EJ.Sparkline id="sparkline1" size = {size} ></EJ.Sparkline>,
  document.getElementById('sparkline')
);
```

### Responsive Sparkline

To resize the Sparkline when the browser or the sparkline container is resized, set the [isResponsive](#) property to **true**, where the sparkline adapts to the changes in size of the container.

#### JAVASCRIPT

```
"use strict";
ReactDOM.render(
  <EJ.Sparkline id="sparkline1" isResponsive = {true}></EJ.Sparkline>,
  document.getElementById('sparkline')
);
```

## Sparkline Types

### Line Type

To render a Line type Sparkline, set the [type](#) as **line**. To change the color and width of the line, you can use the [fill](#) and [width](#) property.

#### HTML

```
"use strict";
var sparklinedata = [
  { employeeId: 1, sales: 25 },
  { employeeId: 2, sales: 28 },
  { employeeId: 3, sales: 34 },
  { employeeId: 4, sales: 32 },
  { employeeId: 5, sales: 40 },
  { employeeId: 6, sales: 32 },
  { employeeId: 7, sales: 35 },
  { employeeId: 8, sales: 55 },
  { employeeId: 9, sales: 38 },
```

```
{ employeeId: 10, sales: 30 }]];
ReactDOM.render(
  <EJ.Sparkline id="sparkline1" dataSource = {sparklinedata} type = 'line'
  fill = "#33ccff"
  xName = "employeeId" yName = "sales" ></EJ.Sparkline>,
  document.getElementById('sparkline')
);
```



### Column Type

To render a Column Sparkline, set the type as **column**. To change the color of the column, you can use the [fill](#) property.

#### HTML

```
"use strict";
ReactDOM.render(
  <EJ.Sparkline id="sparkline1" dataSource = {sparklinedata} type = 'column'
  fill = "#33ccff"
  xName = "employeeId" yName = "sales" ></EJ.Sparkline>,
  document.getElementById('sparkline')
);
```



### Area Type

To render an Area Sparkline, you can specify the type as **area**. To change the Area color, you can use the [fill](#) property.

#### HTML

```
"use strict";
ReactDOM.render(
  <EJ.Sparkline id="sparkline1" dataSource = {sparklinedata} type = 'area'
  fill = "#33ccff" xName = "employeeId" yName = "sales" ></EJ.Sparkline>,
  document.getElementById('sparkline')
);
```



### WinLoss Type

WinLoss Sparkline render as a column segment and it show the positive, negative and neutral values. You can customize the positive and negative color of the win-loss type.

#### HTML

```
"use strict";
ReactDOM.render(
  <EJ.Sparkline id="sparkline1" dataSource = {sparklinedata} type =
  'winloss'
  fill = "#33ccff" xName = "employeeId" yName = "sales" ></EJ.Sparkline>,
  document.getElementById('sparkline')
);
```



### Pie Type

You can create a pie type sparkline by setting the type as **pie**. Colors for the pie can be customize using [palette](#) property.

#### HTML

```
"use strict";
ReactDOM.render(
  <EJ.Sparkline id="sparkline1" dataSource = {sparklinedata} type = 'pie'
    fill = "#33ccff"
    xName = "employeeId" yName = "sales" ></EJ.Sparkline>,
  document.getElementById('sparkline')
);
```



### Axis Customize

The Sparkline axis can be collapsed using visible property in [axisLineSetting](#) and this not applicable for win-loss. You can customize [color](#), [width](#) and [dash array](#) of axis line.

#### HTML

```
"use strict";
var axisLineSettings = {
  visible: true,
  color: "#ff14ae",
};
ReactDOM.render(
  <EJ.Sparkline id="sparkline1" axisLineSettings = {axisLineSettings}
    fill = "#33ccff"></EJ.Sparkline>,
  document.getElementById('sparkline')
);
```



### Marker Customization

You can customize markers by initializing the [markerSettings](#) property. The customization options allow you to change the [fill](#), [width](#), [opacity](#) and [border](#) for marker. This customization only applicable for line, column and area type Sparkline.

#### HTML

```
"use strict";
var axisLineSettings = {
  visible: true,
  color: "#ff14ae",
};
var markerSettings = {
  visible: true,
  width: 4,
  fill: "#ff14ae",
  border: {
    width: 1
  }
};
```

```
};
ReactDOM.render(
  <EJ.Sparkline id="sparkline1" markerSettings =
    {markerSettings}></EJ.Sparkline>,
  document.getElementById('sparkline')
);
```



### Point Customization

You can customize points by initializing the point colors. The customization options allow you to differentiate the [first](#), [last](#), [highest](#), [lowest](#), and [negative](#) points. This customization only applicable for line, column and area type Sparkline.

### JS

```
ReactDOM.render(
  <EJ.Sparkline id="sparkline1" negativePointColor = "red" highPointColor =
    "blue"
    lowPointColor = "orange" startPointColor = "green" endPointColor = "green"
  ></EJ.Sparkline>,
  document.getElementById('sparkline')
);
```



### Sparkline Customization

This section explains you the customization options available to make changes in the Sparkline for getting better appearance.

#### Sparkline background

You can specify the background color for the Sparkline using `background` property. When you don't specify the `background`, it takes "transparent" as background color.

### JAVASCRIPT

```
ReactDOM.render(
  <EJ.Sparkline id="line" background = "green"></EJ.Sparkline>,
  document.getElementById('spark')
);
```



#### Stroke color and width

You can customize the series border color and width using `stroke` and `width`. This is applicable for Sparkline types line and area.

### JAVASCRIPT

```
ReactDOM.render(
  <EJ.Sparkline id="line" stroke = "green"
    width={5}></EJ.Sparkline>,
  document.getElementById('spark')
);
```



### Sparkline border

You can customize the **border** width and height of the Sparkline using **border width** and **border height** properties. This is applicable for column, win-loss and pie series.

#### JAVASCRIPT

```
var border={
  color:'blue',
  width:5
};
ReactDOM.render(
  <EJ.Sparkline id="line" border={border}></EJ.Sparkline>,
  document.getElementById('spark')
);
```



### Opacity

By default **opacity** of the Sparkline is 1. You can specify the opacity value from 0 to 1. This is applicable for all types of series.

#### JAVASCRIPT

```
ReactDOM.render(
  <EJ.Sparkline id="line" opacity = {0.5}></EJ.Sparkline>,
  document.getElementById('spark')
);
```



### Localization

Sparkline is having support for localization as well. Default culture is "en-US". You can modify the culture using the property **locale**.

#### JAVASCRIPT

```
ReactDOM.render(
  <EJ.Sparkline id="line" locale = "en-fr"></EJ.Sparkline>,
  document.getElementById('spark')
);
```

### Padding for Sparkline

**Padding** is used to specify the padding value between the container and Sparkline. By default padding value of the Sparkline is 5.

#### JAVASCRIPT

```
ReactDOM.render(
  <EJ.Sparkline id="line" padding = {20}></EJ.Sparkline>,
  document.getElementById('spark')
);
```



### Canvas support

You can control whether Sparkline has to be rendered as SVG or **Canvas**. Canvas rendering supports all the functionalities supported in SVG rendering.

#### JAVASCRIPT

```
ReactDOM.render(
  <EJ.Sparkline id="line"
    enableCanvasRendering = {true}></EJ.Sparkline>,
  document.getElementById('spark')
);
```

### Themes

You can specify different **themes** for Sparkline control.

#### JAVASCRIPT

```
ReactDOM.render(
  <EJ.Sparkline id="line" theme = "saffron"></EJ.Sparkline>,
  document.getElementById('spark')
);
```



### Tooltip

A tooltip follows the pointer movement and is used to indicate the value of a point. This feature is applicable for line, column, pie, and area Sparkline. You can customize the tooltip [fill color](#), [border](#) and [font](#).

#### HTML

```
"use strict";
var tooltip = {
  visible: true,
};
ReactDOM.render(
  <EJ.Sparkline id="sparkline1" tooltip = {tooltip}></EJ.Sparkline>,
  document.getElementById('sparkline')
);
```



### Tooltip Template

HTML elements can be displayed in the tooltip by using the [template](#) option of the tooltip. The template option takes the value of the id attribute of the HTML element. You can use the **#point.x#** and **#point.y#** as place holders in the HTML element to display the x and y values of the corresponding point.

#### JS

```
<div id="item" style="display: none;">
<div>
```



```

<div>#point.x#</div>
<div>#point.y#</div>
</div>
</div>
"use strict";
var tooltip = {
  visible: true, template: 'item',
};
ReactDOM.render(
  <EJ.Sparkline id="sparkline1" tooltip = {tooltip}></EJ.Sparkline>,
  document.getElementById('sparkline')
);

```



### Range Band

The range band feature is used to highlight a particular range along the y-axis using start and end range property. You can also customize the color and opacity of the range band.

### JS

```

"use strict";
var rangeBandSettings = {
  startRange: 4,
  endRange: 30,
  color: "#ff14ae",
  opacity: 0.4
};
ReactDOM.render(
  <EJ.Sparkline id="sparkline1" rangeBandSettings = {rangeBandSettings}>
</EJ.Sparkline>,
  document.getElementById('sparkline')
);

```



### Methods

#### redraw()

Redraws the entire sparkline. You can call this method whenever you update, add or remove points from the data source or whenever you want to refresh the UI.

### HTML

```
<div id="line"></div>
```

### JAVASCRIPT

```

ReactDOM.render(
  <EJ.Sparkline id="default"></EJ.Sparkline>,
  document.getElementById('line')
);
function SparkLineMethod() {
  var sparkObj = $("#default").data("ejSparkLine");
  sparkObj.redraw();
};

```

## Events

### *load*

Fires before loading the sparkline.

#### **HTML**

```
<div id="line"></div>
```

#### **JAVASCRIPT**

```
ReactDOM.render(  
  <EJ.Sparkline id="default" load = {Load}></EJ.Sparkline>,  
  document.getElementById('line')  
)  
;  
function Load() {  
  // Do Something  
};
```

### *loaded*

Fires after loaded the sparkline.

#### **HTML**

```
<div id="line"></div>
```

#### **JAVASCRIPT**

```
ReactDOM.render(  
  <EJ.Sparkline id="default" loaded = {Loaded}></EJ.Sparkline>,  
  document.getElementById('line')  
)  
;  
function Loaded() {  
  // Do Something  
};
```

### *tooltipInitialize*

Fires before rendering trackball tooltip. You can use this event to customize the text displayed in trackball tooltip.

#### **HTML**

```
<div id="line"></div>
```

#### **JAVASCRIPT**

```
ReactDOM.render(  
  <EJ.Sparkline id="default" tooltipInitialize  
    ={TooltipInitialize}></EJ.Sparkline>,  
  document.getElementById('line')  
)  
;  
function TooltipInitialize() {  
  // Do Something  
};
```

```
};
```

### *seriesRendering*

Fires before rendering a series. This event is fired for each series in Sparkline.

#### **HTML**

```
<div id="line"></div>
```

#### **JAVASCRIPT**

```
ReactDOM.render(
  <EJ.Sparkline id="default" seriesRendering =
    {SeriesRendering}></EJ.Sparkline>,
  document.getElementById('line')
);
function SeriesRendering() {
  // Do Something
};
```

### *pointRegionMouseMove*

Fires when mouse is moved over a point.

#### **HTML**

```
<div id="line"></div>
```

#### **JAVASCRIPT**

```
ReactDOM.render(
  <EJ.Sparkline id="default" pointRegionMouseMove =
    {PointRegionMouseMove}></EJ.Sparkline>,
  document.getElementById('line')
);
function PointRegionMouseMove() {
  // Do Something
};
```

### *pointRegionMouseClick*

Fires on clicking a point in sparkline. You can use this event to handle clicks made on points.

#### **HTML**

```
<div id="line"></div>
```

#### **JAVASCRIPT**

```
ReactDOM.render(
  <EJ.Sparkline id="default" pointRegionMouseClick =
    {PointRegionMouseClick}></EJ.Sparkline>,
  document.getElementById('line')
);
function PointRegionMouseClick() {
  // Do Something
};
```

```
};
```

### [sparklineMouseMove](#)

Fires on moving mouse over the sparkline.

#### HTML

```
<div id="line"></div>
```

#### JAVASCRIPT

```
ReactDOM.render(
  <EJ.Sparkline id="default" sparklineMouseMove =
    {SparklineMouseMove}></EJ.Sparkline>,
  document.getElementById('line')
);
function SparklineMouseMove() {
  // Do Something
};
```

### [sparklineMouseLeave](#)

Fires on moving mouse outside the sparkline.

#### HTML

```
<div id="line"></div>
```

#### JAVASCRIPT

```
ReactDOM.render(
  <EJ.Sparkline id="default" sparklineMouseLeave =
    {SparklineMouseLeave}></EJ.Sparkline>,
  document.getElementById('line')
);
function SparklineMouseLeave() {
  // Do Something
};
```

## SpellCheck

### Overview

The **Essential ReactJS SpellCheck** control helps to find and highlight misspelled words from a given content, typically by comparing it with a stored list of dictionary words. Almost all the features in JS SpellChecker are applicable to SpellChecker in ReactJS too.

#### Key Features

Some of the key features of SpellCheck are as follows,

- **Custom Dictionary** - Custom dictionary file is used to store and maintain your custom words like technical terms, brand names etc. *Ignore Settings - Ignore words composed of alphanumeric*

*characters, mixed case words, uppercase words, URL links, and file path.* **Ignore Words** - Ignore a specific collection of words from the error word consideration.

- **Customize Menu Items** - Adding the user defined menu items in the context menu to correct the error words. *Misspell Word Style* - Highlight the erroneous word based on the user-customized style. **Maximum Suggestion Count** - Display a number of suggestions based on the user-specified count.

## Getting Started

You can create a React application and add necessary scripts and styles with the help of the given [React Getting Started Documentation](#).

## Control Initialization

Define an HTML element for adding SpellCheck in the application and refer the JS file.

### HTML

```
<div id="spellcheck-default"></div>
<script src="app/spellcheck/default.js"></script>
```

Create a JSX file for rendering SpellCheck component using <EJ.SpellCheck> syntax. Add required properties to it in <EJ. SpellCheck> tag element.

### HTML

```
"use strict";
var DefaultDialog = React.createClass({
  render: function () {
    return (
      <div id="dialog_default">
        <EJ.SpellCheck id="SpellCheck1" dictionarySettings-
          dictionaryUrl="http://js.syncfusion.com/demos/ejservices/api/SpellCheck/CheckWords" dictionarySettings-
          customDictionaryUrl="http://js.syncfusion.com/demos/ejservices/api/SpellCheck/AddToDictionary" contextMenuSettings-enable={false}>
          Facebook is a social networking service headquartered in Menlo Park,
          California. Its website was launched on February 4, 2004, by Mark Zuckerberg
          with his Harvard College roommates and fellow students Eduardo, Andrew
          McCollum, Dustin and Chris Hughes.
          The fouders had initially limited the websites membrship to Harvard
          students, but later expandedit to collges in the Boston area, the Ivy
          League, and Stanford Univrsity. It graually added support for students at
          various other universities and later to high-school students.
        </EJ.SpellCheck>
      </div>
    );
  }
});
ReactDOM.render(<DefaultDialog />, document.getElementById('spellcheck-default'));
```

**Note:** The above jsx template needs to be converted from .jsx to .js extension by using gulp nuget package (refer [here](#)) and then it must be referred in the html page.

## Spell Checking

To spell check the content of the target element, you need to add one button and calling any one of the SpellCheck method showInDialog or validate by clicking the button to highlight the error words.

The following code example depicts that checking the spelling of the target element through the "validate" method.

### HTML

```
"use strict";
var SpellMenu = React.createClass({
  checkInTarget: function (e) {
    var obj = $("#SpellCheck2").data("ejSpellCheck");
    obj.validate();
  },
  render: function () {
    return (
      <div id="spell_menu">
        <EJ.SpellCheck id="SpellCheck2" dictionarySettings-
        dictionaryUrl="http://js.syncfusion.com/demos/ejservices/api/SpellCheck/Chec
        kWords" dictionarySettings-
        customDictionaryUrl="http://js.syncfusion.com/demos/ejservices/api/SpellChec
        k/AddToDictionary">
          It is a concept vehicle with Liuid Silver body colour, 20-inch wheels,
          fabric foding roof, electrically-controlled hood, 4-cylinder 2.0 TDI engine
          rated 204 PS (150 kW; 201 hp)
          and 400 (295.02 lbf ft), diesel particulate filter and Bluetec emission
          control system, quattro permanent four-wheel drive system,
          Audi S tronic dual-clutch gearbox, McPherson-strut front axle and a four-
          link rear axle, Audi drive select system with 3 modes (dynamic, sport,
          efficiency),
          MMI control panel with touch pad and dual-view technology, sound system with
          the proinent extending tweeters.
        </EJ.SpellCheck>
        <EJ.Button id="btnCheck" type="button" height={30} text="Spell Check"
        click={this.checkInTarget}>
        </EJ.Button>
      </div>
    );
  }
});
ReactDOM.render(<SpellMenu />, document.getElementById('contextmenu'));
```

## SpellCheck Operations

Essential SpellCheck provides two ways to perform the spellcheck operation(error correction). They are,

- Dialog Mode
- Context Menu Mode

### Dialog Mode

#### Description

SpellCheck provides the dialog mode option to perform the following spellcheck operations.

- Ignore Once
- Ignore All
- Change
- Change All
- Add to Dictionary

### Open Dialog Mode

The following code snippet shows how to open the dialog to spell check the content.

#### HTML

```
"use strict";
var DefaultDialog = React.createClass({
  checkErrors: function (e) {
    var spellObj = $("#SpellCheck1").data("ejSpellCheck");
    spellObj.showInDialog();
  },
  render: function () {
    return (
      <div id="dialog_default">
        <EJ.SpellCheck id="SpellCheck1" dictionarySettings-
          dictionaryUrl="http://js.syncfusion.com/demos/ejservices/api/SpellCheck/CheckWords" dictionarySettings-
          customDictionaryUrl="http://js.syncfusion.com/demos/ejservices/api/SpellCheck/AddToDictionary" contextMenuSettings-enable={false}>
          Facebook is a social networking service headquartered in Menlo Park, California. Its website was launched on February 4, 2004, by Mark Zuckerberg with his Harvard College roommates and fellow students Eduardo, Andrew McCollum, Dustin and Chris Hughes. The fouders had initially limited the websites membrship to Harvard students, but later expanded it to collges in the Boston area, the Ivy League, and Stanford Univrsity. It graually added support for students at various other universities and later to high-school students.
        </EJ.SpellCheck>,
        <EJ.Button id="btnCheck" type="button" height={30} text="Spell Check" click={this.checkErrors}>
        </EJ.Button>
      </div>
    );
  }
});
ReactDOM.render(<DefaultDialog />, document.getElementById('spellcheck-default'));
```

### Handling Dialog Actions

To define the specific actions before the dialog window open, the client-side event `dialogBeforeOpen` can be used as depicted in the below code example.

#### HTML

```
"use strict";
var DefaultDialog = React.createClass({
  checkErrors: function (e) {
    var spellObj = $("#SpellCheck1").data("ejSpellCheck");
    spellObj.showInDialog();
  },
  render: function () {
    return (
      <div id="dialog_default">
        <EJ.SpellCheck id="SpellCheck1" dictionarySettings-
          dictionaryUrl="http://js.syncfusion.com/demos/ejservices/api/SpellCheck/CheckWords" dictionarySettings-
          customDictionaryUrl="http://js.syncfusion.com/demos/ejservices/api/SpellCheck/AddToDictionary" contextMenuSettings-enable={false}>
          Facebook is a social networking service headquartered in Menlo Park, California. Its website was launched on February 4, 2004, by Mark Zuckerberg with his Harvard College roommates and fellow students Eduardo, Andrew McCollum, Dustin and Chris Hughes. The fouders had initially limited the websites membrship to Harvard students, but later expanded it to collges in the Boston area, the Ivy League, and Stanford Univrsity. It graually added support for students at various other universities and later to high-school students.
        </EJ.SpellCheck>,
        <EJ.Button id="btnCheck" type="button" height={30} text="Spell Check" click={this.checkErrors}>
        </EJ.Button>
      </div>
    );
  }
});
ReactDOM.render(<DefaultDialog />, document.getElementById('spellcheck-default'));
```

```

},
onDialogBeforeOpen:function(e) {
if (e.requestType == "dialogBeforeOpen") {
alert("dialog before open event triggered");
}
},
render: function () {
return (
<div id="dialog_default">
<EJ.SpellCheck id="SpellCheck1" dictionarySettings-
dictionaryUrl="http://js.syncfusion.com/demos/ejservices/api/SpellCheck/CheckWords" dictionarySettings-
customDictionaryUrl="http://js.syncfusion.com/demos/ejservices/api/SpellCheck/AddToDictionary" contextMenuSettings-enable={false}
dialogBeforeOpen={this.onDialogBeforeOpen} >
Facebook is a social networking service headquartered in Menlo Park,
California. Its website was launched on February 4, 2004, by Mark Zuckerberg
with his Harvard College roommates and fellow students Eduardo, Andrew
McCollum, Dustin and Chris Hughes.
The founders had initially limited the websites membership to Harvard
students, but later expanded it to colleges in the Boston area, the Ivy
League, and Stanford University. It gradually added support for students at
various other universities and later to high-school students.
</EJ.SpellCheck>,
<EJ.Button id="btnCheck" type="button" height={30} text="Spell Check"
click={this.checkErrors}>
</EJ.Button>
</div>
);
}
});
ReactDOM.render(<DefaultDialog />, document.getElementById('spellcheck-
default'));

```

The client-side event `dialogOpen` can be used to define the specific actions after the dialog window open as shown in the following code example.

### HTML

```

"use strict";
var DefaultDialog = React.createClass({
checkErrors: function (e) {
var spellObj = $("#SpellCheck1").data("ejSpellCheck");
spellObj.showInDialog();
},
onDialogOpen:function(e) {
if (e.requestType == "dialogOpen") {
alert(e.targetText);
}
},
render: function () {
return (
<div id="dialog_default">
<EJ.SpellCheck id="SpellCheck1" dictionarySettings-
dictionaryUrl="http://js.syncfusion.com/demos/ejservices/api/SpellCheck/CheckWords" dictionarySettings-

```



```

customDictionaryUrl="http://js.syncfusion.com/demos/ejservices/api/SpellCheck/
k/AddToDictionary" contextMenuSettings-enable={false}
dialogOpen={this.onDialogOpen} >
Facebook is a social networking service headquartered in Menlo Park,
California. Its website was launched on February 4, 2004, by Mark Zuckerberg
with his Harvard College roommates and fellow students Eduardo, Andrew
McCollum, Dustin and Chris Hughes.
The fouders had initially limited the websites membrship to Harvard
students, but later expanded it to collges in the Boston area, the Ivy
League, and Stanford Univrsity. It graually added support for students at
various other universities and later to high-school students.
</EJ.SpellCheck>,
<EJ.Button id="btnCheck" type="button" height={30} text="Spell Check"
click={this.checkErrors}>
</EJ.Button>
</div>
);
}
});
ReactDOM.render(<DefaultDialog />, document.getElementById('spellcheck-
default'));

```

The following code example used to define some actions after the dialog closing by using the client-side event `dialogClose`.

#### HTML

```

"use strict";
var DefaultDialog = React.createClass({
  checkErrors: function (e) {
    var spellObj = $("#SpellCheck1").data("ejSpellCheck");
    spellObj.showInDialog();
  },
  onDialogClose: function (e) {
    if (e.requestType == "dialogClose") {
      alert(e.updatedText);
    }
  },
  render: function () {
    return (
      <div id="dialog_default">
        <EJ.SpellCheck id="SpellCheck1" dictionarySettings-
        dictionaryUrl="http://js.syncfusion.com/demos/ejservices/api/SpellCheck/Chec
        kWords" dictionarySettings-
        customDictionaryUrl="http://js.syncfusion.com/demos/ejservices/api/SpellChec
        k/AddToDictionary" contextMenuSettings-enable={false}
        dialogClose={this.onDialogClose} >
        Facebook is a social networking service headquartered in Menlo Park,
        California. Its website was launched on February 4, 2004, by Mark Zuckerberg
        with his Harvard College roommates and fellow students Eduardo, Andrew
        McCollum, Dustin and Chris Hughes.
        The fouders had initially limited the websites membrship to Harvard
        students, but later expanded it to collges in the Boston area, the Ivy
        League, and Stanford Univrsity. It graually added support for students at
        various other universities and later to high-school students.
        </EJ.SpellCheck>,

```

```

<EJ.Button id="btnCheck" type="button" height={30} text="Spell Check"
click={this.checkErrors}>
</EJ.Button>
</div>
);
}
});
ReactDOM.render(<DefaultDialog />, document.getElementById('spellcheck-
default'));

```

It is possible to predict the error word details before starting the spellcheck operations through dialog mode by using the client side event `start`. The below code example describes the above behavior.

### HTML

```

"use strict";
var DefaultDialog = React.createClass({
  checkErrors: function (e) {
    var spellObj = $("#SpellCheck1").data("ejSpellCheck");
    spellObj.showInDialog();
  },
  onSpellCheckStart: function (e) {
    if (e.requestType == "spellCheckDialog") {
      alert(JSON.stringify(args.errorWords));
    }
  },
  render: function () {
    return (
      <div id="dialog_default">
        <EJ.SpellCheck id="SpellCheck1" dictionarySettings-
        dictionaryUrl="http://js.syncfusion.com/demos/ejservices/api/SpellCheck/Chec
        kWords" dictionarySettings-
        customDictionaryUrl="http://js.syncfusion.com/demos/ejservices/api/SpellChec
        k/AddToDictionary" contextMenuSettings-enable={false}
        start={this.onSpellCheckStart} >
        Facebook is a social networking service headquartered in Menlo Park,
        California. Its website was launched on February 4, 2004, by Mark Zuckerberg
        with his Harvard College roommates and fellow students Eduardo, Andrew
        McCollum, Dustin and Chris Hughes.
        The fouders had initially limited the websites membrship to Harvard
        students, but later expandedit to collges in the Boston area, the Ivy
        League, and Stanford Univrsity. It graually added support for students at
        various other universities and later to high-school students.
        </EJ.SpellCheck>,
        <EJ.Button id="btnCheck" type="button" height={30} text="Spell Check"
        click={this.checkErrors}>
        </EJ.Button>
      </div>
    );
  }
});
ReactDOM.render(<DefaultDialog />, document.getElementById('spellcheck-
default'));

```

You can get the corrected text content details before updating it into target element with the help of the client side event **complete** as mentioned in the below code example.

### HTML

```
"use strict";
var DefaultDialog = React.createClass({
  checkErrors: function (e) {
    var spellObj = $("#SpellCheck1").data("ejSpellCheck");
    spellObj.showInDialog();
  },
  onSpellCheckComplete: function (e) {
    if (args.requestType == "changeErrorWord") {
      alert(args.targetText);
    }
  },
  render: function () {
    return (
      <div id="dialog_default">
        <EJ.SpellCheck id="SpellCheck1" dictionarySettings-
        dictionaryUrl="http://js.syncfusion.com/demos/ejservices/api/SpellCheck/CheckWords" dictionarySettings-
        customDictionaryUrl="http://js.syncfusion.com/demos/ejservices/api/SpellCheck/AddToDictionary" contextMenuSettings-enable={false}
        start={this.onSpellCheckComplete} >
          Facebook is a social networking service headquartered in Menlo Park,
          California. Its website was launched on February 4, 2004, by Mark Zuckerberg
          with his Harvard College roommates and fellow students Eduardo, Andrew
          McCollum, Dustin and Chris Hughes.
          The founders had initially limited the website's membership to Harvard
          students, but later expanded it to colleges in the Boston area, the Ivy
          League, and Stanford University. It gradually added support for students at
          various other universities and later to high-school students.
        </EJ.SpellCheck>,
        <EJ.Button id="btnCheck" type="button" height={30} text="Spell Check"
        click={this.checkErrors}>
        </EJ.Button>
      </div>
    );
  }
});
ReactDOM.render(<DefaultDialog />, document.getElementById('spellcheck-default'));
```

### Context Menu Mode

SpellCheck provides default context menu options to perform the spellcheck operations. It also allows to define additional custom context menu options.

The options that are available under **contextMenuSettings** are as follows,

- **enable** - Enables/disables the context menu option in SpellCheck.
- **menuItems** - Contains the options to perform spellcheck operations.

### Default Menu Options

The menu items contains the following options to perform the spellcheck operation.

- Ignore All
- Add to Dictionary

The following code snippet shows how to enable the context menu settings in SpellCheck and to make use of it with default available options.

### HTML

```

"use strict";
var SpellMenu = React.createClass({
  checkInTarget: function (e) {
    var obj = $("#SpellCheck2").data("ejSpellCheck");
    obj.validate();
  },
  render: function () {
    return (
      <div id="spell_menu">
        <EJ.SpellCheck id="SpellCheck2" dictionarySettings-
        dictionaryUrl="http://js.syncfusion.com/demos/ejservices/api/SpellCheck/CheckWords" dictionarySettings-
        customDictionaryUrl="http://js.syncfusion.com/demos/ejservices/api/SpellCheck/AddToDictionary" contextMenuSettings-enable={true} contextMenuSettings-
        menuItems=[
          { id: "IgnoreAll", text: "Ignore All" },
          { id: "AddToDictionary", text: "Add To Dictionary" } ]>
        It is a concept vehicle with Liquid Silver body colour, 20-inch wheels,
        fabric foding roof, electrically-controlled hood, 4-cylinder 2.0 TDI engine
        rated 204 PS (150 kW; 201 hp)
        and 400 (295.02 lbf ft), diesel particulate filter and Bluetec emission
        control system, quattro permanent four-wheel drive system,
        Audi S tronic dual-clutch gearbox, McPherson-strut front axle and a four-
        link rear axle, Audi drive select system with 3 modes (dynamic, sport,
        efficiency),
        MMI control panel with touch pad and dual-view technology, sound system with
        the proinent extending tweeters.
      </EJ.SpellCheck>
      <EJ.Button id="btnCheck" type="button" height={30} text="Spell Check"
      click={this.checkInTarget}>
    </EJ.Button>
    </div>
  );
}
});
ReactDOM.render(<SpellMenu />, document.getElementById('contextmenu'));

```

**Note:** For default menu items, the id must be defined the same as mentioned in the above code example – as we have processed the menus based on this id within our source.

### Custom Menu Options

Apart from the default available options, it is also possible to add custom menu options to the context-menu list.

The following code example depicts how to **add the custom menu items** into the context menu settings.

### HTML

```
"use strict";
var SpellMenu = React.createClass({
  checkInTarget: function (e) {
    var obj = $("#SpellCheck2").data("ejSpellCheck");
    obj.validate();
  },
  render: function () {
    return (
      <div id="spell_menu">
        <EJ.SpellCheck id="SpellCheck2" dictionarySettings-
        dictionaryUrl="http://js.syncfusion.com/demos/ejservices/api/SpellCheck/CheckWords" dictionarySettings-
        customDictionaryUrl="http://js.syncfusion.com/demos/ejservices/api/SpellCheck/AddToDictionary" contextMenuSettings-enable={true} contextMenuSettings-
        menuItems=[
          { id: "Ignore", text: "IgnoreOnce" },
          { id: "IgnoreAll", text: "Ignore All" },
          { id: "AddToDictionary", text: "Add To Dictionary" } ]>
        It is a concept vehicle with Liquid Silver body colour, 20-inch wheels,
        fabric foding roof, electrically-controlled hood, 4-cylinder 2.0 TDI engine
        rated 204 PS (150 kW; 201 hp)
        and 400 (295.02 lbf ft), diesel particulate filter and Bluetec emission
        control system, quattro permanent four-wheel drive system,
        Audi S tronic dual-clutch gearbox, McPherson-strut front axle and a four-
        link rear axle, Audi drive select system with 3 modes (dynamic, sport,
        efficiency),
        MMI control panel with touch pad and dual-view technology, sound system with
        the proinent extending tweeters.
      </EJ.SpellCheck>
      <EJ.Button id="btnCheck" type="button" height={30} text="Spell Check"
      click={this.checkInTarget}>
    </EJ.Button>
    </div>
  );
}
});
ReactDOM.render(<SpellMenu />, document.getElementById('contextmenu'));
```

**Note:** The **id** given for the custom menu items **must be unique**.

### Handling Menu Actions

The client-side event `contextClick` can be used to define specific actions when a click made on the custom menu items. The following code example describes the above behavior.

### HTML

```
"use strict";
var SpellMenu = React.createClass({
  checkInTarget: function (e) {
    var obj = $("#SpellCheck2").data("ejSpellCheck");
    obj.validate();
  },
  onCustomMenuClick: function (e) {debugger;
```

```

if (e.selectedOption == "Ignore") {
  alert("Custom menu clicked");
},
render: function () {
  return (
    <div id="spell_menu">
      <EJ.SpellCheck id="SpellCheck2" dictionarySettings-
dictionaryUrl="http://js.syncfusion.com/demos/ejservices/api/SpellCheck/CheckWords" dictionarySettings-
customDictionaryUrl="http://js.syncfusion.com/demos/ejservices/api/SpellCheck/AddToDictionary" contextMenuSettings-enable={true} contextMenuSettings-
menuItems=[
  { id: "Ignore", text: "IgnoreOnce" },
  { id: "IgnoreAll", text: "Ignore All" },
  { id: "AddToDictionary", text: "Add To Dictionary" }
] contextClick={this.onCustomMenuClick}>
    It is a concept vehicle with Liquid Silver body colour, 20-inch wheels,
    fabric foding roof, electrically-controlled hood, 4-cylinder 2.0 TDI engine
    rated 204 PS (150 kW; 201 hp)
    and 400 (295.02 lbf ft), diesel particulate filter and Bluetec emission
    control system, quattro permanent four-wheel drive system,
    Audi S tronic dual-clutch gearbox, McPherson-strut front axle and a four-
    link rear axle, Audi drive select system with 3 modes (dynamic, sport,
    efficiency),
    MMI control panel with touch pad and dual-view technology, sound system with
    the proinent extending tweeters.
  </EJ.SpellCheck>
  <EJ.Button id="btnCheck" type="button" height={30} text="Spell Check"
  click={this.checkInTarget}>
  </EJ.Button>
</div>
);
}
});
ReactDOM.render(<SpellMenu />, document.getElementById('contextmenu'));

```

It is possible to predict the target (error word) on which the right click is made with the use of the event `contextOpen`. The below code example shows how to block the display of context menu, when right clicked on the word by setting `args.cancel` as `true`.

### HTML

```

"use strict";
var SpellMenu = React.createClass({
  checkInTarget: function (e) {
    var obj = $("#SpellCheck2").data("ejSpellCheck");
    obj.validate();
  },
  onMenuOpen: function (e) {
    if (e.selectedErrorWord == "Bluetec") {
      e.cancel = true;
    }
  },
  render: function () {
    return (

```

```

<div id="spell_menu">
<EJ.SpellCheck id="SpellCheck2" dictionarySettings-
dictionaryUrl="http://js.syncfusion.com/demos/ejservices/api/SpellCheck/Chec
kWords" dictionarySettings-
customDictionaryUrl="http://js.syncfusion.com/demos/ejservices/api/SpellChec
k/AddToDictionary" contextMenuSettings-enable={true}
contextOpen={this.onMenuOpen}>
It is a concept vehicle with Liuid Silver body colour, 20-inch wheels,
fabric foding roof, electrically-controlled hood, 4-cylinder 2.0 TDI engine
rated 204 PS (150 kW; 201 hp)
and 400 (295.02 lbf ft), diesel particulate filter and Bluetec emission
control system, quattro permanent four-wheel drive system,
Audi S tronic dual-clutch gearbox, McPherson-strut front axle and a four-
link rear axle, Audi drive select system with 3 modes (dynamic, sport,
efficiency),
MMI control panel with touch pad and dual-view technology, sound system with
the proinent extending tweeters.
</EJ.SpellCheck>
<EJ.Button id="btnCheck" type="button" height={30} text="Spell Check"
click={this.checkInTarget}>
</EJ.Button>
</div>
);
}
});
ReactDOM.render(<SpellMenu />, document.getElementById('contextmenu'));

```

You can get the current spell check operation arguments details with the client-side event `validating`. The following code example describes the way to block the ignoreAll operation for the particular word.

### HTML

```

"use strict";
var SpellMenu = React.createClass({
  checkInTarget: function (e) {
    var obj = $("#SpellCheck2").data("ejSpellCheck");
    obj.validate();
  },
  onSpellChecking: function (e) {
    if (e.requestType == "ignoreAll" && e.ignoreWord == "Bluetec") {
      e.cancel = true;
    }
  },
  render: function () {
    return (
      <div id="spell_menu">
      <EJ.SpellCheck id="SpellCheck2" dictionarySettings-
      dictionaryUrl="http://js.syncfusion.com/demos/ejservices/api/SpellCheck/Chec
      kWords" dictionarySettings-
      customDictionaryUrl="http://js.syncfusion.com/demos/ejservices/api/SpellChec
      k/AddToDictionary" contextMenuSettings-enable={true}
      validating={this.onSpellChecking}>
      It is a concept vehicle with Liuid Silver body colour, 20-inch wheels,
      fabric foding roof, electrically-controlled hood, 4-cylinder 2.0 TDI engine
      rated 204 PS (150 kW; 201 hp)
    
```

```

and 400 (295.02 lbf ft), diesel particulate filter and Bluetec emission
control system, quattro permanent four-wheel drive system,
Audi S tronic dual-clutch gearbox, McPherson-strut front axle and a four-
link rear axle, Audi drive select system with 3 modes (dynamic, sport,
efficiency),
MMI control panel with touch pad and dual-view technology, sound system with
the proinent extending tweeters.
</EJ.SpellCheck>
<EJ.Button id="btnCheck" type="button" height={30} text="Spell Check"
click={this.checkInTarget}>
</EJ.Button>
</div>
);
}
});
ReactDOM.render(<SpellMenu />, document.getElementById('contextmenu'));

```

## Functionalities

### Check Spelling

The main functionality of the SpellCheck control is checking the spelling of a word and returns the suggestions for an error word.

For example, if you pass the sentence that contains misspell words as input, you can know the error words in this sentence and its suggestions (apt words to replace).

### Ignore Words

The SpellCheck control provides the support to ignore the words from an error word consideration and also to ignore an error words whenever needed. Please refer the following options to ignore the words.

#### Ignore

The **ignore** option is used to ignore a specific word once from the given input string.

The following code example describes the above method implementation.

### HTML

```

"use strict";
var DefaultDialog = React.createClass({
  componentDidMount:function() {
    var targetSentence = 'The <span class="errors" e-
errorword">textarea</span> sample uses a dialog to display the spell
errors.';
    var schObj = $("#SpellCheck1").data("ejSpellCheck");
    schObj.ignore("textarea",targetSentence, null);
  },
  render: function () {
    return (
      <div id="dialog_default">
        <EJ.SpellCheck id="SpellCheck1" dictionarySettings-
dictionaryUrl="http://js.syncfusion.com/demos/ejservices/api/SpellCheck/Chec
kWords" dictionarySettings-
customDictionaryUrl="http://js.syncfusion.com/demos/ejservices/api/SpellChec
k/AddToDictionary" contextMenuSettings-enable={false}>
        </EJ.SpellCheck>
      </div>
    );
  }
});

```



```

}
});
ReactDOM.render(<DefaultDialog />, document.getElementById('spellcheck-
default'));

```

### Ignore All

The **ignore all** option is used to ignore all the error word occurrences from the given input string.

The following code example describes the way of using ignore all method.

### HTML

```

"use strict";
var DefaultDialog = React.createClass({
  componentDidMount:function() {
    var targetSentence = 'This <span class="errors" e-
errorword">textarea</span> sample uses a dialog to display all the <span
class="errors" e-errorword">textarea</span> spell errors.';
    var schObj = $("#SpellCheck1").data("ejSpellCheck");
    schObj.ignoreAll("textarea",targetSentence, null);
  },
  render: function () {
    return (
      <div id="dialog_default">
        <EJ.SpellCheck id="SpellCheck1" dictionarySettings-
dictionaryUrl="http://js.syncfusion.com/demos/ejservices/api/SpellCheck/Chec
kWords" dictionarySettings-
customDictionaryUrl="http://js.syncfusion.com/demos/ejservices/api/SpellChec
k/AddToDictionary" contextMenuSettings-enable={false}>
        </EJ.SpellCheck>
      </div>
    );
  }
});
ReactDOM.render(<DefaultDialog />, document.getElementById('spellcheck-
default'));

```

### Word Collection to Ignore

The **ignoreWords** option is used to ignore the collection of words from an error word checking. You can pass the technical terms, brand names which is not present in the dictionary file to this property "ignoreWords" as shown in the following code example and then while checking the spelling these passed words are considered as a correct word.

### HTML

```

"use strict";
var DefaultDialog = React.createClass({
  render: function () {
    return (
      <div id="dialog_default">
        <EJ.SpellCheck id="SpellCheck1" dictionarySettings-
dictionaryUrl="http://js.syncfusion.com/demos/ejservices/api/SpellCheck/Chec
kWords" dictionarySettings-
customDictionaryUrl="http://js.syncfusion.com/demos/ejservices/api/SpellChec

```

```
k/AddToDictionary" contextMenuSettings-enable={true}
ignoreWords={["fouders","graually"]}>
Facebook is a social networking service headquartered in Menlo Park,
California. Its website was launched on February 4, 2004, by Mark Zuckerberg
with his Harvard College roommates and fellow students Eduardo, Andrew
McCollum, Dustin and Chris Hughes.
The fouders had initially limited the websites membrship to Harvard
students, but later expanded it to collges in the Boston area, the Ivy
League, and Stanford Univrsity. It graually added support for students at
various other universities and later to high-school students.
</EJ.SpellCheck>
</div>
);
}
});
ReactDOM.render(<DefaultDialog />, document.getElementById('spellcheck-
default'));
```

### Ignore Settings

The **ignore settings** helps to ignore the uppercase, mixed case words, alphanumeric words, file path and email addresses from the error word checking. Ignore settings contains the following options to ignore the words based on their category.

- **ignoreAlphaNumericWords** - ignoring the alpha numeric words from an error word consideration.
- **ignoreUpperCase** - ignoring the upper case words from an error word consideration.
- **ignoreMixedCaseWords** - ignoring the mixed case words from an error word consideration.
- **ignoreFileNames** - ignoring the file address path from an error word consideration.
- **ignoreUrl** - ignoring the url links from an error word consideration.
- **ignoreEmailAddress** - ignoring the email address from an error word consideration.

The following code example uses to enable the checking of all the words formed with alphanumeric, uppercase, mixed case words and file paths and email addresses.

### HTML

```
"use strict";
var DefaultDialog = React.createClass({
  render: function () {
    return (
      <div id="dialog_default">
        <EJ.SpellCheck id="SpellCheck1" dictionarySettings-
        dictionaryUrl="http://js.syncfusion.com/demos/ejservices/api/SpellCheck/Chec
        kWords" dictionarySettings-
        customDictionaryUrl="http://js.syncfusion.com/demos/ejservices/api/SpellChec
        k/AddToDictionary" contextMenuSettings-enable={true} ignoreSettings-
        ignoreAlphaNumericWords={false} ignoreSettings-ignoreMixedCaseWords={false}
        ignoreSettings-ignoreUpperCase={false} ignoreSettings-ignoreUrl={false}
        ignoreSettings-ignoreEmailAddress={false} ignoreSettings-
        ignoreFileNames={false}>
        Amazon.com, Inc., often refered to as simply Amazon, is an American
        electronic commerce and cloud computing company with headquarters in
        Seattle, Washington. It is the largest web-based retailer in the United
```

```

States. Amazon.com started as bookstore on the web, later diversifying to
sell DVDs, Blu-rays, CDs, video downloads/streaming, MP3
downloads/streaming, audiobook downloads/streaming, software, video games,
electronics, apparel, furniture, food, toys and jewelry. The company also
produces consumer electronics notably, Amazon Kindle e-book readers, Fire
tablets, Fire TV and Fire Phone and is the world's largest provider of cloud
infrastructure services (IaaS). Amazon also sells certain low-end products
like USB cables under its in-house brand AmazonBasics.
Amazon has separate retail websites for United States, United Kingdom and
Ireland, France, Canada, Germany, Italy, Spain, Netherlands, Australia,
Brazil, Japan, China, India and Mexico. Amazon also offers international
shipping to certain other countries for some of its products.
</EJ.SpellCheck>
</div>
);
}
});
ReactDOM.render(<DefaultDialog />, document.getElementById('spellcheck-
default'));

```

### Change Words

The SpellCheck control provides the support to change an error words from its possible suggestions. Please refer the following options to change an error word.

#### Change

The **change** option is used to replace an error word occurrences once from the given input string with the correct word.

The following code example describes the behavior of change method.

#### HTML

```

"use strict";
var DefaultDialog = React.createClass({
  componentDidMount:function(){
    var targetSentence = 'The <span class="errors" e-
    errorword">textarea</span> sample uses a dialog to display the spell
    errors.';
    var schObj = $("#SpellCheck1").data("ejSpellCheck");
    schObj.change("textarea", targetSentence, null);
  },
  render: function () {
    return (
      <div id="dialog_default">
        <EJ.SpellCheck id="SpellCheck1" dictionarySettings-
        dictionaryUrl="http://js.syncfusion.com/demos/ejservices/api/SpellCheck/Chec
        kWords" dictionarySettings-
        customDictionaryUrl="http://js.syncfusion.com/demos/ejservices/api/SpellChec
        k/AddToDictionary">
        </EJ.SpellCheck>
      </div>
    );
  }
});
ReactDOM.render(<DefaultDialog />, document.getElementById('spellcheck-
default'));

```

### Change All

The **change all** option is used to replace all the occurrences of an error word with the correct word(selected from the suggestions list) from the given inputs string.

The following code example uses to change all the error word occurrences.

#### HTML

```
"use strict";
var DefaultDialog = React.createClass({
  componentDidMount:function(){
    var targetSentence = 'This 

```

### Custom Words

The SpellCheck control provides the support to add the custom words into the custom dictionary file.

The **addToDictionary** option is used to add the custom words into the custom dictionary file.

The following code example uses to add the custom word into the custom dictionary file.

#### HTML

```
"use strict";
var DefaultDialog = React.createClass({
  componentDidMount:function(){
    var schObj = $("#SpellCheck1").data("ejSpellCheck");
    schObj.addToDictionary("textarea");
  },
  render: function () {
    return (
      <div id="dialog_default">
        <EJ.SpellCheck id="SpellCheck1" dictionarySettings-
        dictionaryUrl="http://js.syncfusion.com/demos/ejservices/api/SpellCheck/Chec
        kWords" dictionarySettings-
      </div>
    );
  }
});
```

```

customDictionaryUrl="http://js.syncfusion.com/demos/ejservices/api/SpellCheck/
k/AddToDictionary">
</EJ.SpellCheck>
</div>
);
}
});
ReactDOM.render(<DefaultDialog />, document.getElementById('spellcheck-
default'));

```

You can also add the custom words into the custom dictionary file through the dialog mode or context menu mode add to dictionary option.

- Dialog Mode - Add To Dictionary button is available in the dialog window, while highlighting the error word in the given input string and clicking this button then the word will be adding into the custom dictionary file.
- Context Menu Mode - Add To Dictionary option is available while right click on the error word and selecting this option, the word will be adding into the custom dictionary file.

### Checking content on typing

SpellCheck control provides support for checking the content, on pressing the **Enter** and **Space** key. The cursor position will also be properly retained, while processing the SpellCheck operations. If you enable “enableValidateOnType” property, the spellcheck operation will be carried out on typing.

The following code example describes the above behavior.

### HTML

```

"use strict";
var DefaultDialog = React.createClass({
  render: function () {
    return (
      <div id="dialog_default">
        <EJ.SpellCheck id="SpellCheck1" dictionarySettings-
        dictionaryUrl="http://js.syncfusion.com/demos/ejservices/api/SpellCheck/Chec
        kWords" dictionarySettings-
        customDictionaryUrl="http://js.syncfusion.com/demos/ejservices/api/SpellChec
        k/AddToDictionary" contextMenuSettings-enable={true}
        enableValidateOnType={true}>
        </EJ.SpellCheck>
      </div>
    );
  }
});
ReactDOM.render(<DefaultDialog />, document.getElementById('spellcheck-
default'));

```

The following screenshot displays the output for the above code

It is a concept vehicle with Liquid Silver body colour, 20-inch wheels, fabric foding roof, electrically-controlled hood, 4-cylinder 2.0 TDI engine rated 204 PS (150 kW; 201 hp) and 400 (295.02 lb ft), diesel particulate filter and Bluetec emission control system, quattro permanent four-wheel drve system, Audi S tronic dual-clutch gearbox, McPherson-strut front axle and a four-link rear axle, Audi drive select system with 3 modes (dynamic, sport, efficiency), MMI control panel with touch pad and dual-view technology, sound system with the proinent extending tweete.

You can also validate the content within the IFrame element or IFrame element target text, by passing the IFrame element id or class name value to the `controlsToValidate` property.

Detailed information is given [here](#)

### Suggestion Words

The `getSuggestionWords` option is used to retrieve the possible suggestion words for an error word which is provided to correct that spelling.

The following code example describes the above behavior.

### HTML

```
"use strict";
var DefaultDialog = React.createClass({
  componentDidMount:function() {
    var schObj = $("#SpellCheck1").data("ejSpellCheck");
    schObj.getSuggestionWords("textarea");
    setTimeout(function () {
      alert(spellObj._suggestedWords);
    }, 800);
  },
  render: function () {
    return (
      <div id="dialog_default">
        <EJ.SpellCheck id="SpellCheck1" dictionarySettings-
        dictionaryUrl="http://js.syncfusion.com/demos/ejservices/api/SpellCheck/Chec
        kWords" dictionarySettings-
        customDictionaryUrl="http://js.syncfusion.com/demos/ejservices/api/SpellChec
        k/AddToDictionary">
        </EJ.SpellCheck>
      </div>
    );
  }
});
ReactDOM.render(<DefaultDialog />, document.getElementById('spellcheck-
default'));
```

**Note:** You can get the suggestion words after some time interval once this method is called. Since, ajax request processing takes place in the background.

### Synchronous request

On setting `enableAsync` option to false, enables the synchronous request to the server to perform SpellCheck operations.

The following code example describes the above behavior.

**HTML**

```
"use strict";
var DefaultDialog = React.createClass({
  render: function () {
    return (
      <div id="dialog_default">
        <EJ.SpellCheck id="SpellCheck1" dictionarySettings-
          dictionaryUrl="http://js.syncfusion.com/demos/ejservices/api/SpellCheck/Chec
            kWords" dictionarySettings-
            customDictionaryUrl="http://js.syncfusion.com/demos/ejservices/api/SpellChec
              k/AddToDictionary" enableAsync={false} ajaxDataType="json">
          </EJ.SpellCheck>
        </div>
      );
    }
  });
ReactDOM.render(<DefaultDialog />, document.getElementById('spellcheck-
  default'));
```

**Note:** You need to set the `ajaxDataType` value as `json` to retrieve the synchronous request result properly.

**SpellCheck Customization**

The SpellCheck provides option to customize for the following scenarios.

- Misspell Word Appearance
- Restrict Suggestion Count

**Misspell Word Appearance**

The SpellCheck control provide the support(`misspellWordCss`) to display the error word in user defined style. By default displaying the error words with the red underline.

The following code example depicts the way to customize the error word highlight (displaying error word with red color font and lightblue background).

**HTML**

```
"use strict";
var DefaultDialog = React.createClass({
  checkInDialog: function (e) {
    var obj = $("#SpellCheck1").data("ejSpellCheck");
    obj.showInDialog();
  },
  render: function () {
    return (
      <div id="dialog_default">
        <EJ.SpellCheck id="SpellCheck1" dictionarySettings-
          dictionaryUrl="http://js.syncfusion.com/demos/ejservices/api/SpellCheck/Chec
            kWords" dictionarySettings-
            customDictionaryUrl="http://js.syncfusion.com/demos/ejservices/api/SpellChec
              k/AddToDictionary" contextMenuSettings-enable={false}
              misspellWordCss="highlight">
          Facebook is a social networking service headquartered in Menlo Park,
          California. Its website was launched on February 4, 2004, by Mark Zuckerberg
        </EJ.SpellCheck>
      </div>
    );
  }
});
```

```

with his Harvard College roommates and fellow students Eduardo, Andrew
McCollum, Dustin and Chris Hughes.
The fouders had initially limited the websites membrship to Harvard
students, but later expanded it to collges in the Boston area, the Ivy
League, and Stanford Univrsity. It graually added support for students at
various other universities and later to high-school students.
</EJ.SpellCheck>
<EJ.Button id="btnCheck" type="button" height={30} text="Spell Check"
click={this.checkInDialog}>
</EJ.Button>
</div>
);
}
});
ReactDOM.render(<DefaultDialog />, document.getElementById('spellcheck-
default'));

```

Html file:

### HTML

```

<style>
.highlight {
background-color: lightblue;
color: red;
}
</style>

```

### Restrict Suggestion Count

The SpellCheck control provides option (`maxSuggestionCount`) to restrict the count that the number of items displayed in the suggestion list.

The following code example describes the way to control the suggestion count.

### HTML

```

"use strict";
var DefaultDialog = React.createClass({
  checkInDialog: function (e) {
    var obj = $("#SpellCheck1").data("ejSpellCheck");
    obj.showInDialog();
  },
  render: function () {
    return (
      <div id="dialog_default">
        <EJ.SpellCheck id="SpellCheck1" dictionarySettings-
dictionaryUrl="http://js.syncfusion.com/demos/ejservices/api/SpellCheck/Chec
kWords" dictionarySettings-
customDictionaryUrl="http://js.syncfusion.com/demos/ejservices/api/SpellChec
k/AddToDictionary" contextMenuSettings-enable={false}
maxSuggestionCount={3}>
        Facebook is a social networking service headquartered in Menlo Park,
        California. Its website was launched on February 4, 2004, by Mark Zuckerberg
        with his Harvard College roommates and fellow students Eduardo, Andrew
        McCollum, Dustin and Chris Hughes.
      </div>
    );
  }
});

```



```

The fouders had initially limited the websites membrship to Harvard
students, but later expanded it to collges in the Boston area, the Ivy
League, and Stanford Univrsity. It graually added support for students at
various other universities and later to high-school students.
</EJ.SpellCheck>
<EJ.Button id="btnCheck" type="button" height={30} text="Spell Check"
click={this.checkInDialog}>
</EJ.Button>
</div>
);
}
});
ReactDOM.render(<DefaultDialog />, document.getElementById('spellcheck-
default'));

```

## Localization

SpellCheck dialog mode comes with default localization support which allows it to customize the display of text within the SpellCheck dialog in a user-specific culture and locale. The SpellCheck control can be localized in specific culture using the common API `locale` along with the collection of localized words defined for that culture using the `ej.SpellCheck.Locale [culture-code]`.

By default, the SpellCheck control is localized in **en-US** culture. Please find the following table lists the default keywords and its localized text values for en-US culture.

Locale key words	Text
SpellCheckButtonText	Spelling:
NotInDictionary	Not in Dictionary:
SuggestionLabel	Suggestions:
IgnoreOnceButtonText	Ignore Once
IgnoreAllButtonText	Ignore All
AddToDictionary	Add to Dictionary
ChangeButtonText	Change
ChangeAllButtonText	Change All
CloseButtonText	Close
CompletionPopupMessage	Spell check is complete
CompletionPopupTitle	Spell check
Ok	OK
NoSuggestionMessage	No suggestions available
NotValidElement	Specify the valid control id or class name to spell check

To localize SpellCheck into any particular culture, it is necessary to refer the culture-specific script files in your application after the reference of **ej.web.all.min.js** file, which are available under the following location.

*<Installed location>\Syncfusion\Essential Studio\{{ site.releaseversion }}\JavaScript\assets\scripts\i18n*

The following code example shows how to localize the SpellCheck control in **fr-FR** culture.

### HTML

```
"use strict";
var DefaultDialog = React.createClass({
  checkInDialog: function (e) {
    var obj = $("#SpellCheck1").data("ejSpellCheck");
    obj.showInDialog();
  },
  componentWillMount: function (e) {
    ej.SpellCheck.Locales["fr-FR"] = {
      SpellCheckButtonText: "Orthographe",
      NotInDictionary: "Pas dans le dictionnaire",
      SuggestionLabel: "Suggestions",
      IgnoreOnceButtonText: "Ignorer une fois",
      IgnoreAllButtonText: "Ignorer tout",
      AddToDictionary: "Ajouter au dictionnaire",
      ChangeButtonText: "Changement",
      ChangeAllButtonText: "Tout modifier",
      CloseButtonText: "Fermer",
      CompletionPopupMessage: "Vérification orthographique est terminée",
      ErrorPopupMessage: "Vérification orthographique est pas terminée",
      CompletionPopupTitle: "Vérification orthographique alerte",
      OK: "D'accord",
      NoSuggestionMessage: "Aucune suggestion disponible",
    };
  },
  render: function () {
    return (
      <div id="dialog_default">
        <EJ.SpellCheck id="SpellCheck1" dictionarySettings-
        dictionaryUrl="http://js.syncfusion.com/demos/ejservices/api/SpellCheck/CheckWords" dictionarySettings-
        customDictionaryUrl="http://js.syncfusion.com/demos/ejservices/api/SpellCheck/AddToDictionary" contextMenuSettings-enable={false} locale="fr-FR">
        Facebook is a social networking service headquartered in Menlo Park,
        California. Its website was launched on February 4, 2004, by Mark Zuckerberg
        with his Harvard College roommates and fellow students Eduardo, Andrew
        McCollum, Dustin and Chris Hughes.
        The founders had initially limited the websites membership to Harvard
        students, but later expanded it to colleges in the Boston area, the Ivy
        League, and Stanford University. It gradually added support for students at
        various other universities and later to high-school students.
        </EJ.SpellCheck>
        <EJ.Button id="btnCheck" type="button" height={30} text="Spell Check"
        click={this.checkInDialog}>
        </EJ.Button>
      </div>
    );
  }
});
```

```
});
ReactDOM.render(<DefaultDialog />, document.getElementById('spellcheck-
default'));
```

**Note:** Refer the **ej.culture.fr-FR.min.js** file in your HTML application and also define the **locale** property for the SpellCheck control with the appropriate **culture-code [fr-FR]**.

For further information on how to refer the required culture scripts into your application, refer [here](#).

### Localizing Specific Words

To customize or localize only some specific words in the default **ej.SpellCheck.Locale["en-US"]** collection, the words to be localized/customized can be defined in a separate variable and then extended to the original collection as depicted in the following code example.

### HTML

```
"use strict";
var DefaultDialog = React.createClass({
  checkInDialog: function (e) {
    var obj = $("#SpellCheck1").data("ejSpellCheck");
    obj.showInDialog();
  },
  componentWillMount: function (e) {
    var customizationMessage = {
      CompletionPopupMessage: "Completed",
    };
    // Extend only the required changes to the original locale collection
    $.extend(ej.SpellCheck.Locale["en-US"], customizationMessage);
  },
  render: function () {
    return (
      <div id="dialog_default">
        <EJ.SpellCheck id="SpellCheck1" dictionarySettings-
        dictionaryUrl="http://js.syncfusion.com/demos/ejservices/api/SpellCheck/CheckWords" dictionarySettings-
        customDictionaryUrl="http://js.syncfusion.com/demos/ejservices/api/SpellCheck/AddToDictionary" contextMenuSettings-enable={false}>
        Facebook is a social networking service headquartered in Menlo Park,
        California. Its website was launched on February 4, 2004, by Mark Zuckerberg
        with his Harvard College roommates and fellow students Eduardo, Andrew
        McCollum, Dustin and Chris Hughes.
        The fouders had initially limited the websites membership to Harvard
        students, but later expanded it to collges in the Boston area, the Ivy
        League, and Stanford Univrsity. It graually added support for students at
        various other universities and later to high-school students.
        </EJ.SpellCheck>
        <EJ.Button id="btnCheck" type="button" height={30} text="Spell Check"
        click={this.checkInDialog}>
        </EJ.Button>
      </div>
    );
  }
});
ReactDOM.render(<DefaultDialog />, document.getElementById('spellcheck-
default'));
```

## Supported Target Elements

The SpellCheck control has support for spell checking for the HTML element's texts such as div, span, textarea, input, address, article and IFrame elements.

### Multiple Target

The SpellCheck control has support for multiple target HTML element's texts spelling and correct its error words. This can be achieved by passing the target HTML elements "id" values or "CSS Class Name" or combination of both id and class names to the property "controlsToValidate".

The following code example describes the above behavior.

#### HTML

```
"use strict";
var MultipleTargets = React.createClass({
  render: function () {
    return (
      <div id="spell_multiple">
        <div id="control1" class="control1">
          London, one of the most popular tourist destinations in the world for a
          reason. A cultural and historical hub, London has an excellent public
          transportation system that allows visitors to see all the fantastic sights
          without spending a ton of money on a rental car.
          London contains four World Heritage Sites.
        </div><br />
        <span id="control2">
          Rome, one of the world's most fascinating cities. The old adage that Rome was
          not built in a day - and that you won't see it in one or even in three - is
          true. For the intrepid traveler who can keep pace, here is a list of must-
          sees.
          But remember what the Romans say: Even a lifetime isn't enough to see Rome.
        </span><br />
        <EJ.SpellCheck id="SpellCheck3" dictionarySettings-
          dictionaryUrl="http://js.syncfusion.com/demos/ejservices/api/SpellCheck/CheckWords" dictionarySettings-
          customDictionaryUrl="http://js.syncfusion.com/demos/ejservices/api/SpellCheck/AddToDictionary" controlsToValidate=".control1,#control2">
        </EJ.SpellCheck><br />
      </div>
    );
  }
});
ReactDOM.render(<MultipleTargets />,
  document.getElementById('multipletargets'));
```

You need to pass the target HTML element's id value or CSS class name or combination of the id value and class name with comma separator. Passing the class name followed by (.) character and passing the id value followed by the (#) character.

The following code example describes the spell checking of HTML element's.

#### HTML

```
"use strict";
var MultipleTargets = React.createClass({
  // Perform the spell check through dialog mode
```

```

correctErrors: function (e) {
var obj = $("#SpellCheck3").data("ejSpellCheck");
obj.showInDialog();
},
// Perform the spell check through context menu mode
correctErrors: function (e) {
var obj = $("#SpellCheck3").data("ejSpellCheck");
obj.validate();
},
render: function () {
return (
<div id="spell_multiple">
<div id="control1" class="control1">
London, one of the most popular tourist destinations in the world for a
reason. A cultural and historical hub, London has an excellent public
transportation system that allows visitors to see all the fantastic sights
without spending a ton of money on a rental car.
London contains four World Heritage Sites.
</div><br />
<span id="control2">
Rome, one of the world's most fascinating cities. The old adage that Rome was
not built in a day - and that you won't see it in one or even in three - is
true. For the intrepid traveler who can keep pace, here is a list of must-
sees.
But remember what the Romans say: Even a lifetime isn't enough to see Rome.
</span><br />
<EJ.SpellCheck id="SpellCheck3" dictionarySettings-
dictionaryUrl="http://js.syncfusion.com/demos/ejservices/api/SpellCheck/CheckWords" dictionarySettings-
customDictionaryUrl="http://js.syncfusion.com/demos/ejservices/api/SpellCheck/AddToDictionary" controlsToValidate="#control1,#control2">
</EJ.SpellCheck><br />
<EJ.Button id="btnCheck" type="button" height={30} text="Spell Check"
click={this.correctErrors}>
</EJ.Button>
</div>
);
}
});
ReactDOM.render(<MultipleTargets />,
document.getElementById('multipletargets'));

```

The spellcheck component iterates through the target elements which specified in the “controlsToValidate” property, and process the element’s content one by one in the order of CSS elements, and id specified elements.

## Responsive

The SpellCheck control has support for responsive behavior based on client browser’s width and height. To enable responsive, `isResponsive` property should be true.

The following code example describes the above behavior.

## HTML

```

"use strict";
var DefaultDialog = React.createClass({

```

```

render: function () {
  return (
    <div id="dialog_default">
      <EJ.SpellCheck id="SpellCheck1" dictionarySettings-
dictionaryUrl="http://js.syncfusion.com/demos/ejservices/api/SpellCheck/Chec
kWords" dictionarySettings-
customDictionaryUrl="http://js.syncfusion.com/demos/ejservices/api/SpellChec
k/AddToDictionary" contextMenuSettings-enable={false} isResponsive={true}>
        Facebook is a social networking service headquartered in Menlo Park,
        California. Its website was launched on February 4, 2004, by Mark Zuckerberg
        with his Harvard College roommates and fellow students Eduardo, Andrew
        McCollum, Dustin and Chris Hughes.
        The fouders had initially limited the websites membrship to Harvard
        students, but later expanded it to collges in the Boston area, the Ivy
        League, and Stanford Univrsity. It graually added support for students at
        various other universities and later to high-school students.
      </EJ.SpellCheck>
    </div>
  );
}
});
ReactDOM.render(<DefaultDialog />, document.getElementById('spellcheck-
default'));

```

The dialog of spell check control is rendering based on the client browser's width and height. Refer to the following code to render the spellcheck dialog control with responsive.

### HTML

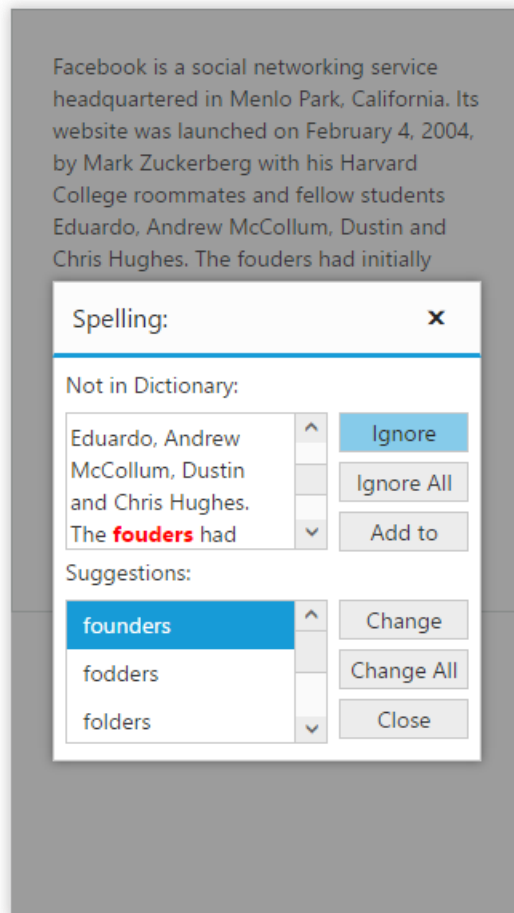
```

"use strict";
var DefaultDialog = React.createClass({
  checkErrors: function (e) {
    var spellObj = $("#SpellCheck1").data("ejSpellCheck");
    spellObj.showInDialog();
  },
  render: function () {
    return (
      <div id="dialog_default">
        <EJ.SpellCheck id="SpellCheck1" dictionarySettings-
dictionaryUrl="http://js.syncfusion.com/demos/ejservices/api/SpellCheck/Chec
kWords" dictionarySettings-
customDictionaryUrl="http://js.syncfusion.com/demos/ejservices/api/SpellChec
k/AddToDictionary" contextMenuSettings-enable={false} isResponsive={true}>
          Facebook is a social networking service headquartered in Menlo Park,
          California. Its website was launched on February 4, 2004, by Mark Zuckerberg
          with his Harvard College roommates and fellow students Eduardo, Andrew
          McCollum, Dustin and Chris Hughes.
          The fouders had initially limited the websites membrship to Harvard
          students, but later expanded it to collges in the Boston area, the Ivy
          League, and Stanford Univrsity. It graually added support for students at
          various other universities and later to high-school students.
        </EJ.SpellCheck>,
        <EJ.Button id="btnCheck" type="button" height={30} text="Spell Check"
click={this.checkErrors}>
        </EJ.Button>
      </div>
    );
  }
});

```

```
);  
}  
});  
</script>
```

Mobile Rendering Screenshot:



## Spreadsheet

### Overview

The Spreadsheet control is a Microsoft Excel-like Spreadsheet component for web. It provides editing experience that is very similar to that of excel and it is able to import and export Excel workbook files.

The Spreadsheet control includes all the important features of Microsoft Excel like editing, sorting, filtering, formulas, data validation, formatting, table, charts, import and export.

### Key Features

The key features of Spreadsheet control for React JS are:

- **Editing** - Provides support to edit the cell content.
- **Sorting** - Helps you to visualize and figure out your data better, organize and find the data quickly that you want.
- **Filtering** - Offers Excel-like filtering to filter data.
- **Formulas** - Allows you to handle calculations in cells.
- **Table** - Allows you to manage and analyze a group of related data.
- **Formatting** - Provides various formatting support like cell formatting, number formatting, text formatting.
- **Data Validation** - Helps you to restrict the users to enter the valid data in a range.
- **Conditional Formatting** - Allows you to highlight range of cells with a certain colors, depending on the provided conditions.
- **Chart** – Allows you to have graphical representation of data.
- **Fill Series** - Allows you to drag fill data based on adjacent cell data.
- **Printing** – Allows you to print whole sheet / selected range of data.
- **Import and Export** - Allows you to import excel files to view and export Spreadsheet as excel files.

### Getting started

This section explains you the steps required to populate the Spreadsheet with data, format, and export it as excel file. This section covers only the minimal features that you need to know to get started with the Spreadsheet.

#### Adding Script Reference

To get more information about react refer [React Getting Started Documentation](#).

Create an **HTML** page and add the scripts references in the order mentioned in the following code example.

#### HTML

```
<!DOCTYPE html>
<html>
<head>
  <!-- Essential Studio for JavaScript theme reference -->
  <link rel="stylesheet" href="http://cdn.syncfusion.com/{
site.releaseversion }}/js/web/bootstrap-theme/ej.web.all.min.css" />
  <!-- react script -->
  <script
src="https://cdnjs.cloudflare.com/ajax/libs/react/15.2.1/react.js"></script>
  <script src="https://cdnjs.cloudflare.com/ajax/libs/react/15.2.1/react-
dom.js"></script>
  <script src="https://cdnjs.cloudflare.com/ajax/libs/babel-
core/5.8.34/browser.min.js"></script>
  <!-- jquery script -->
  <script src="https://code.jquery.com/jquery-1.10.2.min.js"></script>
```



```

<script
src="http://cdn.syncfusion.com/js/assets/external/jquery.globalize.min.js"></script>
<script
src="http://ajax.aspnetcdn.com/ajax/jquery.validate/1.14.0/jquery.validate.min.js"></script>
<!-- jsrender script -->
<script
src="http://cdn.syncfusion.com/js/assets/external/jsrender.min.js"></script>
<!-- Essential JS UI widget -->
<script src="http://cdn.syncfusion.com/{ site.releaseversion
}/js/web/ej.web.all.min.js"></script>
<script src="http://cdn.syncfusion.com/{ site.releaseversion
}/js/common/ej.web.react.min.js"></script>
<!--Add custom scripts here -->
</head>
<body>
</body>
</html>

```

In the above code, `ej.web.all.min.js` script reference has been added for demonstration purpose. It is not recommended to use this for deployment purpose, as its file size is larger since it contains all the widgets. Instead, you can use [CSG](#) utility to generate a custom script file with the required widgets for deployment purpose.

#### Note:

- For details about Spreadsheet internal and external dependencies refer following [link](#)
- `react.js` and `react-dom.js` are the core files needed to create react elements.
- `browser.min.js` file is required for code transform.
- `ej.web.react.min.js` is a react-syncfusion bridge to render Syncfusion components.

#### Initialize Spreadsheet

Create a JSX file for rendering Spreadsheet component using the `<EJ.Spreadsheet>` tag. The Spreadsheet is rendered based on default width and height. You can also customize the Spreadsheet dimension by setting the `width` and `height` property in `scrollSettings`.

#### JS

```

"use strict";
ReactDOM.render(
<EJ.Spreadsheet></EJ.Spreadsheet>,
document.getElementById('spreadsheet')
);

```

Add a `div` container to render the Spreadsheet in HTML file. To translate JSX to plain JavaScript `<script type="text/babel">` is used.

#### HTML

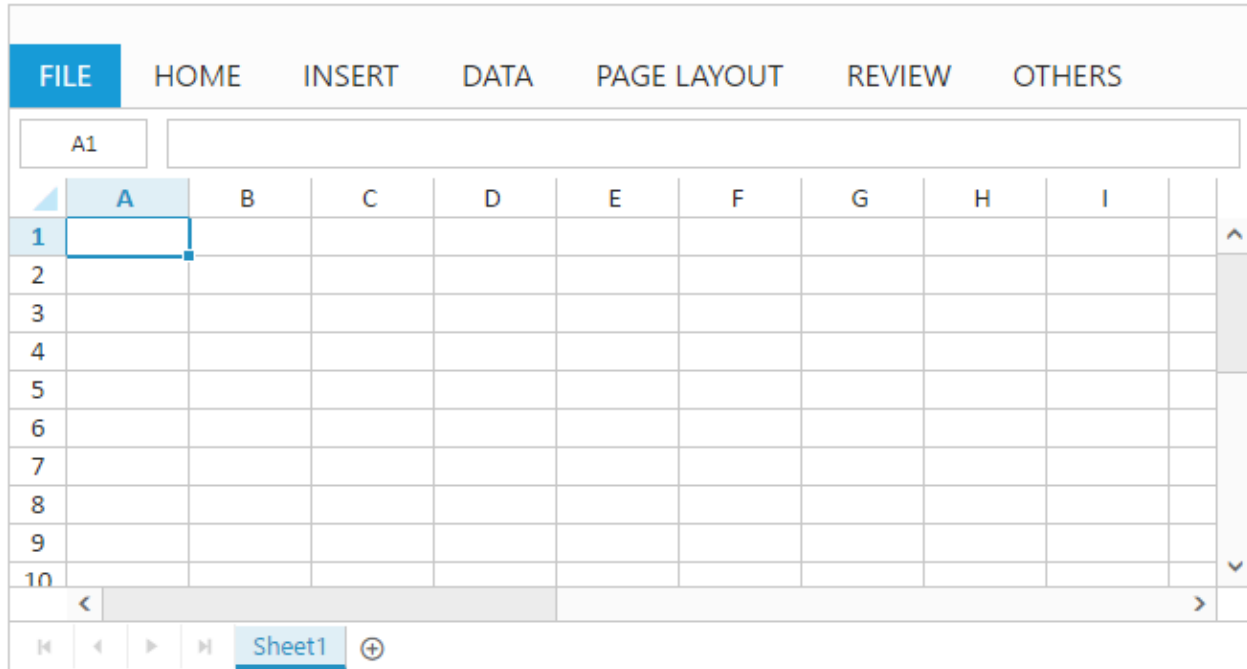
```

<!DOCTYPE html>
<html>
<body>

```

```
<div id="spreadsheet"></div>
<script type="text/babel" src="spreadsheet.jsx">
</body>
</html>
```

Now, the Spreadsheet is rendered with default row and column count.



### Populate Spreadsheet with data

Now, this section explains how to populate JSON data to the Spreadsheet. You can set `dataSource` property in `sheet` settings to populate JSON data in Spreadsheet.

### JS

```
"use strict";
var sheets = [
{
rangeSettings: [{
dataSource: [
{ "Item Name": "Casual Shoes", Date: "02/14/2014", Time: "11:34:32 AM",
Quantity: 10, Price: 20, Amount: 200, Discount: 1, Profit: 10 },
{ "Item Name": "Sports Shoes", Date: "06/11/2014", Time: "05:56:32 AM",
Quantity: 20, Price: 30, Amount: 600, Discount: 5, Profit: 50 },
{ "Item Name": "Formal Shoes", Date: "07/27/2014", Time: "03:32:44 AM",
Quantity: 20, Price: 15, Amount: 300, Discount: 7, Profit: 27 },
{ "Item Name": "Sandals & Floaters", Date: "11/21/2014", Time: "06:23:54
AM", Quantity: 15, Price: 20, Amount: 300, Discount: 11, Profit: 67 },
{ "Item Name": "Flip- Flops & Slippers", Date: "06/23/2014", Time: "12:43:59
AM", Quantity: 30, Price: 10, Amount: 300, Discount: 10, Profit: 70 },
{ "Item Name": "Sneakers", Date: "07/22/2014", Time: "10:55:53 AM",
Quantity: 40, Price: 20, Amount: 800, Discount: 13, Profit: 66 },
```

```

{ "Item Name": "Running Shoes", Date: "02/04/2014", Time: "03:44:34 AM",
  Quantity: 20, Price: 10, Amount: 200, Discount: 3, Profit: 14 },
{ "Item Name": "Loafers", Date: "11/30/2014", Time: "03:12:52 AM", Quantity:
  31, Price: 10, Amount: 310, Discount: 6, Profit: 29 },
{ "Item Name": "Cricket Shoes", Date: "07/09/2014", Time: "11:32:14 AM",
  Quantity: 41, Price: 30, Amount: 1210, Discount: 12, Profit: 166 },
{ "Item Name": "T-Shirts", Date: "10/31/2014", Time: "12:01:44 AM",
  Quantity: 50, Price: 10, Amount: 500, Discount: 9, Profit: 55 },
]
}]
}
];
ReactDOM.render(
  <EJ.Spreadsheet sheets={sheets}></EJ.Spreadsheet>,
  document.getElementById('spreadsheet')
);

```

FILE HOME INSERT DATA PAGE LAYOUT REVIEW OTHERS								
A1	Item Name							
	A	B	C	D	E	F	G	
1	Item Name	Date	Time	Quantity	Price	Amount	Discount	^
2	Casual Shoes	2/14/2014	11:34:32 AM	10	20	200	1	
3	Sports Shoes	6/11/2014	5:56:32 AM	20	30	600	5	
4	Formal Shoes	7/27/2014	3:32:44 AM	20	15	300	7	
5	Sandals & Floaters	11/21/2014	6:23:54 AM	15	20	300	11	
6	Flip- Flops & Slippers	6/23/2014	12:43:59 AM	30	10	300	10	
7	Sneakers	7/22/2014	10:55:53 AM	40	20	800	13	
8	Running Shoes	2/4/2014	3:44:34 AM	20	10	200	3	
9	Loafers	11/30/2014	3:12:52 AM	31	10	310	6	
10	Cricket Shoes	7/9/2014	11:32:14 AM	41	30	1210	12	▼
< >								
Sheet1 ⊕								

### Apply Conditional Formatting

Conditional formatting helps you to apply formats to a cell or range with certain color based on the cells values. You can use `allowConditionalFormats` attribute to enable/disable Conditional formats.

To apply conditional formats for a range use `cFormatRule` property . The following code example illustrates this behavior,

### JS

```

"use strict";
var sheets = [
{
  rangeSettings: [{
    dataSource: [
      { "Item Name": "Casual Shoes", Date: "02/14/2014", Time: "11:34:32 AM",
        Quantity: 10, Price: 20, Amount: 200, Discount: 1, Profit: 10 },

```

```

{ "Item Name": "Sports Shoes", Date: "06/11/2014", Time: "05:56:32 AM",
  Quantity: 20, Price: 30, Amount: 600, Discount: 5, Profit: 50 },
{ "Item Name": "Formal Shoes", Date: "07/27/2014", Time: "03:32:44 AM",
  Quantity: 20, Price: 15, Amount: 300, Discount: 7, Profit: 27 },
{ "Item Name": "Sandals & Floaters", Date: "11/21/2014", Time: "06:23:54
AM", Quantity: 15, Price: 20, Amount: 300, Discount: 11, Profit: 67 },
{ "Item Name": "Flip- Flops & Slippers", Date: "06/23/2014", Time: "12:43:59
AM", Quantity: 30, Price: 10, Amount: 300, Discount: 10, Profit: 70 },
{ "Item Name": "Sneakers", Date: "07/22/2014", Time: "10:55:53 AM",
  Quantity: 40, Price: 20, Amount: 800, Discount: 13, Profit: 66 },
{ "Item Name": "Running Shoes", Date: "02/04/2014", Time: "03:44:34 AM",
  Quantity: 20, Price: 10, Amount: 200, Discount: 3, Profit: 14 },
{ "Item Name": "Loafers", Date: "11/30/2014", Time: "03:12:52 AM", Quantity:
31, Price: 10, Amount: 310, Discount: 6, Profit: 29 },
{ "Item Name": "Cricket Shoes", Date: "07/09/2014", Time: "11:32:14 AM",
  Quantity: 41, Price: 30, Amount: 1210, Discount: 12, Profit: 166 },
{ "Item Name": "T-Shirts", Date: "10/31/2014", Time: "12:01:44 AM",
  Quantity: 50, Price: 10, Amount: 500, Discount: 9, Profit: 55 },
]
}],
cFormatRule: [{ action: ej.Spreadsheet.CFormatRule.GreaterThan, inputs:
["10"], color: ej.Spreadsheet.CFormatHighlightColor.RedFill, range: "D2:D8"
}]
}
};
ReactDOM.render(
<EJ.Spreadsheet sheets={sheets}></EJ.Spreadsheet>,
document.getElementById('spreadsheet')
);

```

FILE HOME INSERT DATA PAGE LAYOUT REVIEW OTHERS							
A1	Item Name						
	A	B	C	D	E	F	G
1	Item Name	Date	Time	Quantity	Price	Amount	Discount
2	Casual Shoes	2/14/2014	11:34:32 AM	10	20	200	1
3	Sports Shoes	6/11/2014	5:56:32 AM	20	30	600	5
4	Formal Shoes	7/27/2014	3:32:44 AM	20	15	300	7
5	Sandals & Floaters	11/21/2014	6:23:54 AM	15	20	300	11
6	Flip- Flops & Slippers	6/23/2014	12:43:59 AM	30	10	300	10
7	Sneakers	7/22/2014	10:55:53 AM	40	20	800	13
8	Running Shoes	2/4/2014	3:44:34 AM	20	10	200	3
9	Loafers	11/30/2014	3:12:52 AM	31	10	310	6
10	Cricket Shoes	7/9/2014	11:32:14 AM	41	30	1210	12
Sheet1							

### Export Spreadsheet as Excel File

The Spreadsheet can save its data, style, format into an excel file. To enable save option in Spreadsheet set `allowExporting` option in `exportSettings` as `true`. Since Spreadsheet uses server side helper to save documents set `excelUrl` in `exportSettings` option. The following code example illustrates this behavior,

**JS**

```
"use strict";
var sheets = [
{
rangeSettings: [{
dataSource: [
{ "Item Name": "Casual Shoes", Date: "02/14/2014", Time: "11:34:32 AM",
Quantity: 10, Price: 20, Amount: 200, Discount: 1, Profit: 10 },
{ "Item Name": "Sports Shoes", Date: "06/11/2014", Time: "05:56:32 AM",
Quantity: 20, Price: 30, Amount: 600, Discount: 5, Profit: 50 },
{ "Item Name": "Formal Shoes", Date: "07/27/2014", Time: "03:32:44 AM",
Quantity: 20, Price: 15, Amount: 300, Discount: 7, Profit: 27 },
{ "Item Name": "Sandals & Floaters", Date: "11/21/2014", Time: "06:23:54
AM", Quantity: 15, Price: 20, Amount: 300, Discount: 11, Profit: 67 },
{ "Item Name": "Flip- Flops & Slippers", Date: "06/23/2014", Time: "12:43:59
AM", Quantity: 30, Price: 10, Amount: 300, Discount: 10, Profit: 70 },
{ "Item Name": "Sneakers", Date: "07/22/2014", Time: "10:55:53 AM",
Quantity: 40, Price: 20, Amount: 800, Discount: 13, Profit: 66 },
{ "Item Name": "Running Shoes", Date: "02/04/2014", Time: "03:44:34 AM",
Quantity: 20, Price: 10, Amount: 200, Discount: 3, Profit: 14 },
{ "Item Name": "Loafers", Date: "11/30/2014", Time: "03:12:52 AM", Quantity:
31, Price: 10, Amount: 310, Discount: 6, Profit: 29 },
{ "Item Name": "Cricket Shoes", Date: "07/09/2014", Time: "11:32:14 AM",
Quantity: 41, Price: 30, Amount: 1210, Discount: 12, Profit: 166 },
{ "Item Name": "T-Shirts", Date: "10/31/2014", Time: "12:01:44 AM",
Quantity: 50, Price: 10, Amount: 500, Discount: 9, Profit: 55 },
]
}],
cFormatRule: [{ action: ej.Spreadsheet.CFormatRule.GreaterThan, inputs:
["10"], color: ej.Spreadsheet.CFormatHighlightColor.RedFill, range: "D2:D8"
}]
}
];
var exportSettings = {
excelUrl:
"http://js.syncfusion.com/demos/ejservices/api/Spreadsheet/ExcelExport"
};
ReactDOM.render(
<EJ.Spreadsheet sheets={sheets}
exportSettings={exportSettings}></EJ.Spreadsheet>,
document.getElementById('spreadsheet')
);
```

Use shortcut **Ctrl + S** to save Spreadsheet as excel file.

## Sunburst Chart

### Getting Started

This section explains you the steps required to populate the Sunburst Chart with data, data labels, legend and title. This section covers only the minimal features that you need to know to get started with the Sunburst Chart.

### Adding Script Reference

Create an **HTML** page and add the scripts references in the order mentioned in the following code example.

- [jQuery](#) 1.10.2 and later versions

The required ReactJS script dependencies as follows. And you can also refer [React](#) to know more about react js.

- react.min.js - <http://cdn.syncfusion.com/js/assets/external/react.min.js>
- react-dom.min.js - <http://cdn.syncfusion.com/js/assets/external/react-dom.min.js>
- ej.web.react.min.js - <http://cdn.syncfusion.com/{{ site.releaseversion }}/js/common/ej.web.react.min.js>

To get started, you can use the `ej.web.all.min.js` file that encapsulates all the `ej` controls and frameworks in one single file.

### HTML

```
<!DOCTYPE html>
<html>
<head>
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<meta name="description" content="Essential Studio for React JS">
<meta name="author" content="Syncfusion">
<title>Getting Started for Ribbon React JS</title>
<!-- Essential Studio for JavaScript theme reference -->
<link href="http://cdn.syncfusion.com/{{ site.releaseversion }}/js/web/flat-azure/ej.web.all.min.css" rel="stylesheet" />
<!-- Essential Studio for JavaScript script references -->
<script src="http://cdn.syncfusion.com/js/assets/external/jquery-3.0.0.min.js"></script>
<script
src="http://cdn.syncfusion.com/js/assets/external/react.min.js"></script>
<script src="http://cdn.syncfusion.com/js/assets/external/react-dom.min.js"></script>
<script src="http://cdn.syncfusion.com/{{ site.releaseversion }}/js/web/ej.web.all.min.js"></script>
<script src="http://cdn.syncfusion.com/{{ site.releaseversion }}/js/common/ej.web.react.min.js"></script>
<!-- Add your custom scripts here -->
</head>
<body>
</body>
</html>
```

**Note:** 1. In production, we highly recommend you to use our [custom script generator](#) to create custom script file with required controls and its dependencies only. Also to reduce the file size further please use [GZip compression](#) in your server.

2. For themes, you can use the `ej.web.all.min.css` CDN link from the code snippet given. To add the themes in your application, please refer to [this link](#).

### Control Initialization

Control can be initialized in two ways.

- Using jsx Template
- Without using jsx Template

### Using jsx Template

By using the jsx template, we can create the html file and jsx file. The `.jsx` file can be convert to `.js` file and it can be referred in html page.

### Create Sunburst Chart

You can easily create the Sunburst Chart widget by following the below steps.

1.Create a

tag.

#### HTML

```
<!DOCTYPE html>
<html>
<body>
<div id="sunburst"></div>
<script src="app/sunburstchart/default.js">
</script>
</body>
</html>
```

2.Initialize the Sunburst chart by using the `EJ.SunburstChart` tag

#### JAVASCRIPT

```
"use strict";
ReactDOM.render(
<EJ.SunburstChart id = "sunburst1">
</EJ.SunburstChart>,
document.getElementById('sunburst')
);
```

### Populate Data source:

The datasource for the Sunburst Chart is populated as a JSON object. The “default\_data” contains the JSON data for rendering the Sunburst Chart as shown in the sample.

#### JS

```
var default_data = [
{ Category: "Employees", Country: "USA", JobDescription: "Sales", JobGroup:
"Executive", EmployeesCount: 50 },
```

```

{ Category: "Employees", Country: "USA", JobDescription: "Sales", JobGroup:
"Analyst", EmployeesCount: 40 },
{ Category: "Employees", Country: "USA", JobDescription: "Marketing",
EmployeesCount: 40 },
{ Category: "Employees", Country: "USA", JobDescription: "Technical",
JobGroup: "Testers", EmployeesCount: 55 },
{ Category: "Employees", Country: "USA", JobDescription: "Technical",
JobGroup: "Developers", JobRole: "Windows", EmployeesCount: 175 },
{ Category: "Employees", Country: "USA", JobDescription: "Technical",
JobGroup: "Developers", JobRole: "Web", EmployeesCount: 70 },
{ Category: "Employees", Country: "USA", JobDescription: "Management",
EmployeesCount: 40 },
{ Category: "Employees", Country: "USA", JobDescription: "Accounts",
EmployeesCount: 60 },
{ Category: "Employees", Country: "India", JobDescription: "Technical",
JobGroup: "Testers", EmployeesCount: 43 },
{ Category: "Employees", Country: "India", JobDescription: "Technical",
JobGroup: "Developers", JobRole: "Windows", EmployeesCount: 125 },
{ Category: "Employees", Country: "India", JobDescription: "Technical",
JobGroup: "Developers", JobRole: "Web", EmployeesCount: 60 },
{ Category: "Employees", Country: "India", JobDescription: "HR Executives",
EmployeesCount: 70 },
{ Category: "Employees", Country: "India", JobDescription: "Accounts",
EmployeesCount: 45 },
{ Category: "Employees", Country: "Germany", JobDescription: "Sales",
JobGroup: "Executive", EmployeesCount: 30 },
{ Category: "Employees", Country: "Germany", JobDescription: "Sales",
JobGroup: "Analyst", EmployeesCount: 40 },
{ Category: "Employees", Country: "Germany", JobDescription: "Marketing",
EmployeesCount: 50 },
{ Category: "Employees", Country: "Germany", JobDescription: "Technical",
JobGroup: "Testers", EmployeesCount: 40 },
{ Category: "Employees", Country: "Germany", JobDescription: "Technical",
JobGroup: "Developers", JobRole: "Windows", EmployeesCount: 65 },
{ Category: "Employees", Country: "Germany", JobDescription: "Technical",
JobGroup: "Developers", JobRole: "Web", EmployeesCount: 27 },
{ Category: "Employees", Country: "Germany", JobDescription: "Management",
EmployeesCount: 33 },
{ Category: "Employees", Country: "Germany", JobDescription: "Accounts",
EmployeesCount: 55 },
{ Category: "Employees", Country: "UK", JobDescription: "Technical",
JobGroup: "Testers", EmployeesCount: 45 },
{ Category: "Employees", Country: "UK", JobDescription: "Technical",
JobGroup: "Developers", JobRole: "Windows", EmployeesCount: 96 },
{ Category: "Employees", Country: "UK", JobDescription: "Technical",
JobGroup: "Developers", JobRole: "Web", EmployeesCount: 55 },
{ Category: "Employees", Country: "UK", JobDescription: "HR Executives",
EmployeesCount: 60 },
{ Category: "Employees", Country: "UK", JobDescription: "Accounts",
EmployeesCount: 30 },
{ Category: "Employees", Country: "France", JobDescription: "Technical",
JobGroup: "Testers", EmployeesCount: 40 },
{ Category: "Employees", Country: "France", JobDescription: "Technical",
JobGroup: "Developers", JobRole: "Windows", EmployeesCount: 65 },
{ Category: "Employees", Country: "France", JobDescription: "Technical",
JobGroup: "Developers", JobRole: "Web", EmployeesCount: 27 },

```



```
{ Category: "Employees", Country: "France", JobDescription: "Marketing",
  EmployeesCount: 50 }
];
```

#### Initialize Sunburst Chart with data

Now, bind the default\_Datasource to [datasource](#) property of the Sunburst Chart. The [levels](#) property determines the number of hierarchical levels. Each hierarchy level is formed based on the property specified in [groupMemberPath](#) property, and each arc segment size is calculated using [valueMemberPath](#).

#### JS

```
"use strict";
var levels=[
{groupMemberPath: "Country" },
{groupMemberPath: "JobDescription"},
{groupMemberPath: "JobGroup"},
{groupMemberPath: "JobRole" }
];
ReactDOM.render(
<EJ.SunburstChart id = "sunburst1" valueMemberPath = "EmployeesCount"
dataSource = {default_data}
levels = {levels} ></EJ.SunburstChart>,
document.getElementById('sunburst')
);
```

#### Add Title to the Sunburst Chart

The title of the Sunburst chart is used to provide quick information to the user about the data being plotted in the Sunburst Chart. You can add it by using the [text](#) property of the [title](#)

#### JS

```
"use strict";
var title = { text:'Employees Count'};
ReactDOM.render(
<EJ.SunburstChart id = "sunburst1"
title = {title}></EJ.SunburstChart>,
document.getElementById('sunburst')
);
```

#### Enable Legend

You can enable or disable the legend by using the [visible](#) property present inside the [legend](#)

#### JS

```
"use strict";
var legend = { visible:true, position: "left" };
ReactDOM.render(
<EJ.SunburstChart id = "sunburst1"
legend = {legend}></EJ.SunburstChart>,
document.getElementById('sunburst')
);
```

### Add Data Labels

The data labels are used to improve the readability of the Sunburst chart. This can be achieved by enabling the [visible](#) property in the [datalabelSettings](#).

#### JS

```
"use strict";
var datalabelSettings = { visible:true};
ReactDOM.render(
  <EJ.SunburstChart id = "sunburst1"
    datalabelSettings = {datalabelSettings}></EJ.SunburstChart>,
  document.getElementById('sunburst')
);
```

Now the Sunburst Chart is rendered along with the specified customizations

![[/js/SunburstChart/Getting-Startedimages/Getting-Startedimg1.png]

[Click](#) here to view the default sample of the SunburstChart

### Sunburst Elements

The Sunburst region represents the entire chart and all its elements. It includes all the chart elements like Legend, DataLabel, Levels etc. The major properties of the Sunburst Chart are as follows

- [datasource](#) – Provides the data that are used to generate the chart.
- [valueMemberPath](#) - Property based on the which the data segments are rendered in the Sunburst chart
- [legend](#) – displays the legend of the Sunburst Chart
- [levels](#)- displays the hierarchical levels for the chart
- [datalabel](#) – displays the datalabel for the Sunburst Chart

### Start and End Angle

#### Start and End Angle

You can change the start and end angle of Sunburst chart using [startAngle](#) and [endAngle](#) property as shown in below code

#### JS

```
"use strict";
//Set startAngle and endAngle to draw the sunburst chart
ReactDOM.render(
  <EJ.SunburstChart id = "sunburst1"
    startAngle = {-90}
    endAngle = {90}
  >
  </EJ.SunburstChart>,
  document.getElementById('sunburst')
);
```

### Sunburst Radius

The Radius of the Sunburst chart can be customized by using the [radius](#) property. The default value of radius is 1 and its value ranges between 0 and 1

**JS**

```
"use strict";
//Set radius to the sunburst chart
ReactDOM.render(
  <EJ.SunburstChart id = "sunburst1"
    radius = {0.8}>
  </EJ.SunburstChart>,
  document.getElementById('sunburst')
);
```

;

**Sunburst Inner Radius**

The Inner Radius of the Sunburst chart can be customized by using the [innerRadius](#) property. The default value of innerRadius is 0.4 and its value ranges between 0 and 1

**JS**

```
"use strict";
//Set inner radius to the sunburst chart
ReactDOM.render(
  <EJ.SunburstChart id = "sunburst1"
    innerRadius = {0.5}
  >
  </EJ.SunburstChart>,
  document.getElementById('sunburst')
);
```

;

**Levels**

Sunburst chart is used to display hierarchical data. You can add more than one hierarchical data by using the [levels](#) property of Sunburst chart. Each level of the hierarchy is represented by circle.

The following code snippet illustrates

**JS**

```
"use strict";
var levels = [
  {
    //... to represent the hierarchical data in different levels
  }
];
ReactDOM.render(
  <EJ.SunburstChart id = "sunburst1"
    levels={levels}
  >
  </EJ.SunburstChart>,
  document.getElementById('sunburst')
);
```

**GroupMemberPath****JS**

```
"use strict";
var levels = [
  { groupMemberPath: "Level 1" },
  { groupMemberPath: "Level 2" },
  { groupMemberPath: "Level 3" },
];
ReactDOM.render(
  <EJ.SunburstChart id = "sunburst1"
    levels ={levels}
  >
</EJ.SunburstChart>,
  document.getElementById('sunburst')
);
```

The following screenshot illustrates the Sunburst Chart with different levels



### Legend

The legend is used to represent the first level of items in the Sunburst Chart. The [legend](#) can be initialized using the below code snippet

### JS

```
"use strict";
var legend = { visible: true };
ReactDOM.render(
  <EJ.SunburstChart id = "sunburst1"
    legend ={legend}
  >
</EJ.SunburstChart>,
  document.getElementById('sunburst')
);
```



### Legend Icon

You can specify different shapes of legend icon by using the [shape](#) property of the legend. By default, legend shape is **Circle**. The Sunburst chart has some predefined shapes such as:

- Circle
- Cross
- Diamond
- Pentagon
- Rectangle
- Triangle

### JS

```
"use strict";
var legend = { shape: "pentagon" };
ReactDOM.render(
  <EJ.SunburstChart id = "sunburst1"
    legend ={legend}
  >
```

```
>
</EJ.SunburstChart>,
document.getElementById('sunburst')
);
```



### Positioning the Legend

By using the [position](#) property, you can position the legend at left, right, top or bottom of the chart.

#### JS

```
"use strict";
var legend = { position:"top"};
ReactDOM.render(
<EJ.SunburstChart id = "sunburst1"
  legend ={legend}
>
</EJ.SunburstChart>,
document.getElementById('sunburst')
);
```



### Customization

#### Legend Item Size and border

You can change the size of the legend items by using the [itemStyle.width](#) and [itemStyle.height](#) properties. To change the legend item border, use [border](#) property of the legend .

#### JS

```
"use strict";
var legend = {position:"top",itemStyle:{height:13,width:13},border: { color:
"#FF0000", width: 1 }};
ReactDOM.render(
<EJ.SunburstChart id = "sunburst1"
  legend ={legend}
>
</EJ.SunburstChart>,
document.getElementById('sunburst')
);
```



### Legend Alignment

You can align the legend to the **center**, **far** or **near** based on its position by using the **legend-alignment** option.

#### JS

```
"use strict";
var legend = {alignment:"near"};
ReactDOM.render(
<EJ.SunburstChart id = "sunburst1"
  legend ={legend}
>
```

```
>
</EJ.SunburstChart>,
document.getElementById('sunburst')
);
```

### Legend Size

By default, legend takes 20% of the height horizontally when it was placed on the top or bottom position and 20% of the width vertically while placing on the left or right position of the chart. You can change this default legend size by using the [size](#) property of the legend.

#### JS

```
"use strict";
var legend = {position:"top",size:{ height:"75",width:"200"}};
ReactDOM.render(
<EJ.SunburstChart id = "sunburst1"
  legend ={legend}
>
</EJ.SunburstChart>,
document.getElementById('sunburst')
);
```



### Legend title

To add the title to the legend, you have to specify the `legend.title` option.

#### JS

```
"use strict";
var legend = {title:{
  //..
}};
ReactDOM.render(
<EJ.SunburstChart id = "sunburst1"
  legend ={legend}
>
</EJ.SunburstChart>,
document.getElementById('sunburst')
);
```

### Customize the legend text

To customize the legend item text and title you can use the `legend-title-font` and `legend-title` options. You can change the legend title alignment by using the `legend-title-textAlignment` option of the legend title.

#### JS

```
"use strict";
var legend = {title:{
  //..
  font:{
    //..
  }
}}
```

```

});
ReactDOM.render(
  <EJ.SunburstChart id = "sunburst1"
  legend ={legend}
  >
</EJ.SunburstChart>,
  document.getElementById('sunburst')
);

```

### Legend Row and Column

You can arrange the legend items horizontally and vertically by using the [rowCount](#) and [columnCount](#) properties of the legend.

- When only the rowCount is specified, the legend items are arranged according to the rowCount and number of columns may vary based on the number of legend items.
- When only the columnCount is specified, the legend items are arranged according to the columnCount and number of rows may vary based on the number of legend items.
- When both the properties are specified, then the one which has higher value is given preference. For example, when the rowCount is 4 and columnCount is 3, legend items are arranged in 4 rows.
- When both the properties are specified and have the same value, the preference is given to the columnCount when it is positioned at the top/bottom position. The preference is given to the rowCount when it is positioned at the left/right position.

### JS

```

"use strict";
var legend = {position:"top",rowCount:"2",columnCount:"3"};
ReactDOM.render(
  <EJ.SunburstChart id = "sunburst1"
  legend ={legend}
  >
</EJ.SunburstChart>,
  document.getElementById('sunburst')
);

```



### LegendInteractivity

You can select a specific category while clicking on corresponding legend item through [`clickAction`](#) property.

It has three types of action

*ToggleSegmentSelection* *ToggleSegmentVisibility* \* *None*

### ToggleSegmentSelection

Used to highlight specific category while clicking on legend item

### JS

```

"use strict";
var legend = {clickAction:"toggleSegmentSelection"};

```

```
ReactDOM.render(
  <EJ.SunburstChart id = "sunburst1"
    legend ={legend}
  >
</EJ.SunburstChart>,
  document.getElementById('sunburst')
);
```



### Toggle Segment Visibility

Used to disable the specific category while clicking on legend item.

#### JS

```
"use strict";
var legend = {clickAction:"toggleSegmentVisibility"};
ReactDOM.render(
  <EJ.SunburstChart id = "sunburst1"
    legend ={legend}
  >
</EJ.SunburstChart>,
  document.getElementById('sunburst')
);
```



### Data Labels

Sunburst data labels are used to display the data related to the segment. It helps to provide the information about the data points to the users.

You can enable or disable the data labels by setting the [visible](#) property of the [dataLabelSettings](#) to true as shown in the below code

#### JS

```
"use strict";
var dataLabelSettings = { visible:true};
ReactDOM.render(
  <EJ.SunburstChart id = "sunburst1"
    dataLabelSettings ={dataLabelSettings}
  >
</EJ.SunburstChart>,
  document.getElementById('sunburst')
);
```



### Label Overflow mode

When you represent huge data with data labels, they may intersect each other. You can avoid this using the [labelOverflowMode](#) property.

The following properties are used to avoid the overlapping.

*Trim* – To trim the large data labels. *Hide* – To hide the overlapped data labels.



The following code shows how to set Hide and Trim mode.

**JS**

```
"use strict";
var dataLabelSettings = { visible:true,labelOverflowMode:"hide"};
ReactDOM.render(
  <EJ.SunburstChart id = "sunburst1"
    dataLabelSettings ={dataLabelSettings}
  >
</EJ.SunburstChart>,
document.getElementById('sunburst')
);
```



**JS**

```
"use strict";
var dataLabelSettings = { visible:true,labelOverflowMode:"trim"};
ReactDOM.render(
  <EJ.SunburstChart id = "sunburst1"
    dataLabelSettings ={dataLabelSettings}
  >
</EJ.SunburstChart>,
document.getElementById('sunburst')
);
```



### Label Rotation Mode

You can rotate the data label by using [labelRotationMode](#) property. By default, the labelRotationMode is set as **angle**.

The following code shows how to set labelRotationMode as normal and angle.

**JS**

```
"use strict";
var dataLabelSettings = { visible:true,labelRotationMode:"normal"};
ReactDOM.render(
  <EJ.SunburstChart id = "sunburst1"
    dataLabelSettings ={dataLabelSettings}
  >
</EJ.SunburstChart>,
document.getElementById('sunburst')
);
```



**JS**

```
"use strict";
var dataLabelSettings = { visible:true,labelRotationMode:"angle"};
ReactDOM.render(
  <EJ.SunburstChart id = "sunburst1"
    dataLabelSettings ={dataLabelSettings}
  >
```

```
>
</EJ.SunburstChart>,
document.getElementById('sunburst')
);
```



### Customizing the data labels

You can customize the appearance of the data point using the [font](#) property.

#### JS

```
"use strict";
var dataLabelSettings = {visible: true, font:
{color:"black",fontWeight:"bold",size:"15px"}};
ReactDOM.render(
<EJ.SunburstChart id = "sunburst1"
dataLabelSettings ={dataLabelSettings}
>
</EJ.SunburstChart>,
document.getElementById('sunburst')
);
```



### Sunburst Chart Title & Subtitle

#### Title & TextAlignment

By using the title option, you can add the **title-text** as well as customize its **title-border**, **title-background** and **title-font**.

You can change the title alignment to center, far and near by using the **title-textAlignment** property of the Title.

#### JS

```
"use strict";
var title = {visible: true, font:
{color:"black",fontWeight:"bold",size:"15px"},
border:{color:'green',width:5}};
ReactDOM.render(
<EJ.SunburstChart id = "sunburst1"
title ={title}
>
</EJ.SunburstChart>,
document.getElementById('sunburst')
);
```

#### Sub Title & TextAlignment

By using the subTitle option, you can add the **title-subTitle-text** as well as customize its **title-subTitle-border**, **title-subTitle-background** and **title-subTitle-font**.

#### JS

```
"use strict";
```

```

var title = { subTitle:{visible: true, font:
{color:"black",fontWeight:"bold",size:"15px"}},
border:{color:'green',width:5}};
ReactDOM.render(
<EJ.SunburstChart id = "sunburst1"
title ={title}
>
</EJ.SunburstChart>,
document.getElementById('sunburst')
);

```

### Tooltip

[Tooltip](#) allows you to display any information over a sunburst segment. It appears when mouse hovered over or touch any chart segment. By default, it displays the corresponding segment category name and its value

### JS

```

"use strict";
var tooltip = {visible: true};
ReactDOM.render(
<EJ.SunburstChart id = "sunburst1"
tooltip ={tooltip}
>
</EJ.SunburstChart>,
document.getElementById('sunburst')
);

```



### Tooltip Template

HTML elements can be displayed in the tooltip by using the [template](#) property of the tooltip. The template property takes the value of the id attribute of the HTML element. You can use the **#point.x#** and **#point.y#** as place holders in the HTML element to display the x and y values of the corresponding point.

### JS

```

<body>
<div id="Tooltip" style="display: none;">
<div id="value" style="background-color:red;padding-top:3px;padding-right:3px">
<div>
<label id="efpercentage" style="color:white">
&#160;&#160;Category:&#160;&#160;#point.x#
<br />&#160;&#160;Value:#point.y#
</label>
</div>
</div>
</div>
</body>
"use strict";
var tooltip = { visible: true,
template:'Tooltip'};
ReactDOM.render(

```

```
<EJ.SunburstChart id = "sunburst1"
tooltip = {tooltip}>
</EJ.SunburstChart>,
document.getElementById('sunburst')
);
```



### Customize the appearance of tooltip

The `fill` and `border` options are used to customize the background color and border of the tooltip respectively. The `font` option in the tooltip is used to customize the font of the tooltip text.

#### JS

```
"use strict";
var tooltip = { visible: true,
border:{
color:'red',
width:2
},
font:{
//..
}
};
ReactDOM.render(
<EJ.SunburstChart id = "sunburst1"
tooltip = {tooltip}>
</EJ.SunburstChart>,
document.getElementById('sunburst')
);
```

### Highlight

EjSunburstChart provides highlighting support for the points on mouse hover. To enable the highlighting, set the `enable` property to true in the `highlightSettings`.

#### JS

```
"use strict";
var highlightSettings = { enable: true };
ReactDOM.render(
<EJ.SunburstChart id = "sunburst1"
highlightSettings ={highlightSettings}
>
</EJ.SunburstChart>,
document.getElementById('sunburst')
);
```



### Highlight Display mode

You can customize the highlighted segment appearance by using color or opacity. You can choose between color or opacity using the `type` property in the highlight Settings.

*HighlightByColor* – To display the highlighted segment appearance using color. *HighlightByOpacity* – To display the highlighted segment appearance using opacity.

**JS**

```
"use strict";
var highlightSettings = { enable: true, type:"color",color:"red" };
ReactDOM.render(
  <EJ.SunburstChart id = "sunburst1"
  highlightSettings ={highlightSettings}
  >
</EJ.SunburstChart>,
  document.getElementById('sunburst')
);
```



**Highlight Mode**

Sunburst chart provides multiple options to represent the highlighted categories. You can highlight the segment categories by using the [mode](#) property in highlightSettings

*Child* – To highlight the child of selected parent. *All* – To highlight the entire categories in group. *Parent* – To highlight the parent of selected child. *Single* - To highlight single item in the category.

**Child**

The following code shows how to set the highlight type as child

**JS**

```
"use strict";
var highlightSettings = { enable: true,mode:"child" };
ReactDOM.render(
  <EJ.SunburstChart id = "sunburst1"
  highlightSettings ={highlightSettings}
  >
</EJ.SunburstChart>,
  document.getElementById('sunburst')
);
```



**Parent**

The parent mode can be enabled by using the below code

**JS**

```
"use strict";
var highlightSettings = { enable: true,mode:"parent"};
ReactDOM.render(
  <EJ.SunburstChart id = "sunburst1"
  highlightSettings ={highlightSettings}
  >
</EJ.SunburstChart>,
  document.getElementById('sunburst')
);
```



#### Point

To highlight the particular segment, the point mode of the highlight settings is used.

#### JS

```
"use strict";
var highlightSettings = { enable: true, mode: "point" };
ReactDOM.render(
  <EJ.SunburstChart id = "sunburst1"
    highlightSettings = {highlightSettings}
  >
</EJ.SunburstChart>,
  document.getElementById('sunburst')
);
```



#### All

The following code snippet is used for the all mode of highlight settings

#### JS

```
"use strict";
var highlightSettings = { enable: true, mode: "all" };
ReactDOM.render(
  <EJ.SunburstChart id = "sunburst1"
    highlightSettings = {highlightSettings}
  >
</EJ.SunburstChart>,
  document.getElementById('sunburst')
);
```



[Click](#) here to view the online demo sample of Highlight in the Sunburst Chart

#### Selection

EjSunburstChart provides selection support for the points on mouse click. To enable the selection, set the [enable](#) property to true in the [selectionSettings](#).

#### JS

```
"use strict";
var selectionSettings = { enable: true };
ReactDOM.render(
  <EJ.SunburstChart id = "sunburst1"
    selectionSettings = {selectionSettings}
  >
</EJ.SunburstChart>,
  document.getElementById('sunburst')
);
```



### Selection Display mode

You can customize the selected segment appearance by using color or opacity. You can choose between color or opacity using the [type](#) property in the selection Settings.

*selectionByColor* – To display the selected segment appearance using color. *selectionByOpacity* – To display the selected segment appearance using opacity.

#### JS

```
"use strict";
var selectionSettings = { enable: true, type:"color",color:"red" };
ReactDOM.render(
  <EJ.SunburstChart id = "sunburst1"
  selectionSettings ={selectionSettings}
  >
  </EJ.SunburstChart>,
  document.getElementById('sunburst')
);
```



### Selection Mode

Sunburst chart provides multiple options to represent the selected categories. You can select the segment categories by using the [mode](#) property in selectionSettings

*Child* – To selection the child of selected parent. *All* – To selection the entire categories in group. *Parent* – To selection the parent of selected child. *Single* - To selection single item in the category.

#### Child

The following code shows how to set the selection type as child

#### JS

```
"use strict";
var selectionSettings = { enable: true,mode:"child"};
ReactDOM.render(
  <EJ.SunburstChart id = "sunburst1"
  selectionSettings ={selectionSettings}
  >
  </EJ.SunburstChart>,
  document.getElementById('sunburst')
);
```



#### Parent

The parent mode can be enabled by using the below code

#### JS

```
"use strict";
var selectionSettings = { enable: true,mode:"parent"};
ReactDOM.render(
  <EJ.SunburstChart id = "sunburst1"
  selectionSettings ={selectionSettings}
  >
```

```
</EJ.SunburstChart>,
document.getElementById('sunburst')
);
```



#### Point

To selection the particular segment, the point mode of the selection settings is used.

#### JS

```
"use strict";
var selectionSettings = { enable: true, mode: "point" };
ReactDOM.render(
  <EJ.SunburstChart id = "sunburst1"
  selectionSettings ={selectionSettings}
  >
  </EJ.SunburstChart>,
  document.getElementById('sunburst')
);
```



#### All

The following code snippet is used for the all mode of selection settings

#### JS

```
"use strict";
var selectionSettings = { enable: true, mode: "all" };
ReactDOM.render(
  <EJ.SunburstChart id = "sunburst1"
  selectionSettings ={selectionSettings}
  >
  </EJ.SunburstChart>,
  document.getElementById('sunburst')
);
```



[Click](#) here to view the online demo sample of Selection in the Sunburst Chart

#### Zooming

Sunburst chart provides zooming (drill down) experience with animation for both mouse and touch enabled devices. It allows you to virtualizing large sets of data into minimum data view. The zooming is achieved by using the property of [zoomSettings](#)

The following code shows how to initialize the zooming.

#### JS

```
"use strict";
var zoomSettings = { enable: true };
ReactDOM.render(
  <EJ.SunburstChart id = "sunburst1"
  zoomSettings ={zoomSettings}
  >
```



```
>
</EJ.SunburstChart>,
document.getElementById('sunburst')
);
```



### Zooming toolbar

By default, zooming toolbar will be enabled while zooming the segment. It contains both back and reset option.

You can align the zooming toolbar position by using [toolbarHorizontalAlignment](#) and [toolbarVerticalAlignment](#) property.

### JS

```
"use strict";
var zoomSettings = { enable: true, toolbarHorizontalAlignment: "left" };
ReactDOM.render(
  <EJ.SunburstChart id = "sunburst1"
  zoomSettings = {zoomSettings}
  >
</EJ.SunburstChart>,
document.getElementById('sunburst')
);
```



[Click](#) here to view the online demo sample of Zooming in the Sunburst Chart

### Animation

Sunburst chart allows you to animate the chart segments. You can enable animation using [enableAnimation](#) property.

### JS

```
"use strict";
ReactDOM.render(
  <EJ.SunburstChart id = "sunburst1"
  enableAnimation = {true}
  >
</EJ.SunburstChart>,
document.getElementById('sunburst')
);
```

### Animation Types

Sunburst chart provide options to animate the chart segments in different ways using [animationType](#) property.

FadeIn – It gradually changes opacity of the chart segment.

Rotation – During an animation, control rotate from 0 to 360 angle.

#### Fade In

The Fade In animation is enabled as follows

### JS

```
"use strict";
ReactDOM.render(
  <EJ.SunburstChart id = "sunburst1"
    enableAnimation = {true}
    animationType = "fadeIn">
  </EJ.SunburstChart>,
  document.getElementById('sunburst')
);
```



### Rotation

The following example shows how to enable rotation animation in ejSunburstChart

### JS

```
"use strict";
ReactDOM.render(
  <EJ.SunburstChart id = "sunburst1"
    enableAnimation = {true}
    animationType = "rotation">
  </EJ.SunburstChart>,
  document.getElementById('sunburst')
);
```



[Click](#) here to view the online demo sample of Animation in the Sunburst Chart.

### Appearance

The appearance of the Sunburst Chart can be customized as shown below

### Palette

The Sunburst Chart displays different segments in different colors by default. You can customize the color of each segment by providing a custom color palette of your choice by using the [palette](#) property.

### JS

```
"use strict";
var palette = ["#002e4d", "#005c99", "#008ae6", "#33adff", "#80ccff"];
ReactDOM.render(
  <EJ.SunburstChart id = "sunburst1"
    palette = {palette}>
  </EJ.SunburstChart>,
  document.getElementById('sunburst')
);
```

The Sunburst Chart rendered with palette colors



### Built-in Themes

The Sunburst Chart supports different themes.

*flatlight* flatdark *gradientlight* gradientdark *azure* azuredark *lime* limedark *saffron* saffrondark *gradient-azure* gradient-azuredark *gradient-lime* gradient-limedark *gradient-saffron* gradient-saffrondark

You can set your desired theme by using the [theme](#) property. **Flat light** is the default theme used in the Sunburst Chart.

### JS

```
"use strict";
ReactDOM.render(
  <EJ.SunburstChart id = "sunburst1"
    theme = "flatdark">
  </EJ.SunburstChart>,
  document.getElementById('sunburst')
);
```



### Methods

#### *redraw()*

Redraws the entire sunburst. You can call this method whenever you update, add or remove points from the data source or whenever you want to refresh the UI.

### HTML

```
<div id="sun"></div>
```

### JAVASCRIPT

```
ReactDOM.render(
  <EJ.SunburstChart id="default"></EJ.SunburstChart>,
  document.getElementById('sun')
);
function SunburstChartMethod() {
  var sunObj = $("#default").data("ejSunburstChart");
  sunObj.redraw();
};
```

#### *\_destroy()*

destroy the sunburst

### HTML

```
<div id="sun"></div>
```

### JAVASCRIPT

```
ReactDOM.render(
  <EJ.SunburstChart id="default"></EJ.SunburstChart>,
  document.getElementById('sun')
);
function SunburstChartMethod() {
  var sunObj = $("#default").data("ejSunburstChart");
  sunObj.destroy();
};
```

```
};
```

## Events

### *load*

Fires before loading.

#### **HTML**

```
<div id="sun"></div>
```

#### **JAVASCRIPT**

```
ReactDOM.render(  
  <EJ.SunburstChart id="default" load = {Load}></EJ.SunburstChart>,  
  document.getElementById('sun')  
)  
;  
function Load() {  
  // Do Something  
};
```

### *preRender*

Fires before rendering sunburst.

#### **HTML**

```
<div id="sun"></div>
```

#### **JAVASCRIPT**

```
ReactDOM.render(  
  <EJ.SunburstChart id="default" preRender = {PreRender}></EJ.SunburstChart>,  
  document.getElementById('sun')  
)  
;  
function PreRender() {  
  // Do Something  
};
```

### *loaded*

Fires after rendering sunburst.

#### **HTML**

```
<div id="sun"></div>
```

#### **JAVASCRIPT**

```
ReactDOM.render(  
  <EJ.SunburstChart id="default" loaded = {Loaded}></EJ.SunburstChart>,  
  document.getElementById('sun')  
)  
;  
function Loaded() {  
  // Do Something  
};
```

*dataLabelRendering*

Fires before rendering the data label

**HTML**

```
<div id="sun"></div>
```

**JAVASCRIPT**

```
ReactDOM.render(  
  <EJ.SunburstChart id="default" dataLabelRendering =  
    {DataLabelRendering}></EJ.SunburstChart>,  
  document.getElementById('sun')  
);  
function DataLabelRendering() {  
  // Do Something  
};
```

*segmentRendering*

Fires before rendering each segment

**HTML**

```
<div id="sun"></div>
```

**JAVASCRIPT**

```
ReactDOM.render(  
  <EJ.SunburstChart id="default" segmentRendering =  
    {SegmentRendering}></EJ.SunburstChart>,  
  document.getElementById('sun')  
);  
function SegmentRendering() {  
  // Do Something  
};
```

*titleRendering*

Fires before rendering sunburst title.

**HTML**

```
<div id="sun"></div>
```

**JAVASCRIPT**

```
ReactDOM.render(  
  <EJ.SunburstChart id="default" titleRendering =  
    {TitleRendering}></EJ.SunburstChart>,  
  document.getElementById('sun')  
);  
function TitleRendering() {  
  // Do Something  
};
```

*tooltipInitialize*

Fires during initialization of tooltip.

**HTML**

```
<div id="sun"></div>
```

**JAVASCRIPT**

```
ReactDOM.render(  
  <EJ.SunburstChart id="default" tooltipInitialize =  
    {TooltipInitialize}></EJ.SunburstChart>,  
  document.getElementById('sun')  
);  
function TooltipInitialize() {  
  // Do Something  
};
```

*pointRegionClick*

Fires after clicking the point in sunburst

**HTML**

```
<div id="sun"></div>
```

**JAVASCRIPT**

```
ReactDOM.render(  
  <EJ.SunburstChart id="default" pointRegionClick =  
    {PointRegionClick}></EJ.SunburstChart>,  
  document.getElementById('sun')  
);  
function PointRegionClick() {  
  // Do Something  
};
```

*pointRegionMouseMove*

Fires while moving the mouse over sunburst points

**HTML**

```
<div id="sun"></div>
```

**JAVASCRIPT**

```
ReactDOM.render(  
  <EJ.SunburstChart id="default" pointRegionMouseMove =  
    {PointRegionMouseMove}></EJ.SunburstChart>,  
  document.getElementById('sun')  
);  
function PointRegionMouseMove() {  
  // Do Something  
};
```

*drillDownClick*

Fires when clicking the point to perform drilldown.

**HTML**

```
<div id="sun"></div>
```

**JAVASCRIPT**

```
ReactDOM.render(  
  <EJ.SunburstChart id="default" drillDownClick =  
    {DrillDownClick}></EJ.SunburstChart>,  
  document.getElementById('sun')  
);  
function DrillDownClick() {  
  // Do Something  
};
```

*drillDownBack*

Fires when resetting drilldown points.

**HTML**

```
<div id="sun"></div>
```

**JAVASCRIPT**

```
ReactDOM.render(  
  <EJ.SunburstChart id="default" drillDownBack =  
    {DrillDownBack}></EJ.SunburstChart>,  
  document.getElementById('sun')  
);  
function DrillDownBack() {  
  // Do Something  
};
```

*drillDownReset*

Fires after resetting the sunburst points

**HTML**

```
<div id="sun"></div>
```

**JAVASCRIPT**

```
ReactDOM.render(  
  <EJ.SunburstChart id="default" drillDownReset =  
    {DrillDownReset}></EJ.SunburstChart>,  
  document.getElementById('sun')  
);  
function DrillDownReset() {  
  // Do Something  
};
```

## Slider

### Getting Started

This section helps to get started of the Slider component in a React application

#### Create a Slider

Refer the common React Getting Started Documentation to create an application and add necessary scripts and styles for rendering our ReactJS components.

Create a JSX file and use <EJ.Slider> syntax to render React Slider component. Add required properties to <EJ.Slider> tag element.

#### HTML

```
ReactDOM.render(  
  <EJ.Slider>  
  </EJ.Slider>,  
  document.getElementById('default-slider')  
)
```

Define an HTML element for adding Slider in the application and refer the JSX file created.

#### HTML

```
<div id="default-slider"></div>  
<script type="text/babel" src="sample.jsx">
```

This will render an default Slider component on executing.

#### Configure Properties

In the JSX, need to declare the Slider properties. Refer to the following code,.

#### HTML

```
ReactDOM.render(  
  <EJ.Slider value={20} >  
  </EJ.Slider>,  
  document.getElementById('default-slider')  
)
```

Run the above code to render the following output,



*Note: You can find the Slider properties from the [API reference](#) document.*

## Tab

### Overview

The **Tab** control is an interface where list of items are expanded from a single item. Each **Tab** panel defines its header text or header template, as well as a content template. **Tab** items are dynamically



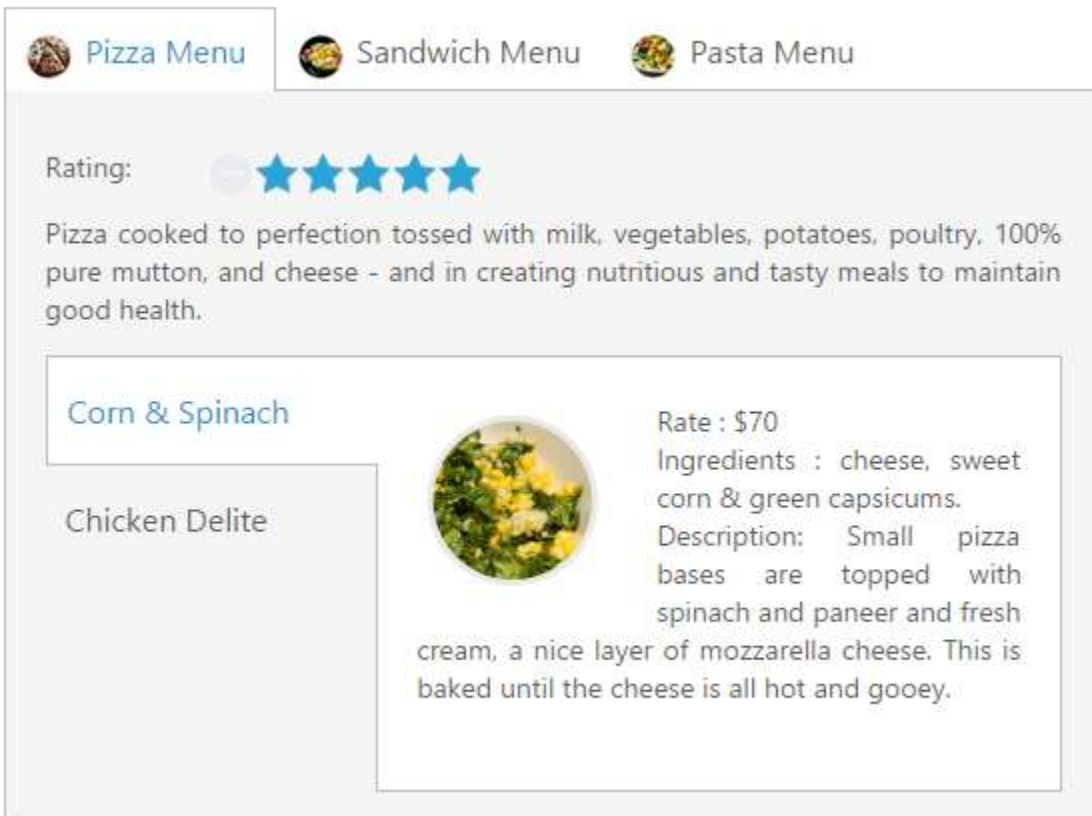
added and removed. **Tabs** can be loaded with AJAX content that is useful for building dashboards where space is limited.

### Key Features

- **Collapsible header:** All headers are collapsible.
- **AJAX load:** Load AJAX content in the Tab content panel.
- **Close button:** Provides a close button for each Tab item.
- **Custom event for expanding header:** Activate a Tab item on a single click, or bind custom events such as mouse over or mouse up to activate a Tab.
- **Header orientation:** Change the header position to the top or bottom of the control.
- **Persist:** Save the current model value to browser cookies to maintain the state.
- **Height style:** Adjust the content panel height.
- **RTL:** Tab headers and content can display right-to-left languages.
- **Add Tab dynamically:** New Tab items can be added and removed at run time.
- **Theme:** Essential JavaScript controls feature 17 built-in themes, six flat themes, six with gradient effects, bootstrap theme, two high-contrast, material theme and also support custom skin options to set user-defined themes.
- **Keyboard navigation:** You can interact with the control using the keyboard.

### Getting Started

This section explains briefly about how to create a **Tab** Control in your application with **JavaScript**. The **Essential JavaScript Tab** control is an interface that displays the content in multiple sections. Each **TabPanel** consists of **HeaderText** or **HeaderTemplate** as well as a **ContentTemplate**. **Tab** is useful for a dashboard that is having limited space. The following section guides you to on-demand customize the **Tab** for displaying Hotel menu items, its rating details and ingredients.



### Create Tab Control in React JS

**Essential JavaScript Tab** widget basically builds a dynamic interactive menu-driven interface from your content. The content can be text, graphics or HTML. You can create the **Tab** headers using `<UL>` and `<LI>` tags and content panels using `<div>` tags.

You can create a React application and add necessary scripts and styles with the help of the given [React Getting Started Documentation](#).

Create a JSX file for rendering Tab component using `<EJ.Tab>` syntax. Add required properties to it in `<EJ.Tab>` tag element

### HTML

```
var DefaultTab = React.createClass({
  render: function () {
    return (
      <div id="tab_default">
        <EJ.Tab width="100%">
          <ul>
            <li><a href="#pizza">Pizza Menu</a></li>
            <li><a href="#sandwich">Sandwich Menu</a></li>
            <li><a href="#pasta">Pasta Menu</a></li>
          </ul>
        </EJ.Tab>
      </div>
    );
  }
});
```

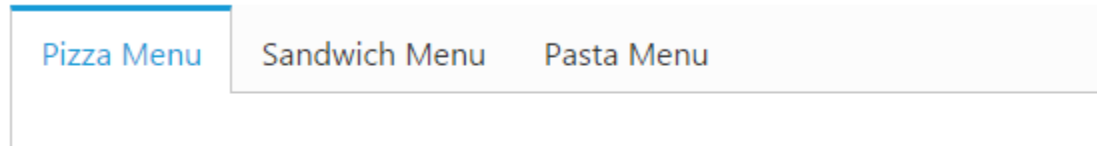
```
ReactDOM.render(<DefaultTab />, document.getElementById('tab-default'));
```

Define an HTML element for adding Tab in the application and refer the JSX file.

### HTML

```
<div id="tab-default"></div>
<script src="app/tab/default.js">
```

The following screen shot illustrates the **Tab** control with Header.



### Configure Content

In this application a detailed description is provided to each item. You can specify the contents in the **Tab** section within the <div> element.

### HTML

```
<div id="pizza" style="background-color: #F5F5F5">
  <!--Food item description-->
  <p>Pizza cooked to perfection tossed with milk, vegetables, potatoes,
  poultry, 100% pure mutton, and cheese - and in creating nutritious and tasty
  meals to maintain good health.</p>
</div>
```

You can provide more customization to the **Tab** with **rating** control as content in it for describing the item rating value.

### Create the Rating

The **Essential JavaScript Rating** control provides you an intuitive rating experience that allows you to select the number of stars that represents the rating. For more information about the rating please refer the following link:

<http://help.syncfusion.com/js/rating/getting-started>

The following code example explains you the **rating** control creation. The input element is used to create the **rating** control. Render the input element as **rating** control using the input element id. The code example can be placed within the content description <div> element to declare the rating control and description in the **Tab** section and it can be appended with the header initialization code section <div> element.

### HTML

```
var DefaultTab = React.createClass({
  render: function () {
    return (
      <div id="tab_default">
```

```

<EJ.Tab width="100%">
<ul>
<li><a href="#pizza">Pizza Menu</a></li>
<li><a href="#sandwich">Sandwich Menu</a></li>
<li><a href="#pasta">Pasta Menu</a></li>
</ul>
<div id="pizza" style="background-color: #F5F5F5">
<p>Rating:</p>
<!--Rating control declaration-->
<div class="dishRating">
<EJ.Rating value={3}></EJ.Rating>
</div>
<!--Food item description-->
<p>Pizza cooked to perfection tossed with milk, vegetables, potatoes,
poultry, 100% pure mutton, and cheese - and in creating nutritious and tasty
meals to maintain good health.</p>
</div>
</EJ.Tab>
</div>
);
}
});

```

To render the **rating** controls in the first **Tab** element refer the styles mentioned in the following code example.

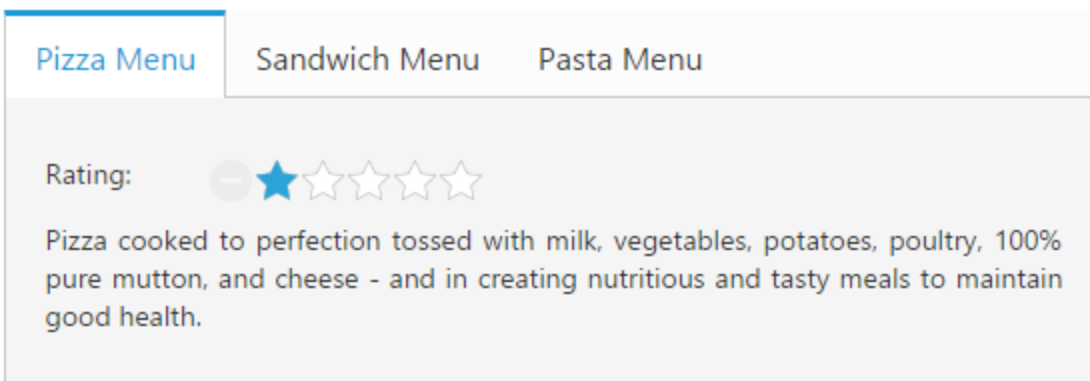
### CSS

```

<style type="text/css" class="cssStyles">
.dishRating {
position: absolute;
margin: -31px 0px 0px 80px;
}
</style>

```

The following screenshot illustrates the **Tab** content with rating control.



### Ajax Content Load (Load On Demand)

You can change the contents in sub **Tab** element periodically and you are provided with a support to change the contents without any problems. To achieve the content load, use the Load on Demand concept.

In Load On-Demand, the external HTML file with the necessary details is referred in <href> section during **Tab** header declaration section. When you click the **Tab** header, the Ajax automatically calls the content from the external files and displays in a **Tab** content section.

#### Sub Tab with Ajax Content

Each item is having a variety of options and these options are also specified in the limited space. So you can choose the **Tab** control that is used within the root **Tab** to specify more details.

The following code example illustrates to create the **Tab** control within the root **Tab** element. This HTML code is appended within the previous HTML code section. To render the child **Tab** with its header, add this code example within the first **Tab** <div> element.

#### HTML

```
var DefaultTab = React.createClass({
  render: function () {
    return (
      <div id="tab_default">
        <EJ.Tab width="100%">
          <ul>
            <li><span class="cornSpinach"></span>
              <a href="http://js.syncfusion.com/UG/Web/Tab-Content/cornSpainach.html">Corn
                & Spinach</a></li>
            <li><span class="chickenDelite"></span>
              <a href="http://js.syncfusion.com/UG/Web/Tab-Content/ChickenDelite.html">Chicken Delite</a></li>
          </ul>
        </EJ.Tab>
      </div>
    );
  }
});
```

The Load On-Demand supported HTML file content (cornSpinach.html)

#### HTML

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<body>
  <div class="e-content">
    
    <div class="ingredients">
      Rate      : $70<br />
      Ingredients : cheese, sweet corn &#38; green capsicums.
    <br />
    Description: Small pizza bases are topped with spinach and paneer and fresh cream, a nice layer of mozzarella cheese. This is baked until the cheese is all hot and gooey.
    </div>
```

```

</div>
</body>
</html>

```

The Load On Demand supported html file content (chickenDelite.html)

### HTML

```

<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<body>
<div class="e-content">

<div class="ingredients">
Rate      : $100<br />
Ingredients : cheese, chicken chunks, onions &#38; pineapple chunks.
<br />
Description: This is a tasty, elegant chicken dish that is easy to prepare.
</div>
</div>
</body>
</html>

```

**Note:** In Load On Demand, when the external files are referred from local the following error occurs.

XMLHttpRequest cannot load [http://js.syncfusion.com/UG/Web/Tab-Content/cornSpainach.html?\\_1399883825133](http://js.syncfusion.com/UG/Web/Tab-Content/cornSpainach.html?_1399883825133). No 'Access-Control-Allow-Origin' header is present on the requested resource.

To avoid these errors, the external files are hosted in the server to run this application.

The following code example is used to position the image and content in Load On Demand.

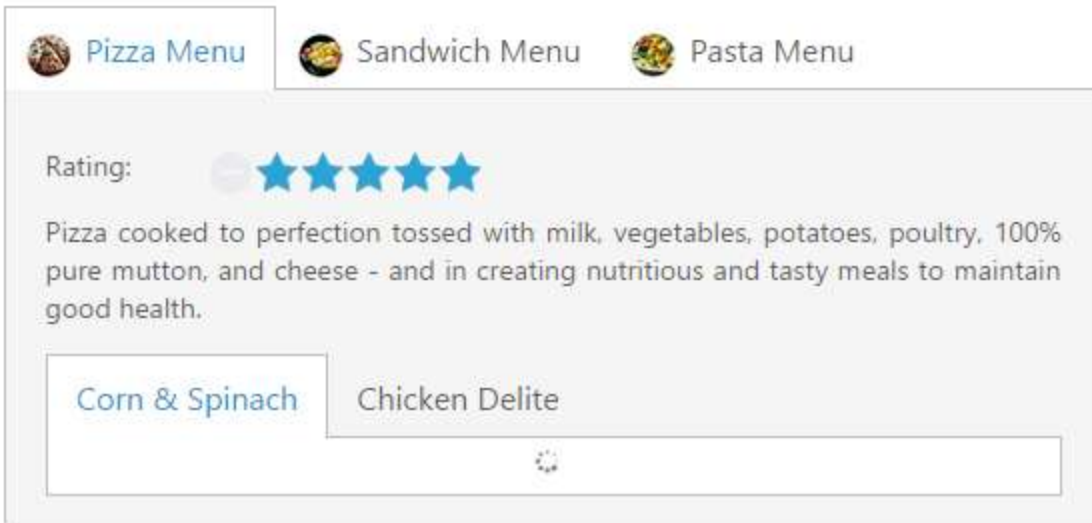
### CSS

```

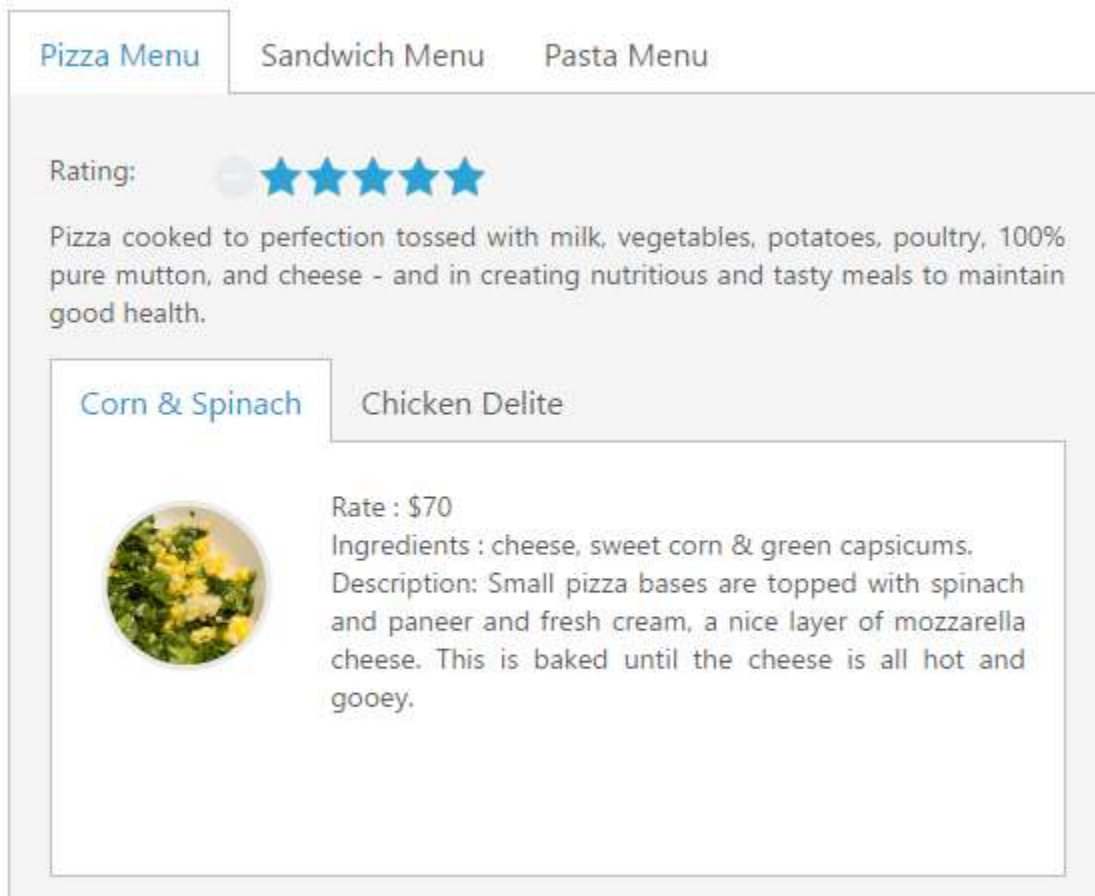
<style type="text/css" class="cssStyles">
/*reuse the previous rating control style section code*/
.ingredients {
height: 180px;
margin-top: 8px;
}
img {
float: left;
margin: 10px 26px 5px 1px;
}
</style>

```

At the time of Ajax call, the content fetched from external file referenced location is illustrated in the following screenshot.



The following screenshot illustrates the First **Tab** with the sub **Tab** control using Load on Demand.



### Orientation Change

In this application, the sub **Tab** orientation needs to be vertical. By default, **Tab** control renders in horizontal orientation. Change the orientation to vertical using the “**headerPosition**” property. The **Tab** header orientation is set as “**left**”.

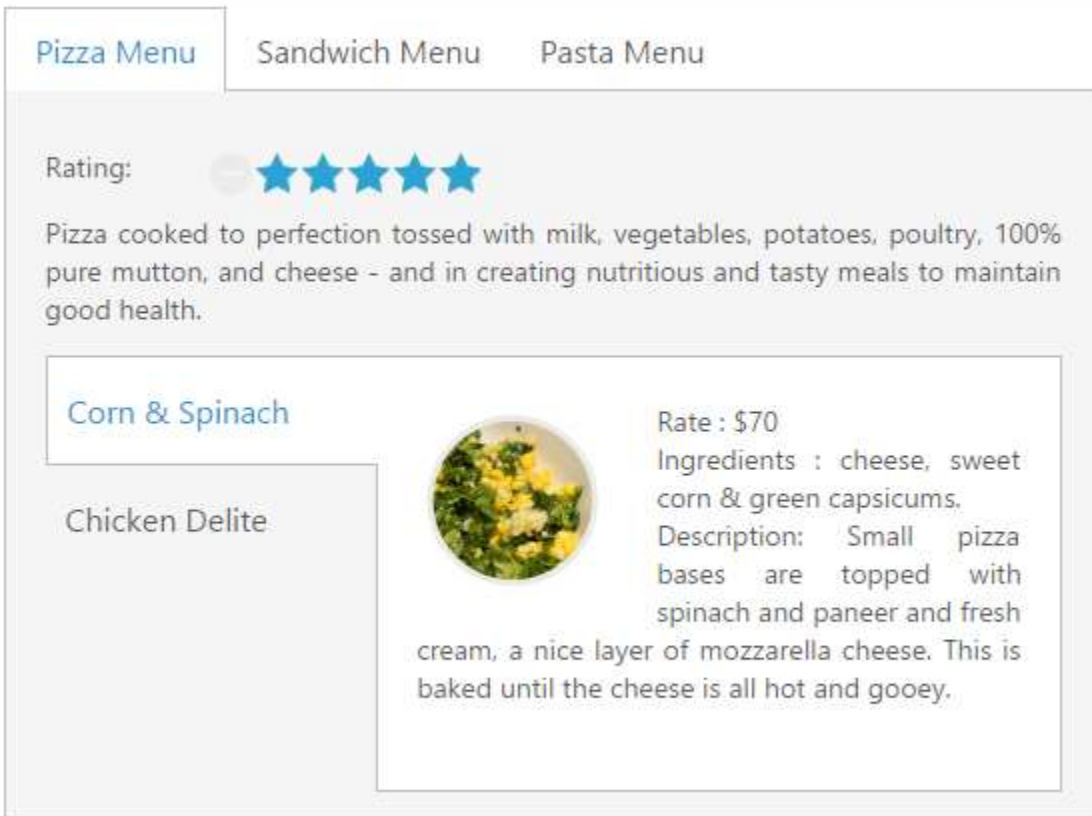
The following code example is used to render the sub **Tab** element in the vertical orientation.

#### JS

```
var DefaultTab = React.createClass({
  render: function () {
    return (
      <div id="tab_default">
        <EJ.Tab width="100%">
          <ul>
            <li><a href="#pizza">Pizza Menu</a></li>
            <li><a href="#sandwich">Sandwich Menu</a></li>
            <li><a href="#pasta">Pasta Menu</a></li>
          </ul>
          <div id="pizza" style="background-color: #F5F5F5">
            <p>Rating:</p>
            <!--Rating control declaration-->
            <div class="dishRating">
              <EJ.Rating value={3}></EJ.Rating>
            </div>
            <!--Food item description-->
            <p>Pizza cooked to perfection tossed with milk, vegetables, potatoes,
            poultry, 100% pure mutton, and cheese - and in creating nutritious and tasty
            meals to maintain good health.</p>
            <EJ.Tab headerPosition="left" height={200} heightAdjustMode="fill">
              <ul>
                <li><span class="cornSpinach"></span>
                <a href="http://js.syncfusion.com/UG/Web/Tab-Content/cornSpainach.html">Corn
                & Spinach </a></li>
                <li><span class="chickenDelite"></span>
                <a href="http://js.syncfusion.com/UG/Web/Tab-
                Content/ChickenDelite.html">Chicken Delite </a></li>
              </ul>
            </EJ.Tab>
          </div>
        </EJ.Tab>
      </div>
    );
  }
});
```

The following screenshot illustrates the sub **Tab** with vertical orientation.





### Header Image Customization

In this application, you have to set the **Tab** header image for each Tab item to specify image in `<span>` tag element during the **Tab** header declaration time.

The following code example is used for customizing the header image.

### CSS

```
<style type="text/css" class="cssStyles">
.dish {
display: inline-block;
vertical-align: middle;
margin: 0px -9px 0px 9px;
}
.pizzaImg {
background: url("http://js.syncfusion.com/UG/Web/Content/rsz_chicken-
delite.png") no-repeat;
height: 25px;
width: 25px;
}
/*reuse the previous header orientation code*/
</style>
```

The following code example is used to add the header image for the root **Tab** header element.

### HTML

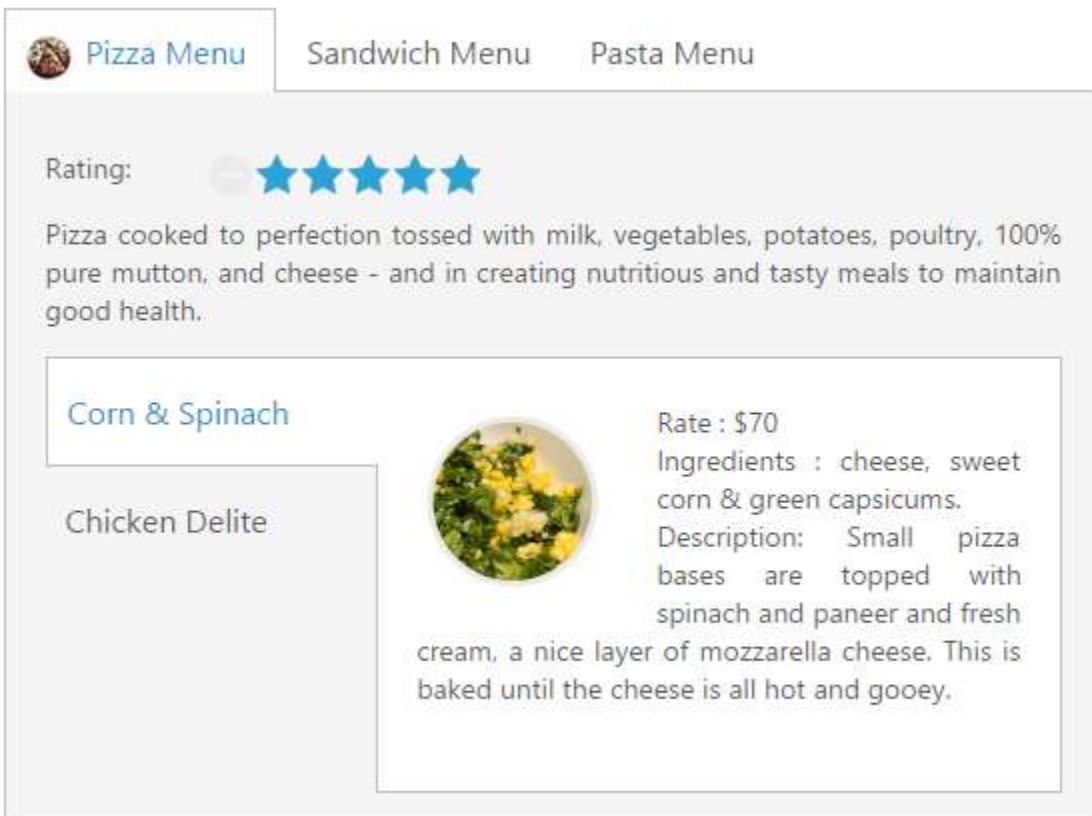
```
<div id="dishType" style="width: 550px">
```

```

<ul>
<li><span class="dish pizzaImg"></span><a href="#pizza">Pizza Menu</a></li>
<!-- reuse the remaining tab header -->
</ul>
<!-- reuse the previously defined first tab html content section-->
</div>

```

The following screenshot illustrates the **Tab** with the customized header image.



#### Configuring Contents to remaining Tab items

The second and third **Tab** contents are declared in the same method as of the first **Tab** content declaration. These **Tabs** also consist of rating and sub **Tab** controls. The sub **Tab** control contents are loaded in Load On Demand support.

The following code example can be placed within the previous image customization **HTML** code section.

#### HTML

```

<!--reuse the first tab header defined in previous image customization -->
<li><span class="dish sandwichImg"></span><a href="#sandwich">Sandwich
Menu</a></li>
<li><span class="dish pastaImg"></span><a href="#pasta">Pasta Menu</a></li>

```

Add the second **Tab** contents in <div> element during initialization.

#### HTML

```

<div id="sandwich" style="background-color: #F5F5F5">
<p>Rating:</p>
<!--Rating control declaration-->
<div class="dishRating">
<EJ.Rating value={3}></EJ.Rating>
</div>
<!--dish description-->
<p>Sandwich cooked to perfection tossed with bread, milk, vegetables,
potatoes, poultry, 100% pure mutton, and cheese - and in creating nutritious
and tasty meals to maintain good health.</p>
<!--sub Tab control, the contents loaded with load on demand-->
<EJ.Tab id="sandwichType">
<ul>
<li>
<a href="http://js.syncfusion.com/UG/Web/Tab-
Content/gardenVeggie.html">Garden Veggie </a></li>
<li>
<a href="http://js.syncfusion.com/UG/Web/Tab-
Content/chickenTikka.html">Chicken Tikka </a></li>
<li>
<a href="http://js.syncfusion.com/UG/Web/Tab-
Content/paneerTikka.html">Paneer Tikka </a></li>
</ul>
</EJ.Tab>
</div>

```

Add third **Tab** contents in <div> element during initialization.

### HTML

```

<div id="pasta" style="background-color: #F5F5F5">
<p>Rating:</p>
<!--Rating control declaration-->
<div class="dishRating">
<EJ.Rating value={3}></EJ.Rating>
</div>
<!--dish description-->
<p>Pasta cooked to perfection tossed with milk, vegetables, potatoes,
poultry, 100% pure mutton, and cheese - and in creating nutritious and tasty
meals to maintain good health.</p>
<!--sub Tab control, the contents loaded with load on demand-->
<EJ.Tab id="pastaType">
<ul>
<li>
<a href="http://js.syncfusion.com/UG/Web/Tab-
Content/khemmaPasta.html">Kheema Pasta </a></li>
<li>
<a href="http://js.syncfusion.com/UG/Web/Tab-Content/tunaPasta.html">Tuna
Pasta</a></li>
<li>
<a href="http://js.syncfusion.com/UG/Web/Tab-
Content/channaPasta.html">Channa Pasta
</a></li>
</ul>
</EJ.Tab>
</div>

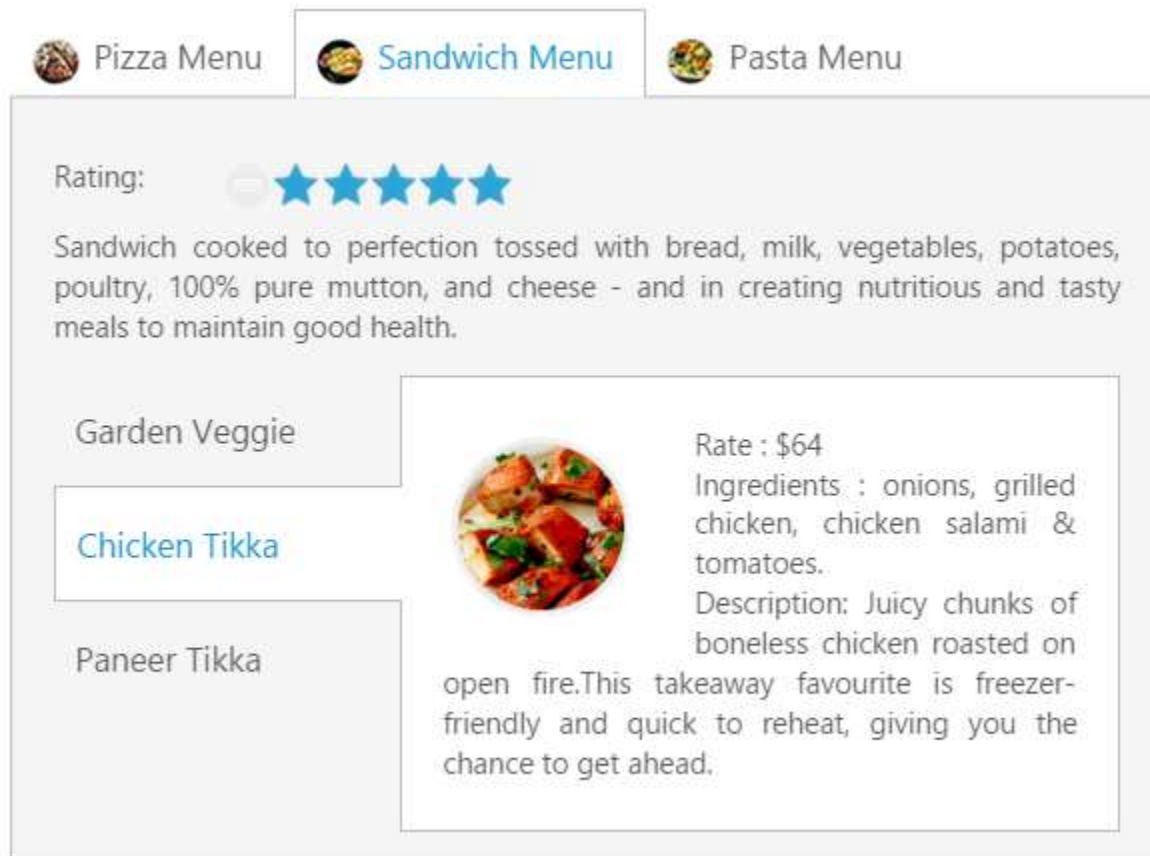
```

Apply the following styles to the **Tab**.

### CSS

```
<style type="text/css" class="cssStyles">
/*to reuse the previous style section code and following css*/
.sandwichImg, .pastaImg {
height: 25px;
width: 25px;
}
.sandwichImg {
background: url("http://js.syncfusion.com/UG/Web/Content/rsz_garden-
fresh.png") no-repeat;
}
.pastaImg {
background: url("http://js.syncfusion.com/UG/Web/Content/rsz_garden-
veggie.png") no-repeat;
}
</style>
```

The following screenshot illustrates you the second **Tab** contents in **Tab** and the final hotel menu with rating, description and ingredients of the item in the Tab interface.



## TagCloud

### Overview

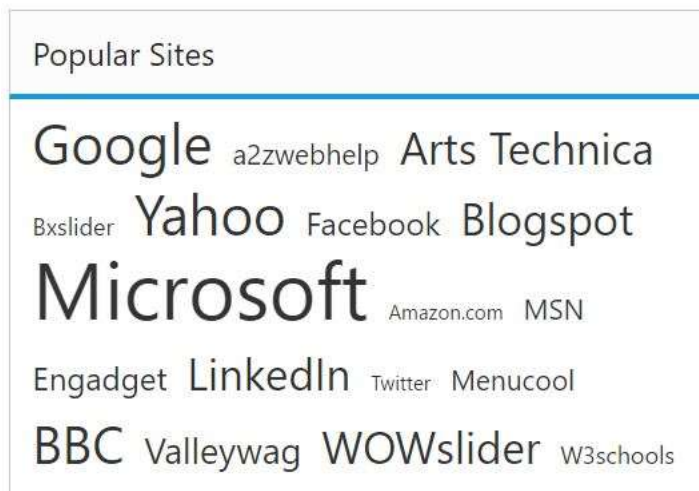
The JavaScript TagCloud control allows the user to display a list of links or tags with a structured cloud format where the importance of the tags can differentiate with varied font sizes, colors, and styles.

### Key Features

- **Data binding:** Supports data binding with JSON data as well as remote data.
- **Accessible:** Supports adding or removing tags dynamically.
- **Format:** Supports tag structure in both cloud format and list format.
- **Font-size:** Supports limiting the font size with a minimum and maximum range.
- **RTL:** Supports changing the TagCloud direction for right-to-left alignment.
- **Themes:** JavaScript controls include 12 built-in themes (6 flat and 6 gradient effects) and also support a custom skin option to set user defined themes.

### Getting Started

This section explains briefly about how to create a **TagCloud** in your application with **JavaScript**. The **TagCloud** can be easily configured to the div element in which the tags are placed. The following screenshot illustrates the functionality of a **TagCloud** widget with a list of the topmost search engines.



### Create TagCloud widget in React JS

You can create a React application and add necessary scripts and styles with the help of the given [React Getting Started Documentation](#).

Create a JSX file for rendering TagCloud component using <EJ.TabCloud> syntax. Add required properties to it in <EJ.TagCloud> tag element

### JAVASCRIPT

```
"use strict";
var websiteCollection = [
  { text: "Google", url: "http://www.google.co.in", frequency: 20 },
  { text: "a2zwebhelp", url: "http://www.a2zwebhelp.com", frequency: 3 },
  { text: "Arts Technica", url: "http://arstechnica.com/", frequency: 8 },
  { text: "Bxslider", url: "http://bxslider.com/examples", frequency: 2 },
  { text: "Yahoo", url: "http://in.yahoo.com/", frequency: 12 },
```

```

{ text: "Facebook", url: "https://www.facebook.com/", frequency: 5 },
{ text: "Blogspot", url: "http://www.blogspot.com/", frequency: 8 },
{ text: "Microsoft", url: "http://www.microsoft.com/", frequency: 20 },
{ text: "Amazon.com", url: "http://www.amazon.com/", frequency: 1 },
{ text: "MSN", url: "http://www.msn.com/", frequency: 3 },
{ text: "Engadget", url: "http://www.engadget.com/", frequency: 5 },
{ text: "LinkedIn", url: "http://www.linkedin.com/", frequency: 9 },
{ text: "Twitter", url: "http://www.Twitter.com/", frequency: 0 },
{ text: "Menucool", url: "http://www.menucool.com", frequency: 3 },
{ text: "BBC", url: "http://www.bbc.co.uk/", frequency: 11 },
{ text: "Valleywag", url: "http://valleywag.gawker.com/", frequency: 6 },
{ text: "WOWslider", url: "http://wowslider.com", frequency: 9 },
{ text: "W3schools", url: "http://www.w3schools.com/", frequency: 2 }
];
ReactDOM.render(
<EJ.TagCloud width="100%" cssClass="alignc" titleText= "Tech Sites"
dataSource={websiteCollection}>
</EJ.TagCloud>,
document.getElementById('tagcloud-default')
);

```

Define an HTML element for adding TabCloud in the application and refer the JSX file.

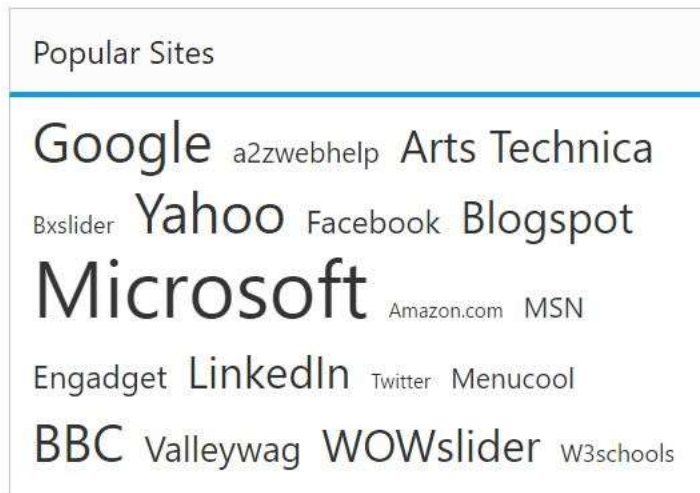
#### HTML

```

<div id="tagcloud-default"></div>
<script src="app/tagcloud/default.js">

```

The following screenshot displays the output of the above code example.



#### Set Min and Max Font Size

In the above code example, the **frequency** properties are used to set the min and max font size to the **TagCloud** list item.

#### JAVASCRIPT

```

var websiteCollection = [
{ text: "Google", url: "http://www.google.co.in", frequency: 20 },
{ text: "a2zwebhelp", url: "http://www.a2zwebhelp.com", frequency: 3 },

```

```

{ text: "Arts Technica", url: "http://arstechnica.com/", frequency: 8 },
{ text: "Bxslider", url: "http://bxslider.com/examples", frequency: 2 },
{ text: "Yahoo", url: "http://in.yahoo.com/", frequency: 12 },
{ text: "Facebook", url: "https://www.facebook.com/", frequency: 5 },
{ text: "Blogspot", url: "http://www.blogspot.com/", frequency: 8 },
{ text: "Microsoft", url: "http://www.microsoft.com/", frequency: 20 },
{ text: "Amazon.com", url: "http://www.amazon.com/", frequency: 1 },
{ text: "MSN", url: "http://www.msn.com/", frequency: 3 },
{ text: "Engadget", url: "http://www.engadget.com/", frequency: 5 },
{ text: "LinkedIn", url: "http://www.linkedin.com/", frequency: 9 },
{ text: "Twitter", url: "http://www.Twitter.com/", frequency: 0 },
{ text: "Menucool", url: "http://www.menucool.com", frequency: 3 },
{ text: "BBC", url: "http://www.bbc.co.uk/", frequency: 11 },
{ text: "Valleywag", url: "http://valleywag.gawker.com/", frequency: 6 },
{ text: "WOWslider", url: "http://wowslider.com", frequency: 9 },
{ text: "W3schools", url: "http://www.w3schools.com/", frequency: 2 }
];
ReactDOM.render(
<EJ.TagCloud width="100%" cssClass="alignnc" titleText= "Tech Sites"
dataSource={websiteCollection}>
</EJ.TagCloud>,
document.getElementById('tagcloud-default')
);

```

In the above code, the min font size is 0 and max font size is 20.

Set event to perform an operation

Here, you can set the **TagCloud** events such as create, mouseover, mouseout, click.

### HTML

```

<div id="tagcloud-default"></div>
<script src="app/tagcloud/default.js">

```

### JAVASCRIPT

```

var websiteCollection = [
{ text: "Google", url: "http://www.google.co.in", frequency: 12 },
{ text: "a2zwebhelp", url: "http://www.a2zwebhelp.com", frequency: 3 },
{ text: "Arts Technica", url: "http://arstechnica.com/", frequency: 8 },
{ text: "Bxslider", url: "http://bxslider.com/examples", frequency: 2 },
{ text: "Yahoo", url: "http://in.yahoo.com/", frequency: 12 },
{ text: "Facebook", url: "https://www.facebook.com/", frequency: 5 },
{ text: "Blogspot", url: "http://www.blogspot.com/", frequency: 8 },
{ text: "Microsoft", url: "http://www.microsoft.com/", frequency: 20 },
{ text: "Amazon.com", url: "http://www.amazon.com/", frequency: 1 },
{ text: "MSN", url: "http://www.msn.com/", frequency: 3 },
{ text: "Engadget", url: "http://www.engadget.com/", frequency: 5 },
{ text: "LinkedIn", url: "http://www.linkedin.com/", frequency: 9 },
{ text: "Twitter", url: "http://www.Twitter.com/", frequency: 0 },
{ text: "Menucool", url: "http://www.menucool.com", frequency: 3 },
{ text: "BBC", url: "http://www.bbc.co.uk/", frequency: 11 },
{ text: "Valleywag", url: "http://valleywag.gawker.com/", frequency: 6 },
{ text: "WOWslider", url: "http://wowslider.com", frequency: 9 },
{ text: "W3schools", url: "http://www.w3schools.com/", frequency: 2 }
];

```

```

var DefaultTagcloud = React.createClass({
  componentDidMount: function () {
    $scope.onCreate=function() {
      alert();
    }
    $scope.onmouseover=function() {
      alert();
    }
    $scope.onmouseout=function() {
      alert();
    }
    $scope.onclick=function() {
      alert();
    }
  },
  render: function () {
    return (
      <EJ.TagCloud width="100%" cssClass="alignc" titleText= "Tech Sites"
        dataSource={websiteCollection} create={oncreate} mouseOver={onmouseover}
        mouseOut={onmouseout} click={onclick}>
      </EJ.TagCloud>,
    );
  }
});
ReactDOM.render(<DefaultTagcloud />, document.getElementById('tagcloud-
default'));

```

In the above code example, the **alert()** function is used to show the events that happened.

## Tile

### Overview

Our Essential ReactJS Tile component displays opaque rectangles or squares and they are arrayed on the start screen like a grid pattern. Tapping or selecting a Tile, launches the app or does some other action that is represented by the Tile. Tiles are arranged in a group separated by columns that looks like a start screen of a device and it can be either static or live.

### Key Features

- **Live tile support:** Here the tiles are changed dynamically at specific time interval. *Badge value:* It is used to show the content count present behind the Tile. **Caption support:** Caption is used to set title for the corresponding tile.

### Getting Started

This section helps to get started with Essential ReactJS Tile component.

#### Create a Tile

Refer the common ReactJS [Getting Started](#) Documentation to create an application and add necessary scripts and styles for rendering our ReactJS components.

Create a JSX file and use <EJ.Tile> syntax to render React Tile component. Add required properties to <EJ.Tile> tag element.



**JAVASCRIPT**

```
ReactDOM.render(
  <EJ.Tile id="tileview">
  </EJ.Tile>,
  document.getElementById('tile')
);
```

Define an HTML element for adding Tile in the application and refer the JSX file created.

**HTML**

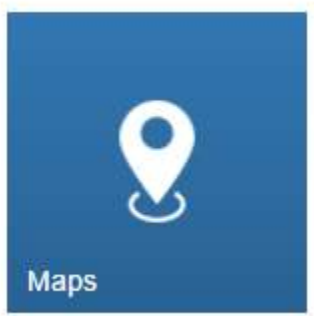
```
<div id="tile"></div>
<script type="text/babel" src="sample.jsx">
```

## Configuring properties

In the JSX, need to declare the Tile properties. Refer to the following code,

**JAVASCRIPT**

```
ReactDOM.render(
  <EJ.Tile id="tileview" imagePosition="fill" tileSize="medium" text="People">
  </EJ.Tile>,
  document.getElementById('tile')
);
```



You can easily design a home page using tile component for easy navigation. Therefore, you require many different sizes of tiles aligned in a grid-like manner. The tiles will be aligned automatically to define the necessary tile elements inside the wrapper element, that contains a column class. You can define all columns elements under the wrapper element with the tile group class to make 'n' number of tiles as a grouped tile. Add the below mentioned code to the corresponding view page.

**JAVASCRIPT**

```
ReactDOM.render(
  <div align="center" className="tileCenter">
  <div className="e-tile-group" id="groupTile">
  <div className="e-tile-column">
  <EJ.Tile id="tile1" imagePosition="fill" caption-text="People"
  tileSize="medium"
  imageUrl='http://js.syncfusion.com/ug/web/content/tile/people_1.png'
  text="People">
```

```

</EJ.Tile>
<div className="e-tile-small-col-2">
<EJ.Tile id="tile2" imagePosition="center" tileSize="small"
imageUrl='http://js.syncfusion.com/ug/web/content/tile/alerts.png'>
</EJ.Tile>
<EJ.Tile id="tile3" imagePosition="center" tileSize="small"
imageUrl='http://js.syncfusion.com/ug/web/content/tile/bing.png'>
</EJ.Tile>
<EJ.Tile id="tile4" tileSize="small"
imageUrl='http://js.syncfusion.com/ug/web/content/tile/camera.png'>
</EJ.Tile>
<EJ.Tile id="tile5" tileSize="small" imagePosition="center"
imageUrl='http://js.syncfusion.com/ug/web/content/tile/messages.png'>
</EJ.Tile>
</div>
<EJ.Tile id="tile6" tileSize="medium" imagePosition="center"
imageUrl='http://js.syncfusion.com/ug/web/content/tile/games.png'
text="Play">
</EJ.Tile>
<EJ.Tile id="tile7" tileSize="medium"
imageUrl='http://js.syncfusion.com/ug/web/content/tile/map.png' text="Maps">
</EJ.Tile>
<EJ.Tile id="tile8" tileSize="wide"
imageUrl='http://js.syncfusion.com/ug/web/content/tile/sports.png'
text="Sports" imagePosition="fill">
</EJ.Tile>
</div>
<div className="e-tile-column">
<EJ.Tile id="tile9" tileSize="medium" imagePosition="fill"
imageUrl='http://js.syncfusion.com/ug/web/content/tile/people_2.png'
text="People">
</EJ.Tile>
<EJ.Tile id="tile10" tileSize="medium" imagePosition="center"
imageUrl='http://js.syncfusion.com/ug/web/content/tile/pictures.png'
text="Photo">
</EJ.Tile>
<EJ.Tile id="tile11" tileSize="wide" imagePosition="center"
imageUrl='http://js.syncfusion.com/ug/web/content/tile/weather.png'
text="Weather">
</EJ.Tile>
<EJ.Tile id="tile12" tileSize="medium" imagePosition="center"
imageUrl='http://js.syncfusion.com/ug/web/content/tile/music.png'
text="Music">
</EJ.Tile>
<EJ.Tile id="tile13" tileSize="medium" imagePosition="center"
imageUrl='http://js.syncfusion.com/ug/web/content/tile/favs.png'
text="Favorites">
</EJ.Tile>
</div>
</div>
</div>,
document.getElementById('tile')
);

```

Run the above code to get the following output.



### Template support with Tile Component

To customize the Tile images with template feature by using `imageTemplateId` property of Tile component. Add the below mentioned code to the corresponding view page.

#### HTML

```
ReactDOM.render(
  <div class="e-tile-group" id="groupTile">
    <div class="e-tile-column">
      <EJ.Tile id="tileview" imagePosition="fill" tileSize="wide"
        imageTemplateId="imageTemplate" text="Windows Store">
      </EJ.Tile>
    </div>
    <div id="imageTemplate">
      <div id="appimage">
      </div>
      <div class="tileMargin">
        <span class="caption">Google Search</span><br />
        <span class="description">The world's information</span><br />
        <span class="sub">Free</span>
      </div>
    </div>
  </div>,
  document.getElementById('tile')
```

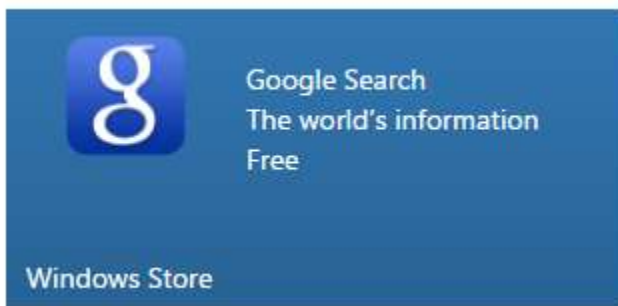
```
);
```

Add the following styles to customize the tile images with template support.

### CSS

```
<style>
#appimage {
background-image:
url("http://js.syncfusion.com/UG/mobile/content/google.png");
background-position: center center;
background-repeat: no-repeat;
background-size: 50% auto;
display: table-cell;
width: 45%;
}
.tileMargin {
display: table-cell;
padding-top: 25px;
}
.e-tile-template {
display: table;
height: 100%;
width: 100%;
}
</style>
```

Run the above code to get the following output.



**Note:** To get the complete API list for all the Syncfusion component's properties from the [API reference](#)

## TimePicker

### Overview

**TimePicker** control allows you to select the time value with a specific format.

### Key Features

- Time format: Supports all valid time formats.
- Localization: Supports localization to different cultures.
- Persist: Supports state maintenance during page refresh.

- RTL: Support for Right to Left alignment of content in TimePicker control.
- Themes: Essential JavaScript controls feature 12 built-in themes (six flat themes and six with gradient effects), and also supports custom skin options to set user-defined themes.

## Getting Started

This section explains you how to render and configure TimePicker component in a ReactJS application.

### Adding Scripts and Style references

Create an HTML page and refer the necessary script and CSS dependency files in your application with the help of given [React Getting Started Documentation](#).

#### Example

##### HTML

```
<body>
<input id="timepicker"/>
</body>
```

### Configure Properties

#### Value

This property specifies the list of time range to be disabled in TimePicker control. To know more about TimePicker properties please refer the [API reference](#)

##### HTML

```
ReactDOM.render(
  <EJ.TimePicker id="timepicker" value={"10:30 PM"} >
</EJ.TimePicker>,
  document.getElementById('dtp')
);
```

#### DisableTimeRanges

This property specifies the list of time range to be disabled in TimePicker control. To know more about TimePicker properties please refer the [API reference](#)

##### HTML

```
var disableTime= [{ startTime: "3:00 AM", endTime: "6:00 AM" },
{ startTime: "1:00 PM", endTime: "3:00 PM" },
{ startTime: "8:00 PM", endTime: "10:00 PM" }];
ReactDOM.render(
  <EJ.TimePicker id="timepicker" disableTimeRanges={disableTime} >
</EJ.TimePicker>,
  document.getElementById('dtp')
);
```

In the JSX, need to declare the Timepicker properties. Refer to the following code.,

##### HTML

```
ReactDOM.render(
  <EJ.TimePicker value={20} >
</EJ.TimePicker>,
```

```
document.getElementById('timepicker')
);
```

With ReactJS, components can be initialized in two ways.

1. Using jsx Template
2. Without using jsx Template

#### *Using jsx Template*

You can render EJ component by using JSX template, wherein the document will be converted to its equivalent JS file.

1. Create an input element with an ID in the HTML file.

#### **HTML**

```
<body>
<div id="timepicker"></div>
</body>
```

2. Create a JSX file and render Timepicker component by using the below code snippet.

#### **HTML**

```
ReactDOM.render(
  <EJ.TimePicker id="timepicker">
</EJ.TimePicker>,
  document.getElementById('timepicker')
);
```

3. Refer the JSX file created in last step in the HTML file as given below.

#### **HTML**

```
<body>
<div id="timepicker"></div>
<!-- timepicker.jsx created in previous step-->
<script type="text/babel" src="timepicker.jsx">
</script>
</body>
```

Now the jsx file will be compiled into its equivalent JavaScript file by means of Babel.

#### *Without using jsx Template*

The Timepicker can be created from a HTML **INPUT** element with the HTML **id** attribute set to it. Refer to the following code example.

#### **HTML**

```
<body>
```

```
<input id="timepicker"/>
</body>
```

## JAVASCRIPT

```
<script>
ReactDOM.render(
  React.createElement(EJ.Chart, {id: "timepicker"}
),
document.getElementById('timepicker')
);
</script>
```

Execute the above code to render Timepicker component.



*Note: You can find the TimePicker properties from the [API reference](#) document.*

## ToggleButton

### Overview

The **ToggleButton** allows you to perform the toggle option by using checked and unchecked state. This **ToggleButton** can be helpful to you to check their states. The **ToggleButton** control displays both text and images. The text displayed on the **ToggleButton** is contained in the text property. The **ToggleButton** control displays images using the sprite CSS Class and **ImagePosition** properties. The **ToggleButton** has theme support.

### Key Features

- **Trendy Look** : Rich appearance with theme Support
- **RTL** : Supports for right to left alignment
- **Text and Image** : Supports both text and image as ToggleButton content in both On/Off state
- **Built-in Icon** : Supports the built-in icon libraries
- **Easy Customization** : The customization of button control to any form is made simple

### Getting Started

Using the following steps, you can create a React ToggleButton component. The basic rendering of React ToggleButton can be achieved with default functionality.

#### Create an ToggleButton

You can create a React application and add necessary scripts and styles with the help of the given [React Getting Started Documentation](#).

Create a JSX file for rendering ToggleButton component using <EJ.ToggleButton> syntax. Add required properties to it in <EJ.ToggleButton> tag element

## HTML

```
ReactDOM.render(
  <EJ.ToggleButton>
</EJ.ToggleButton>,
document.getElementById('togglebtn')
```

```
);
```

Define an HTML element for adding ToggleButton in the application and refer the JSX file created.

### HTML

```
<div id="togglebtn"></div>
<script type="text/babel" src="sample.jsx">
```

This will render an empty ToggleButton component on executing.

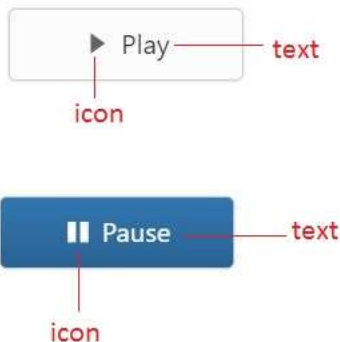
### Configure Properties

In the JSX, need to declare the ToggleButton properties. Refer to the following code,

### HTML

```
ReactDOM.render(
  <EJ.ToggleButton id="togglebtn" ejHtmlAttributes={{type:"checkbox"}}
    size={ej.ButtonSize.Large} showRoundedCorner={true}
    contentType={ej.ContentType.TextAndImage} defaultPrefixIcon="e-icon e-
mediaplay" activePrefixIcon="e-icon e-mediapause" defaultText="Play"
    activeText="Pause"></EJ.ToggleButton>,
  document.getElementById('togglebtn')
);
```

Run the above code to render the following output,



*Note: You can find the ToggleButton properties from the [API reference](#) document.*

## TreeGrid

### Overview

**The Essential React JS TreeGrid** is an efficient control designed for representing the hierarchical data in a tabular format, combining the visual representation of Grid and TreeView controls; it represents the data from datasource such as array of JSON objects, ej.DataManager or self-referential datasource.

### Key Features

- **Editing** - Offers cell editing Mode for editing the item along each column
- **Sorting** - Supports *n* levels of sorting.



- **Column Template** - Offers to render the customized column for an item also with customized expand-collapse icon.
- **Virtualization** - Supports rendering huge amount of hierarchical data at once.
- **User Interaction** - Supports tooltip for each cell or even only for expander cell in the grid, expand collapse at ease, single and multiple row selection, columns resizing.

## Getting Started

This section helps to understand the getting started of the React JS TreeGrid with the step-by-step instructions.

### Create your first TreeGrid in React JS

To get started Syncfusion React JS application refer [this](#) page for basic control integration and script references.

The **Essential React JS TreeGrid** has been designed to represent and edit the hierarchical data.

This section explains how to create a TreeGrid widget in your application with hierarchical data source and enable sorting and editing. The following screenshot displays the output.

Task Id	Task Name	Start Date	End Date	Progress
1	▲ Planning	02/03/2014	02/07/2014	100
2	Plan timeline	02/03/2014	02/07/2014	100
3	Plan budget	02/03/2014	02/07/2014	100
4	Allocate resources	02/03/2014	02/07/2014	100
5	Planning complete	02/07/2014	02/07/2014	0
6	▲ Design	02/10/2014	02/14/2014	86
7	Software Specification	02/10/2014	02/12/2014	60
8	Develop prototype	02/10/2014	02/12/2014	100
9	Get approval from customer	02/13/2014	02/14/2014	100
10	Design Documentation	02/13/2014	02/14/2014	100
11	Design complete	02/14/2014	02/14/2014	0

The required ReactJS script dependencies as follows. And you can also refer [React](#) to know more about react js.

- react.min.js - <http://cdn.syncfusion.com/js/assets/external/react.min.js>
- react-dom.min.js - <http://cdn.syncfusion.com/js/assets/external/react-dom.min.js>
- browser.min.js - <http://cdnjs.cloudflare.com/ajax/libs/babel-core/5.8.34/browser.min.js>
- ej.web.react.min.js - <http://cdn.syncfusion.com/{{ site.releaseversion }}/js/common/ej.web.react.min.js> 1. Create HTML file and add the following necessary script and css files to the HTML file.

### HTML

```
<!DOCTYPE html>
```

```

<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta name="viewport" content="width=device-width, initial-scale=1.0"/>
<meta charset="utf-8" />
<link href="http://cdn.syncfusion.com/{{ site.releaseversion
}}/js/web/flat-azure/ej.web.all.min.css" rel="stylesheet"/>
<script
src="http://cdn.syncfusion.com/js/assets/external/react.min.js"></script>
<script src="http://cdn.syncfusion.com/js/assets/external/react-
dom.min.js"></script>
<script src="https://cdnjs.cloudflare.com/ajax/libs/babel-
core/5.8.34/browser.min.js"></script>
<script src="http://cdn.syncfusion.com/js/assets/external/jquery-
1.10.2.min.js"></script>
<script
src="http://cdn.syncfusion.com/js/assets/external/jsrender.min.js"></script>
<script
src="http://cdn.syncfusion.com/js/assets/external/jquery.globalize.min.js">
</script>
<script
src="http://cdn.syncfusion.com/js/assets/external/jquery.easing.1.3.min.js">
</script>
<script src="http://cdn.syncfusion.com/{{ site.releaseversion
}}/js/web/ej.web.all.min.js" type="text/javascript"></script>
<script src="http://cdn.syncfusion.com/{{ site.releaseversion
}}/js/common/ej.web.react.min.js"></script>
</head>
<body>
<!--Add TreeGrid control here -->
</body>
</html>

```

### Create a TreeGrid

Control can be initialized in two ways.

- Using jsx Template
- Without using jsx Template

### Using jsx Template

By using the jsx template, we can create the html file and jsx file. The .jsx file can be convert to .js file and it can be referred in html page.

Please refer to the code of HTML file.

### HTML

```

<div id="TreeGrid-default"></div>
<script type="text/babel" src="app.jsx">
</script>

```

Create an app.JSX file and paste the following content

### JS

```

var ProjectData = [{

```

```

taskID: 2,
taskName: "Planning",
startDate: "02/03/2014",
endDate: "02/07/2014",
duration: 10,
progress: 56,
subtasks: [{
  taskID: 3,
  taskName: "Plan timeline",
  startDate: "02/03/2014",
  endDate: "02/07/2014",
  duration: 5,
  progress: "100"
},
//...
}]
var columns = [{ field: 'taskID', headerText: 'Task Id', width: '45'},
{ field: 'taskName', headerText: 'Task Name'},
{ field: 'startDate', headerText: 'Start Date'},
{ field: 'endDate', headerText: 'End Date'},
{ field: 'duration', headerText: 'Duration'},
{ field: 'progress', headerText: 'Progress'}
];
ReactDOM.render(
<EJ.TreeGrid dataSource={projectData} childMapping="subtasks"
columns={columns} treeColumnIndex={1}>
</EJ.TreeGrid>,
document.getElementById('TreeGrid-default')
);

```

TreeGrid widget is displayed as the output in the following screenshot.

Task Id	Task Name	Start Date	End Date	Progress
1	▲ Planning	02/03/2014	02/07/2014	100
2	Plan timeline	02/03/2014	02/07/2014	100
3	Plan budget	02/03/2014	02/07/2014	100
4	Allocate resources	02/03/2014	02/07/2014	100
5	Planning complete	02/07/2014	02/07/2014	0
6	▲ Design	02/10/2014	02/14/2014	86
7	Software Specification	02/10/2014	02/12/2014	60
8	Develop prototype	02/10/2014	02/12/2014	100
9	Get approval from customer	02/13/2014	02/14/2014	100
10	Design Documentation	02/13/2014	02/14/2014	100
11	Design complete	02/14/2014	02/14/2014	0

### Enable Sorting




The TreeGrid control has sorting functionality, to arrange the data in ascending or descending order based on a particular column.

### Multicolumn Sorting

Enable the multicolumn sorting in TreeGrid by setting [allowMultiSorting](#) as `true`. You can sort multiple columns in TreeGrid, by selecting the desired column header while holding the `Ctrl` key.

### JS

```
ReactDOM.render(
  <EJ.TreeGrid allowSorting={true} allowMultiSorting={true}>
    </EJ.TreeGrid>,
  document.getElementById('TreeGrid-default')
);
```

Task Id	Task Name  	Start Date	End Date 	Progress
1	Planning	02/03/2014	02/07/2014	100
4	Allocate resources	02/03/2014	02/07/2014	100
3	Plan budget	02/03/2014	02/07/2014	100
2	Plan timeline	02/03/2014	02/07/2014	100
5	Planning complete	02/07/2014	02/07/2014	0
6	Design	02/10/2014	02/14/2014	86
8	Develop prototype	02/10/2014	02/12/2014	100
7	Software Specification	02/10/2014	02/12/2014	60
11	Design complete	02/14/2014	02/14/2014	0
10	Design Documentation	02/13/2014	02/14/2014	100
9	Get approval from customer	02/13/2014	02/14/2014	100

### Enable Editing

You can enable Editing in TreeGrid by using the [editSettings](#) property as follows.

### JS

```
var editSettings = {
  allowAdding: true,
  allowEditing: true,
  allowDeleting: true,
  editMode: 'cellEditing',
  rowPosition: 'belowSelectedRow'
};
ReactDOM.render(
  <EJ.TreeGrid editSettings={editSettings} >
    </EJ.TreeGrid>,
  document.getElementById('TreeGrid-default')
);
```

And also, the following editors are provided for editing support in TreeGrid control.

- string
- boolean
- numeric
- dropdown
- datepicker
- datetimepicker

You can set the editor type for a particular column as follows.

### JS

```
var ProjectData = [{
  taskID: 2,
  taskName: "Planning",
  startDate: "02/03/2014",
  endDate: "02/07/2014",
  duration: 10,
  progress: 56,
  subtasks: [{
    taskID: 3,
    taskName: "Plan timeline",
    startDate: "02/03/2014",
    endDate: "02/07/2014",
    duration: 5,
    progress: "100"
  },
  //...
}]
var columns = [{ field: 'taskID', headerText: 'Task Id', width: '45',
editType: 'numericedit' },
{ field: 'taskName', headerText: 'Task Name', editType: 'stringedit' },
{ field: 'startDate', headerText: 'Start Date', editType: 'datepicker' },
{ field: 'endDate', headerText: 'End Date', editType: 'datepicker'},
{ field: 'duration', headerText: 'Duration', editType: 'numericedit' },
{ field: 'progress', headerText: 'Progress', editType: 'numericedit' }
];
ReactDOM.render(
<EJ.TreeGrid columns={columns}>
</EJ.TreeGrid>,
document.getElementById('TreeGrid-default')
);
```

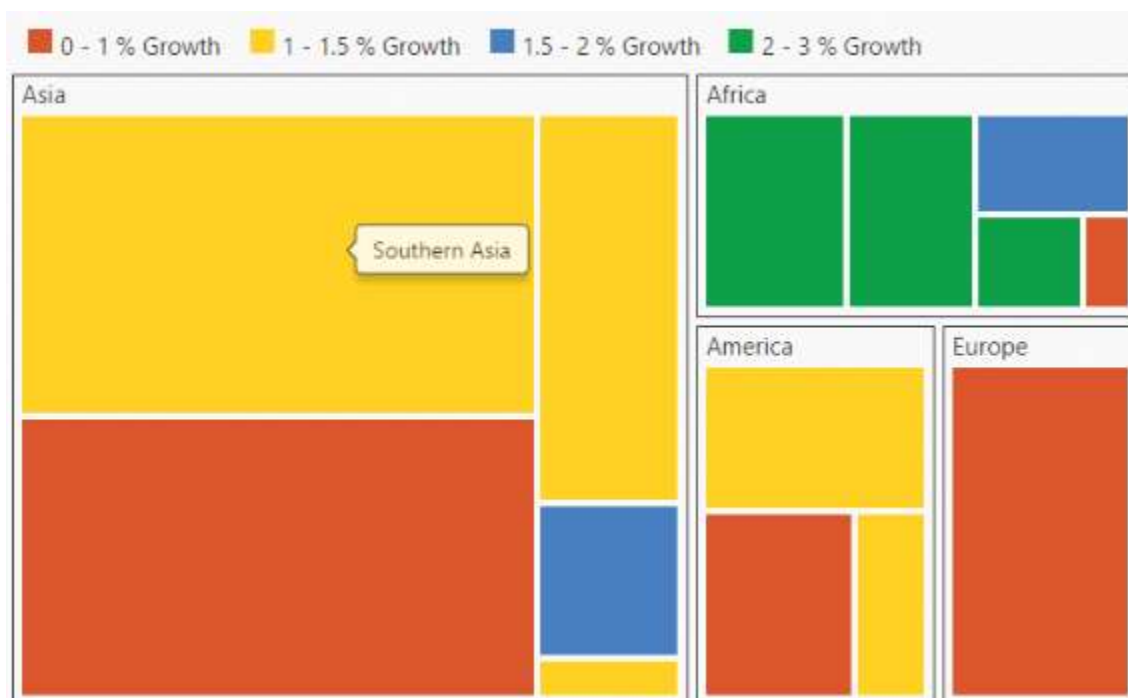
The output of the DateTimePicker editor in TreeGrid control is as follows.

Task Id	Task Name	Start Date	End Date	Progress
1	Planning	02/03/2014	02/07/2014	100
2	Plan timeline	02/03/2014	02/07/2014	100
3	Plan budget	02/03/2014	02/07/2014	100
4	Allocate resources	February 2014		100
5	Planning complete	Su Mo Tu We Th Fr Sa		0
6	Design	26 27 28 29 30 31 1		86
7	Software Specification	2 3 4 5 6 7 8		60
8	Develop prototype	9 10 11 12 13 14 15		100
9	Get approval from customer	16 17 18 19 20 21 22		100
10	Design Documentation	23 24 25 26 27 28 1		100
11	Design complete	Today		0

## TreeMap

### Getting Started

- This section explains briefly about how to create a TreeMap in your application with ReactJS.
- Here you can learn how to configure a TreeMap control in a real-time scenario where it is used to visually represent the percentage of growth in population in each continent.
- It also provides a walk-through on some of the customization features available in TreeMap control.



## Create a TreeMap

You can easily create the TreeMap widget by following the below steps.

### Adding Script Reference

Create an **HTML** page and add the scripts references in the order mentioned in the following code example.

- [jQuery](#) 1.10.2 and later versions
- [jsRender](#) - to render the templates

The required ReactJS script dependencies as follows. And you can also refer [React](#) to know more about react js.

- react.min.js - <http://cdn.syncfusion.com/js/assets/external/react.min.js>
- react-dom.min.js - <http://cdn.syncfusion.com/js/assets/external/react-dom.min.js>
- ej.web.react.min.js - <http://cdn.syncfusion.com/{{ site.releaseversion }}/js/common/ej.web.react.min.js>

To get started, you can use the `ej.web.all.min.js` file that encapsulates all the `ej` controls and frameworks in one single file.

### HTML

```
<!DOCTYPE html>
<html>
<head>
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<meta name="description" content="Essential Studio for React JS">
<meta name="author" content="Syncfusion">
<title>Getting Started for Ribbon React JS</title>
<!-- Essential Studio for JavaScript theme reference -->
<link href="http://cdn.syncfusion.com/{{ site.releaseversion }}/js/web/flat-
azure/ej.web.all.min.css" rel="stylesheet" />
<!-- Essential Studio for JavaScript script references -->
<script src="http://cdn.syncfusion.com/js/assets/external/jquery-
3.0.0.min.js"></script>
<script
src="http://cdn.syncfusion.com/js/assets/external/react.min.js"></script>
<script src="http://cdn.syncfusion.com/js/assets/external/react-
dom.min.js"></script>
<script src="http://cdn.syncfusion.com/{{ site.releaseversion
}}/js/web/ej.web.all.min.js"></script>
<script src="http://cdn.syncfusion.com/{{ site.releaseversion
}}/js/common/ej.web.react.min.js"></script>
<!-- Add your custom scripts here -->
</head>
<body>
</body>
</html>
```

**Note:** 1. In production, we highly recommend you to use our [custom script generator](#) to create custom script file with required controls and its dependencies only. Also to reduce the file size further please use [GZip compression](#) in your server.

2. For themes, you can use the `ej.web.all.min.css` CDN link from the code snippet given. To add the themes in your application, please refer to [this link](#).

### Control Initialization

Control can be initialized in two ways.

- Using jsx Template
- Without using jsx Template

### Using jsx Template

By using the jsx template, we can create the html file and jsx file. The `.jsx` file can be convert to `.js` file and it can be referred in html page.

### Initialize TreeMap

1.Create a

tag with specific id

#### HTML

```
<!DOCTYPE html>
<html>
<body>
<div id="treemap-default" style="height:99%;"></div>
<script src="app/treemap/default.js"></script>
</body>
</html>
```

2.The `dataSource` property of the TreeMap accepts the collection values as input.Populate the datasource of the TreeMap data as JSON object. For example, you can use population data of countries to generate TreeMap data as illustrated in the following code sample.

#### HTML

```
var population_data = [
{ Continent: "Asia", Region: "Southern Asia", Growth: 1.32, Population: 1749046000 },
{ Continent: "Asia", Region: "Eastern Asia", Growth: 0.57, Population: 1620807000 },
{ Continent: "Asia", Region: "South-Eastern Asia", Growth: 1.20, Population: 618793000 },
{ Continent: "Asia", Region: "Western Asia", Growth: 1.98, Population: 245707000 },
{ Continent: "Asia", Region: "Central Asia", Growth: 1.43, Population: 64370000 },
{ Continent: "Europe", Region: "Europe", Growth: 0.10, Population: 742452000 },
{ Continent: "America", Region: "South America", Growth: 1.06, Population: 406740000 },
{ Continent: "America", Region: "Northern America", Growth: 0.85, Population: 355361000 },
```



```
{ Continent: "America", Region: "Central America", Growth: 1.40, Population: 167387000 },
{ Continent: "Africa", Region: "Eastern Africa", Growth: 2.89, Population: 373202000 },
{ Continent: "Africa", Region: "Western Africa", Growth: 2.78, Population: 331255000 },
{ Continent: "Africa", Region: "Northern Africa", Growth: 1.70, Population: 210002000 },
{ Continent: "Africa", Region: "Middle Africa", Growth: 2.79, Population: 135750000 },
{ Continent: "Africa", Region: "Southern Africa", Growth: 0.91, Population: 60425000 }
];
```

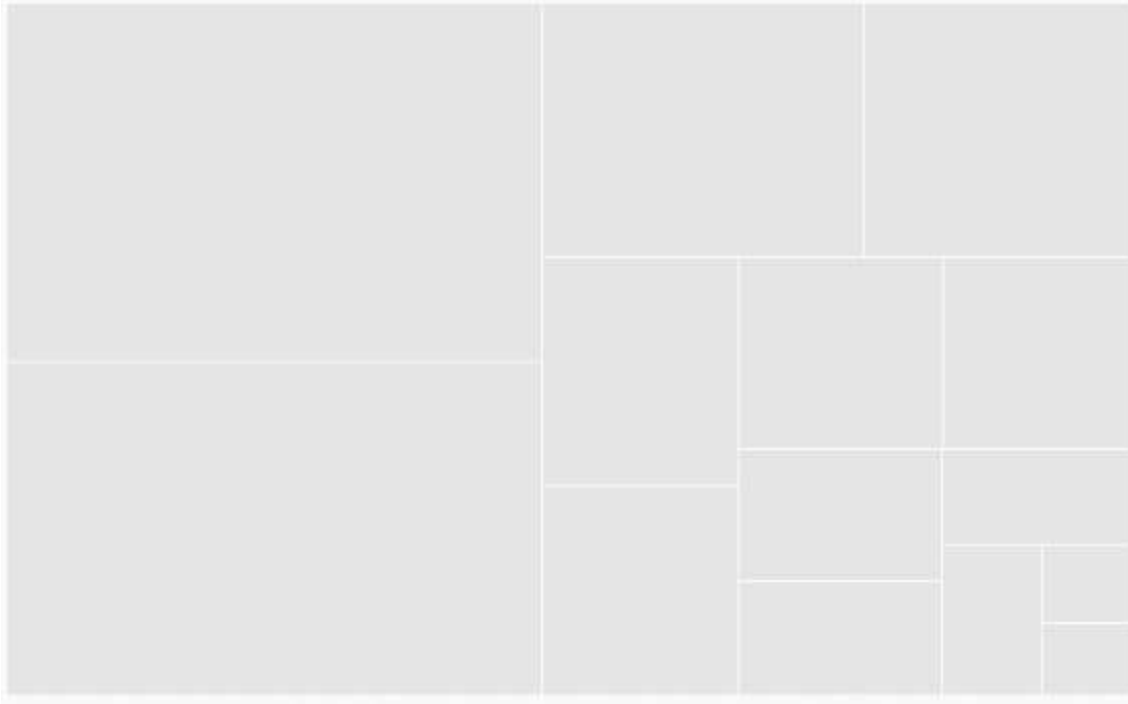
**Note:** Population data is referred from [List of continents by population](#)

3.The size of an object can be calculated by using the **WeightValuePath** of TreeMap 4.Initialize the TreeMap by using the **EJ.TreeMap** tag.

#### JAVASCRIPT

```
<script type="text/babel">
<!DOCTYPE html>
<html>
<body>
<script type="text/babel">
ReactDOM.render(
<div className="default">
<EJ.TreeMap id="treemap1" dataSource = {population_data} weightValuePath
="Population" ></EJ.TreeMap>,
</div>,
document.getElementById('treemap-default')
);
</script>
</body>
</html>
```

The following image displays a TreeMap with default properties using the above code.



### GroupTreeMap Items using Levels

You can group TreeMap Items using levels in TreeMap.

#### Group Path

You can use `groupPath` property for every flat level of the TreeMap control. It is a path to a field on the source object that serves as the “group” for the level specified. You can group the data based on the `groupPath` in the TreeMap control. When the `groupPath` is not specified, then the items are not grouped and the data is displayed in the order specified in the `dataSource`.

#### Group Gap

You can use `groupGap` property to separate the items from every flat level and to differentiate the levels mentioned in the TreeMap control.

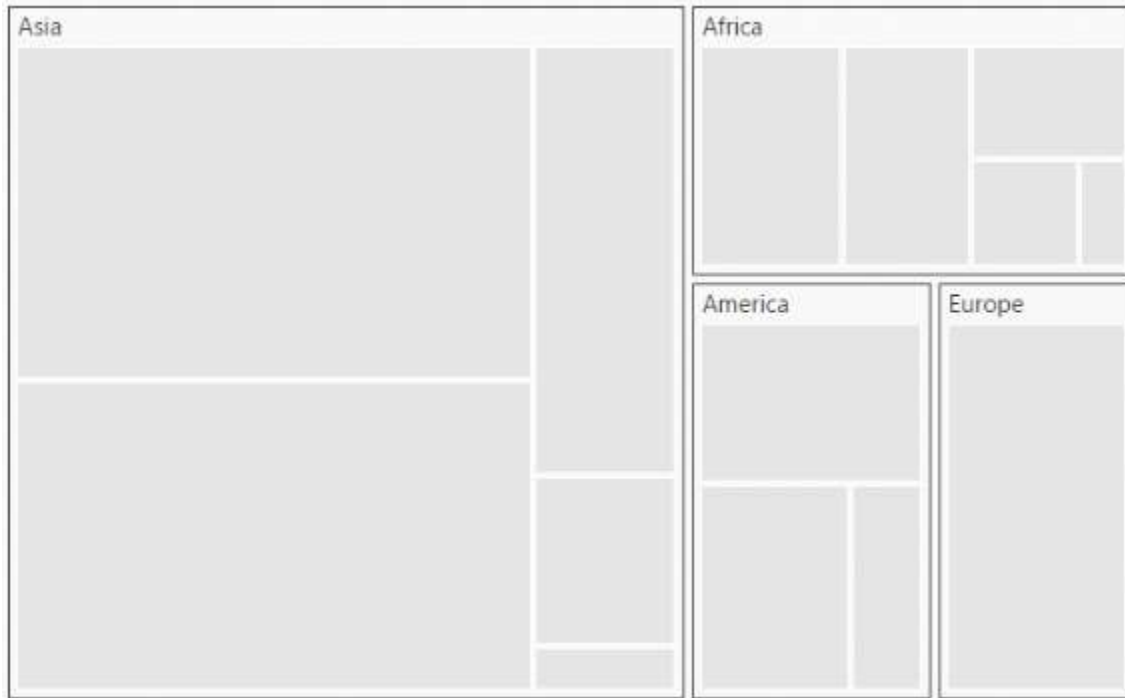
The following code sample explains how to group TreeMap Items using ‘Levels’

#### JAVASCRIPT

```
<script type="text/babel">
var levels = [
{ groupPath: "Continent", groupGap: 5}
];
<!DOCTYPE html>
<html>
<body>
<script type="text/babel">
ReactDOM.render(
<div className="default">
<EJ.TreeMap id="treemap1" dataSource = {population_data} weightValuePath
="Population"
levels = {levels} ></EJ.TreeMap>,
</div>,
document.getElementById('treemap-default')
```

```
);
</script>
</body>
</html>
```

The following screenshot displays grouping of TreeMapItems using Levels.



### Customize TreeMap Appearance by Range

You can differentiate the nodes based on its value and color ranges using Range color. You can also define the color value range using From and To properties.

#### Color Value Path

The `colorValuePath` of TreeMap is a path to a field on the source object. You can determine the color for the object using `colorValuePath` of TreeMap.

The following code sample explains how to customize TreeMap Appearance by Range.

#### JAVASCRIPT

```
<script type="text/babel">
var levels = [
{ groupPath: "Continent", groupGap: 5}
];
var rangeColorMapping = [
{ color: "#DC562D", from: "0", to: "1" },
{ color: "#FED124", from: "1", to: "1.5" },
{ color: "#487FC1", from: "1.5", to: "2" },
{ color: "#0E9F49", from: "2", to: "3" }
];
<!DOCTYPE html>
<html>
<body>
```

```

<script type="text/babel">
ReactDOM.render(
  <div className="default">
    <EJ.TreeMap id="treemap1"  dataSource = {population_data}    weightValuePath
    ="Population"
    levels = {levels} rangeColorMapping = {rangeColorMapping}
    colorValuePath="Growth" ></EJ.TreeMap>,
  </div>,
  document.getElementById('treemap-default')
);
</script>
</body>
</html>

```

The following screenshot displays customized TreeMap Appearance by Range



#### Enable Tooltip

You can enable the tooltip by setting `showTooltip` property to 'true'. By default, it takes the property of the bound object that is referred in the `weightValuePath` and displays its content when the corresponding node is hovered. You can customize the template for tooltip using `tooltipTemplate` property.

#### Leaf Item Settings

You can customize the Leaf level TreeMap items using `leafItemSettings`. The Label and tooltip values take the property of bound object that is referred in the `labelPath` when defined.

The following code sample displays how the tooltip is enabled.

#### JAVASCRIPT

```

<script type="text/babel">
var levels = [

```

```

{ groupPath: "Continent", groupGap: 5}
];
var rangeColorMapping = [
{ color: "#DC562D", from: "0", to: "1" },
{ color: "#FED124", from: "1", to: "1.5" },
{ color: "#487FC1", from: "1.5", to: "2" },
{ color: "#0E9F49", from: "2", to: "3" }
];
var leafItemSettings = { labelPath: "Region" };
<!DOCTYPE html>
<html>
<body>
<script type="text/babel">
ReactDOM.render(
<div className="default">
<EJ.TreeMap id="treemap1" dataSource = {population_data} weightValuePath
="Population"
levels = {levels} rangeColorMapping = {rangeColorMapping}
colorValuePath="Growth" leafItemSettings={leafItemSettings}
showTooltip={true} ></EJ.TreeMap>,
</div>,
document.getElementById('treemap-default')
);
</script>
</body>
</html>

```

The following screenshot displays the TreeMap when the Tooltip is enabled.



## Legend

You can set the color value of leaf nodes using `TreeMap Legend`. This legend is appropriate only for the `TreeMap` whose leaf nodes are colored using `rangeColorMapping`.

You can set `ShowLegend` property value to `'true'` to make a Legend visible.

## Label for Legend

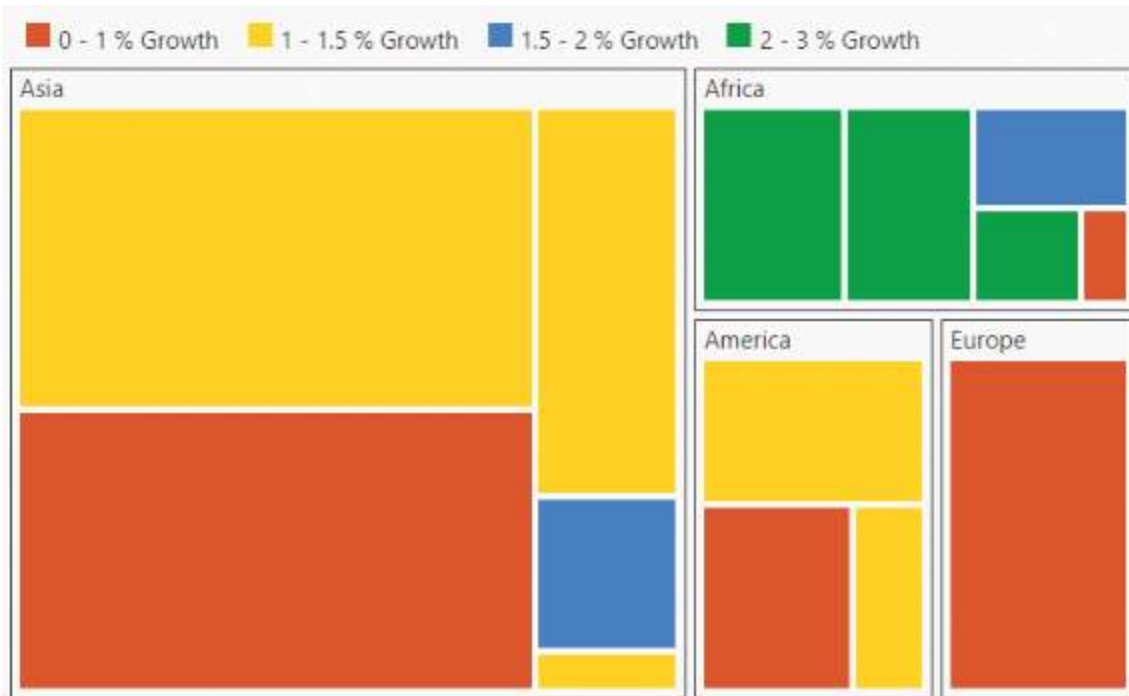
You can customize the labels of the legend item using `legendLabel` property of `rangeColorMapping`.

The following code sample displays how to add labels for legend in a `TreeMap`.

## JAVASCRIPT

```
<script type="text/babel">
var levels = [
{ groupPath: "Continent", groupGap: 5}
];
var rangeColorMapping = [
{ color: "#DC562D", from: "0", to: "1" },
{ color: "#FED124", from: "1", to: "1.5" },
{ color: "#487FC1", from: "1.5", to: "2" },
{ color: "#0E9F49", from: "2", to: "3" }
];
var leafItemSettings = { labelPath: "Region" };
var legendSettings = {
showLegend:true,
height:38,
width:690,
};
<!DOCTYPE html>
<html>
<body>
<script type="text/babel">
ReactDOM.render(
<div className="default">
<EJ.TreeMap id="treemap1"
dataSource = {population_data}
weightValuePath ="Population"
levels = {levels}
rangeColorMapping = {rangeColorMapping}
colorValuePath="Growth"
leafItemSettings={leafItemSettings}
showTooltip={true}
showLegend = {true}
legendSettings={legendSettings}></EJ.TreeMap>,
</div>,
document.getElementById('treemap-default')
);
</script>
</body>
</html>
```

The following screenshot displays the `TreeMap` when Labels are enabled.



Without using jsx Template

The TreeMap can be created from a HTML `DIV` element with the HTML `id` attribute set to it. Refer to the following code example.

#### HTML

```
<div id="treemap-default"></div>
```

#### JAVASCRIPT

```
<script type="text/babel">
var levels = [
  { groupPath: "Continent", groupGap: 5 }
];
var rangeColorMapping = [
  { color: "#DC562D", from: "0", to: "1" },
  { color: "#FED124", from: "1", to: "1.5" },
  { color: "#487FC1", from: "1.5", to: "2" },
  { color: "#0E9F49", from: "2", to: "3" }
];
var leafItemSettings = { labelPath: "Region" };
var legendSettings = {
  showLegend: true,
  height: 38,
  width: 690,
};
ReactDOM.render(
  React.createElement(EJ.TreeMap, { id: "treeMapDefault",
    dataSource: population_data,
    colorValuePath: "Growth",
    weightValuePath: "Population",
    showTooltip: true,
```

```

showLegend: true,
leafItemSettings: leafItemSettings,
legendSettings: legendSettings,
rangeColorMapping: rangeColorMapping,
levels: levels
},
),
document.getElementById('treemap-default')
);
</script>

```

On running the above code the TreeMap will rendered along with labels



## DataBinding

**TreeMap** control supports Data Binding and it can be achieved using **dataSource** property.

The **dataSource** property accepts the collection values as input. For example, you can provide the list of objects as input. The following code illustrates you on how to bind a flat collection as datasource for **TreeMap**.

## JS

```

"use strict";
ReactDOM.render(
  <EJ.TreeMap id="treemap1" dataSource = {population_data} weightValuePath
    ="population" ></EJ.TreeMap>,
  document.getElementById('treemaps')
);
var population_data = [
  { Continent: "Asia", Region: "Southern Asia", Growth: 1.32, Population:
    1749046000 },

```



```
{ Continent: "Asia", Region: "Eastern Asia", Growth: 0.57, Population:
1620807000 },
{ Continent: "Asia", Region: "South-Eastern Asia", Growth: 1.20, Population:
618793000 },
{ Continent: "Asia", Region: "Western Asia", Growth: 1.98, Population:
245707000 },
{ Continent: "Asia", Region: "Central Asia", Growth: 1.43, Population:
64370000 },
{ Continent: "Europe", Region: "Europe", Growth: 0.10, Population: 742452000
},
{ Continent: "America", Region: "South America", Growth: 1.06, Population:
406740000 },
{ Continent: "America", Region: "Northern America", Growth: 0.85,
Population: 355361000 },
{ Continent: "America", Region: "Central America", Growth: 1.40, Population:
167387000 },
{ Continent: "Africa", Region: "Eastern Africa", Growth: 2.89, Population:
373202000 },
{ Continent: "Africa", Region: "Western Africa", Growth: 2.78, Population:
331255000 },
{ Continent: "Africa", Region: "Northern Africa", Growth: 1.70, Population:
210002000 },
{ Continent: "Africa", Region: "Middle Africa", Growth: 2.79, Population:
135750000 },
{ Continent: "Africa", Region: "Southern Africa", Growth: 0.91, Population:
60425000 }
];
```

## TreeMapLevels

The levels of **TreeMap** can be categorized into two types as,

- FlatLevel
- Hierarchical Level

### Flat Level

#### GroupPath

You can use `groupPath` property for every flat level of the **TreeMap** control. It is a path to a field on the source object that serves as the “Group” for the level specified. You can group the data based on the `groupPath` in the **TreeMap** control. When the `groupPath` is not specified, then the items are not grouped and the data is displayed in the order specified in the `dataSource`.

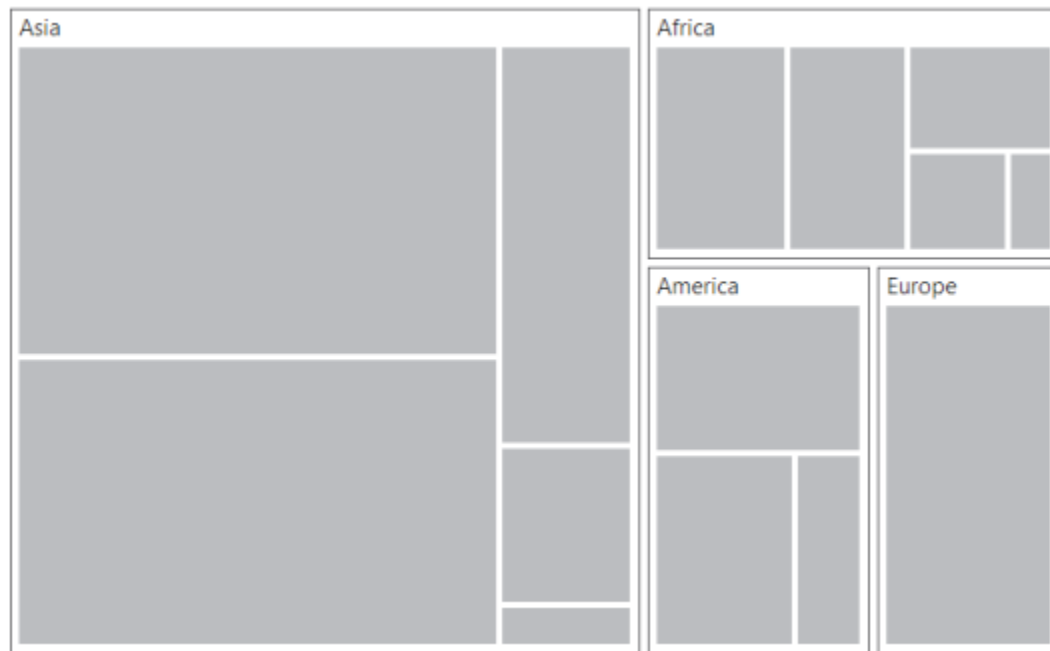
#### GroupGap

You can use `groupGap` property to separate the items from every flat level and to differentiate the levels mentioned in the **TreeMap** control.

### JS

```
"use strict";
var levels= [
{ groupPath: "Continent",groupGap:5},
];
ReactDOM.render(
```

```
<EJ.TreeMap id="treemap1" dataSource = {population_data} weightValuePath =
"Population"
levels = {levels}> </EJ.TreeMap>,
document.getElementById('treemaps')
);
```



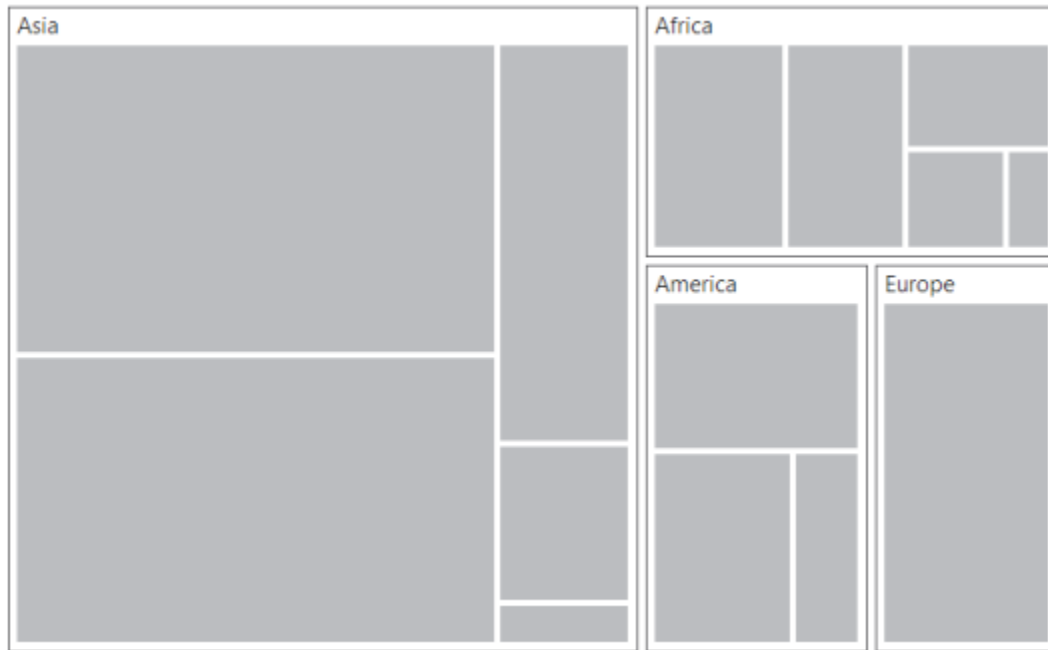
### Hierarchical Level

**TreeMap** Hierarchical level is used to define levels for hierarchical data collection that contains tree-structured data.

### JS

```
"use strict";
ReactDOM.render(
<EJ.TreeMap id="treemap1" dataSource = {population_data} weightValuePath =
"Population"> </EJ.TreeMap>,
document.getElementById('treemaps')
);
var population_data = [
{Asia:[
{Region: "Southern Asia", Growth: 1.32, Population: 1749046000 },
{Region: "Eastern Asia", Growth: 0.57, Population: 1620807000 },
{Region: "South-Eastern Asia", Growth: 1.20, Population: 618793000 },
{Region: "Western Asia", Growth: 1.98, Population: 245707000 },
{Region: "Central Asia", Growth: 1.43, Population: 64370000 }
] },
{America:[
{Region: "South America", Growth: 1.06, Population: 406740000 },
{Region: "Northern America", Growth: 0.85, Population: 355361000 },
{Region: "Central America", Growth: 1.40, Population: 167387000 },
]},
{Africa:[
{Region: "Eastern Africa", Growth: 2.89, Population: 373202000 },
```

```
{Region: "Western Africa", Growth: 2.78, Population: 331255000 },
{Region: "Northern Africa", Growth: 1.70, Population: 210002000 },
{Region: "Middle Africa", Growth: 2.79, Population: 135750000 },
{Region: "Southern Africa", Growth: 0.91, Population: 60425000 }
]}
];
```



## Layout

You can decide on the visual representation of nodes belonging to all the treemap levels using the `itemsLayoutMode` property of the `TreeMap`.

There are four different **TreeMap** layouts such as

- Squarified
- SliceAndDiceAuto
- SliceAndDiceHorizontal
- SliceAndDiceVertical

## Squarified

**Squarified** layout creates rectangles with best aspect ratio.

## JS

```
"use strict";
ReactDOM.render(
  <EJ.TreeMap id="treemap1" itemsLayoutMode ="squarified"> </EJ.TreeMap>,
  document.getElementById('treemaps')
);
```



### SliceAndDiceAuto

**SliceAndDiceAuto** layout creates rectangles with high aspect ratio and displays them sorted both horizontally and vertically.

#### JS

```
"use strict";
ReactDOM.render(
  <EJ.TreeMap id="treemap1" itemsLayoutMode ="sliceanddiceauto">
</EJ.TreeMap>,
  document.getElementById('treemaps')
);
```



### SliceAndDiceHorizontal

**SliceAndDiceHorizontal** layout creates rectangles with high aspect ratio and displays them sorted horizontally.

#### JS

```
"use strict";
ReactDOM.render(
  <EJ.TreeMap id="treemap1" itemsLayoutMode ="sliceanddicehorizontal">
</EJ.TreeMap>,
  document.getElementById('treemaps')
);
```



### SliceAndDiceVertical

**SliceAndDiceVertical** layout creates rectangles with high aspect ratio and displays them sorted vertical.

#### JS

```
"use strict";
ReactDOM.render(
  <EJ.TreeMap id="treemap1" itemsLayoutMode ="sliceanddicevertical">
</EJ.TreeMap>,
  document.getElementById('treemaps')
);
```



### Customization

**TreeMap** control supports color customization to determine the exact combination of colors for tree nodes displayed in **TreeMap** and tooltip support to display additional information of treemap data.

#### Color

You can customize the colors of the leaf nodes of **TreeMap** using the ColorMapping support of the **TreeMap**.

ColorMapping is categorized into three different types such as,

- uniColorMapping

- rangeBrushColorMapping
- desaturationColorMapping

### Uni Color Mapping

You can color, all the leaf nodes with the same color by setting the `color` value of the `uniColorMapping` property of the **TreeMap**.

#### JS

```
"use strict";
var uniColorMapping = { color: "Crimson" };
ReactDOM.render(
  <EJ.TreeMap id="treemap1" uniColorMapping = {uniColorMapping}>
</EJ.TreeMap>,
  document.getElementById('treemaps')
);
```



### Range Color Mapping

You can group the leaf nodes based on the range of the data's color values. You can set a unique color for every ranges. To achieve this, specify the `to` and `from` values as range bound and `color` value to fill the leaf nodes of the particular range, through the `rangeColorMapping` property of the **TreeMap**.

#### JS

```
"use strict";
var rangeColorMapping = [
  { color: "#77D8D8", from: "0", to: "1" },
  { color: "#AED960", from: "0", to: "2" },
  { color: "#FFAF51", from: "0", to: "3" },
  { color: "#F3D240", from: "0", to: "4" }
];
ReactDOM.render(
  <EJ.TreeMap id="treemap1" rangeColorMapping =
  {rangeColorMapping}></EJ.TreeMap>,
  document.getElementById('treemaps')
);
```



### Desaturation Color Mapping

You can differentiate all the leaf nodes using the `desaturationColorMapping` property of the **TreeMap**. Differentiation is achieved, even though same color is applied for all the leaf nodes by varying the opacity of the leaf nodes based on the color value specified in the color value range using `rangeMinimum` and `rangeMaximum` value of the data collection. You can also bound the opacity range by setting `from` and `to` property of the `desaturationColorMapping`.

#### JS

```
"use strict";
var desaturationColorMapping = {
  color: "DeepSkyBlue", from: "1", to: "0.2", rangeMinimum: "0",
  rangeMaximum: "4"
};
```

```
};
ReactDOM.render(
  <EJ.TreeMap id="treemap1" desaturationColorMapping =
    {desaturationColorMapping}></EJ.TreeMap>,
  document.getElementById('treemaps')
);
```



### Tooltip

You can enable the tooltip support for the TreeMap by setting the `showTooltip` property to true. By default, it takes the property of the bound object that is referred to in the `groupPath` and displays its content when the corresponding node is tapped. The `tooltipTemplate` is a **HTML** element that is used to expose the custom template for the tooltip.

### Leaf Item Setting

You can customize the **Leaf level TreeMap items** using `leafItemSettings`. The Label and tooltip values take the property of bound object that is referred in the `labelPath` when defined.

You can specify the border color using `BorderBrush` property.

- For customizing border thickness, you can use `BorderThickness` property.
- To customize the gap between the leaf items, you can use `Gap` property.
- You can specify the label template for the leaf item using `ItemTemplate` property.
- The Label and tooltip values take the property of bound object that is referred in the `LabelPath` when defined.
- You can specify the position of the leaf labels using `LabelPosition` property.
- You can control the mode of label visibility of the labels using `LabelVisibilityMode` property.
- To show or hide the visibility of the leaf item labels you can use `ShowLabels` property.
- For specifying over flow action of left item labels you can use `TextOverflow` property.

### JS

```
"use strict";
var leafItemSettings = { labelPath: "Region" };
ReactDOM.render(
  <EJ.TreeMap id="treemap1" leafItemSettings = {leafItemSettings}
  showTooltip = {true}></EJ.TreeMap>,
  document.getElementById('treemaps')
);
```



### Border Brush

You can able to customize the border color of the treemap using the property `BorderBrush`.

### JS

```
"use strict";
ReactDOM.render(
  <EJ.TreeMap id="treemap1" borderBrush = "white"></EJ.TreeMap>,
  document.getElementById('treemaps')
);
```

```
);
```

### Border Thickness

For customizing the border thickness of the treemap, you can use the `BorderThickness` property.

#### JS

```
"use strict";
ReactDOM.render(
  <EJ.TreeMap id="treemap1" borderThickness = {1}></EJ.TreeMap>,
  document.getElementById('treemaps')
);
```

### Dock Position

You can position the legend at top, bottom, left and right side of the treemap as per your requirement. For changing the position as per your requirement, you can use `DockPosition` property.

```
<ts name="ej.datavisualization.TreeMap.DockPosition"/>
```

Specifies the dockPosition for legend

#### JS

```
"use strict";
ReactDOM.render(
  <EJ.TreeMap id="treemap1" dockPosition = "top"></EJ.TreeMap>,
  document.getElementById('treemaps')
);
```

### Clicking and Dragging

You can select the single treemap element on click and drag. To click and drag treemap items, you have to enable the `draggingOnSelection` property.

#### JS

```
"use strict";
ReactDOM.render(
  <EJ.TreeMap id="treemap1" draggingOnSelection = {false}></EJ.TreeMap>,
  document.getElementById('treemaps')
);
```

For selecting the group element of treemap while clicking and dragging, you can use `draggingGroupOnSelection` property.

#### JS

```
"use strict";
ReactDOM.render(
  <EJ.TreeMap id="treemap1" draggingGroupOnSelection = {false}></EJ.TreeMap>,
  document.getElementById('treemaps')
);
```

### Fill with Gradient

You can customize that whether gradient color have to be applied for treemap or not. This can be customized using the property `enableGradient`.

#### JS

```
"use strict";
ReactDOM.render(
  <EJ.TreeMap id="treemap1" enableGradient = {true}></EJ.TreeMap>,
  document.getElementById('treemaps')
);
```

### Responsive Treemap

You can customize whether treemap have to be responsive or not while resizing the container. For making treemap responsive you can use `enableResize` or `isResponsive` property.

#### JS

```
"use strict";
ReactDOM.render(
  <EJ.TreeMap id="treemap1" enableResize = {true}></EJ.TreeMap>,
  document.getElementById('treemaps')
);
```

### GroupColorMapping

You can customize the color of the each group using `GroupColorMapping` property. To use group color mapping, kindly specify `GroupID` and `RangeColorMapping` inside the `GroupColorMapping`.

#### JS

```
"use strict";
var groupColorMapping={
  //..
};
ReactDOM.render(
  <EJ.TreeMap id="treemap1" groupColorMapping =
  {groupColorMapping}></EJ.TreeMap>,
  document.getElementById('treemaps')
);
```

### GroupSelectionMode

You can specifies the selection mode of the treemap using `groupSelectionMode` property. You can set either group selection mode value as `Default` or `Multiple`.

#### JS

```
"use strict";
ReactDOM.render(
  <EJ.TreeMap id="treemap1" groupSelectionMode = {true}></EJ.TreeMap>,
  document.getElementById('treemaps')
);
```



### Header

You can specify the header for the parent item using the property **Header**. This is applicable only for hierarchical data source.

#### JS

```
"use strict";
ReactDOM.render(
  <EJ.TreeMap id="treemap1" header = ""></EJ.TreeMap>,
  document.getElementById('treemaps')
);
```

### Specifying HierarchicalDatasource

You can specify whether data source bound for the treemap is hierarchical or not using the property **IsHierarchicalDatasource**.

#### JS

```
"use strict";
ReactDOM.render(
  <EJ.TreeMap id="treemap1" isHierarchicalDatasource = {true}></EJ.TreeMap>,
  document.getElementById('treemaps')
);
```

### Treemap Items

You can specify the treemap items which you want to display in the treemap using the property **TreeMapItems**.

#### JS

```
"use strict";
var treeMapItems={
  //..
};
ReactDOM.render(
  <EJ.TreeMap id="treemap1" treeMapItems = {treeMapItems}></EJ.TreeMap>,
  document.getElementById('treemaps')
);
```

## TreeMap Elements

TreeMap contains various elements such as,

- Legend
- Headers
- Labels

### Legend

You can set the color value of **leaf nodes** using **treeMapLegend**. This legend is appropriate only for the **TreeMap** whose leaf nodes are colored using **rangeColorMapping**.

You can set **showLegend** property value to **"true"** to enable or disable legend visibility.

### TreeMap Legend

You can decide the size of the legend icons by setting `iconWidth` and `iconHeight` properties of the `treeMapLegend` property avail in **TreeMap**.

### Label for Legend

You can customize the labels of the **legend item** using `legendLabel` property of `rangeColorMapping`.

### JS

```
"use strict";
var legendSettings = {
  height: 40,
  width: 700
};
var rangeColorMapping = [
  { color: "#77D8D8", from: "0", to: "1" },
  { color: "#AED960", from: "0", to: "2" },
  { color: "#FFAF51", from: "0", to: "3" },
  { color: "#F3D240", from: "0", to: "4" }
];
ReactDOM.render(
  <EJ.TreeMap id="treemap1" showLegend = {true} rangeColorMapping
    = {rangeColorMapping} legendSettings = {legendSettings}> </EJ.TreeMap>,
  document.getElementById('treemaps')
);
```

![[/js/TreeMap/TreeMap-Elementsimages/TreeMap-Elementsimg1.png]

### Interactive Legend

The legends can be made interactive with an arrow mark indicating the exact range color in the legend when the mouse hovers over the corresponding treemap items. You can enable this option by setting `mode` property in `legendSettings` value as “interactive” and default value of `mode` property is “default” to enable the normal legend.

### Title for Interactive Legend

You can provide the title for interactive legend by using `title` property in `legendSettings`.

### Label for Interactive Legend

You can provide the left and right labels to interactive legend by using `leftLabel` and `rightLabel` properties in `legendSettings`.

### JS

```
"use strict";
var legendSettings = {
  height: 15,
  width: 150,
  mode: "interactive",
  title: "Population",
  leftLabel: "0.5M",
  rightLabel: "40M",
  dockPosition: "top"
};
var rangeColorMapping = [
  { color: "#77D8D8", from: "0", to: "1" },
```

```
{ color: "#AED960", from: "0", to: "2" },
{ color: "#FFAF51", from: "0", to: "3" },
{ color: "#F3D240", from: "0", to: "4" }
];
ReactDOM.render(
  <EJ.TreeMap id="treemap1" showLegend = {true} rangeColorMapping
={rangeColorMapping} legendSettings = {legendSettings}></EJ.TreeMap>,
  document.getElementById('treemaps')
);
```



## Header

You can set headers for each level by setting the `showHeader` property of the each **TreeMap** levels. The `headerHeight` property helps to set the height of the header and Group path value determines the header value. You can customize the default header appearance by setting the `headerTemplate` of the **TreeMap** levels.

## JS

```
<div id="treemap" style="width: 950px; height: 500px; "></div>
"use strict";
var levels = [{
  groupPath: "Continent",
  groupGap: 2,
  headerHeight: 25,
  headerTemplate: 'headerTemplate'
}];
ReactDOM.render(
  <EJ.TreeMap id="treemap1" levels = {levels}></EJ.TreeMap>,
  document.getElementById('treemaps')
);
<script id="headertemplate" type="application/jsrender">
<div style="background-color: white; margin:5px">
<label style="color:black;font-size:large;" >{:header}</label><br />
</div>
</script>
```



## Customizing the header

The text in the header can be customized by triggering the event `headerTemplateRendering` of the **TreeMap**. This event is triggered before rendering the header template.

## JS

```
<div id="treemap" style="width: 950px; height: 500px; "></div>
"use strict";
function loadTemplate() {
  //...
}
var levels = [{
  groupPath: "Continent",
  groupGap: 2,
  headerHeight: 25,
```

```
headerTemplate: 'headerTemplate'
}];
ReactDOM.render(
<EJ.TreeMap id="treemap1" levels = {levels} headerTemplateRendering
='loadTemplate'></EJ.TreeMap>,
document.getElementById('treemaps')
);
<script id="headertemplate" type="application/jsrender">
<div style="background-color: white; margin:5px">
<label style="color:black;font-size:large;" >{{:header}}</label><br />
</div>
</script>
```



### Label

You can also set labels for the leaf nodes by setting the `showLabels` property as true. Group path value is displayed as a label for leaf nodes. You can customize the default label appearance by setting the `labelTemplate` of the **TreeMap** levels.

### JS

```
<div id="treemap" style="width: 1100px; height: 550px; "></div>
"use strict";
var levels = [{
  groupPath: "Continent",
  showLabels: true,
  groupGap: 2,
  headerHeight: 20,
  headerTemplate: 'headerTemplate',
  labelPosition:"topLeft",
}];
var leafItemSettings = { labelPath: "Region", showLabels: true};
var legendSettings = {
  height:40,
  width:700
};
ReactDOM.render(
<EJ.TreeMap id="treemap1" leafItemSettings = {leafItemSettings}
legendSettings = {legendSettings} levels = {levels}></EJ.TreeMap>,
document.getElementById('treemaps')
);
<script id="headertemplate" type="application/jsrender">
<div style="background-color: white; margin:5px">
<label style="color:black;font-size:medium;" >{{:header}}</label><br />
</div>
</script>
```



### Customizing the Overflow labels

You can handle the label overflow, by specifying any one of the following values to the property `textOverflows`

**None** - By specifying `textOverflow` as “none”, it displays the default label text.

**Hide** - By specifying textOverflow as “hide”, You can hide the label, when it exceeds the header width.

**Wrap** - By specifying textOverflow as “wrap”, you can wrap the label text.

**WrapByWord** - By specifying textOverflow as “wrap by word”, you can wrap the label text by word.

### JS

```
<div id="treemap" style="width: 1100px; height: 550px; "></div>
"use strict";
var levels = [{
  groupPath: "Continent",
  showLabels: true,
  groupGap: 2,
  headerHeight: 20,
  headerTemplate: 'headerTemplate',
  labelPosition: "topLeft",
}];
var leafItemSettings = { labelPath: "Region", showLabels: true};
var legendSettings = {
  height: 40,
  width: 700,
  textOverFlow: 'Wrap'
};
ReactDOM.render(
  <EJ.TreeMap id="treemap1" leafItemSettings = {leafItemSettings}
  legendSettings = {legendSettings} levels = {levels}></EJ.TreeMap>,
  document.getElementById('treemaps')
);
<script id="headertemplate" type="application/jsrender">
<div style="background-color: white; margin: 5px">
<label style="color: black; font-size: medium;" >{{:header}}</label><br />
</div>
</script>
```

## Drill Down Support

**Treemap** enables drill down to expose the hierarchy achieved by clicking on a node and this results in enabling the **Treemap** to move to the next level or sub level and can return back to the normal **Treemap** view by clicking on the node header. Only a single level of the **Treemap** is visible at once.

### Enable Drill Down

**Treemap** elements can be drilled down by setting the `enableDrillDown` property to true. You can view the hierarchy of the **Treemap** by clicking on the treemap items and can move to the previous level by clicking on the drill down header. The header color can be customized by changing the values in the property `drillDownHeaderColor` and the selection color can be done by changing the `drillDownSelectionColor` property.

Property	Type	Description
<code>enableDrillDown</code>	bool	Gets or sets a value that indicates whether the drill down feature is enabled or not
<code>drillDownHeaderColor</code>	string	Gets or sets a color for header during drill down

drillDownSelectionColor	string	Gets or sets a color for highlighting tree map item during drill down.
-------------------------	--------	--

**JS**

```
"use strict";
var uniColorMapping = { color: "#CCDFE3" };
var levels= [{
  groupPath: "Continent",
  groupGap:5,
  showLabels: true,
  headerHeight: 25,
  showHeader: true
}];
ReactDOM.render(
  <EJ.TreeMap id="treemap1" dataSource = {population_data} enableDrillDown =
  {true} drillDownHeaderColor = "#199DAF" drillDownSelectionColor = "#199DAF"
  uniColorMapping = {uniColorMapping} weightValuePath = "Population"
  levels = {levels}></EJ.TreeMap>,
  document.getElementById('treemaps')
);
```



*Before Drill Down*



*After Drill Down*

**Methods**

*refresh()*

Method to reload treemap with updated values.

**HTML**

```
<div id="tree"></div>
```

**JAVASCRIPT**

```
ReactDOM.render (
  <EJ.TreeMap id="default"></EJ.TreeMap>,
  document.getElementById('map')
);
function TreeTreeMapMethod() {
  var treeObj = $("#default").data("ejTreeMap");
  treeObj.refresh();
};
```

**Events**

*treeMapItemSelected*

Triggers on treemap item selected.

**HTML**

```
<div id="tree"></div>
```

### JAVASCRIPT

```
ReactDOM.render(  
  <EJ.TreeMap id="default" treeMapItemSelected =  
    {TreeMapItemSelected}></EJ.TreeMap>,  
  document.getElementById('tree')  
);  
function TreeMapItemSelected() {  
  // Do Something  
};
```

#### *drillStarted*

Triggers when drilldown is started

### HTML

```
<div id="tree"></div>
```

### JAVASCRIPT

```
ReactDOM.render(  
  <EJ.TreeMap id="default" drillStarted = {DrillStarted}></EJ.TreeMap>,  
  document.getElementById('tree')  
);  
function DrillStarted() {  
  // Do Something  
};
```

#### *drillDownItemSelected*

Triggers on treemap drilldown item selected.

### HTML

```
<div id="tree"></div>
```

### JAVASCRIPT

```
ReactDOM.render(  
  <EJ.TreeMap id="default" drillDownItemSelected =  
    {DrillDownItemSelected}></EJ.TreeMap>,  
  document.getElementById('tree')  
);  
function DrillDownItemSelected() {  
  // Do Something  
};
```

#### *headerTemplateRendering*

Triggers before rendering the treemap drilldown header template

### HTML

```
<div id="tree"></div>
```

### JAVASCRIPT

```
ReactDOM.render(  
  <EJ.TreeMap id="default" headerTemplateRendering =  
    {HeaderTemplateRendering}></EJ.TreeMap>,  
  document.getElementById('tree')  
);  
function HeaderTemplateRendering() {  
  // Do Something  
};
```

#### *refreshed*

Triggers after refreshing the treemap items.

### HTML

```
<div id="tree"></div>
```

### JAVASCRIPT

```
ReactDOM.render(  
  <EJ.TreeMap id="default" refreshed = {Refreshed}></EJ.TreeMap>,  
  document.getElementById('tree')  
);  
function Refreshed() {  
  // Do Something  
};
```

#### *treeMapGroupSelected*

Triggers when the group selection is performed on treemap items.

### HTML

```
<div id="tree"></div>
```

### JAVASCRIPT

```
ReactDOM.render(  
  <EJ.TreeMap id="default" treeMapGroupSelected =  
    {TreeMapGroupSelected}></EJ.TreeMap>,  
  document.getElementById('tree')  
);  
function TreeMapGroupSelected() {  
  // Do Something  
};
```

## TreeView

### Getting Started

Using the following steps, you can create a React TreeView component. The basic rendering of React TreeView is achieved with default functionality.



### Create an TreeView

You can create a React application and add necessary scripts and styles with the help of the given [React Getting Started Documentation](#).

Create a JSX file for rendering TreeView component using <EJ.TreeView> syntax. Add required properties to it in <EJ.TreeView> tag element

#### JS

```
ReactDOM.render (
  <EJ.TreeView>
    <ul id="treeView">
      <li id="1" class="expanded">Artwork
        <ul>
          <li id="2">
            Abstract
            <ul>
              <li id="3">2 Acrylic Mediums</li>
              <li>Creative Acrylic</li>
              <li>Modern Painting</li>
              <li>Canvas Art</li>
              <li>Black white</li>
            </ul>
            </li>
          <li>Children
            <ul>
              <li>Preschool Crafts</li>
              <li>School-age Crafts</li>
              <li>Fabulous Toddler</li>
            </ul>
            </li>
          <li>Comic / Cartoon
            <ul>
              <li>Batman</li>
              <li>Adventures of Superman</li>
              <li>Super boy</li>
            </ul>
            </li>
          </ul>
          </li>
          <li class="expanded">Books
            <ul>
              <li>Comics
                <ul>
                  <li>The Flash</li>
                  <li>Human Target</li>
                  <li>Birds of Prey</li>
                </ul>
                </li>
              <li>Entertaining</li>
              <li>Design</li>
            </ul>
            </li>
          <li>Music
            <ul>
              <li>Classical
                <ul>
```

```

<li>Medieval</li>
<li>Orchestral</li>
</ul>
</li>
<li>Mass</li>
<li>Folk</li>
</ul>
</li>
</ul>
</EJ.TreeView>,
document.getElementById('treeview')
);

```

Define an HTML element for adding TreeView in the application and refer the JSX file.

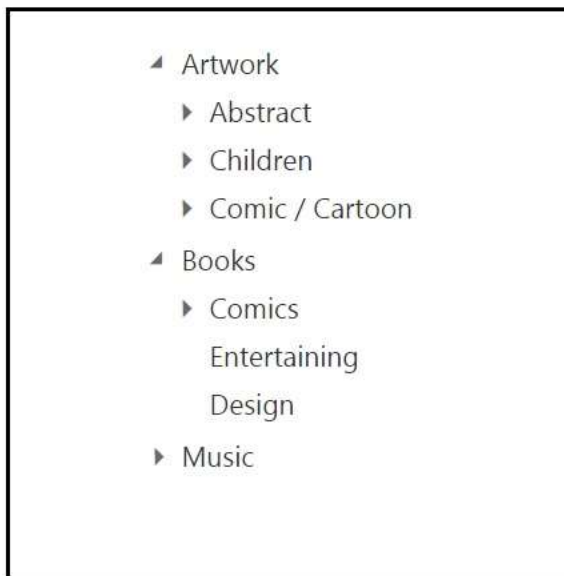
### HTML

```

<div id="treeview"></div>
<script type="text/babel" src="treeview.jsx"></script>

```

Run the above code to render the following output.



### Data Binding

The data for TreeView which can be populated using the `dataSource` property.

The `beforeLoad` event will be triggered before loading nodes into TreeView.

### JS

```

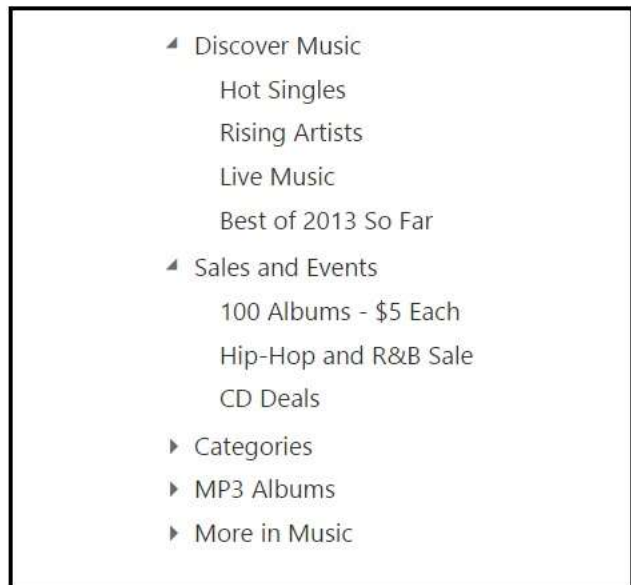
"use strict";
var localData = [
  { id: 1, name: "Discover Music", hasChild: true, expanded: true },
  { id: 2, pid: 1, name: "Hot Singles" },
  { id: 3, pid: 1, name: "Rising Artists" },
  { id: 4, pid: 1, name: "Live Music" },
  { id: 6, pid: 1, name: "Best of 2013 So Far" },
  { id: 7, name: "Sales and Events", hasChild: true, expanded: true },

```

```

{ id: 8, pid: 7, name: "100 Albums - $5 Each" },
{ id: 9, pid: 7, name: "Hip-Hop and R&B Sale" },
{ id: 10, pid: 7, name: "CD Deals" },
{ id: 11, name: "Categories", hasChild: true },
{ id: 12, pid: 11, name: "Songs" },
{ id: 13, pid: 11, name: "Bestselling Albums" },
{ id: 14, pid: 11, name: "New Releases" },
{ id: 15, pid: 11, name: "Bestselling Songs" },
{ id: 16, name: "MP3 Albums", hasChild: true },
{ id: 17, pid: 16, name: "Rock" },
{ id: 18, pid: 16, name: "Gospel" },
{ id: 19, pid: 16, name: "Latin Music" },
{ id: 20, pid: 16, name: "Jazz" },
{ id: 21, name: "More in Music", hasChild: true },
{ id: 22, pid: 21, name: "Music Trade-In" },
{ id: 23, pid: 21, name: "Redeem a Gift Card" },
{ id: 24, pid: 21, name: "Band T-Shirts" },
{ id: 25, pid: 21, name: "Mobile MVC" }];
var fields={id: "id", parentId: "pid", text: "name", hasChild: "hasChild",
dataSource: localData, expanded: "expanded"};
ReactDOM.render(
<EJ.TreeView id="treeview" fields={fields}>
</EJ.TreeView>,
document.getElementById('treeview')
);

```



**Note:** You can find the TreeView properties from the [API reference](#) document

## Toolbar

### Overview

The Toolbar control supports displaying a list of tools within a web page. This control is capable of customizing toolbar items with any functionality by using enriched client-side methods. This control is composed of collection of unordered lists containing text and images contained into a div.

## Key Features

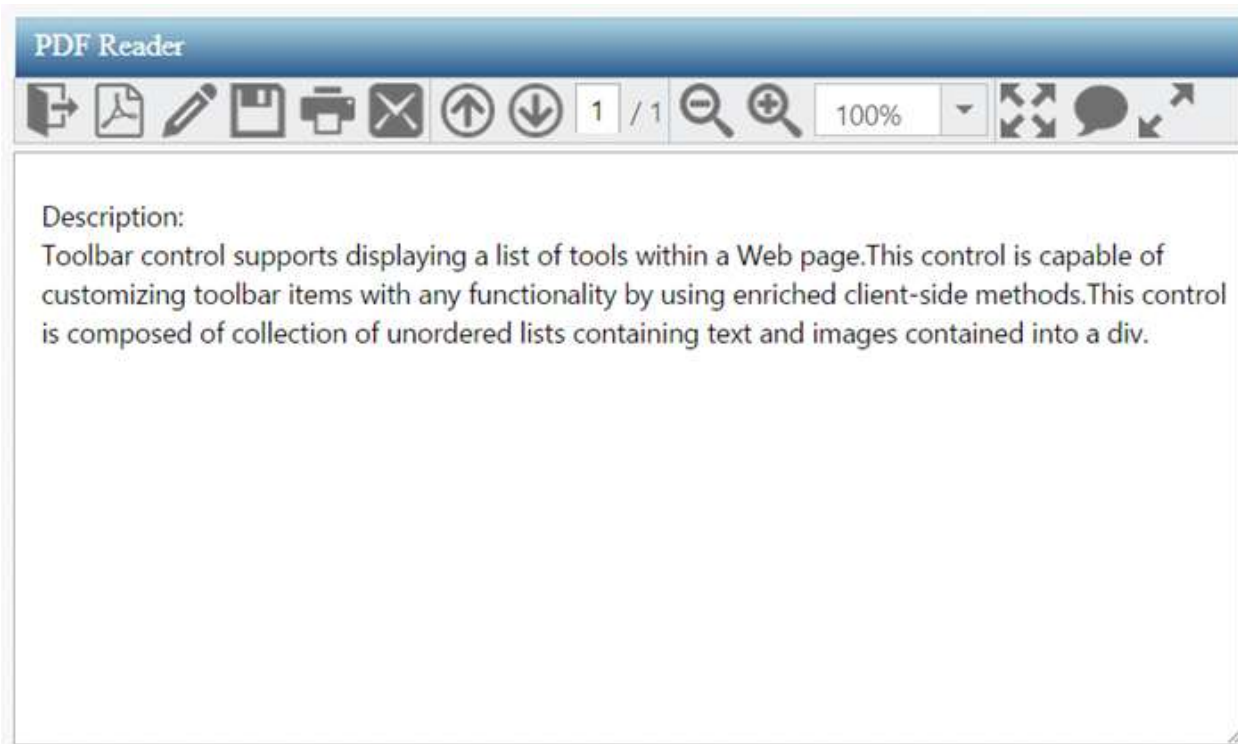
- **Modern look:** Rich appearance with theme support.
- **RTL:** Supports right-to-left alignment.
- **Text and Image:** Supports both text and images as toolbar content.
- **Easy Customization:** The customization of toolbar control to any form is made simple.
- **Data binding:** Supports for data binding with local data and remote data.
- **Template:** Supports for Customizing toolbar with user needed tools.
- **Keyboard navigation:** Support for navigation through keyboard.

## Getting Started

This section explains briefly about how to create a **Toolbar** in your application with **JavaScript**.

### Create Toolbar for PDF Reader

**Toolbar** control supports displaying a list of tools in a Web page. This control is capable of customizing toolbar items with any functionality by using enriched **client-side** methods. This control consists of a collection of **unordered lists** contains text and images into a **<div>**. From the following section, you can learn how to customize **toolbar** control for a **PDF reader** scenario. The following screen shot shows the appearance of **toolbar** in **PDF reader** simulator application.



### Create Toolbar control in React JS

You can create a React application and add necessary scripts and styles with the help of the given [React Getting Started Documentation](#).

Create a JSX file for rendering Toolbar component using `<EJ.Toolbar>` syntax. Add required properties to it in `<EJ.Toolbar>` tag element

### JS

```
"use strict";
ReactDOM.render(
  <EJ.Toolbar width="100%">
</EJ.Toolbar>,
  document.getElementById('toolbar-default')
);
```

Define an HTML element for adding Toolbar in the application and refer the JSX file.

### HTML

```
<div id="toolbar-default"></div>
<script src="app/toolbar/default.js">
```

Output of the above steps



### Initialize Toolbar Items

**Toolbar** consists of a list of items. From the following section, you can learn how to initialize the toolbar items with **UL LI** template.

Initialize the Toolbar items with **UL LI** template as follows.

### HTML

```
"use strict";
ReactDOM.render(
  <EJ.Toolbar width="100%">
    <ul>
      <li id="OtherFormat" title="Convert PDF files to Word or Excel Online..">
        <div class="PdfDocument e-icon convertToOthers "></div>
      </li>
      <li id="PDFOnline" title="Convert files to PDF Online">
        <div class="PdfDocument e-icon convertToPdf "></div>
      </li>
      <li id="Signature" title="Sign, add text or send a document for signature">
        <div class=" PdfDocument e-icon signature "></div>
      </li>
      <li id="Save" title="Save file ( Ctrl+S )">
        <div class=" PdfDocument e-icon save "></div>
      </li>
      <li id="Print" title="Print file ( Ctrl+P ) ">
        <div class=" PdfDocument e-icon print "></div>
      </li>
      <li id="Message" title="Message">
        <div class=" PdfDocument e-icon msg "></div>
      </li>
    </ul>
  </EJ.Toolbar>,
  document.getElementById('toolbar-default')
);
```

Apply the given styles in the code table to show the **toolbar items** as follows. You can refer images from any location. In the following code sample, the images are referred from the given location.

<http://js.syncfusion.com/UG/Web/Content/>

### CSS

```
<style type="text/css" class="cssStyles">
.e-tooltxt .PdfDocument.e-icon {
background-image: url('http://js.syncfusion.com/UG/Web/Content/pdf-
icon.png');
background-repeat: no-repeat;
display: block;
height: 30px;
width: 30px;
}
.e-tooltxt .PdfDocument.e-icon:hover {
background-image: url('http://js.syncfusion.com/UG/Web/Content/pdf-icon-
white.png');
}
.PdfDocument.e-icon.convertToOthers {
background-position: -349px 0px;
}
.PdfDocument.e-icon.convertToPdf {
background-position: -527px 0px;
}
.PdfDocument.e-icon.signature {
background-position: 2px 0px;
}
.PdfDocument.e-icon.save {
background-position: -87px 0px;
}
.PdfDocument.e-icon.msg {
background-position: -483px 0px;
}
</style>
```

After updating the **Toolbar items** with their **CSS** styles, you can render the toolbar inside **<script>** tag.

Execute the code to render a toolbar with a list of **toolbar items**.



Toolbar with list of toolbar items

### Render remaining Toolbar items

In the above output only few **toolbar items** are rendered, but you need to render all the **toolbar items** to achieve the requirements. You can separate or group the toolbar items. The separation or grouping of toolbar items is achieved when you give toolbar items as a list of **UL LI** values inside the toolbar **<div>** or **span** element. From the following section, you can learn how to initialize the remaining toolbar items with **UL LI** template and how to group the toolbar items.

Initialize the Toolbar items with **UL LI** template as follows.

**HTML**

```

"use strict";
var percentList = ["10%", "25%", "50%", "100%", "400%", "800%", "1600%",
"3200%", "6400%"];
ReactDOM.render(
<EJ.Toolbar width="100%" height="33px" enableSeparator={true}>
<!--Initializes toolbar items from above code example -->
<!-- Separator is added at the end of each ul inside the toolbar element-->
<!-- list of Remaining toolbar items with item separator -->
<ul>
<li id="Previous" title="Show previous page ( Left Arrow )">
<div class=" PdfDocument e-icon previous "></div>
</li>
<li id="Next" title="Show next page ( Right Arrow )">
<div class="PdfDocument e-icon next "></div>
</li>
<li id="page">
<div class="PdfDocument">
<input type="text" value="1" />
</div>
</li>
<li id="count">
<span>/ 1</span>
</li>
</ul>
<ul>
<li id="ZoomOut" title="Zoom Out">
<div class=" PdfDocument e-icon zoomOut "></div>
</li>
<li id="ZoomIn" title="Zoom In">
<div class=" PdfDocument e-icon zoomIn "></div>
</li>
<li id="ZoomValue">
<div class=" PdfDocument">
<!-- input element for rendering Zoom value dropdown -->
<EJ.DropDownList id="selectPercent" dataSource={percentList} value="100%"
width="90px" height="27px"></EJ.DropDownList>
</div>
</li>
</ul>
<ul>
<li id="FitFull" title="Fit one full page to window">
<div class=" PdfDocument e-icon fitOne "></div>
</li>
<li id="StickyNote" title="Add stick note ( Ctrl+6 ) ">
<div class=" PdfDocument e-icon sticky "></div>
</li>
<li id="ReadMode" title="View File in Read Mode">
<div class=" PdfDocument e-icon readMode "></div>
</li>
</ul>
</EJ.Toolbar>,
document.getElementById('toolbar-default')
);

```

Add the following styles in the code table to display the toolbar items as follows.

### CSS

```
<style>
.PdfDocument.e-icon.previous {
background-position: -395px 0px;
}
.PdfDocument.e-icon.next {
background-position: -439px 0px;
}
.PdfDocument.e-icon.zoomIn {
background-position: -175px 0px;
}
.PdfDocument.e-icon.zoomOut {
background-position: -219px 0px;
}
.PdfDocument.e-icon.fitOne {
background-position: -264px 0px;
}
.PdfDocument.e-icon.sticky {
background-position: -131px -1px;
}
.PdfDocument.e-icon.readMode {
background-position: -308px 0px;
}
.PdfDocument.e-icon.print {
background-position: -43px 0px;
}
#ZoomValue .PdfDocument {
width: 90px;
}
#page .PdfDocument input {
text-align: center;
width: 20px;
height: 21px;
}
#count span {
width: 30px;
height: 30px;
position: relative;
top: 2px;
text-align: center;
vertical-align: middle;
}
</style>
```

After updating the Toolbar items with their **CSS** styles, you can render the toolbar inside the **<script>** tag and also need to render the drop down list control for **select zoom value**. Basically, dropdown list control is rendered with input element. **Set Zoom value** is one of the items in the toolbar. The following code example shows how to render and initialize **drop down control** with list of **zoom values**.

Execute the code to render a **toolbar items** with separator.





### Add Actions to Toolbar Items

Now the **toolbar** is rendered so you need to render the header and content area to create a **PDF reader**. From the following section, you can learn how to render the **header** (Toolbar), **contentsection** (PDF viewer area) and how to set the action to toolbar items.

You are not going to deal with PDF reading or rendering task here. You will only simulate the PDF Reader app to demonstrate the Toolbar control usage and will completely ignore the PDF rendering area.

Initialize the content area and header as specified in the code table.

### HTML

```
<!-- control class used for aligns the pdf reader in center of a page. -->
<div class="control">
<div class="ctrllabel"></div>
<!-- Here Initialize the Toolbar items as like above code sample -->
<div id="contentSection">
<textarea id="content" rows="10" cols="30">
Description:
Toolbar control supports displaying a list of tools within a Web page.This
control is capable of
customizing toolbar items with any functionality by using enriched client-
side methods.This control
is composed of collection of unordered lists containing text and images
contained into a div.
</textarea>
</div>
</div>
```

You can apply the following styles with the above styles to design the **PDF header** and **content area**. The desired output is shown as follows.

### CSS

```
<style type="text/css" class="cssStyles">
#content {
float: left;
height: 300px;
width: 628px;
position: absolute;
}
.control {
margin: 110px 320px 0;
position: relative;
}
.ctrllabel {
background-image: url("http://js.syncfusion.com/UG/Web/Content/pdf-
header.png");
background-repeat: no-repeat;
width: 634px;
height: 32px;
```

```
}
</style>
```

Execute the given code to render a **PDF reader** as follows.



So far, you have added the required toolbar items and configured its appearance. When you click on **toolbar items**, the operation is performed through **client side click event**. The following code example explains how to perform operations when you click on the **toolbar items**.

#### JAVASCRIPT

```
"use strict";
var percentList = ["10%", "25%", "50%", "100%", "400%", "800%", "1600%",
"3200%", "6400%"];
function onClick(args) {
switch (option) {
case "OtherFormat":
//writes a code for Convert pdf files to Other format.
case "PDFOnline":
//writes a code for Convert files to Pdf online.
case "Signature":
//writes a code for Send a document for signature.
case "Save":
//writes a code for Save content.
case "Print":
//writes a code for Print content.
case "Message":
//writes a code for Send a Message.
case "Previous":
//writes a code for Show previous page.
case "Next":
//writes a code for Show Next page.
case "ZoomOut":
//writes a code for Zoom out the page.
case "ZoomIn":
//writes a code for Zoom In the page.
case "FitFull":
//writes a code for Fit one full page to window.
```

```

case "StickyNote":
  //writes a code for Add Sticky Note.
case "ReadMode":
  //writes a code for view file in read mode.
}
}
ReactDOM.render(
  <EJ.Toolbar width="100%" height="33px" enableSeparator={true}
  click={onClick}>
    <!--Initializes toolbar items from above code example -->
    <!-- Separator is added at the end of each ul inside the toolbar element-->
    <!-- list of Remaining toolbar items with item separator -->
    <ul>
      <li id="Previous" title="Show previous page ( Left Arrow )">
        <div class=" PdfDocument e-icon previous "></div>
      </li>
      <li id="Next" title="Show next page ( Right Arrow )">
        <div class="PdfDocument e-icon next "></div>
      </li>
      <li id="page">
        <div class="PdfDocument">
          <input type="text" value="1" />
        </div>
      </li>
      <li id="count">
        <span>/ 1</span>
      </li>
    </ul>
    <ul>
      <li id="ZoomOut" title="Zoom Out">
        <div class=" PdfDocument e-icon zoomOut "></div>
      </li>
      <li id="ZoomIn" title="Zoom In">
        <div class=" PdfDocument e-icon zoomIn "></div>
      </li>
      <li id="ZoomValue">
        <div class=" PdfDocument">
          <!-- input element for rendering Zoom value dropdown -->
          <EJ.DropDownList id="selectPercent" dataSource={percentList} value="100%"
          width="90px" height="27px"></EJ.DropDownList>
        </div>
      </li>
    </ul>
    <ul>
      <li id="FitFull" title="Fit one full page to window">
        <div class=" PdfDocument e-icon fitOne "></div>
      </li>
      <li id="StickyNote" title="Add stick note ( Ctrl+6 ) ">
        <div class=" PdfDocument e-icon sticky "></div>
      </li>
      <li id="ReadMode" title="View File in Read Mode">
        <div class=" PdfDocument e-icon readMode "></div>
      </li>
    </ul>
  </EJ.Toolbar>,
  document.getElementById('toolbar-default')
);

```

## Uploadbox

### Overview

The **Essential JavaScript Uploadbox** control supports uploading files into the designated server, regardless of the file format and size. The **Uploadbox** control helps you with the selection of files to upload to the server. The **Uploadbox** control has theme support.

### Key Features

- **Modern look:** Rich appearance with theme support.
- **RTL:** Supports right-to-left alignment.
- **Auto Upload:** Support for manual and automatic uploading of files is included.
- **Sync/Async Upload:** Supports synchronous and asynchronous uploading of files.
- **Multiple files:** Supports uploading multiple files.

### Getting Started

This section explains briefly about how to create an **Uploadbox** in your application with **Angular 2**. **Essential JavaScript Uploadbox** widget provides support to upload files or photos within your web page. From the following guidelines, you can learn how to upload the files that are used in a Resume Upload scenario. This helps you to restrict some file extensions when you upload the resume in server by using **Uploadbox** control.

The following screenshot demonstrates the functionality of **Uploadbox** with the file extension.

Extensions:

<input type="text" value=".txt"/>	<input type="button" value="Allow"/>	
<input type="text" value="Format"/>	<input type="button" value="Deny"/>	<input type="button" value="Browse"/>

Upload Box

Name	Size	Status
Tools.txt	1.1KB\1.1KB	<input checked="" type="checkbox"/>

In the above screenshot, you can upload a resume that allows **.txt** files. This helps you to avoid unsupported resume formats getting uploaded in a server.

**Note:** To get upload the file, you should either run this sample in Visual Studio IDE or host in local IIS.

### Create Uploadbox widgets in React JS

**Essential JavaScript Uploadbox** widget basically renders built-in features like upload multiple files, and deletes the files from **Uploadbox**. You can know the status of uploading the file whether it is completed or failed and you can retry uploading the files. You can easily create the **Uploadbox** widget by using the following steps.

You can create a React application and add necessary scripts and styles with the help of the given [React Getting Started Documentation](#).

Create a JSX file for rendering Uploadbox component using `<EJ.Uploadbox>` syntax. Add required properties to it in `<EJ.Uploadbox>` tag element

### JS

```
var DefaultUploadbox = React.createClass({
  render: function () {
    return (
      <div id="tooltip_default">
        <div class="posupload">Select a file to upload </div>
        <EJ.Uploadbox id="UploadDefault"></EJ.Uploadbox>
      </div>
    );
  }
});
ReactDOM.render(<DefaultUploadbox />, document.getElementById('uploadbox-default'));
```

Define an HTML element for adding Uploadbox in the application and refer the JSX file.

### HTML

```
<div id="uploadbox-default"></div>
<script src="app/uploadbox/default.js">
```

### CSS

```
<style>
#targetElement {
width: 500px;
height: 500px;
margin: 0 auto;
}
#UploadDefault {
margin: 0 auto;
}
</style>
```

Create a new handler file (.ashx) and save it as **saveFiles.ashx** and then copy the following code into it.

### C#

```

SaveFiles.ashx
public void ProcessRequest(HttpContext context)
{
    string targetFolder = HttpContext.Current.Server.MapPath("uploadfiles");
    if (!Directory.Exists(targetFolder))
    {
        Directory.CreateDirectory(targetFolder);
    }
    HttpRequest request = context.Request;
    HttpFileCollection uploadedFiles = context.Request.Files;
    if (uploadedFiles != null && uploadedFiles.Count > 0)
    {
        for (int i = 0; i < uploadedFiles.Count; i++)
        {
            if (uploadedFiles[i].FileName != null && uploadedFiles[i].FileName != "")
            {
                string fileName = uploadedFiles[i].FileName;
                int indx = fileName.LastIndexOf("\\");
                if (indx > -1)
                {
                    fileName = fileName.Substring(indx + 1);
                }
                uploadedFiles[i].SaveAs(targetFolder + "\\" + fileName);
            }
        }
    }
}

```

Create a new handler file (.ashx) and save it as **removeFiles.ashx** and then copy the following code into it.

### **C#**

```

removeFiles.ashx
public void ProcessRequest(HttpContext context)
{
    System.Collections.Specialized.NameValueCollection s =
    context.Request.Params;
    string fileName = s["fileNames"];
    string targetFolder = HttpContext.Current.Server.MapPath("uploadfiles");
    if (Directory.Exists(targetFolder))
    {
        string physicalPath = targetFolder + "\\" + fileName;
        if (System.IO.File.Exists(physicalPath))
        {
            System.IO.File.Delete(physicalPath);
        }
    }
}

```

Add the following code example to assign saveUrl and removeUrl for Uploadbox

### **HTML**

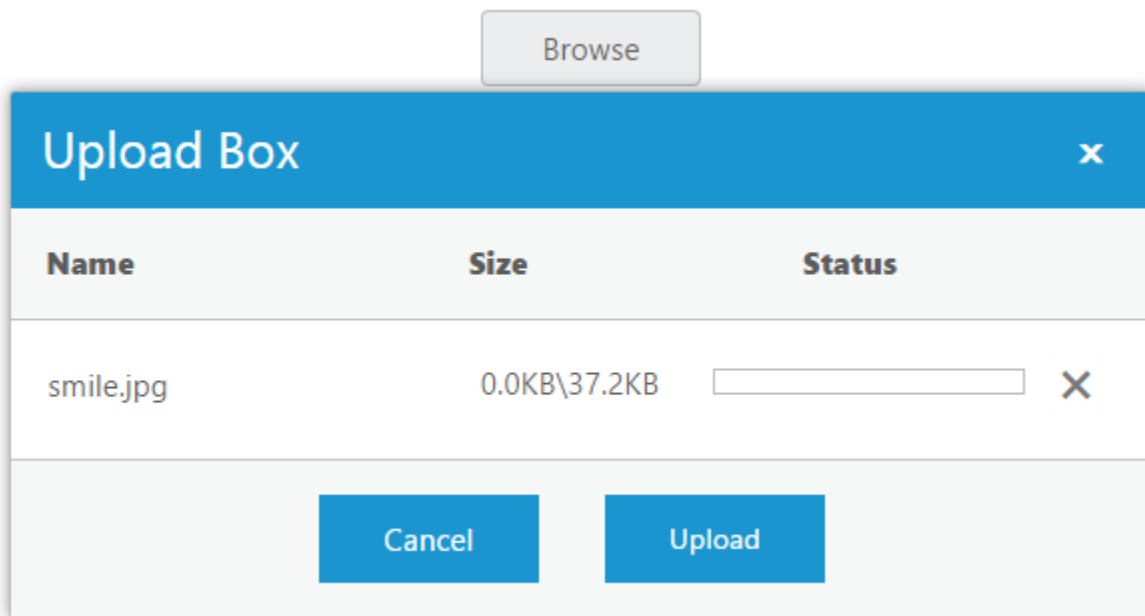
```

var DefaultUploadbox = React.createClass({
    render: function () {

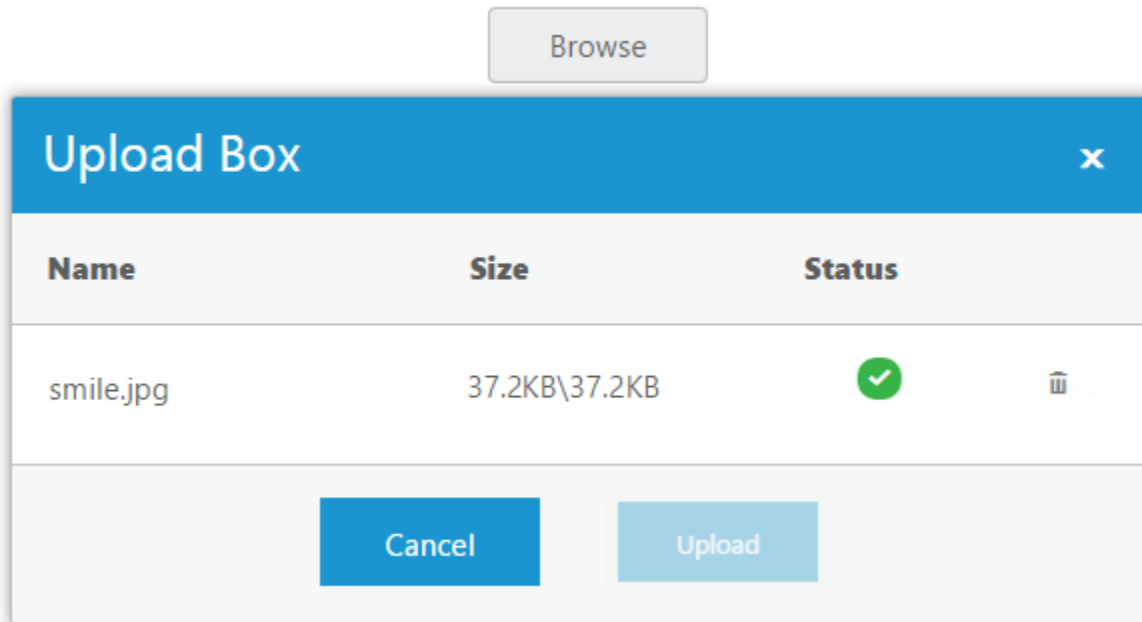
```

```
return (  
  <div id="tooltip_default">  
    <div class="posupload">Select a file to upload </div>  
    <EJ.Uploadbox id="UploadDefault" saveUrl= {savefiles} removeUrl=  
    {removefiles}></EJ.Uploadbox>  
  </div>  
);  
}  
});  
ReactDOM.render(<DefaultUploadbox />, document.getElementById('uploadbox-  
default'));
```

The following screenshot displays an **Uploadbox** control.



After you upload the files, the following screen shot is displayed.



**Note:** The above screenshot displays the Uploadbox control that shows the files are uploaded successfully.

#### Set Restriction for File Extension

In a real-time scenario, some file extensions are restricted. You can allow files and restrict files by using the following two properties **extensionsAllow** and **extensionsDeny** enabled in **Uploadbox**.

**Note:** The SaveUrl and RemoveUrl are the same as above (see step 4)

Add input elements to create elements for file extension.

**Note:** Add the following input elements and two button elements to give file extensions that should support uploading.

#### JS

```
var DefaultUploadbox = React.createClass({
  function onClickAllow(args) {
    var uploadobject = $("#uploadDefault").data("ejUploadbox");
    uploadobject.option('extensionsAllow', $("#fileallow").val());
    uploadobject.option('extensionsDeny', "");
  }
  function onclickDeny() {
    var uploadobject = $("#uploadDefault").data("ejUploadbox");
    uploadobject.option('extensionsAllow', "");
    uploadobject.option('extensionsDeny', $("#filedeny").val());
  }
  render: function () {
    return (
      <div id="uploadbox_default"><table id="uploadTable">
        <tr>
          <td>
            Extensions:
          </td>
        </tr>
      </table>
    )
  }
});
```



```

</tr>
<tr>
<td>
<input type="text" id="fileallow" class="ejinputtext" placeholder="Format"
/>
<input ej-button type="submit" name="allow" value="Allow"
click={onClickAllow} />
</td>
</tr>
<tr>
<td>
<input type="text" id="filedeny" class="ejinputtext" placeholder="Format" />
<input ej-button type="submit" name="deny" value="Deny" click={onclickDeny}
/>
</td>
<td>
<div>Select a file to upload</div>
<div style="width:100px;height:35px;">
<EJ.Uploadbox id="UploadDefault" saveUrl= {savefiles} removeUrl=
{removefiles}></EJ.Uploadbox>
</td>
</tr>
</table>
</div>
);
}
});
ReactDOM.render(<DefaultUploadbox />, document.getElementById('uploadbox-
default'));

```

Add the given styles to display the **Uploadbox** with margin alignments.

### CSS

```

<style>
#targetElement {
width: 520px;
height: 500px;
margin: 0 auto;
}
#UploadDefault {
float: right;
}
#uploadTable {
width: 100%;
}
#fileallow, #filedeny {
width: 150px;
height: 20px;
padding: 5px;
}
</style>

```

**Note:** You can restrict one or more files at a time by giving it as .html,.txt

The following screenshot displays an **Uploadbox** control with the file extension.

Extensions:

Upload Box

Name	Size	Status
Tools.txt	1.1KB\1.1KB	<div><div>✓</div><div>🗑</div></div>

Cancel

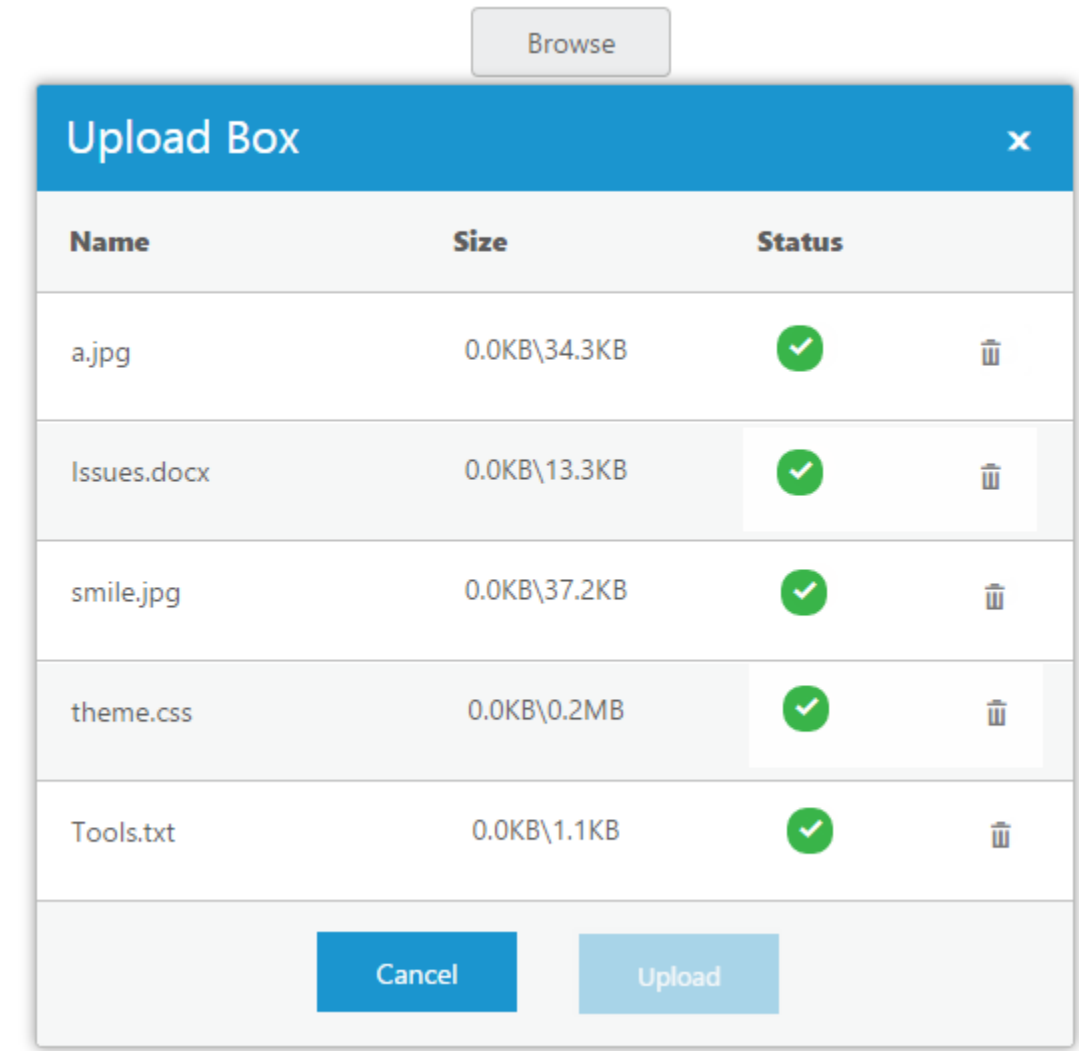
Upload

The above screenshot shows the **Uploadbox** that allows “.txt” file formats. You can give number of file formats in both allow and deny textbox elements.

#### Upload Multiple Files

You can click the **Browse** button and select files to upload multiple files in **Uploadbox** control. You can see the selected files in **Uploadbox** control and you can upload all the files.

The following screenshot displays an **Uploadbox** control with multiple files.



## WaitingPopup

### Overview

The **WaitingPopup control for JavaScript** is a visual element that provides support for displaying a pop-up indicator over a target area and preventing the end user's interaction with the target area while loading.

### Key Features

- **Custom text:** Supports custom text inside the pop-up panel.
- **Template:** Supports including HTML content instead of the default image.
- **Transparency:** Supports customizing the transparency and opacity level.
- **Themes:** JavaScript controls include 12 built-in themes (6 flat and 6 gradient effects) and also support the custom skin option to set user-defined themes.

### Getting Started

This section explains briefly about how to create a **WaitingPopup** in your application with **JavaScript**.

**Essential JavaScript WaitingPopup** provides support to display a **WaitingPopup** within your web page. From the following guidelines, you can learn how to create a **WaitingPopup** in a real-time login page authentication scenario.

The following screenshot illustrates the functionality of a **WaitingPopup** with login page scenario.



You can give the Username and Password in the **login page**. When you click the **Login** button, you get the **WaitingPopup**. After loading, the alert box pops up with the message "Signed in successfully".

#### Create Username and Password

**Essential JavaScript WaitingPopup** widget basically renders built-in features like blocking the other actions until the page is loaded. You can easily create the **WaitingPopup** widget by using simple **<div>** element as follows.

You can create a React application and add necessary scripts and styles with the help of the given [React Getting Started Documentation](#).

Create a JSX file for rendering WaitingPopup component using **<EJ.WaitingPopup>** syntax. Add required properties to it in **<EJ.WaitingPopup>** tag element

#### JS

```
var Defaultwait = React.createClass({
  componentDidMount: function () {
  },
  render: function () {
    return (
      <EJ.WaitingPopup></EJ.WaitingPopup>
    );
  }
});
ReactDOM.render(<Defaultwait />, document.getElementById('waitingpopup-default'));
```

Define an HTML element for adding WaitingPopup in the application and refer the JSX file.

#### HTML

```
<div id="targetElement">
  <table class="loginTable">
    <tr>
      <td>Username</td>
      <td><input type="text" /></td>
    </tr>
    <tr>
      <td>Password</td>
```

```
<td><input type="password"/></td>
</tr>
<tr>
<td></td>
<td><button id="target">Login</button></td>
</tr>
</table>
<div id="popup"></div>
</div>
```

Initialize **Click function** using the following code example.

### JS

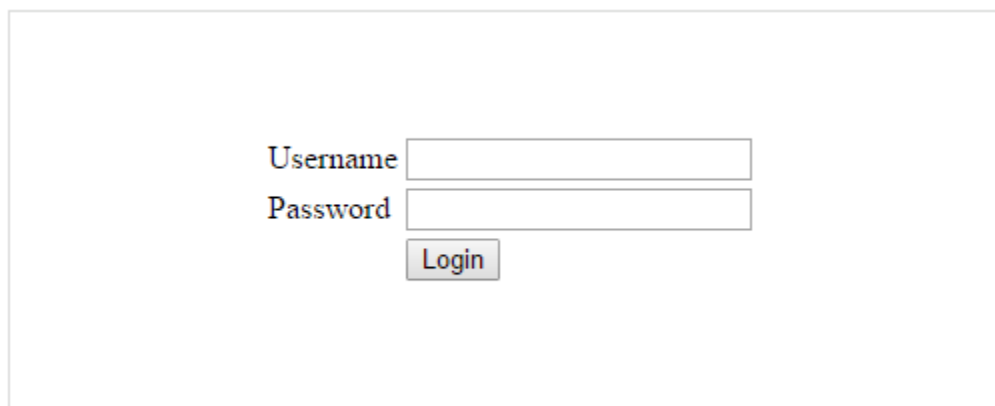
```
$(document).ready(function () {
$("#target").click(function () {
/*Add waiting popup*/
});
});
```

Apply the following styles to show the **WaitingPopup**.

### CSS

```
<style type="text/css" class="cssStyles">
#targetElement {
width: 500px;
height: 200px;
margin: 50px;
border: 1px solid #dbdcdb;
}
.loginTable {
margin: 60px auto;
}
#popup_WaitingPopup .e-image {
display: block;
height: 70px;
}
</style>
```

The following screenshot displays a **User login**.

A screenshot of a user login form. It features two input fields: one for 'Username' and one for 'Password'. Below the password field is a 'Login' button. The form is centered on a light gray background.

### Add WaitingPopup Widget

In a real-time login page scenario, when you click the Login button, the WaitingPopup is displayed.

#### JS

```
var Defaultwait = React.createClass({
  componentDidMount: function () {
    $("#target").click(function () {
      $("#popup").ejWaitingPopup({
        showOnInit: true,
        target: "#targetElement"
      });
      function success() {
        var obj = $("#popup").data("ejWaitingPopup");
        alert("Signed in successfully");
        obj.hide();
      }
      setTimeout(success, 5000);
    });
  },
  render: function () {
    return (
      <EJ.WaitingPopup></EJ.WaitingPopup>
    );
  }
});
ReactDOM.render(<Defaultwait />, document.getElementById('waitingpopup-default'));
```

The following screenshot shows the output of the above code example.

