OxyPlot Documentation

Release 2015.1

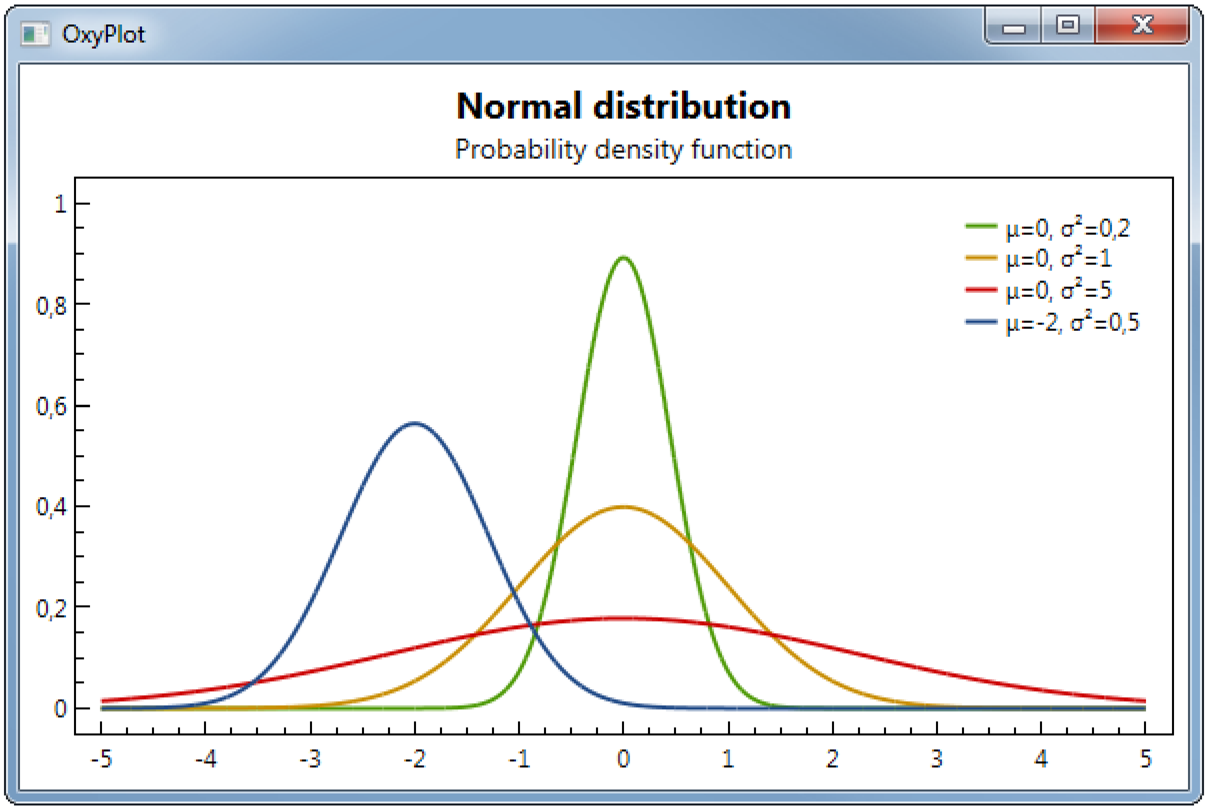
OxyPlot Contributors

April 19, 2016

Contents

Note: The documentation is under construction. Please contribute!

OxyPlot is a cross-platform plotting library for .NET



CHAPTER 1

Content

1.1 Introduction

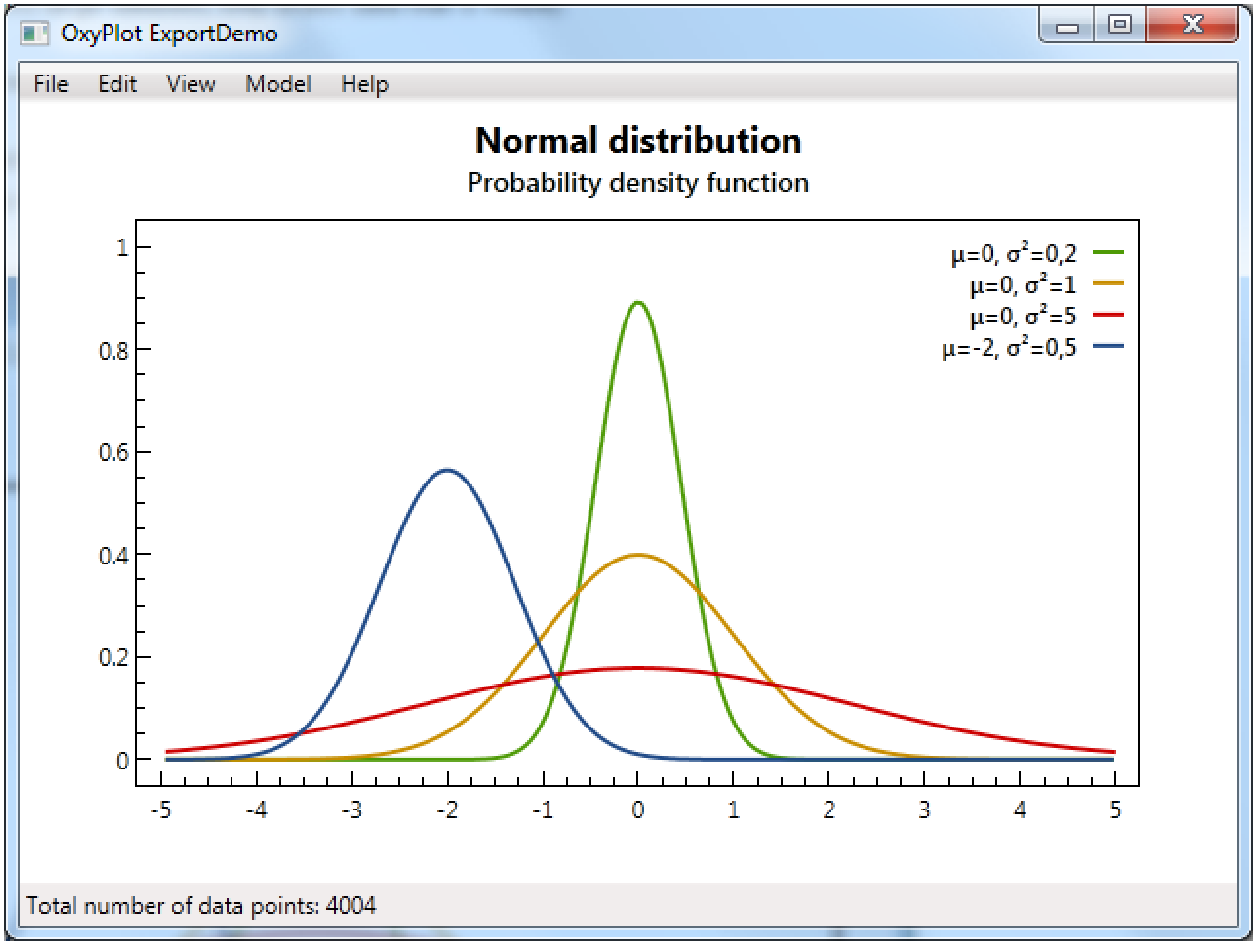
1.1.1 Introduction

OxyPlot is an open source plot generation library that is licensed under the MIT license. The MIT license is very permissive and permits use in proprietary software.

The library is based on .NET and targets multiple platforms. The core library is a portable library, which makes it easy to re-use plotting code on different platforms.

The goal has been to provide a plotting model that is simple to use and open for extension. It should be possible to change the appearance of your plots by changing simple properties. More customization can be achieved by deriving new subclasses that have full control of both behaviour and rendering.

The features of the library can be explored in the example applications included in the source code.



1.1.2 Background

OxyPlot was started in 2010 as a simple WPF plotting component, focusing on simplicity, performance and visual appearance.

The style of the plots is inspired by the books written by Edward Tufte and Stephen Few. Their principles have been imporant when designing this component.

OxyPlot is primarily focused on two-dimensional coordinate systems, that's the reason for the 'xy' in the name!

1.1.3 Features

Note: This section is under construction. Please contribute!

Plot types

- XY (horizontal and vertical axes)
- Cartesian (same scale on X and Y axis)
- Polar

- Pie chart

Axes

- Multiple axes
- It is possible to extend with custom axis types.

Series

- Different types of series can be added to the same plot.
- It is possible to extend with custom series types.

Annotations

- It is possible to extend with custom annotation types.

Export

The plots can be exported to the following raster and vector file formats:

- png
- svg
- pdf

Limitations

- the plot controls are not observing changes on properties and collections. You must manually refresh the plots when changing your data.
- animations are not supported
- gradient and hatch brushes are not supported

1.1.4 License

OxyPlot is licensed under the MIT license. This is the shortest and probably broadest of all the popular open-source licenses. Its terms are very loose and more permissive than most other licenses. It is also compatible with GPL.

The license can be found in the LICENSE file that is located in the root folder of the repository:

The license is explained in a short guide to open source licenses:

- You can use, copy and modify the software however you want. No one can prevent you from using it on any project, from copying it however many times you want and in whatever format you like, or from changing it however you want.
- You can give the software away for free or sell it. You have no restrictions on how to distribute it.
- The only restriction is that it be accompanied by the license agreement.

The MIT License is the least restrictive license out there. It basically says that anyone can do whatever they want with the licensed material, as long as it is accompanied by the license.

See also the explanation on tl;dr.

Authors

Information about the copyright holders is found in the AUTHORS file.

Contributors

Information about the contributors is found in the CONTRIBUTORS file.

1.1.5 Getting Support

Community Support

OxyPlot has a small community that may help you out if you have questions. If you can't find your answer in the documentation, you can...

- Ask your question on the forum. This is a good idea if you have multiple questions or a question that might have a more subjective answer (e.g., "recommendations" sorts of things).
- You can also try StackOverflow. Be sure to use the `oxyplot` tag in your question.

Commercial Support

- We are not aware of anyone offering commercial support

If you would like your company listed here, please let us know.

1.1.6 Frequently asked questions

Note: This section is under construction. Please contribute!

See also the FAQ section at http://discussion.oxyplot.org.

Where is the source code?

The source code can be found at GitHub: https://github.com/oxyplot/oxyplot.

Where can I find examples?

All solutions contain an "ExampleBrowser" project that can be used to render the examples defined in the "ExampleLibrary" project.

The example library contains platform independent code and the source can be found in ~Source\Examples\ExampleLibrary\Examples. See also the platform specific examples in the Examples folder.

Why are the plots not showing up in my Xamarin.Forms project?

Did you forget to add a reference to the OxyPlot platform projects and initialize the renderers? See getting started for more information.

1.1.7 Glossary

Note: This section is under construction. Please contribute!

The goal of this page is to help keep documentation, discussions, and APIs consistent.

Term	Meaning
Plot	Also called a graph or chart
Plot model	A model represents the contents of a plot
Plot element	An element of the plot model that can be displayed (e.g. a series, annotation or axis)
Plot controller	Handles user input
Plot view	The custom control that displays the plot model and communicates with the plot controller
Axis	A plot element that displays an axis
Series	A plot element that displays data
Annotation	Displays content that is not a series. Annotations are not included in the legend and not used by the tracker
Plot area	The area where the series are displayed
Legend	Displays the titles and symbol of the series.
Tracker	When the user hovers or clicks on a series, the tracker shows the actual values at that point

Note: Alternative layout (definition syntax)

Plot Also called a graph or chart

Plot model A model represents the contents of a plot

Plot element An element of the plot model that can be displayed (e.g. a series, annotation or axis)

Plot controller Handles user input

Plot view The custom control that displays the plot model and communicates with the plot controller

Axis A plot element that displays an axis

Series A plot element that displays data

Annotation Displays content that is not a series. Annotations are not included in the legend and not used by the tracker

Plot area The area where the series are displayed

Legend Displays the titles and symbol of the series.

Tracker When the user hovers or clicks on a series, the tracker shows the actual values at that point

Wild deviations from these terms in the API or code should be fixed or raised as issues to fix in a future version.

1.2 Getting started

Note: This section is under construction. Please contribute!

OxyPlot is provided as NuGet packages. To use the library, you must add a reference to the NuGet package that supports the platform of your application.

<TODO> Show an overview of NuGet packages... Which one to choose...

<TODO> General info about how to get started

The following sections contain platform specific descriptions about how to get started:

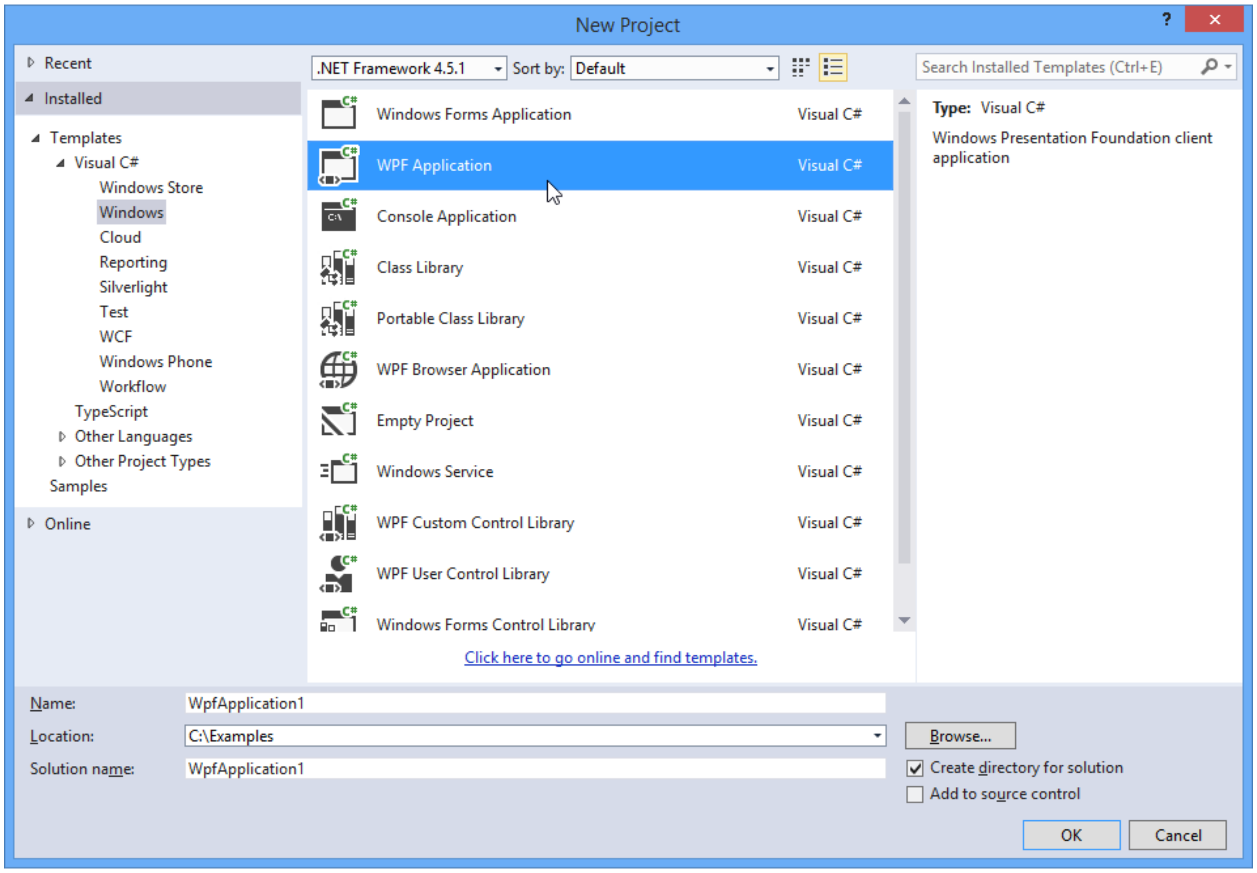
1.2.1 WPF

Note: This section is under construction. Please contribute!

This example shows how to create a WPF application with a `PlotView` control, where the content of the plot is defined in code as a `PlotModel`. If you want to define the content of the plot in XAML, see WPF (XAML).

Create project

Start Visual Studio and select “FILE -> New -> Project...” to create a new WPF application:


New Project
Recent
Installed
Templates
Visual C#
Windows Store
Windows
Cloud
Reporting
Silverlight
Test
WCF
Windows Phone
Workflow
TypeScript
Other Languages
Other Project Types
Samples
Online
.NET Framework 4.5.1
Sort by: Default
Search Installed Templates (Ctrl+E)
Windows Forms Application Visual C#
WPF Application Visual C#
Console Application Visual C#
Class Library Visual C#
Portable Class Library Visual C#
WPF Browser Application Visual C#
Empty Project Visual C#
Windows Service Visual C#
WPF Custom Control Library Visual C#
WPF User Control Library Visual C#
Windows Forms Control Library Visual C#
Click here to go online and find templates.
Browse...
OK
Cancel

Type: Visual C#

Windows Presentation Foundation client application

Name: WpfApplication1

Location: C:\Examples

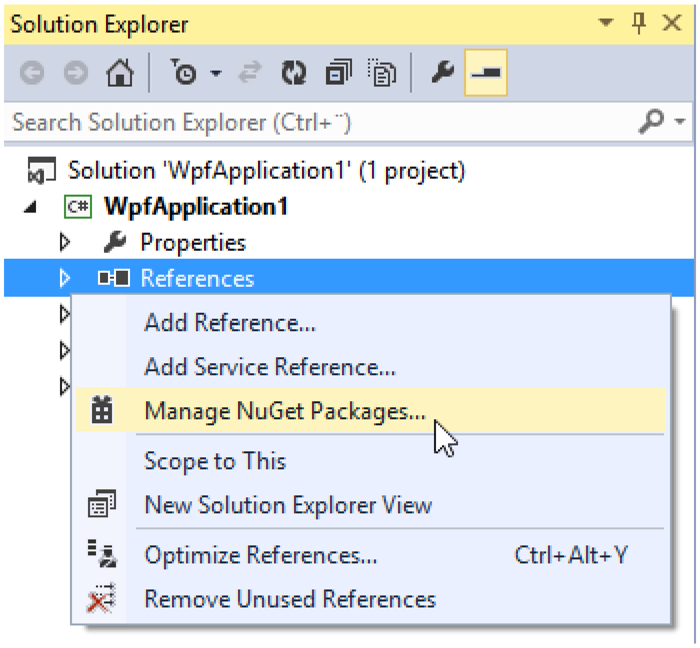
Solution name: WpfApplication1

[✓] Create directory for solution

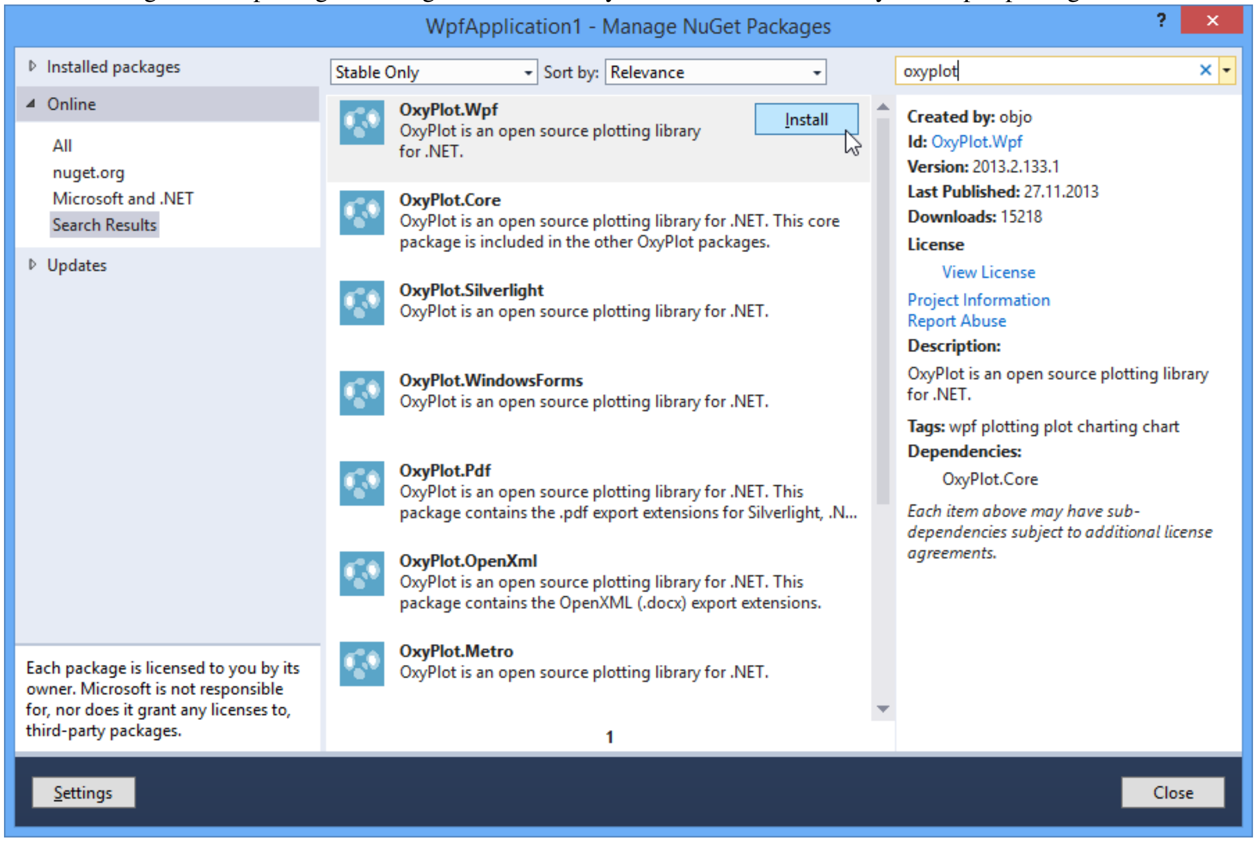
[ ] Add to source control

Add references

You need references to the OxyPlot and OxyPlot.Wpf assemblies. The easiest way to add these is to right click on the "References" item in the Solution Explorer and select "Manage NuGet Packages..." (this requires that the "NuGet Package Manager" extension is installed):



In the "Manage NuGet packages" dialog, search for "OxyPlot" and select the "OxyPlot.Wpf" package:


WpfApplication1 - Manage NuGet Packages

Installed packages
Online
All
nuget.org
Microsoft and .NET
Search Results
Updates

Stable Only · Sort by: Relevance · oxyplot

OxyPlot.Wpf
OxyPlot is an open source plotting library for .NET.
Install

OxyPlot.Core
OxyPlot is an open source plotting library for .NET. This core package is included in the other OxyPlot packages.

OxyPlot.Silverlight
OxyPlot is an open source plotting library for .NET.

OxyPlot.WindowsForms
OxyPlot is an open source plotting library for .NET.

OxyPlot.Pdf
OxyPlot is an open source plotting library for .NET. This package contains the .pdf export extensions for Silverlight, .N...

OxyPlot.OpenXml
OxyPlot is an open source plotting library for .NET. This package contains the OpenXML (.docx) export extensions.

OxyPlot.Metro
OxyPlot is an open source plotting library for .NET.

1

Created by: objo
Id: OxyPlot.Wpf
Version: 2013.2.133.1
Last Published: 27.11.2013
Downloads: 15218
License
View License
Project Information
Report Abuse
Description:
OxyPlot is an open source plotting library for .NET.
Tags: wpf plotting plot charting chart
Dependencies:
OxyPlot.Core
Each item above may have sub-dependencies subject to additional license agreements.

Each package is licensed to you by its owner. Microsoft is not responsible for, nor does it grant any licenses to, third-party packages.

Settings · Close

You can also use the Package Manager Console to install the package:

```
PM> Install-Package OxyPlot.Wpf -Pre
```

Create a view model

Add a class that creates a `PlotModel` with a `FunctionSeries`.

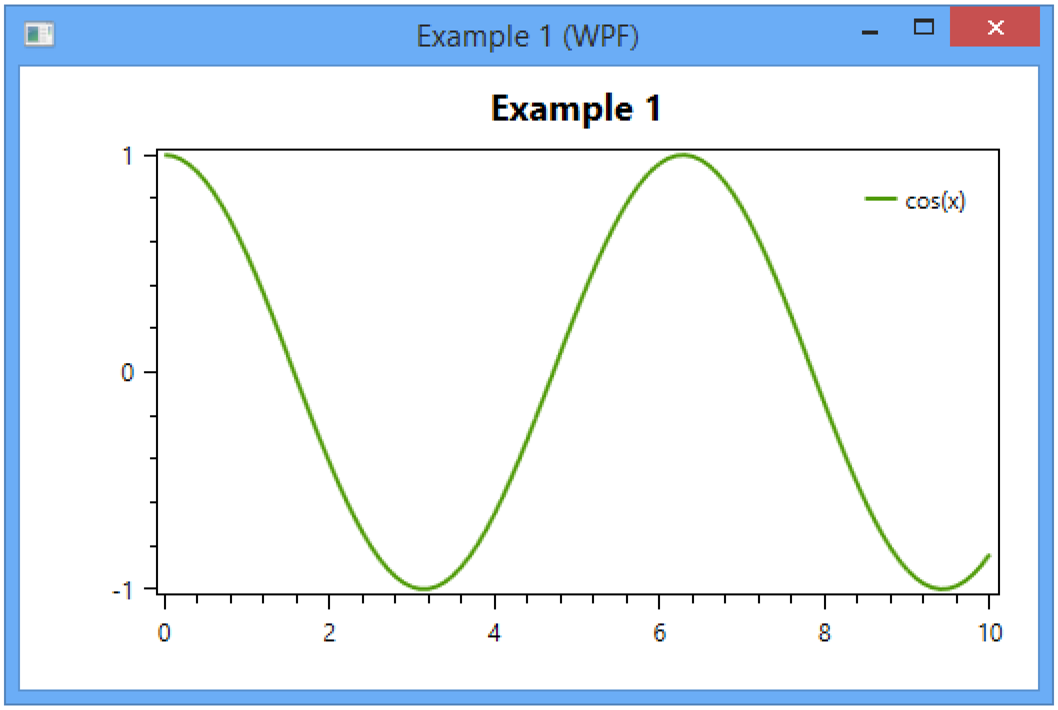
```
namespace WpfApplication1
{
    using System;

    using OxyPlot;
    using OxyPlot.Series;

    public class MainViewModel
    {
        public MainViewModel()
        {
            this.MyModel = new PlotModel { Title = "Example 1" };
            this.MyModel.Series.Add(new FunctionSeries(Math.Cos, 0, 10, 0.1, "cos(x)"));
        }

        public PlotModel MyModel { get; private set; }
    }
}
```

Create the view

Define the namespace in the `Window` element, set the `DataContext` and add a `PlotView` control:

```
<Window x:Class="WpfApplication1.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml" xmlns:oxy="http://oxyplot.org/wpf"
        xmlns:local="clr-namespace:WpfApplication1"
        Title="Example 1 (WPF)" Height="350" Width="525">
    <Window.DataContext>
        <local:MainViewModel/>
    </Window.DataContext>
    <Grid>
        <oxy:PlotView Model="{Binding MyModel}"/>
    </Grid>
</Window>
```

The application should now look like this:



The source code can be found in the HelloWorld\WpfApplication1 folder in the documentation-examples repository.

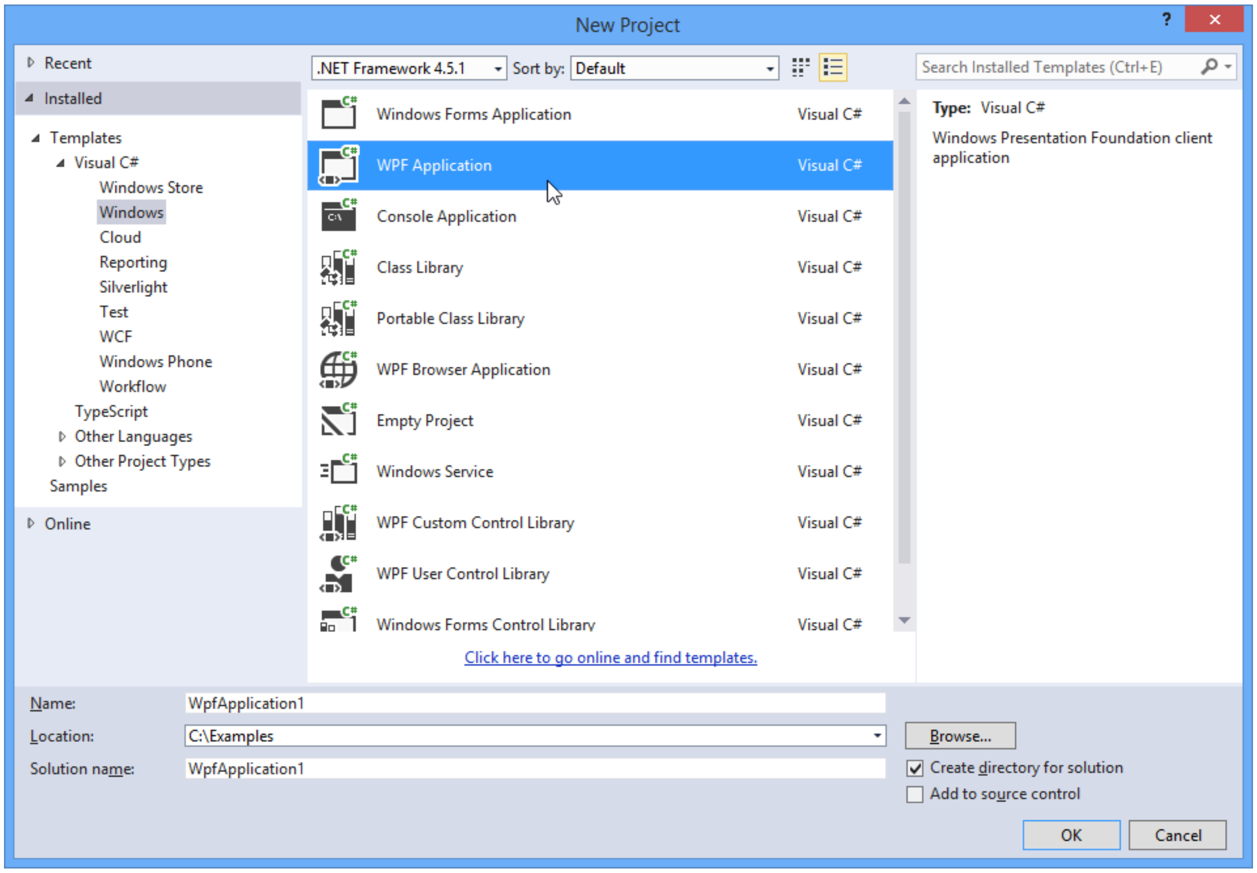
1.2.2 WPF (XAML)

Note: This section is under construction. Please contribute!

This example shows how to create a WPF application with a `Plot` control, where the content of the plot (axes, series, annotations) is defined in XAML.

Create project

Start Visual Studio and select "FILE -> New -> Project..." to create a new WPF application:


New Project
Recent
Installed
Templates
Visual C#
Windows Store
Windows
Cloud
Reporting
Silverlight
Test
WCF
Windows Phone
Workflow
TypeScript
Other Languages
Other Project Types
Samples
Online
Sort by:
Search Installed Templates (Ctrl+E)
Windows Forms Application Visual C#
WPF Application Visual C#
Console Application Visual C#
Class Library Visual C#
Portable Class Library Visual C#
WPF Browser Application Visual C#
Empty Project Visual C#
Windows Service Visual C#
WPF Custom Control Library Visual C#
WPF User Control Library Visual C#
Windows Forms Control Library Visual C#
Click here to go online and find templates.
Browse...
OK
Cancel

.NET Framework 4.5.1

Sort by: Default

Type: Visual C#

Windows Presentation Foundation client application

Name: WpfApplication1

Location: C:\Examples

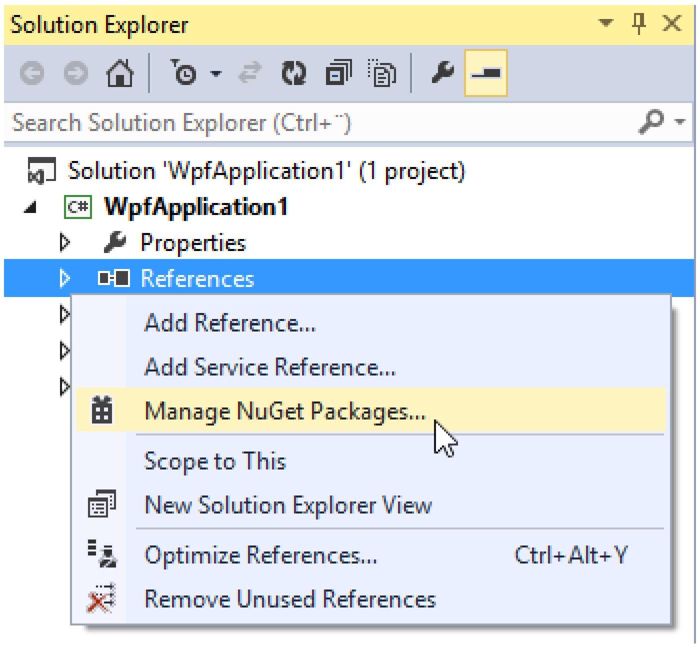
Solution name: WpfApplication1

[✓] Create directory for solution

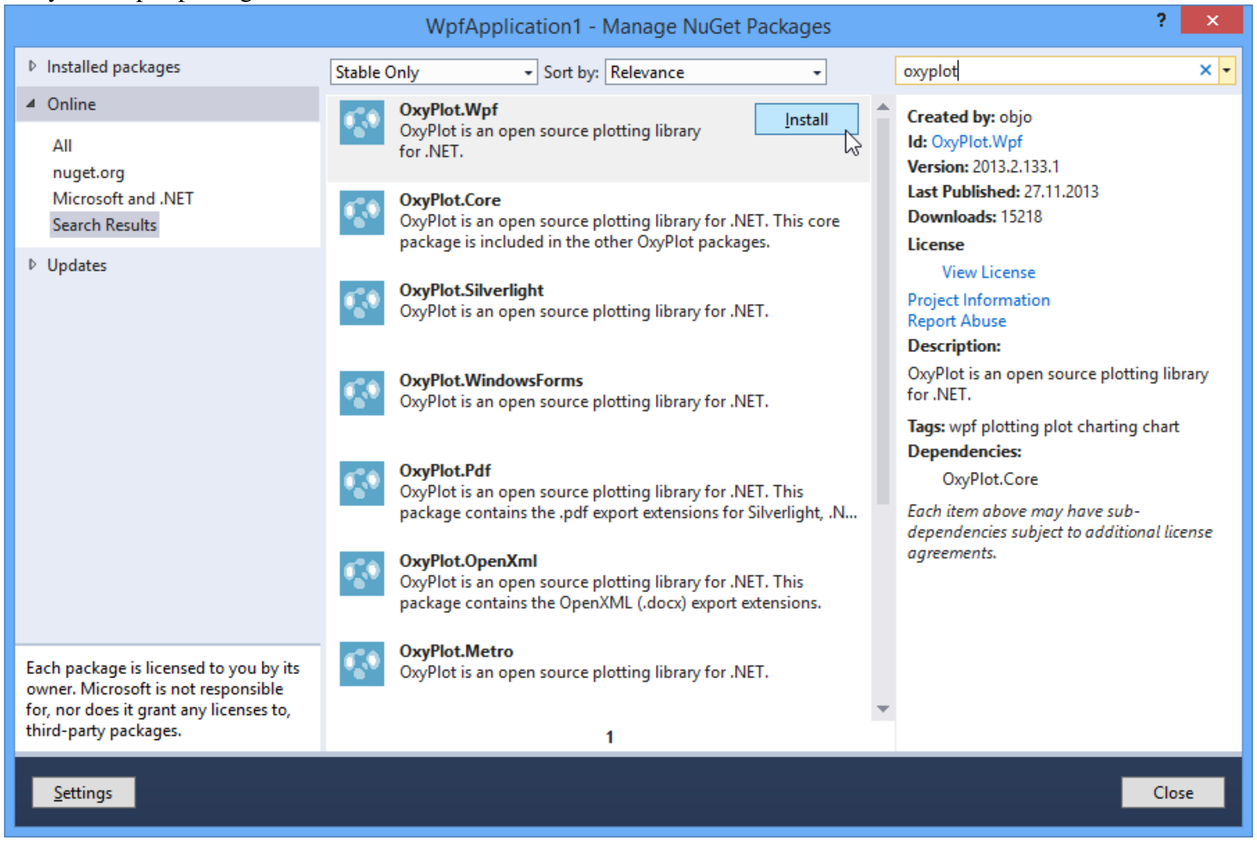
[ ] Add to source control

Add references

You need references to the OxyPlot and OxyPlot.Wpf assemblies. The easiest way to add these is to right click on the “References” item in the Solution Explorer and select “Manage NuGet Packages...” (this requires that the “NuGet Package Manager” extension is installed):



In the "Manage NuGet packages" dialog, search for "OxyPlot" (activate the Include prerelease option) and select the "OxyPlot.Wpf" package:



OxyPlot.Wpf
OxyPlot is an open source plotting library for .NET.

OxyPlot.Core
OxyPlot is an open source plotting library for .NET. This core package is included in the other OxyPlot packages.

OxyPlot.Silverlight
OxyPlot is an open source plotting library for .NET.

OxyPlot.WindowsForms
OxyPlot is an open source plotting library for .NET.

OxyPlot.Pdf
OxyPlot is an open source plotting library for .NET. This package contains the .pdf export extensions for Silverlight, .N...

OxyPlot.OpenXml
OxyPlot is an open source plotting library for .NET. This package contains the OpenXML (.docx) export extensions.

OxyPlot.Metro
OxyPlot is an open source plotting library for .NET.

Created by: objo
Id: OxyPlot.Wpf
Version: 2013.2.133.1
Last Published: 27.11.2013
Downloads: 15218
License
View License
Project Information
Report Abuse
Description:
OxyPlot is an open source plotting library for .NET.
Tags: wpf plotting plot charting chart
Dependencies:
OxyPlot.Core

Each item above may have sub-dependencies subject to additional license agreements.

Each package is licensed to you by its owner. Microsoft is not responsible for, nor does it grant any licenses to, third-party packages.

You can also use the Package Manager Console to install the package:

```
PM> Install-Package OxyPlot.Wpf -Pre
```

Create a view model

Then create a class that defines the data to be plotted.

```
namespace WpfApplication2
{
    using System.Collections.Generic;

    using OxyPlot;

    public class MainViewModel
    {
        public MainViewModel()
        {
            this.Title = "Example 2";
            this.Points = new List<DataPoint>
                              {
                                  new DataPoint(0, 4),
                                  new DataPoint(10, 13),
                                  new DataPoint(20, 15),
                                  new DataPoint(30, 16),
                                  new DataPoint(40, 12),
                                  new DataPoint(50, 12)
                              };
        }

        public string Title { get; private set; }

        public IList<DataPoint> Points { get; private set; }
    }
}
```

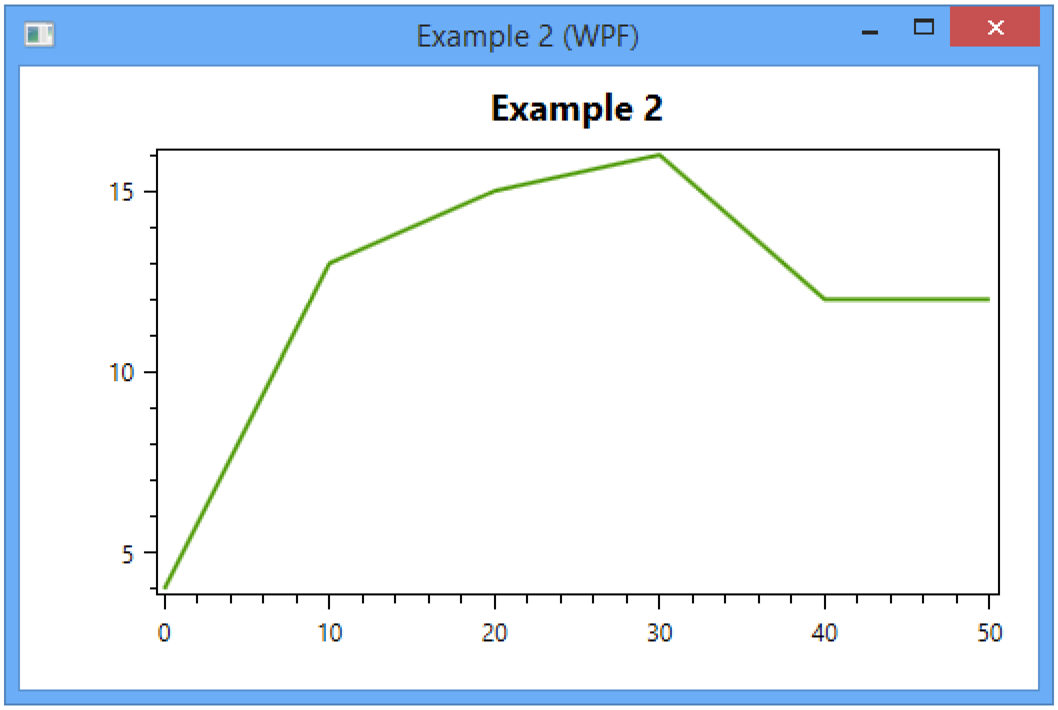
Create the view

Define the namespace in the `Window` element, set the `DataContext` and add a `Plot` control:

```
<Window x:Class="WpfApplication2.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml" xmlns:oxy="http://oxyplot.org/wpf"
        xmlns:local="clr-namespace:WpfApplication2"
        Title="Example 2 (WPF)" Height="350" Width="525">
    <Window.DataContext>
        <local:MainViewModel/>
    </Window.DataContext>
    <Grid>
        <oxy:Plot Title="{Binding Title}">
            <oxy:Plot.Series>
                <oxy:LineSeries ItemsSource="{Binding Points}"/>
            </oxy:Plot.Series>
        </oxy:Plot>
    </Grid>
</Window>
```

If you want to add a `Plot` control in the design view, press `Choose Items...` in the Toolbox and browse for the `OxyPlot.Wpf.dll` file. If you used NuGet, it should be located in the *packages* folder in your solution folder.

The application should now look like this:



The source code can be found in the HelloWorld\WpfApplication2 folder in the documentation-examples repository.

1.2.3 WPF (VB.NET)

Note: This section is under construction. Please contribute!

This example shows how to create a WPF application in VB.NET where the content of the plot is defined in code.

Create a new VB.NET Windows/WPF project and add a reference to the OxyPlot.Wpf NuGet package. Then create a view model class:

```
Imports OxyPlot
Imports OxyPlot.Series

Public Class MainViewModel

    Private mmodel As PlotModel

    Public Sub New()

        Model = New PlotModel()

        Model.Title = "Simple example"
        Model.Subtitle = "using OxyPlot in VB.NET"

        Dim series1 = New LineSeries()
        series1.Title="Series 1"
        series1.MarkerType = MarkerType.Circle
        series1.Points.Add(New DataPoint(0, 0))
        series1.Points.Add(New DataPoint(10, 18))
        series1.Points.Add(New DataPoint(20, 12))
        series1.Points.Add(New DataPoint(30, 8))
        series1.Points.Add(New DataPoint(40, 15))

        Dim series2 = New LineSeries()
        series2.Title="Series 2"
        series2.MarkerType = MarkerType.Square
        series2.Points.Add(New DataPoint(0, 4))
        series2.Points.Add(New DataPoint(10, 12))
        series2.Points.Add(New DataPoint(20, 16))
        series2.Points.Add(New DataPoint(30, 25))
        series2.Points.Add(New DataPoint(40, 5))

        Model.Series.Add(series1)
        Model.Series.Add(series2)

    End Sub

    Property Model() As PlotModel
        Get
            Return mmodel
        End Get
        Set(value As PlotModel)
            mmodel = value
        End Set
    End Property

End Class
```

and edit the main window XAML:

```
<Window x:Class="MainWindow"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:oxy="http://oxyplot.org/wpf"
    xmlns:local="clr-namespace:SimleDemoVB"
    Title="OxyPlot in VB.NET" Height="480" Width="640" Icon="OxyPlot.ico">
    <Window.DataContext>
        <local:MainViewModel/>
    </Window.DataContext>
    <Grid>
        <oxy:PlotView Model="{Binding Model}"/>
    </Grid>
</Window>
```

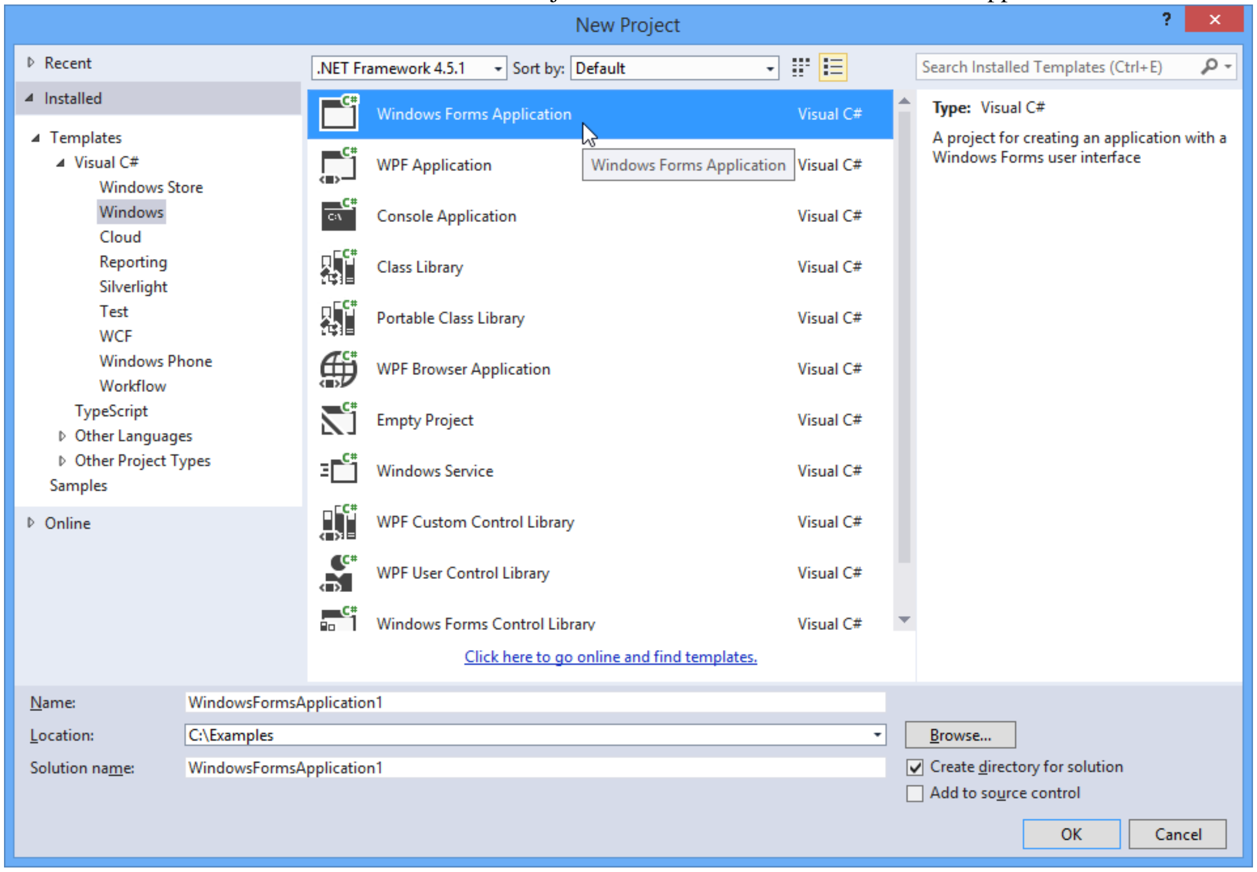
The source code can be found in the Source\Examples\WPF\SimpleDemoVB folder in the oxyplot repository.

1.2.4 Windows.Forms

Note: This section is under construction. Please contribute!

Create project

Start Visual Studio and select “FILE -> New -> Project...” to create a new Windows Forms application:


New Project
Recent
Installed
Templates
Visual C#
Windows Store
Windows
Cloud
Reporting
Silverlight
Test
WCF
Windows Phone
Workflow
TypeScript
Other Languages
Other Project Types
Samples
Online
.NET Framework 4.5.1
Sort by: Default
Search Installed Templates (Ctrl+E)
Windows Forms Application Visual C#
WPF Application Visual C#
Console Application Visual C#
Class Library Visual C#
Portable Class Library Visual C#
WPF Browser Application Visual C#
Empty Project Visual C#
Windows Service Visual C#
WPF Custom Control Library Visual C#
WPF User Control Library Visual C#
Windows Forms Control Library Visual C#
Click here to go online and find templates.
Browse...
OK
Cancel

Type: Visual C#

A project for creating an application with a Windows Forms user interface

Name: WindowsFormsApplication1

Location: C:\Examples

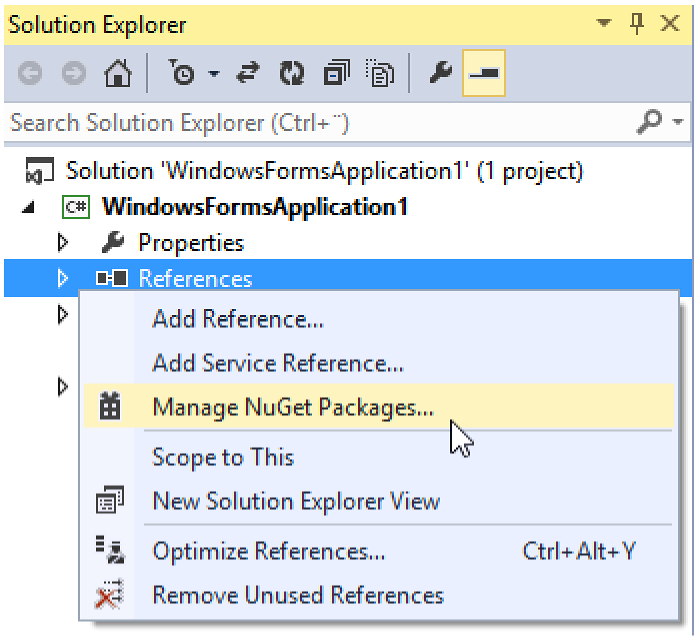
Solution name: WindowsFormsApplication1

[✓] Create directory for solution

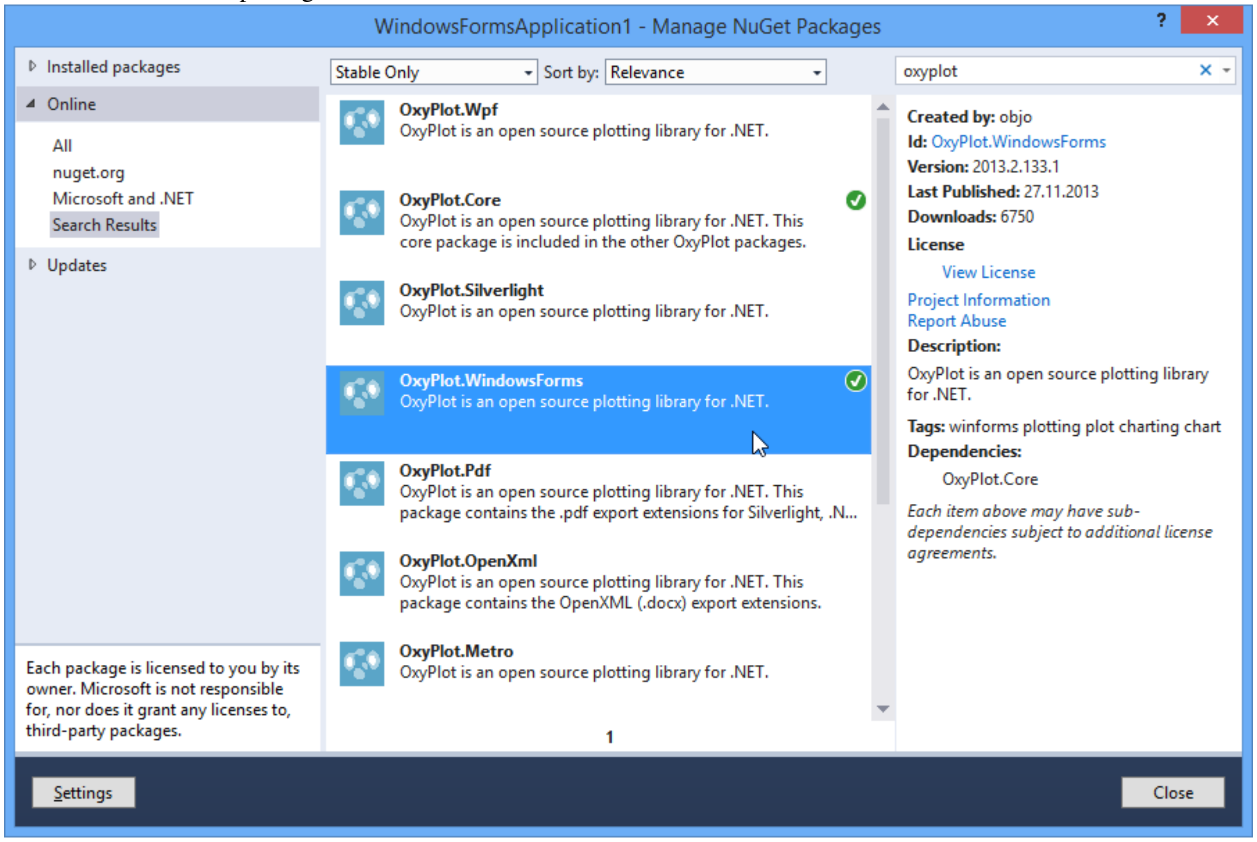
[ ] Add to source control

Add references

You need references to the OxyPlot and OxyPlot.WindowsForms assemblies. The easiest way to do this is to right click on the “References” item in the Solution Explorer and select “Manage NuGet Packages...” (this requires that the “NuGet Package Manager” extension is installed):



In the "Manage NuGet packages" dialog, search for "OxyPlot" in the top-right search textbox. Select the "OxyPlot.WindowsForms" package and click install:



OxyPlot.Wpf
OxyPlot is an open source plotting library for .NET.

OxyPlot.Core
OxyPlot is an open source plotting library for .NET. This core package is included in the other OxyPlot packages.

OxyPlot.Silverlight
OxyPlot is an open source plotting library for .NET.

OxyPlot.WindowsForms
OxyPlot is an open source plotting library for .NET.

OxyPlot.Pdf
OxyPlot is an open source plotting library for .NET. This package contains the .pdf export extensions for Silverlight, .N...

OxyPlot.OpenXml
OxyPlot is an open source plotting library for .NET. This package contains the OpenXML (.docx) export extensions.

OxyPlot.Metro
OxyPlot is an open source plotting library for .NET.

Created by: objo
Id: OxyPlot.WindowsForms
Version: 2013.2.133.1
Last Published: 27.11.2013
Downloads: 6750
License
View License
Project Information
Report Abuse
Description:
OxyPlot is an open source plotting library for .NET.
Tags: winforms plotting plot charting chart
Dependencies:
OxyPlot.Core

Each item above may have sub-dependencies subject to additional license agreements.

Each package is licensed to you by its owner. Microsoft is not responsible for, nor does it grant any licenses to, third-party packages.

You can also use the Package Manager Console to install the package:

```
PM> Install-Package OxyPlot.WindowsForms
```

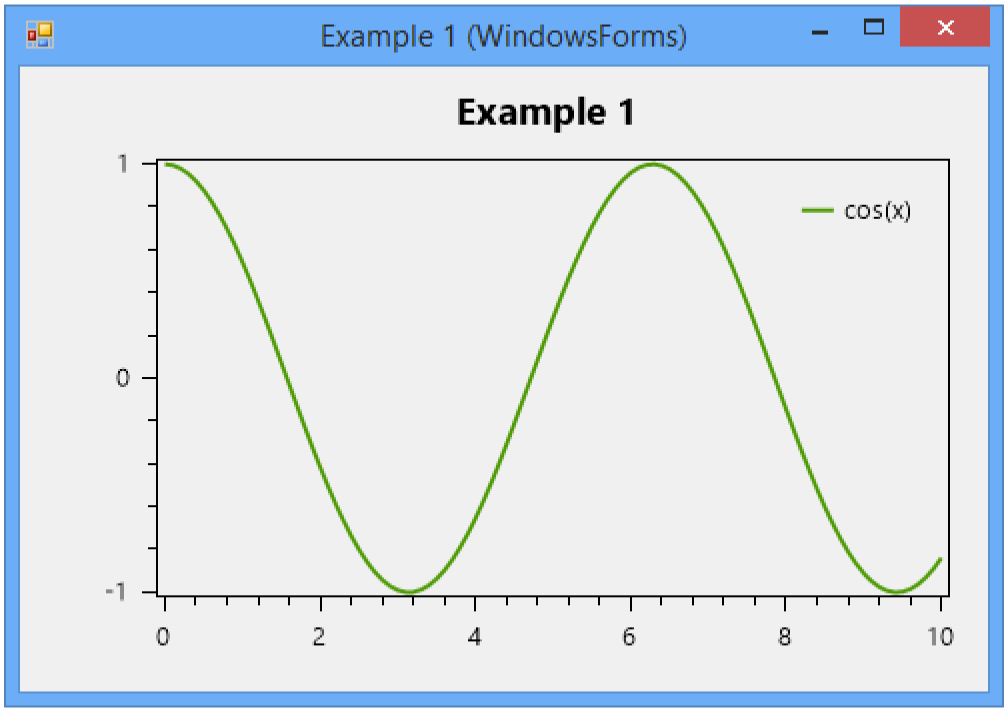
Add a plot view

TODO (add component to toolbox?)

Create the plot model

```
namespace WindowsFormsApplication1
{
    using System;
    using System.Windows.Forms;

    using OxyPlot;
    using OxyPlot.Series;

    public partial class Form1 : Form
    {
        public Form1()
        {
            this.InitializeComponent();
            var myModel = new PlotModel { Title = "Example 1" };
            myModel.Series.Add(new FunctionSeries(Math.Cos, 0, 10, 0.1, "cos(x)"));
            this.plot1.Model = myModel;
        }
    }
}
```

The application should now look like this:



The source code can be found in the HelloWorld\WindowsFormsApplication1 folder in the documentation-examples repository.

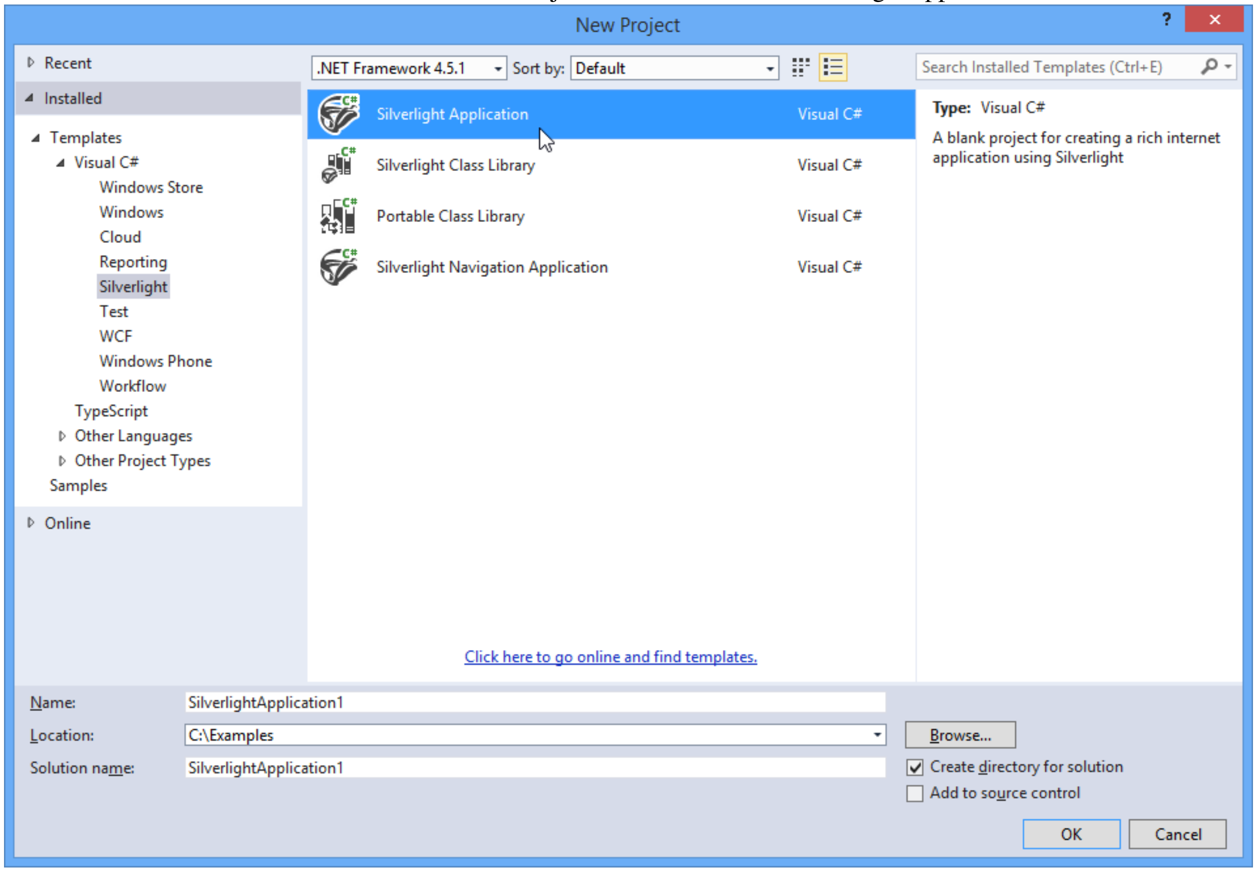
1.2.5 Silverlight

Note: This section is under construction. Please contribute!

This example shows how to create a Silverlight application where the content of the plot is defined in code.

Create project

Start Visual Studio and select "FILE -> New -> Project..." to create a new Silverlight application:



Type: Visual C#

A blank project for creating a rich internet application using Silverlight

Name: SilverlightApplication1

Location: C:\Examples

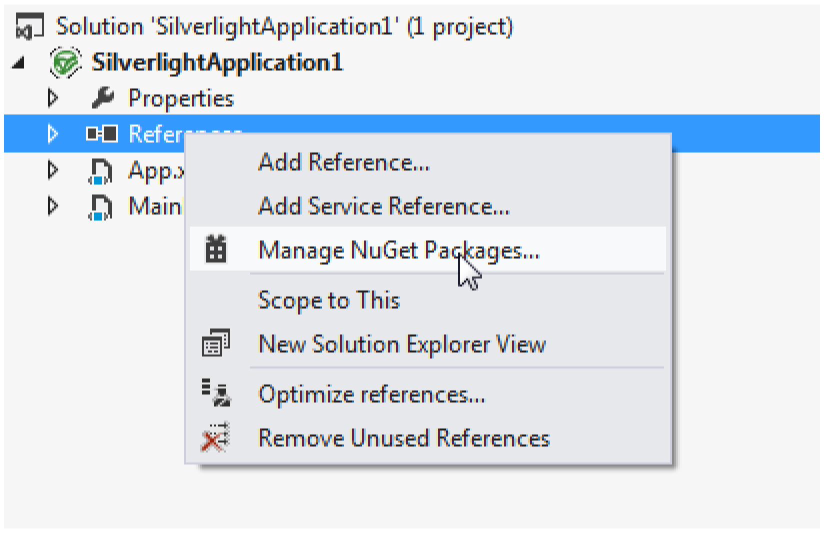
Solution name: SilverlightApplication1

[✓] Create directory for solution

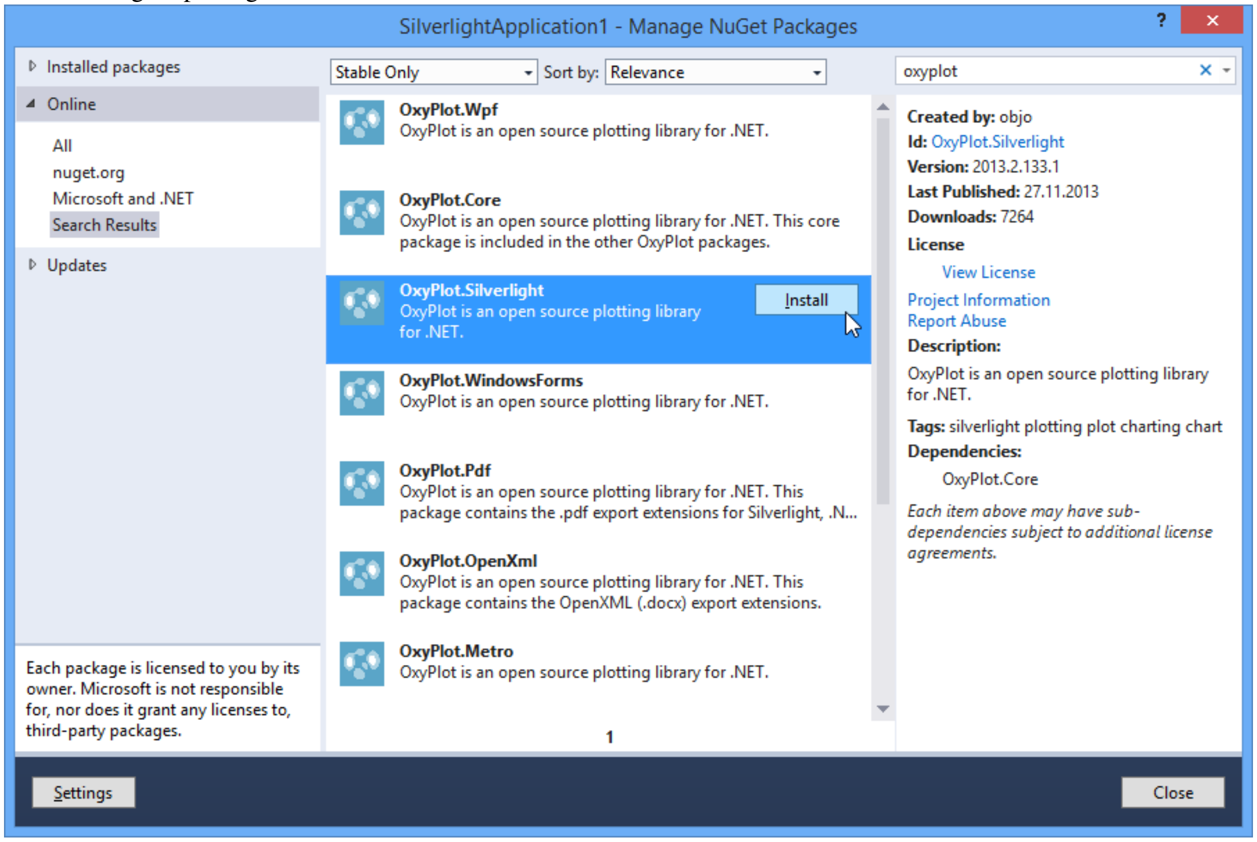
[ ] Add to source control

Add references

You need references to the OxyPlot and OxyPlot.Silverlight assemblies. The easiest way to do this is to right click on the "References" item in the Solution Explorer and select "Manage NuGet Packages..." (this requires that the "NuGet Package Manager" extension is installed):



In the "Manage NuGet packages" dialog, search for "OxyPlot" in the top-right search textbox. Select the "OxyPlot.Silverlight" package and click install:



OxyPlot.Wpf
OxyPlot is an open source plotting library for .NET.

OxyPlot.Core
OxyPlot is an open source plotting library for .NET. This core package is included in the other OxyPlot packages.

OxyPlot.Silverlight
OxyPlot is an open source plotting library for .NET.

OxyPlot.WindowsForms
OxyPlot is an open source plotting library for .NET.

OxyPlot.Pdf
OxyPlot is an open source plotting library for .NET. This package contains the .pdf export extensions for Silverlight, .N...

OxyPlot.OpenXml
OxyPlot is an open source plotting library for .NET. This package contains the OpenXML (.docx) export extensions.

OxyPlot.Metro
OxyPlot is an open source plotting library for .NET.

Created by: objo
Id: OxyPlot.Silverlight
Version: 2013.2.133.1
Last Published: 27.11.2013
Downloads: 7264
License
View License
Project Information
Report Abuse
Description:
OxyPlot is an open source plotting library for .NET.
Tags: silverlight plotting plot charting chart
Dependencies:
OxyPlot.Core

Each item above may have sub-dependencies subject to additional license agreements.

Each package is licensed to you by its owner. Microsoft is not responsible for, nor does it grant any licenses to, third-party packages.

You can also use the [Package Manager Console](http://docs.nuget.org/docs/start-here/using-the-package-manager-console) to install the package:

```
PM> Install-Package OxyPlot.Silverlight
```

Create a view model

Add a class that creates a `PlotModel` and a `FunctionSeries`.

```
namespace SilverlightApplication1
{
    using System;

    using OxyPlot;
    using OxyPlot.Series;

    public class MainViewModel
    {
        public MainViewModel()
        {
            this.MyModel = new PlotModel { Title = "Example 1" };
            this.MyModel.Series.Add(new FunctionSeries(Math.Cos, 0, 10, 0.1, "cos(x)"));
        }

        public PlotModel MyModel { get; set; }
    }
}
```

Create the view

Define the namespace in the `Window` element, set the `DataContext` and add a `PlotView` control:

```
<UserControl x:Class="SilverlightApplication1.MainPage"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
             xmlns:oxy="clr-namespace:OxyPlot.Silverlight;assembly=OxyPlot.Silverlight"
             xmlns:local="clr-namespace:SilverlightApplication1"
             mc:Ignorable="d"
    d:DesignHeight="300" d:DesignWidth="400">
    <UserControl.DataContext>
        <local:MainViewModel/>
    </UserControl.DataContext>

    <Grid x:Name="LayoutRoot" Background="White">
        <oxy:PlotView Model="{Binding MyModel}"/>
    </Grid>
</UserControl>
```

The application should now look like this:

The source code can be found in the HelloWorld\SilverlightApplication1 folder in the documentation-examples repository.

1.2.6 Windows Phone Silverlight

Note: This section is under construction. Please contribute!

This example shows how to create a Windows Phone Silverlight app.

Create project

Start Visual Studio and select "FILE -> New -> Project...". Select "Store Apps -> Windows Phone Apps -> Blank App (Windows Phone Silverlight)" to create a new projects for Windows Phone Silverlight.

Add references

You need references to the *OxyPlot* and *OxyPlot.WP8* assemblies. The easiest way to do this is to right click on the "References" item in the Solution Explorer and select "Manage NuGet Packages..." (this requires that the "NuGet Package Manager" extension is installed).

In the "Manage NuGet packages" dialog, search for "oxyplot windows phone" in the top-right search textbox. Select the "OxyPlot for Windows Phone Silverlight apps" package and click install.

You can also use the Package Manager Console to install the package:

```
PM> Install-Package OxyPlot.WP8
```

Create a view model

Add a class that creates a `PlotModel` and a `FunctionSeries`.

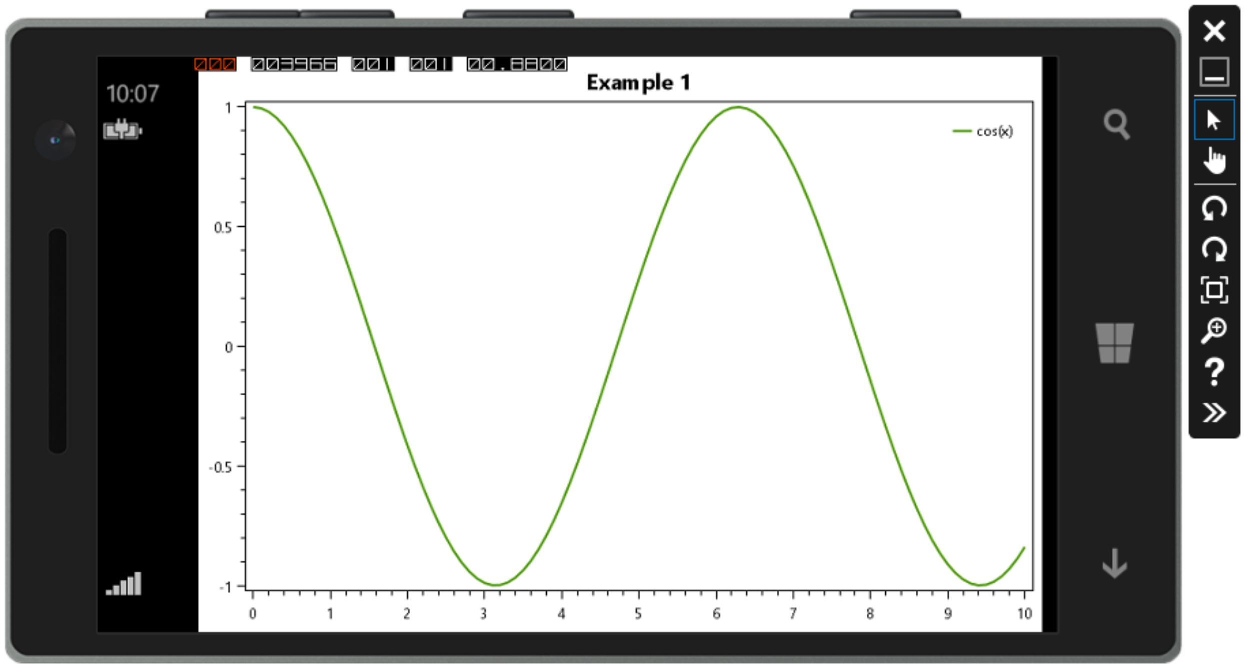
```
namespace WinPhoneApp1
{
    using System;

    using OxyPlot;
    using OxyPlot.Series;

    public class MainViewModel
    {
        public MainViewModel()
        {
            this.MyModel = new PlotModel { Title = "Example 1" };
            this.MyModel.Series.Add(new FunctionSeries(Math.Cos, 0, 10, 0.1, "cos(x)"));
        }

        public PlotModel MyModel { get; private set; }
    }
}
```

Create the view

Define the namespace in the `PhoneApplicationPage` element, set the `DataContext` and add a `PlotView` control:

```
<phone:PhoneApplicationPage
    x:Class="WinPhoneApp1.MainPage"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:phone="clr-namespace:Microsoft.Phone.Controls;assembly=Microsoft.Phone"
    xmlns:shell="clr-namespace:Microsoft.Phone.Shell;assembly=Microsoft.Phone"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    xmlns:oxy="clr-namespace:OxyPlot.WP8;assembly=OxyPlot.WP8"
    xmlns:local="clr-namespace:WinPhoneApp1"
    mc:Ignorable="d"
    FontFamily="{StaticResource PhoneFontFamilyNormal}"
    FontSize="{StaticResource PhoneFontSizeNormal}"
    Foreground="{StaticResource PhoneForegroundBrush}"
    SupportedOrientations="Landscape" Orientation="Landscape"
    shell:SystemTray.IsVisible="True">
    <phone:PhoneApplicationPage.DataContext>
        <local:MainViewModel/>
    </phone:PhoneApplicationPage.DataContext>

    <Grid x:Name="ContentPanel" Margin="12,0,12,0">
        <oxy:PlotView Model="{Binding MyModel}"/>
    </Grid>

</phone:PhoneApplicationPage>
```

The application should now look like this (on the Windows Phone emulator):



The source code can be found in the HelloWorld\WinPhoneApp1 folder in the documentation-examples repository.

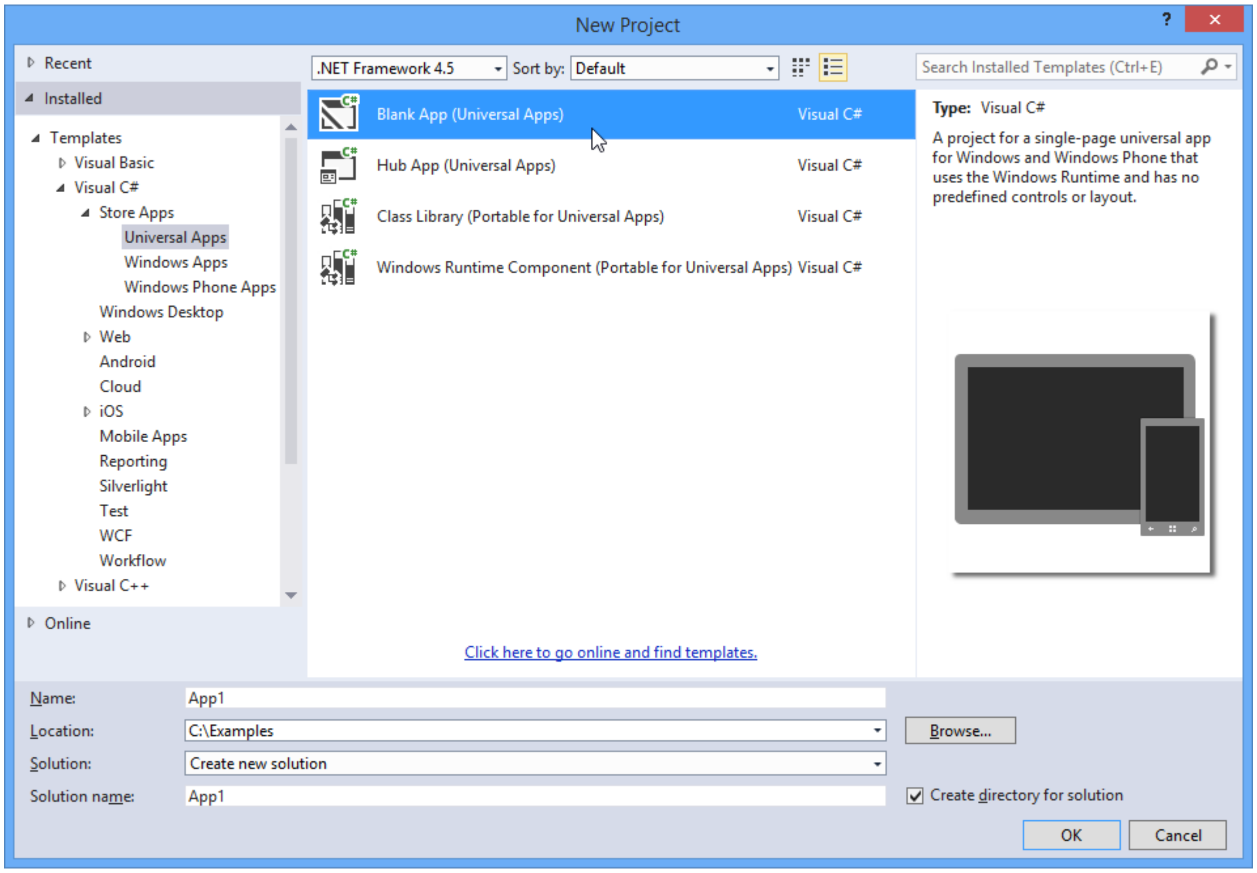
1.2.7 Windows Universal

Note: This section is under construction. Please contribute!

This example shows how to create apps for Windows and Windows Phone.

Create project

Start Visual Studio and select “FILE -> New -> Project...”. Select “Store Apps -> Universal Apps -> Blank App” to create new projects for Windows and Windows Phone:


New Project
Recent
Installed
Templates
Visual Basic
Visual C#
Store Apps
Universal Apps
Windows Apps
Windows Phone Apps
Windows Desktop
Web
Android
Cloud
iOS
Mobile Apps
Reporting
Silverlight
Test
WCF
Workflow
Visual C++
Online
Search Installed Templates (Ctrl+E)
Blank App (Universal Apps) Visual C#
Hub App (Universal Apps) Visual C#
Class Library (Portable for Universal Apps) Visual C#
Windows Runtime Component (Portable for Universal Apps) Visual C#
Click here to go online and find templates.
Browse...
OK
Cancel

.NET Framework 4.5 Sort by: Default

Type: Visual C#

A project for a single-page universal app for Windows and Windows Phone that uses the Windows Runtime and has no predefined controls or layout.

Name: App1

Location: C:\Examples

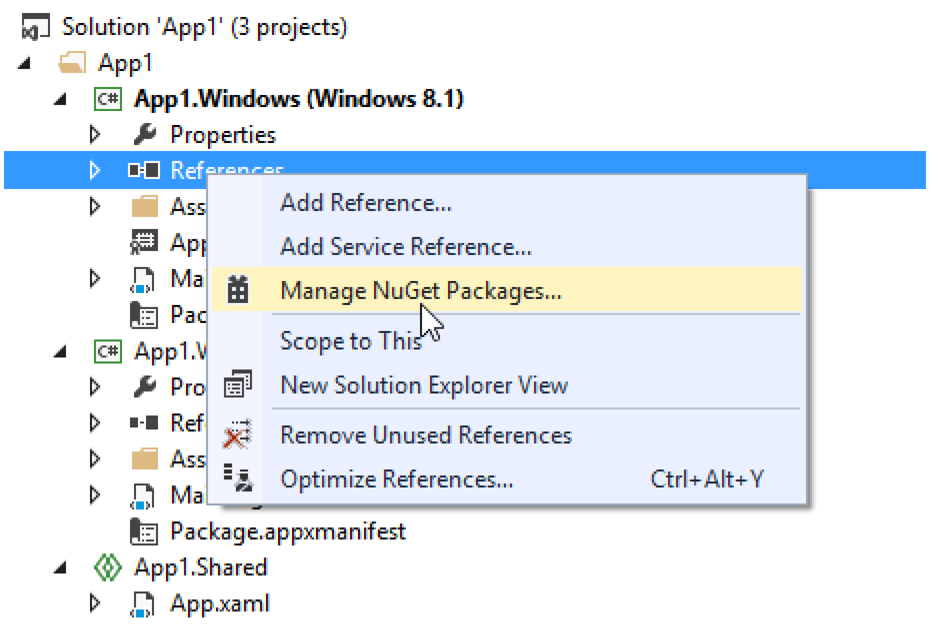
Solution: Create new solution

Solution name: App1

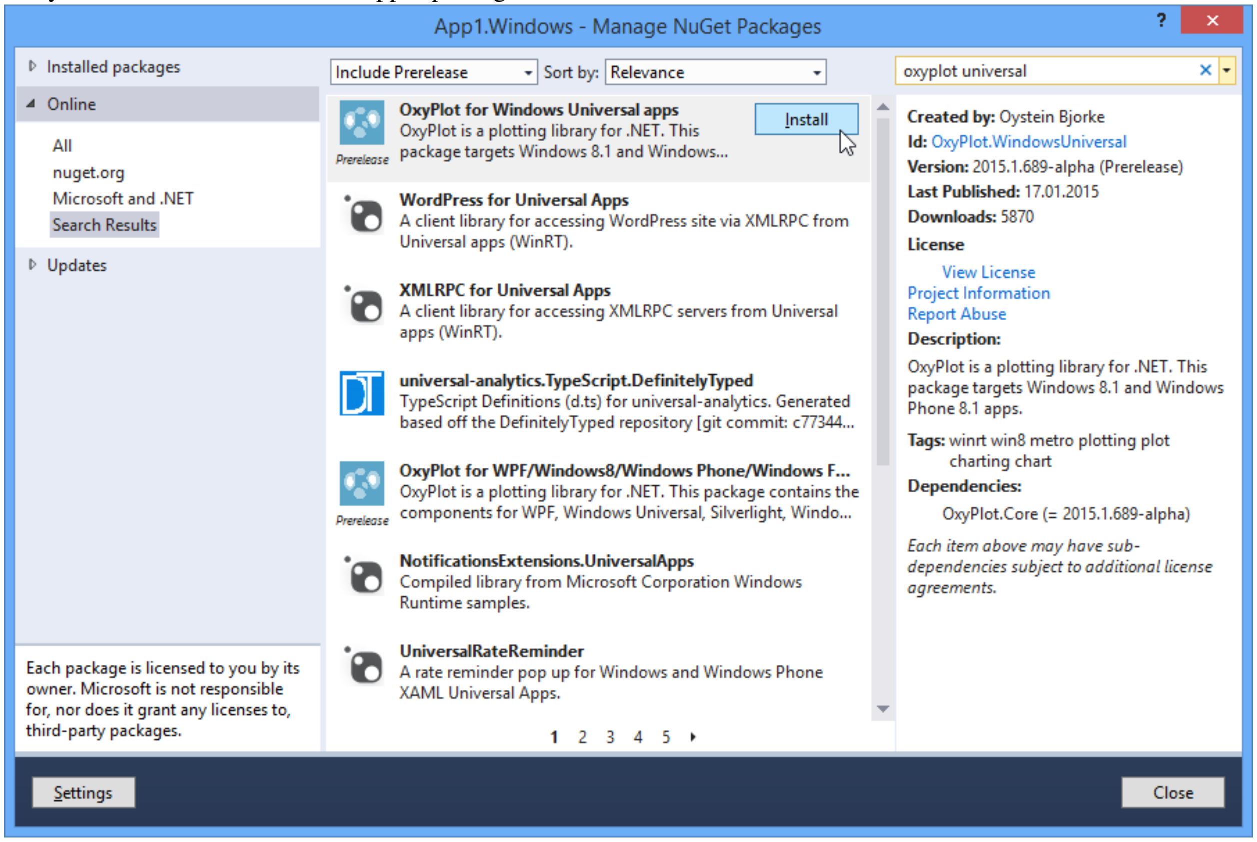
[✓] Create directory for solution

Add references

You need references to the *OxyPlot* and *OxyPlot.WindowsUniversal* assemblies. The easiest way to do this is to right click on the "References" item in the Solution Explorer and select "Manage NuGet Packages..." (this requires that the "NuGet Package Manager" extension is installed):



In the "Manage NuGet packages" dialog, search for "oxyplot universal" in the top-right search textbox. Select the "OxyPlot for Windows Universal apps" package and click install:



OxyPlot for Windows Universal apps
OxyPlot is a plotting library for .NET. This package targets Windows 8.1 and Windows...

WordPress for Universal Apps
A client library for accessing WordPress site via XMLRPC from Universal apps (WinRT).

XMLRPC for Universal Apps
A client library for accessing XMLRPC servers from Universal apps (WinRT).

universal-analytics.TypeScript.DefinitelyTyped
TypeScript Definitions (d.ts) for universal-analytics. Generated based off the DefinitelyTyped repository [git commit: c77344...

OxyPlot for WPF/Windows8/Windows Phone/Windows F...
OxyPlot is a plotting library for .NET. This package contains the components for WPF, Windows Universal, Silverlight, Windo...

NotificationsExtensions.UniversalApps
Compiled library from Microsoft Corporation Windows Runtime samples.

UniversalRateReminder
A rate reminder pop up for Windows and Windows Phone XAML Universal Apps.

Created by: Oystein Bjorke
Id: OxyPlot.WindowsUniversal
Version: 2015.1.689-alpha (Prerelease)
Last Published: 17.01.2015
Downloads: 5870
License
View License
Project Information
Report Abuse
Description:
OxyPlot is a plotting library for .NET. This package targets Windows 8.1 and Windows Phone 8.1 apps.
Tags: winrt win8 metro plotting plot charting chart
Dependencies:
OxyPlot.Core (= 2015.1.689-alpha)

Each item above may have sub-dependencies subject to additional license agreements.

Each package is licensed to you by its owner. Microsoft is not responsible for, nor does it grant any licenses to, third-party packages.

You can also use the Package Manager Console to install the package:

```
PM> Install-Package OxyPlot.WindowsUniversal
```

Create a view model

Add a class that creates a `PlotModel` and a `FunctionSeries`.

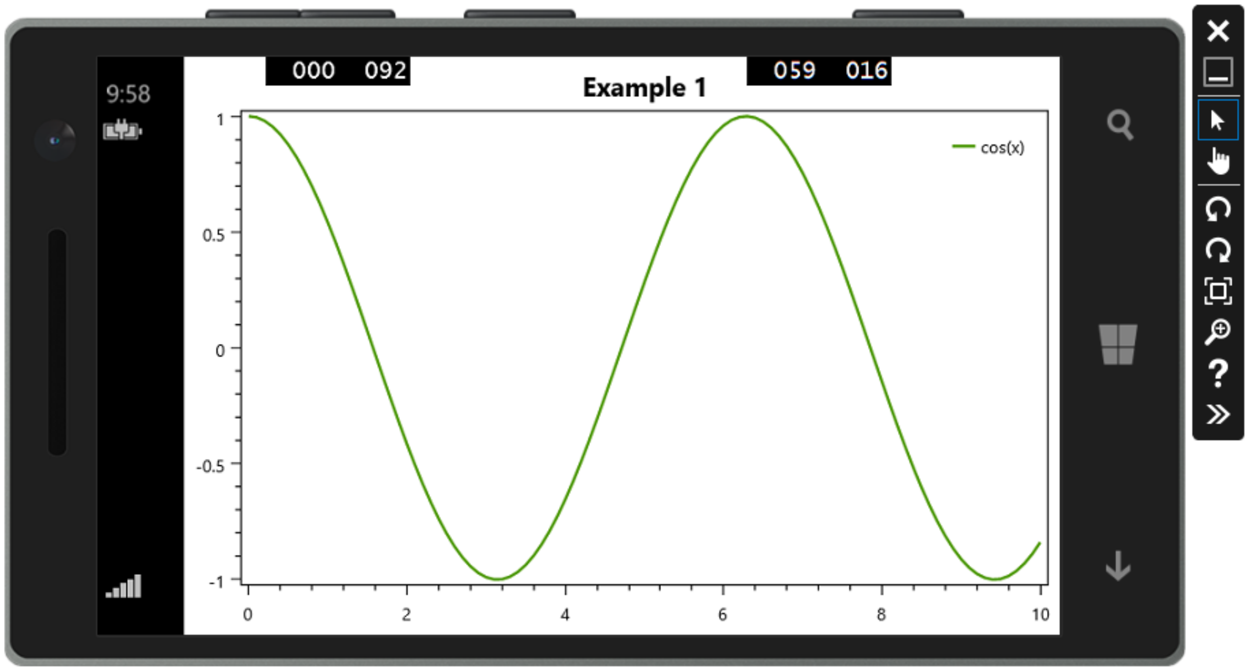
```
namespace UniversalApp1
{
    using System;

    using OxyPlot;
    using OxyPlot.Series;

    public class MainViewModel
    {
        public MainViewModel()
        {
            this.MyModel = new PlotModel { Title = "Example 1" };
            this.MyModel.Series.Add(new FunctionSeries(Math.Cos, 0, 10, 0.1, "cos(x)"));
        }

        public PlotModel MyModel { get; private set; }
    }
}
```

Create the view

Define the namespace in the `Page` element, set the `DataContext` and add a `PlotView` control:

```
<Page
    x:Class="UniversalApp1.MainPage"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:local="using:UniversalApp1"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    xmlns:oxy="using:OxyPlot.WindowsUniversal"
    mc:Ignorable="d">
    <Page.DataContext>
        <local:MainViewModel/>
    </Page.DataContext>

    <Grid Background="{ThemeResource ApplicationPageBackgroundThemeBrush}">
        <oxy:PlotView Model="{Binding MyModel}"/>
    </Grid>
</Page>
```

The application should now look like this (on the Windows Phone emulator):



The source code can be found in the HelloWorld\UniversalApp1 folder in the documentation-examples repository.

1.2.8 Xamarin.iOS

Note: This section is under construction. Please contribute!

OxyPlot supports both the classic (based on MonoTouch) and unified APIs for iOS.

Add the OxyPlot package

Add the `OxyPlot.Xamarin.iOS` NuGet package to the project. References to the portable `OxyPlot.dll` and the Android-specific `OxyPlot.Xamarin.iOS` libraries will be added.

Add a PlotView control

Bind to a PlotModel

References

The source code can be found in the HelloWorld\iOSApp1 folder in the documentation-examples repository.

1.2.9 Xamarin.Android

Note: This section is under construction. Please contribute!

Add the OxyPlot package

Add the `OxyPlot.Xamarin.Android` NuGet package to the project. References to the portable `OxyPlot.dll` and the Android-specific `OxyPlot.Xamarin.Android` libraries will be added.

Add a PlotView control

Bind to a PlotModel

References

The source code can be found in the HelloWorld\AndroidApp1 folder in the documentation-examples repository.

1.2.10 Xamarin.Forms

Note: This section is under construction. Please contribute!

This example shows how to create a Xamarin.Forms app where both the user interface and the plot is defined in portable code.

Create the project

Start Visual Studio and select the blank Xamarin.Forms app template.

Update and add references

Update the `Xamarin.Forms` NuGet packages to the latest version.

Add the `OxyPlot.Xamarin.Forms` NuGet package in both the portable and platform specific projects.

Initialize renderers

You need to initialize the OxyPlot renderers by adding the following call just after `Xamarin.Forms.Forms.Init()`:

- iOS (Unified API): `OxyPlot.Xamarin.Forms.Platform.iOS.PlotViewRenderer.Init();`
- Android: `OxyPlot.Xamarin.Forms.Platform.Android.PlotViewRenderer.Init();`
- Windows Phone: `OxyPlot.Xamarin.Forms.Platform.WP8.PlotViewRenderer.Init();`

Add the PlotView to a page (in code)

In the portable/shared app project, add the plot view to a page:

```
public App()
{
    this.MainPage = new ContentPage
    {
        Content = new PlotView
        {
            Model = new PlotModel { Title = "Hello, Forms!" },
            VerticalOptions = LayoutOptions.Fill,
            HorizontalOptions = LayoutOptions.Fill,
        },
    };
}
```

Add the PlotView to a page (XAML)

Add a "Forms Xaml Page" to your project. In the page element, add a namespace declaration:

```
xmlns:oxy="clr-namespace:OxyPlot.Xamarin.Forms;assembly=OxyPlot.Xamarin.Forms"
```

Then add the plot view:

```
<oxy:PlotView Model="{Binding Model}" VerticalOptions="Center" HorizontalOptions="Center" />
```

This view will now be bound to a `PlotModel` in the binding context of the page.

References

The source code can be found in the HelloWorldXamarinFormsApp1 folder in the documentation-examples repository.

1.2.11 Xamarin.Mac

Note: This section is under construction. Please contribute!

OxyPlot supports the unified API for Xamarin.Mac.

Add the OxyPlot package

Add the `OxyPlot.Xamarin.Mac` NuGet package to the project. References to the portable `OxyPlot.dll` and the Mac-specific `OxyPlot.Xamarin.Mac` libraries will be added.

Add a PlotView control

Bind to a PlotModel

References

The source code can be found in the HelloWorldMacApp1 folder in the documentation-examples repository.

1.2.12 Xwt

Note: This section is under construction. Please contribute!

- Adding the "OxyPlot Xwt" NuGet package

- Add a *PlotView* control
- Bind to a *PlotModel*

1.2.13 GTK#

Note: This section is under construction. Please contribute!

- Adding the “OxyPlot GTK#” NuGet package
- Add a *PlotView* control
- Bind to a *PlotModel*

1.3 Model

1.3.1 PlotModel

Note: This section is under construction. Please contribute!

- What is the PlotModel?
- is it MVC, MVP, MVVM, MV*?
- A `PlotModel` cannot be used in more than one view

Properties

- axes collection
- series collection
- annotations collection
- fonts
- default colors
- margins
- legends

About units

OxyPlot uses device independent units to specify line thicknesses etc.

In OxyPlot, a device indepent unit is related to a 96 dpi display. On such a display, 1 unit should be rendered as 1 pixel.

1.3.2 Legend

Note: This section is under construction. Please contribute!

1.3.3 Axes

Note: This section is under construction. Please contribute!

// TODO - add more info, clean text, add examples

OxyPlot contains the following axis types:

LinearAxis Represents a numerical axis with a linear scale. LogarithmicAxis Represents a numerical axis with a logarithmic scale. DateTimeAxis Represents a date/time axis based on *DateTime* values. TimeSpanAxis Represents a time axis based on *TimeSpan* values. CategoryAxis Represents an axis that displays categories (typically used for bar/column series). LinearColorAxis Represents an axis that displays a linear color scale. RangeColorAxis Represents an axis that displays a colors for specified ranges. MagnitudeAxis Represents the radial axis in polar plots AngleAxis Represents the angular axis in polar plots

Position

The most important propert of the axis is the *Position*. This property determines where the axis is drawn. A standard XY plot requires a horizontal axis (bottom or top position) and a vertical axis (left or right position).

Title

The title is shown next to the axis.

About position and rotation of titles.

About `Unit`

Minimum/Maximum

These properties defines the minimum and maximum values on the axis. If any of them are not specified, they will be calculated from the data. In that case, a "padding" value will be included to make sure there is some whitespace outisde the extreme values.

Major/minor intervals

The major intervals define the steps between the numeric labels on the axis. The minor intervals define the sub-division between the labels.

Major and minor ticks may be drawn at each interval. The style can be defined (inside, outside, crossing or none)

Grid lines can also be drawn at each interval. These will be drawn across the whole plot area. The style can be defined by color, thickness and line style.

StartPosition/EndPosition

The start and end position properties are used to define the relative position of the axis. The default values [0,1] will fill the available plot area.

To reverse the direction of an axis, set *StartPosition = 1* and *EndPosition = 0*

Axis keys

How to use *AxisKey*...

Adding axes in XAML

```
<oxy:Plot Title="Linear axes">
    <oxy:Plot.Axes>
        <oxy:LinearAxis Position="Bottom" Minimum="-20" Maximum="80" />
        <oxy:LinearAxis Position="Left" Minimum="-10" Maximum="10" />
     </oxy:Plot.Axes>
</oxy:Plot>
```

Adding axes to a PlotModel

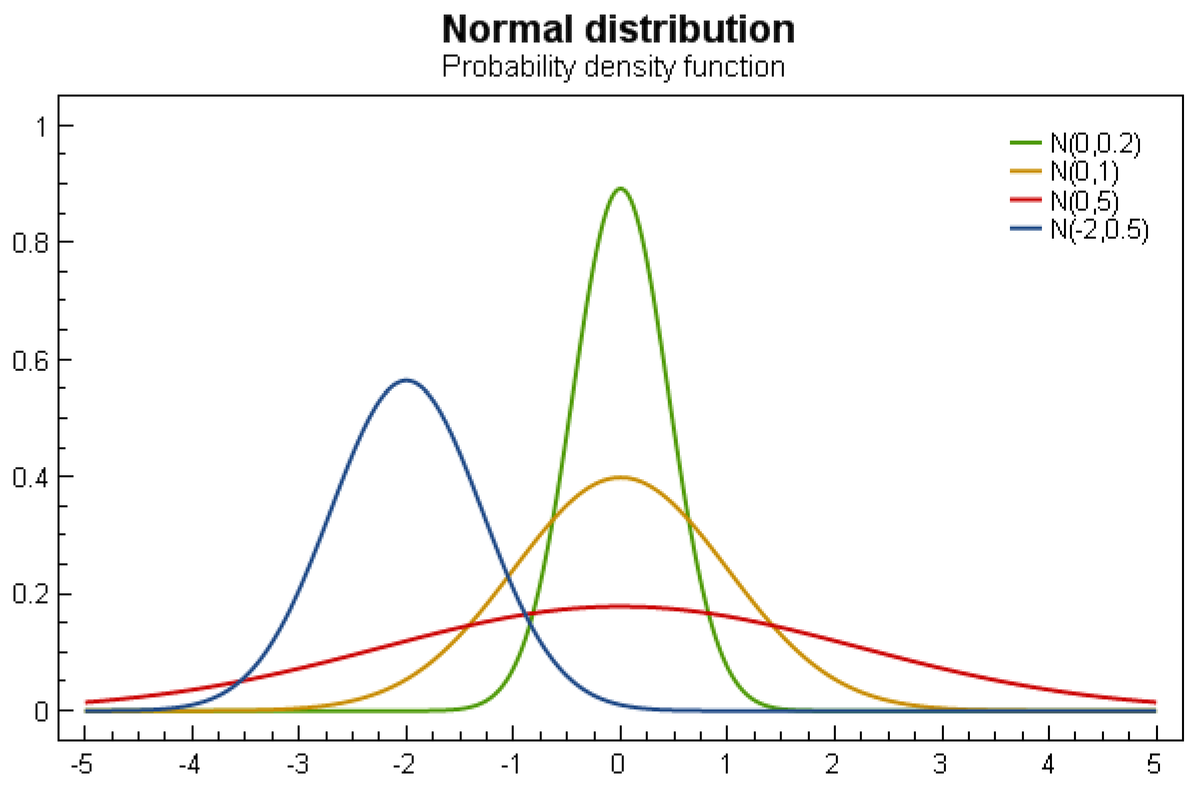
```
var model = new PlotModel();
model.Axes.Add(new LinearAxis(AxisPosition.Bottom, -20, 80));
model.Axes.Add(new LinearAxis(AxisPosition.Left, -10, 10));
```

If no axes are defined, linear axes will be added to the bottom and left.

Axis types

LinearAxis

Note: This section is under construction. Please contribute!

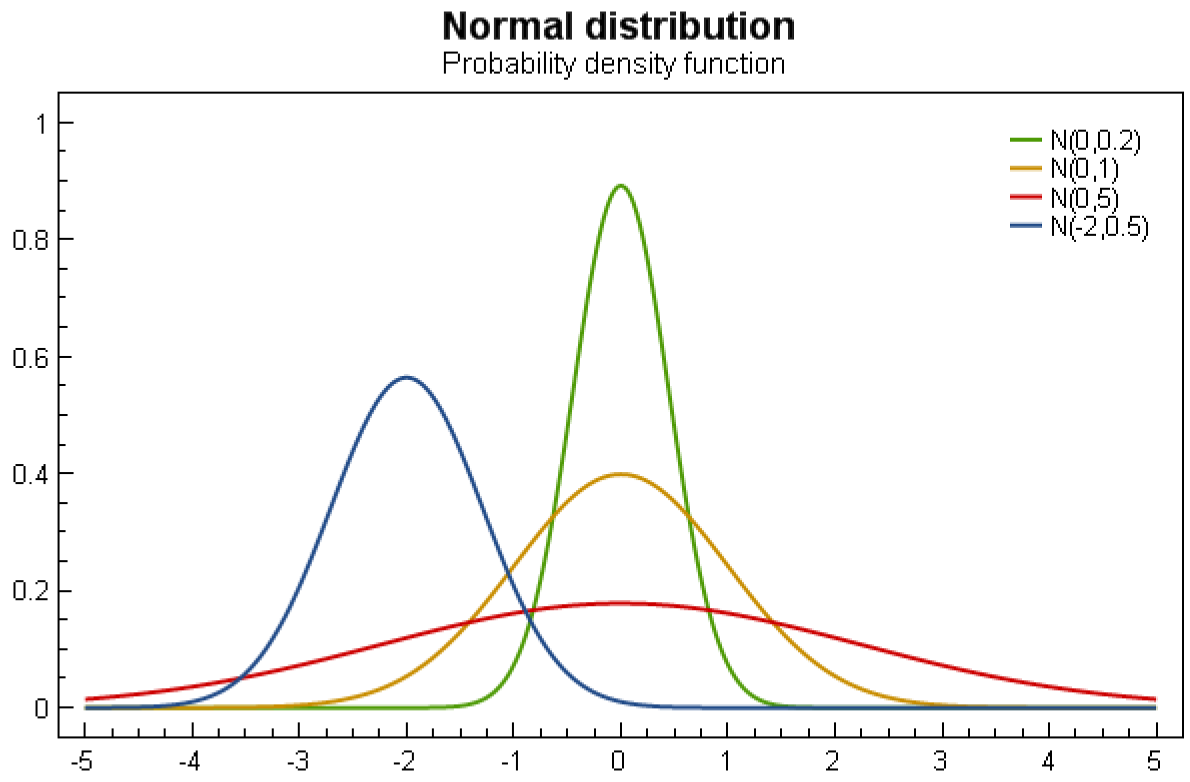


Example

```
var model = new PlotModel { Title = "LinearAxis" };
model.Axes.Add(new LinearAxis { Position = AxisPosition.Bottom, Minimum = -20, Maximum = 80});
model.Axes.Add(new LinearAxis { Position = AxisPosition.Left, Minimum = -10, Maximum = 10});
```

LogarithmicAxis

Note: This section is under construction. Please contribute!



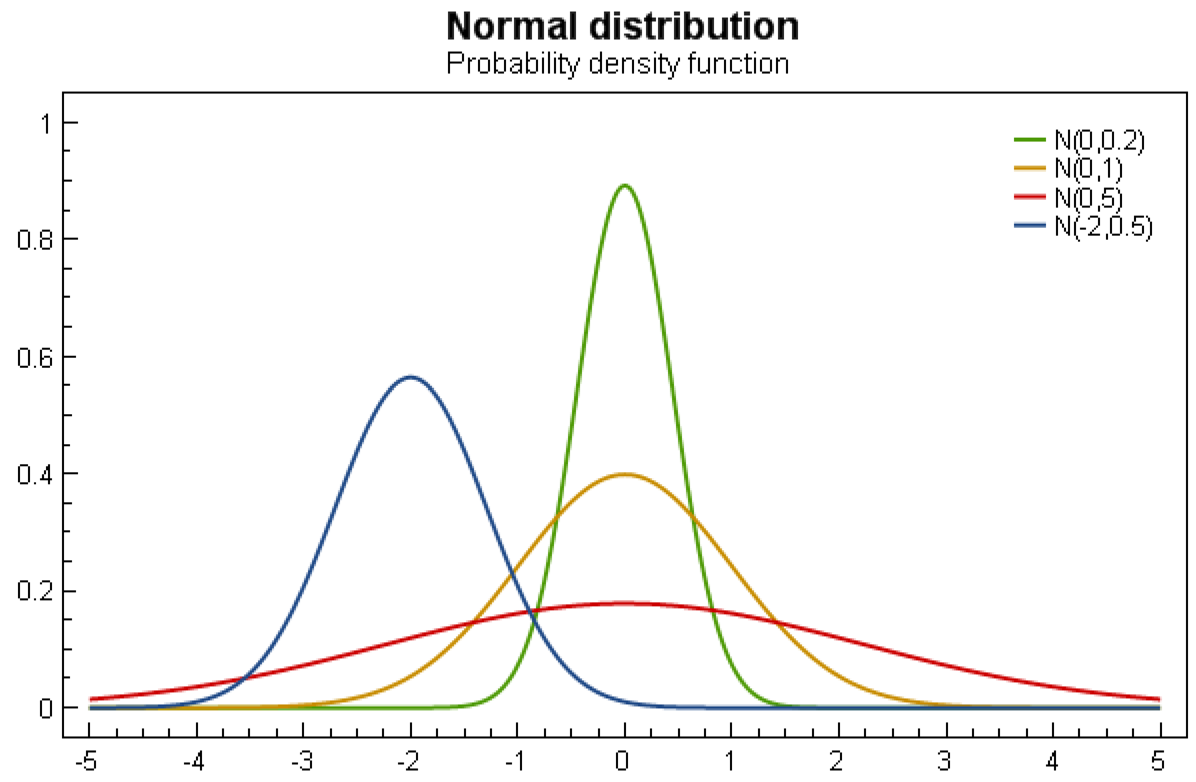
Example

```
var model = new PlotModel { Title = "LogarithmicAxis" };
model.Axes.Add(new LinearAxis { Position = AxisPosition.Bottom, Minimum = -20, Maximum = 80});
model.Axes.Add(new LinearAxis { Position = AxisPosition.Left, Minimum = -10, Maximum = 10});
```

DateTimeAxis

Note: This section is under construction. Please contribute!

This will show date/time values on the axis.



If you are adding *DataPoint's to a series, the 'DateTime* values should be converted to numeric values by the *DateTimeAxis.ToDouble* method.

```
mySeries.Points.Add(new DataPoint(DateTimeAxis.ToDouble(myDateTime),myValue))
```

If you are binding *DataFieldX* or *DataFieldY* to a *DateTime*, OxyPlot will handle the conversion.

You can control the axis intervals by *IntervalType*, *MinorIntervalType* and the formatting of the axis labels by the *StringFormat* property.

The following codes are accepted by the `StringFormat` property

Code	Description	Example
yyyy	year	"2011"
yy	year	"11"
MM	month	"01"
MMM	month	"Jan"
MMMM	month	"January"
w	week number	"4"
ww	week number	"04"
dd	day	"26"
hh	hour	"04"
HH	hour	"16"
mm	minute	"37"
ss	seconds	"23"
yyyy-MM-dd		"2011-01-26"
MM/dd/yyyy		"01/26/2011"

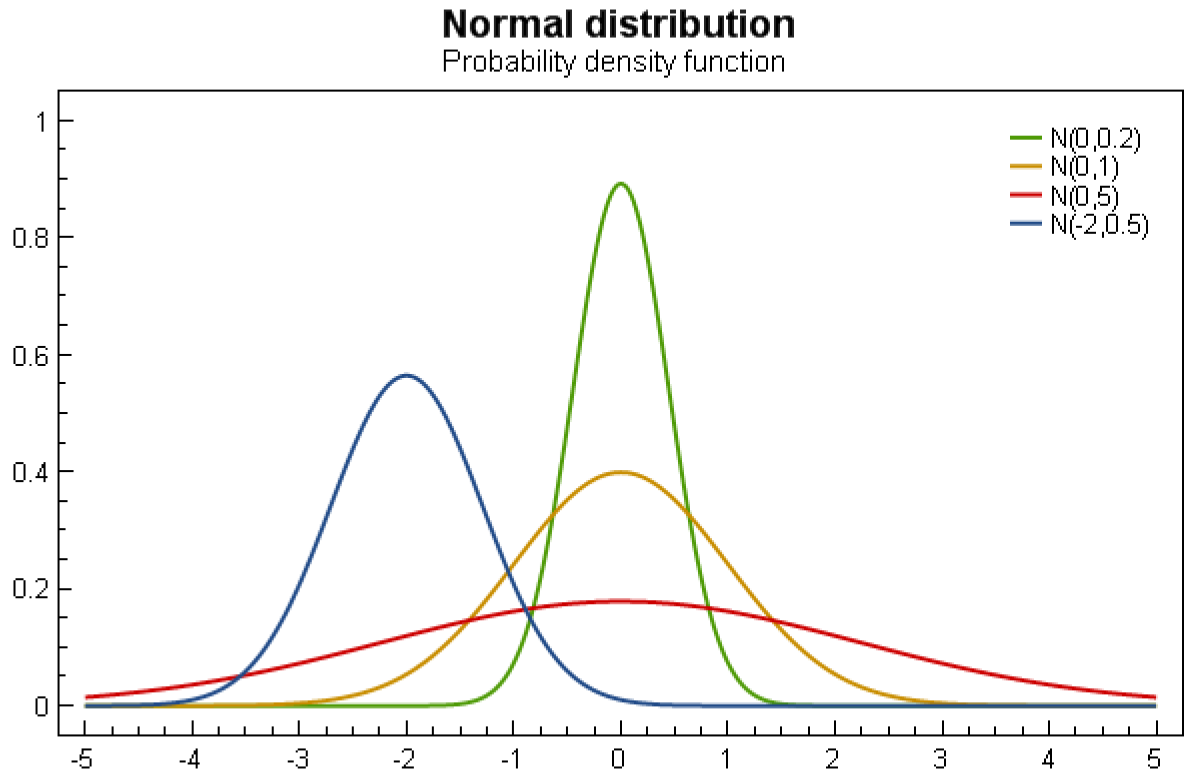
Example

```
var model = new PlotModel { Title = "DateTimeAxis" };
model.Axes.Add(new LinearAxis { Position = AxisPosition.Bottom, Minimum = -20, Maximum = 80});
model.Axes.Add(new LinearAxis { Position = AxisPosition.Left, Minimum = -10, Maximum = 10});
```

TimeSpanAxis

Note: This section is under construction. Please contribute!

This axis will show `TimeSpan` values on the axis.



If you are adding *DataPoint's to a series, the 'TimeSpan* values should be converted to numeric values by the *TimeSpanAxis.ToDouble* method.

```
mySeries.Points.Add(new DataPoint(TimeSpanAxis.ToDouble(myTimeSpan), myValue))
```

If you are using *DataFieldX* or *DataFieldY* to bind to a *TimeSpan*, OxyPlot will handle the conversion.

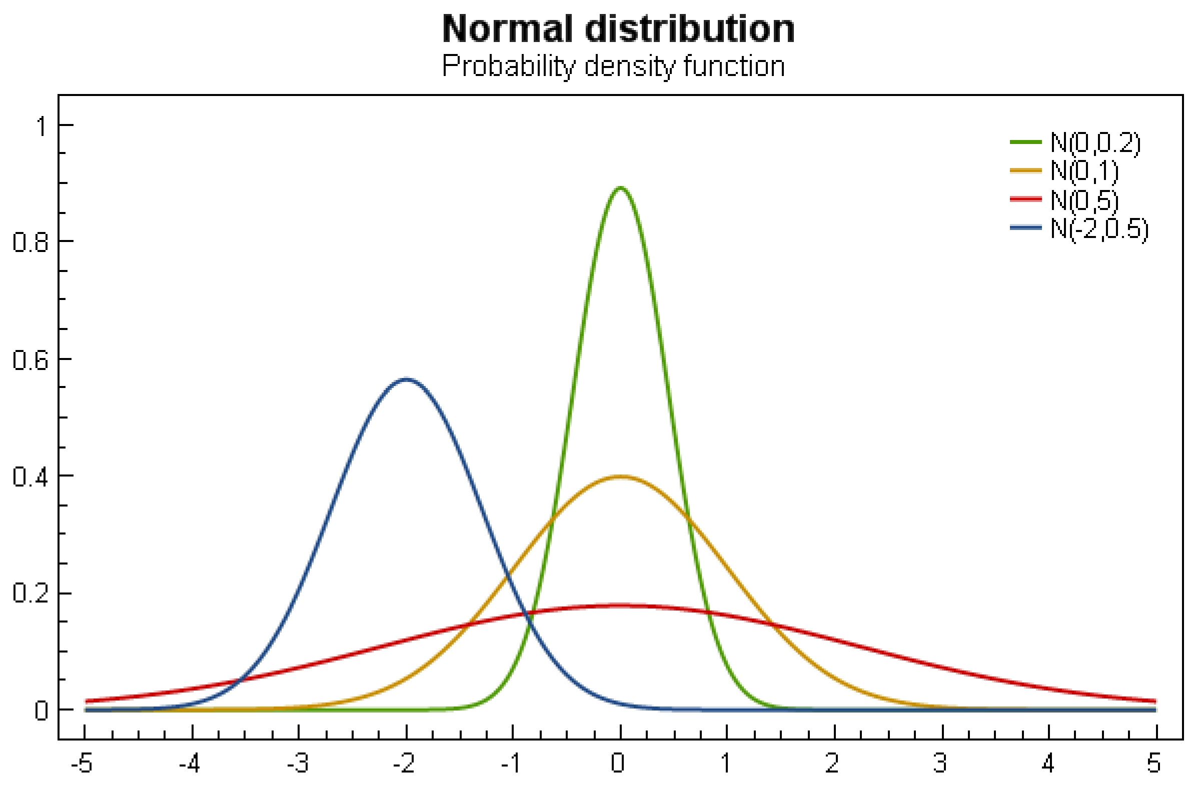
The formatting of the axis labels can be controlled by the *StringFormat* property.

Example

```
var model = new PlotModel { Title = "TimeSpanAxis" };
model.Axes.Add(new LinearAxis { Position = AxisPosition.Bottom, Minimum = -20, Maximum = 80});
model.Axes.Add(new LinearAxis { Position = AxisPosition.Left, Minimum = -10, Maximum = 10});
```

CategoryAxis

Note: This section is under construction. Please contribute!

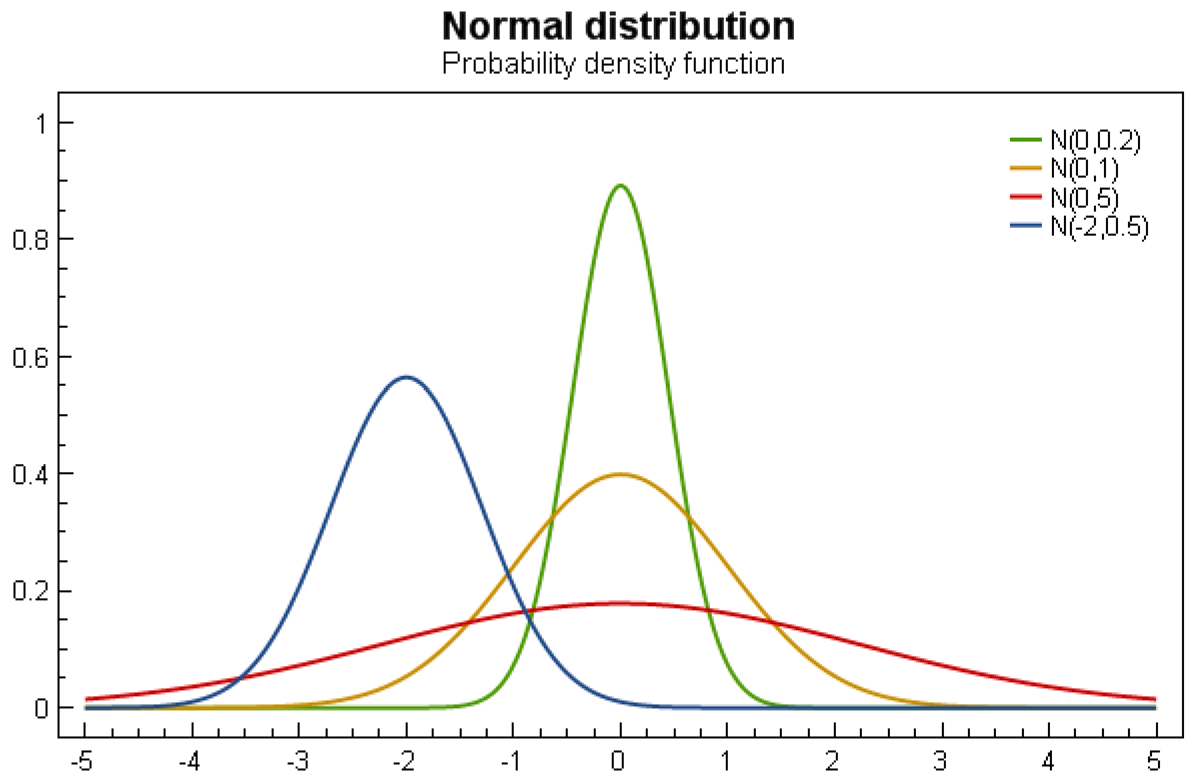


Example

```
var model = new PlotModel { Title = "CategoryAxis" };
model.Axes.Add(new LinearAxis { Position = AxisPosition.Bottom, Minimum = -20, Maximum = 80});
model.Axes.Add(new LinearAxis { Position = AxisPosition.Left, Minimum = -10, Maximum = 10});
```

LinearColorAxis

Note: This section is under construction. Please contribute!

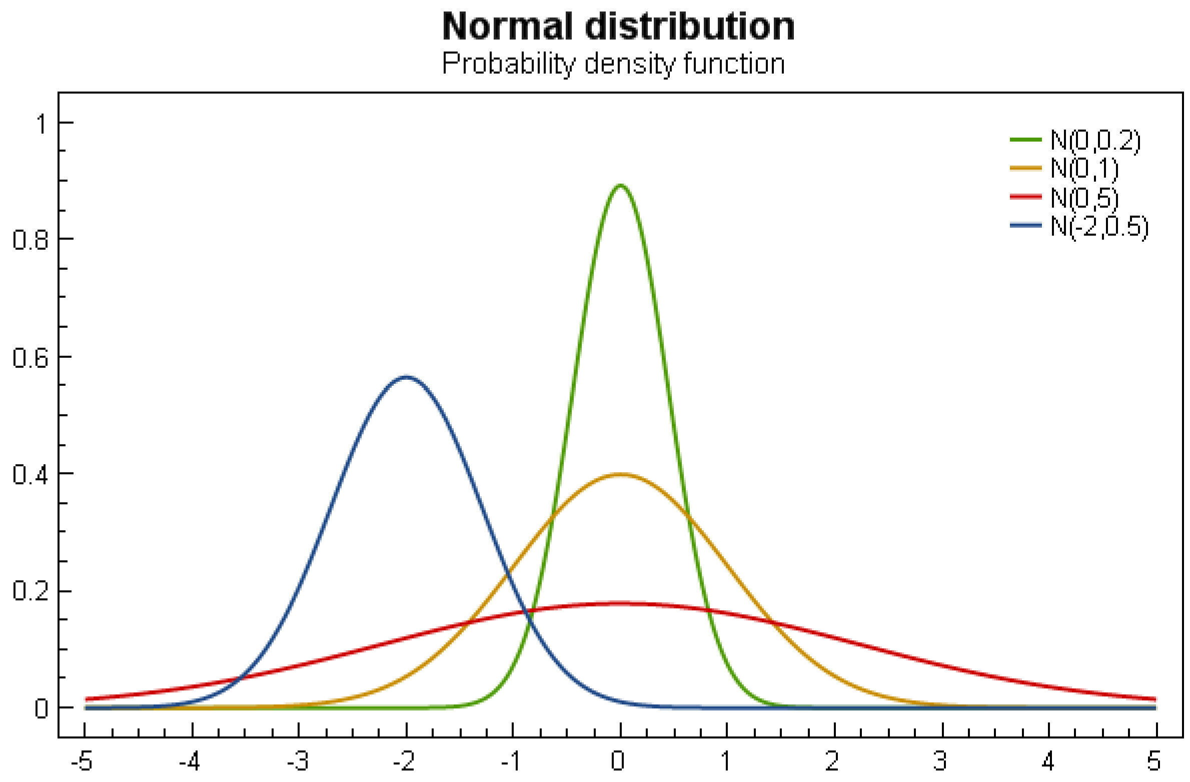


Example

```
var model = new PlotModel { Title = "LinearColorAxis" };
model.Axes.Add(new LinearAxis { Position = AxisPosition.Bottom, Minimum = -20, Maximum = 80});
model.Axes.Add(new LinearAxis { Position = AxisPosition.Left, Minimum = -10, Maximum = 10});
```

RangeColorAxis

Note: This section is under construction. Please contribute!

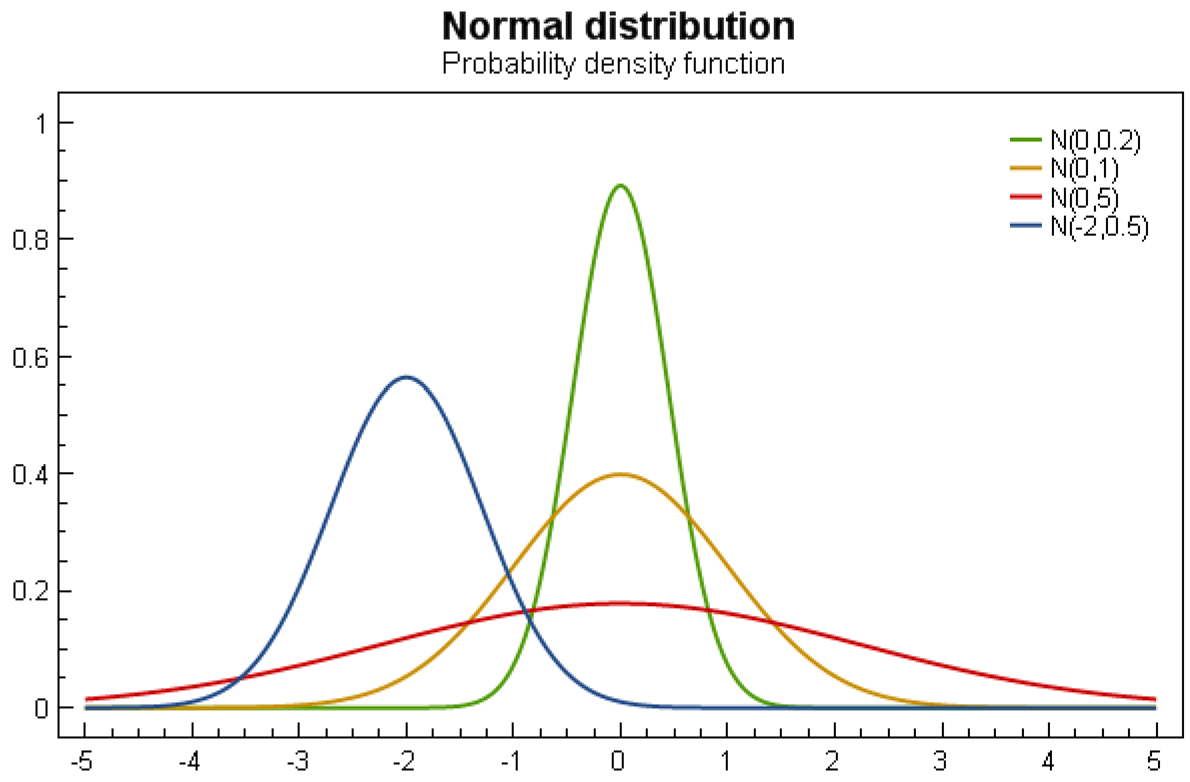


Example

```
var model = new PlotModel { Title = "RangeColorAxis" };
model.Axes.Add(new LinearAxis { Position = AxisPosition.Bottom, Minimum = -20, Maximum = 80});
model.Axes.Add(new LinearAxis { Position = AxisPosition.Left, Minimum = -10, Maximum = 10});
```

MagnitudeAxis

Note: This section is under construction. Please contribute!

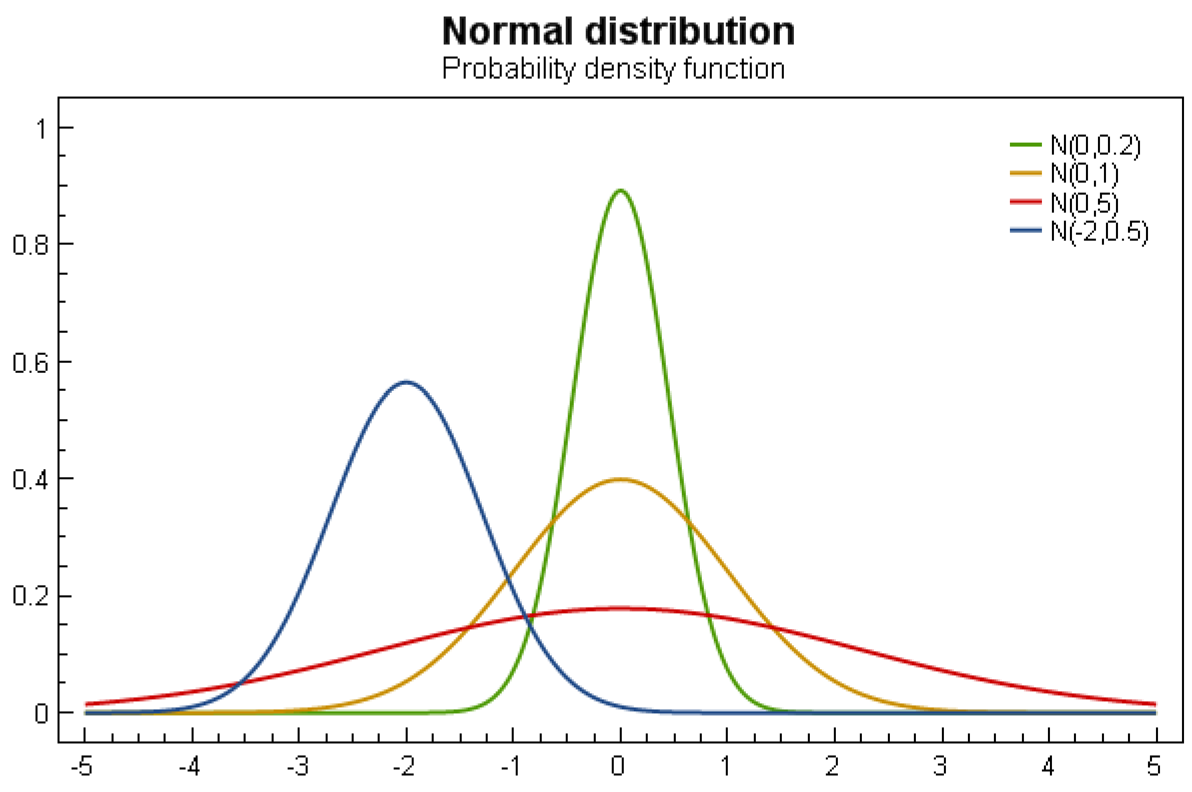


Example

```
var model = new PlotModel { Title = "MagnitudeAxis" };
model.Axes.Add(new LinearAxis { Position = AxisPosition.Bottom, Minimum = -20, Maximum = 80});
model.Axes.Add(new LinearAxis { Position = AxisPosition.Left, Minimum = -10, Maximum = 10});
```

AngleAxis

Note: This section is under construction. Please contribute!



Example

```
var model = new PlotModel { Title = "AngleAxis" };
model.Axes.Add(new LinearAxis { Position = AxisPosition.Bottom, Minimum = -20, Maximum = 80});
model.Axes.Add(new LinearAxis { Position = AxisPosition.Left, Minimum = -10, Maximum = 10});
```

1.3.4 Series

Note: This section is under construction. Please contribute!

Position

The position of the axes is defined by the `Position` property.

Visibility

The visibility of the series can be controlled by the `IsVisible` property. The default value is `true`. If set to `false`, the series will not be rendered.

Title

The `Title` property defines the title to show in the plot Legend. The default value is `null` (not shown in legend).

Background

If the `Background` property is set to a color, the area defined by the X and Y axes will be filled with the specified color. The default value is `Undefined` (not showing a background).

Tracker

The *TrackerFormatString* property is used to format the string shown in the Tracker. The arguments that can be used for the format string is documented for each series.

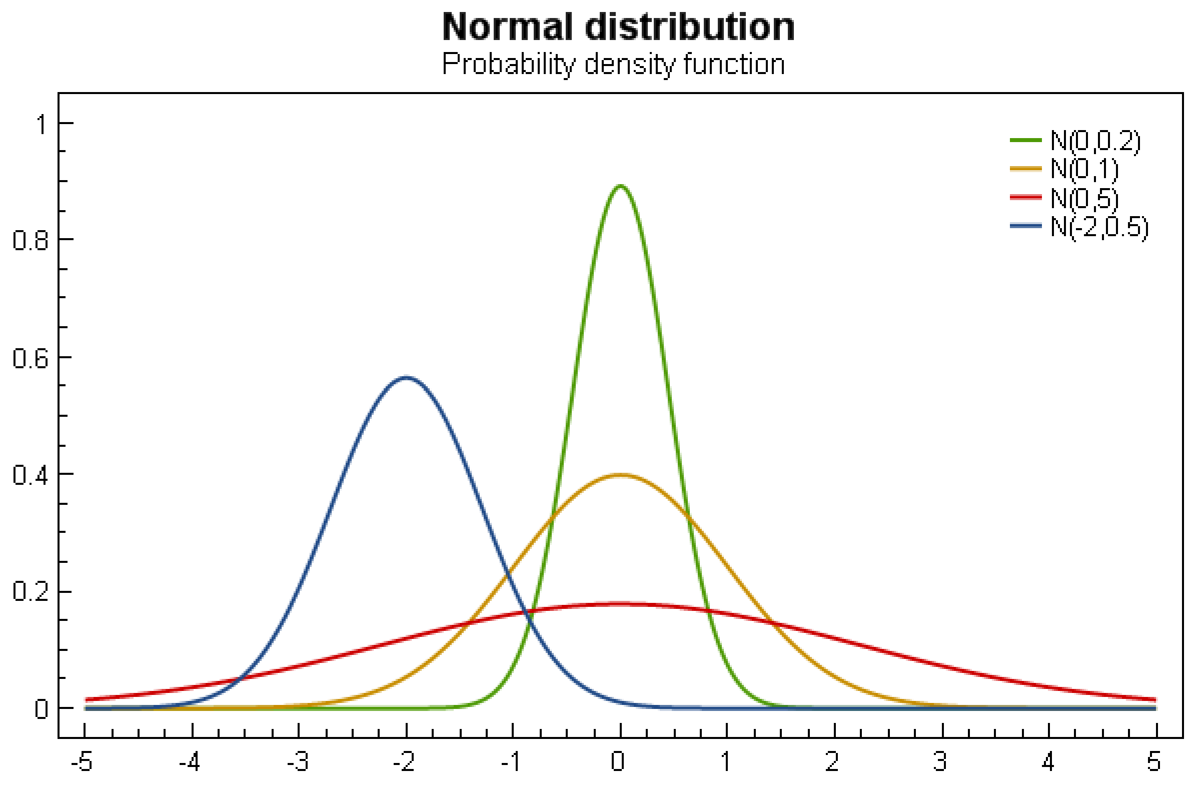
If an item was hit, it is also possible to use the extended format string syntax, e.g. `{PropertyX:0.##}`, where the value of `PropertyX` will be found by reflection of the item.

See MSDN for more information about format strings.

AreaSeries

Note: This section is under construction. Please contribute!

A `AreaSeries` shows an area between two sets of points, or between a set of point and a baseline.



Tracker The `TrackerFormatString` property is used to format the string shown in the tracker. The format string may use the following arguments:

- `{0}` the title of the series

- `{1}` the title of the x-axis
- `{2}` the x-value
- `{3}` the title of the y-axis
- `{4}` the y-value
- `{PropertyX}` the value of `PropertyX` in the item (extended format string syntax)

To show the x and y values with one digit, use the format string `"{2:0.0},{4:0.0}"`.

If an item was hit, it is also possible to use the extended format string syntax, e.g. `{PropertyX:0.##}`, where the value of `PropertyX` will be found by reflection of the item.

The default format string for `AreaSeries` is `"{0}\n{1}:  {2}\n{3}:  {4}"`

See MSDN for more information about format strings.

The `TrackerKey` property may be used to specify a custom tracker. This makes it possible to use different trackers for each series.

Example

```
var model = new PlotModel { Title = "AreaSeries" };
var areaSeries = new AreaSeries());
...
model.Series.Add(areaSeries);
```

BarSeries

Note: This section is under construction. Please contribute!

A `BarSeries` shows the data as horizontal bars.

Axes A vertical `CategoryAxis` and a horizontal `LinearAxis` is required.

Tracker The format string may use the following arguments:

- `{0}` the title of the series
- `{1}` the category
- `{2}` the bar value
- `{PropertyX}` the value of `PropertyX` in the item (extended format string syntax)

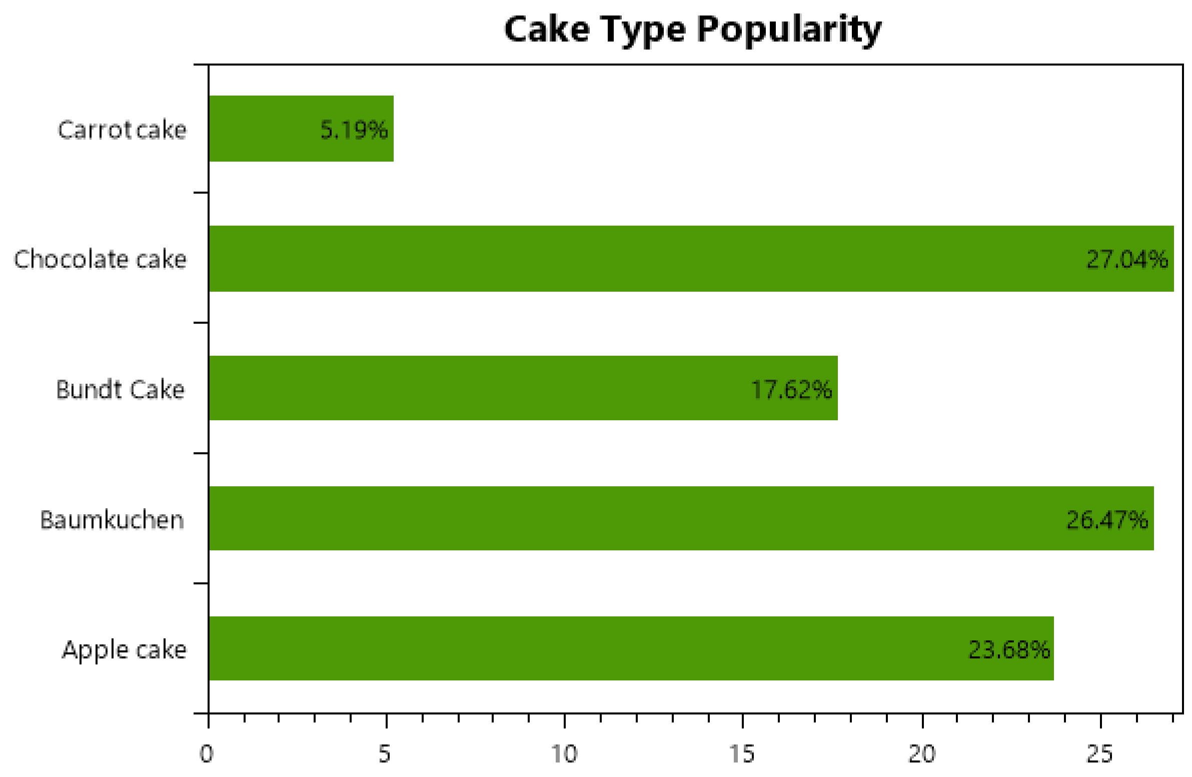
To show only the value with one digit, use the format string `"{2:0.0}"`.

If an item was hit, it is also possible to use the extended format string syntax, e.g. `{PropertyX:0.##}`, where the value of `PropertyX` will be found by reflection of the item.

The default format string for `BarSeries` is `"{0}\n{1}:  {2}"`

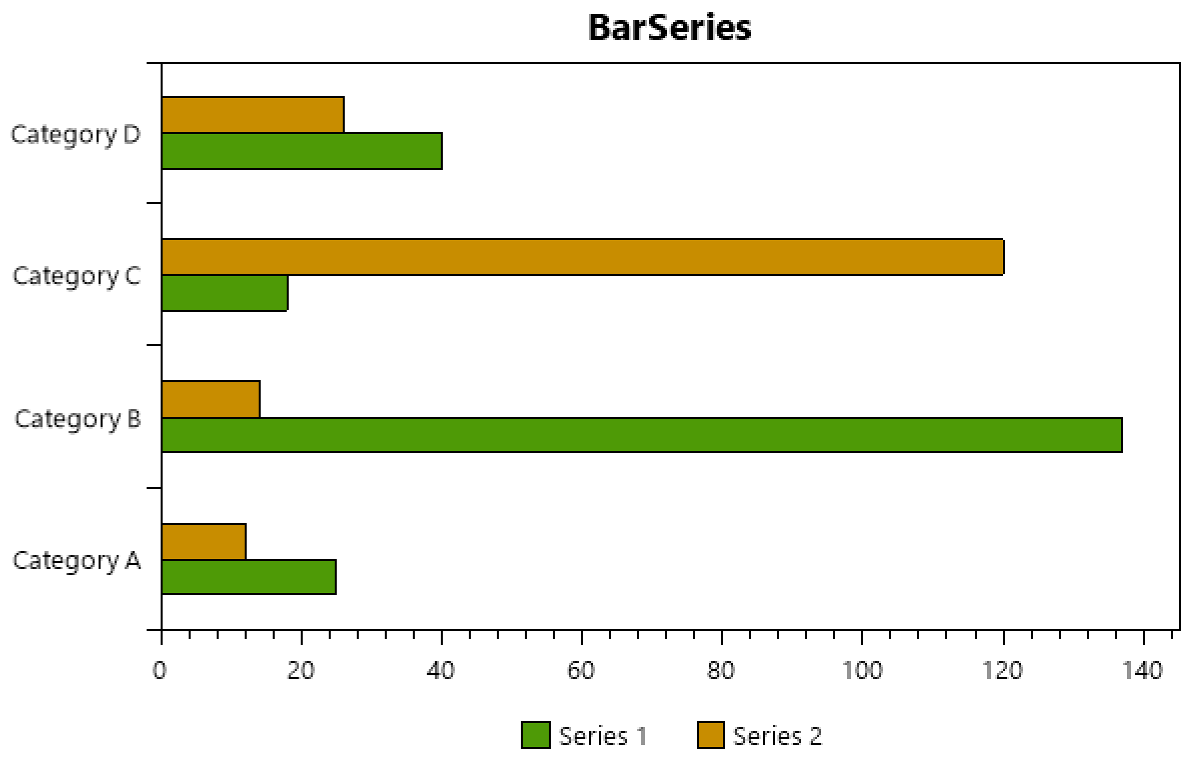
See MSDN for more information about format strings.

Example(simple) Here a simple example making use of very basic functionality to visualize the popularity of 5 different types of cake.



```
var model = new PlotModel{ Title = "Cake Type Popularity" };

//generate a random percentage distribution between the 5
//cake-types (see axis below)
var rand = new Random();
double[] cakePopularity = new double[5];
for(int i = 0; i < 5; ++i) {
        cakePopularity[i] = rand.NextDouble();
}
var sum = cakePopularity.Sum();

var barSeries = new BarSeries
{
        ItemsSource = new List<BarItem>(new[]
        {
                new BarItem{ Value = (cakePopularity[0] / sum * 100) },
                new BarItem{ Value = (cakePopularity[1] / sum * 100) },
                new BarItem{ Value = (cakePopularity[2] / sum * 100) },
                new BarItem{ Value = (cakePopularity[3] / sum * 100) },
                new BarItem{ Value = (cakePopularity[4] / sum * 100) }
        }),
        LabelPlacement = LabelPlacement.Inside,
        LabelFormatString = "{0:.00}%"
};
model.Series.Add(barSeries);

model.Axes.Add(new CategoryAxis
{
        Position = AxisPosition.Left,
        Key = "CakeAxis",
        ItemsSource = new[]
        {
                "Apple cake",
                "Baumkuchen",
                "Bundt Cake",
                "Chocolate cake",
                "Carrot cake"
        }
});
```

Example(grouped) Here a more advanced example making use of grouping to always compare the two series' values.



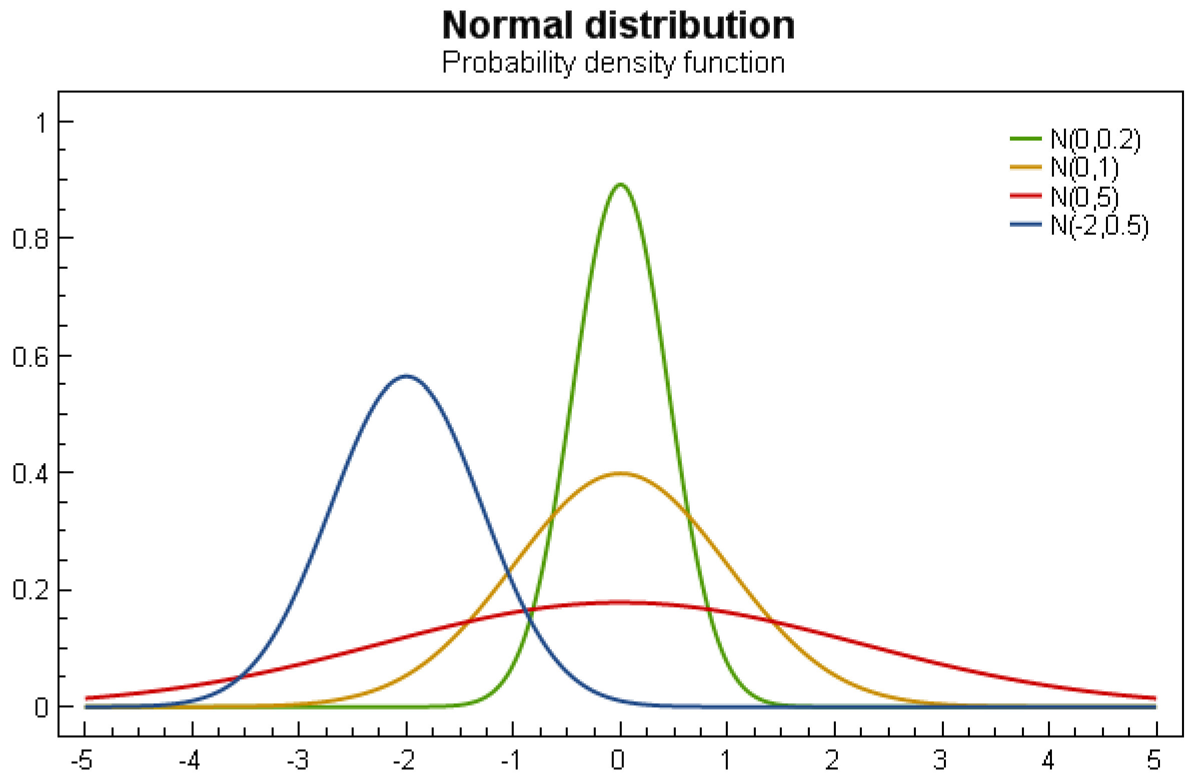
```
var model = new PlotModel
{
        Title = "BarSeries",
        LegendPlacement = LegendPlacement.Outside,
        LegendPosition = LegendPosition.BottomCenter,
        LegendOrientation = LegendOrientation.Horizontal,
        LegendBorderThickness = 0
};

var s1 = new BarSeries { Title = "Series 1", StrokeColor = OxyColors.Black, StrokeThickness = 1 };
s1.Items.Add(new BarItem { Value = 25 });
s1.Items.Add(new BarItem { Value = 137 });
s1.Items.Add(new BarItem { Value = 18 });
s1.Items.Add(new BarItem { Value = 40 });
```

```
var s2 = new BarSeries { Title = "Series 2", StrokeColor = OxyColors.Black, StrokeThickness = 1 };
s2.Items.Add(new BarItem { Value = 12 });
s2.Items.Add(new BarItem { Value = 14 });
s2.Items.Add(new BarItem { Value = 120 });
s2.Items.Add(new BarItem { Value = 26 });

var categoryAxis = new CategoryAxis { Position = AxisPosition.Left };
categoryAxis.Labels.Add("Category A");
categoryAxis.Labels.Add("Category B");
categoryAxis.Labels.Add("Category C");
categoryAxis.Labels.Add("Category D");
var valueAxis = new LinearAxis { Position = AxisPosition.Bottom, MinimumPadding = 0, MaximumPadding =
model.Series.Add(s1);
model.Series.Add(s2);
model.Axes.Add(categoryAxis);
model.Axes.Add(valueAxis);
```

BoxPlotSeries

Note: This section is under construction. Please contribute!

A `BoxPlotSeries` shows a set of points. The points can also have a size and color value.



Axes

- TODO

Data

- TODO

Tracker The tracker format string may use the following arguments:

- `{0}` the title of the series
- `{1}` the title of the x-axis
- `{2}` the x-value
- `{3}` the value of the upper whisker
- `{4}` the value of the third quartil
- `{5}` the value of the median
- `{6}` the value of the first quartil
- `{7}` the value of the lower whisker
- `{PropertyX}` the value of `PropertyX` in the item (extended format string syntax)

To show only the median value with one digit, use the format string `"{5:0.0}"`.

The default format string for `BoxPlotSeries` is `"{0}\n{1}: {2}\nUpper Whisker: {3:0.00}\nThird Quartil: {4:0.00}\nMedian: {5:0.00}\nFirst Quartil: {6:0.00}\nLower Whisker: {7:0.00}"`

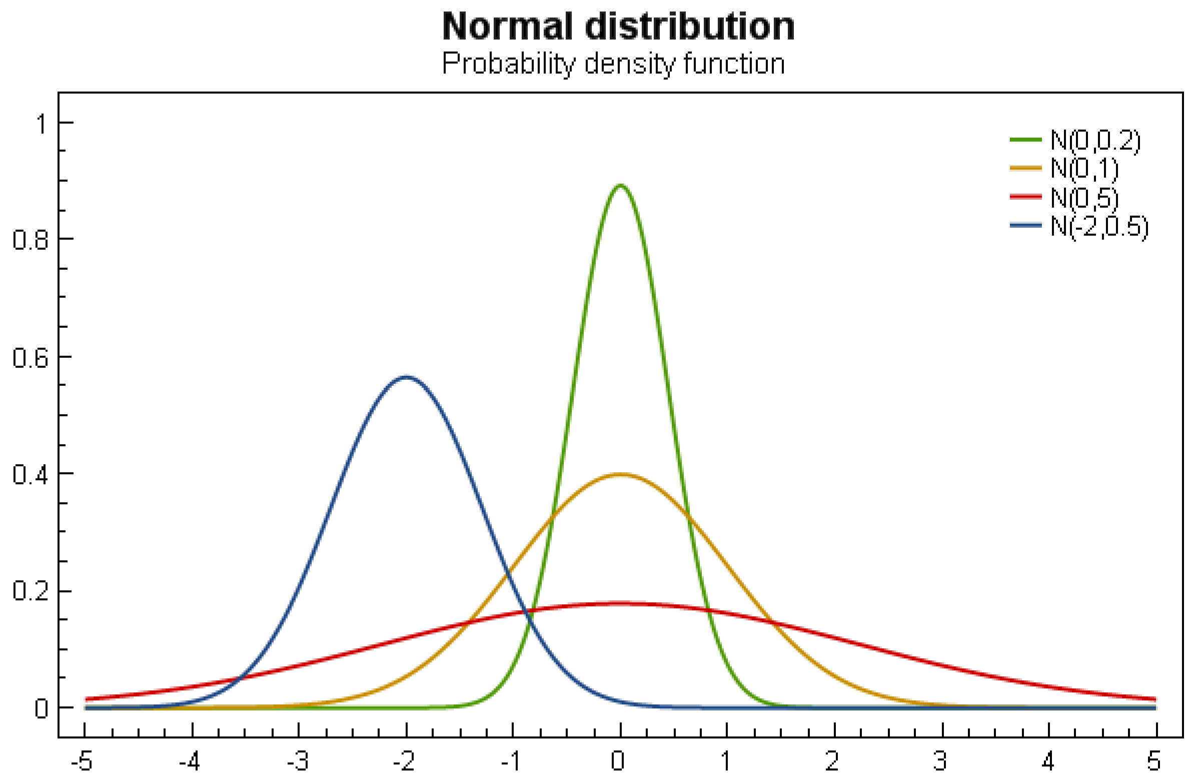
Example

```
var model = new PlotModel { Title = "BoxPlotSeries" };
```

CandleStickSeries

Note: This section is under construction. Please contribute!

A `CandleStickSeries` shows a set of points. The points can also have a size and color value.



Tracker The format string may use the following arguments:

- `{0}` the title of the series
- `{1}` the title of the x-axis
- `{2}` the x-value
- `{3}` the high value
- `{4}` the low value
- `{5}` the open value
- `{6}` the close value
- **`{PropertyX}` the value of `PropertyX` in the item (extended** format string syntax)

To show the close value with one digit, use the format string `"{6:0.0}"`.

The default format string for `CandleStickSeries` is `"{0}\n{1}: {2}\nHigh: {3:0.###}\nLow: {4:0.###}\nOpen: {5:0.###}\nClose: {6:0.###}"`

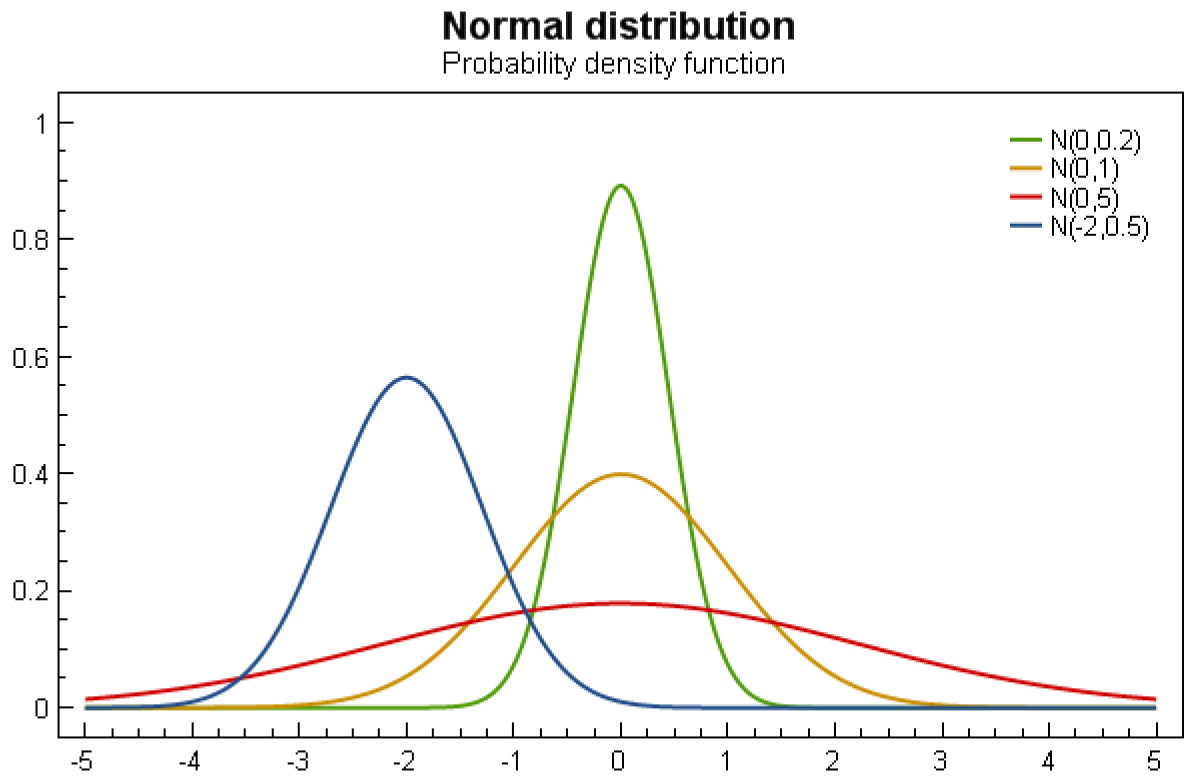
Example

```
var model = new PlotModel { Title = "CandleStickSeries" };
```

ColumnSeries

Note: This section is under construction. Please contribute!

A `ColumnSeries` shows the data set as vertical columns.



Axes

Data

Tracker The format string may use the following arguments:

- `{0}` the title of the series
- `{1}` the category
- `{2}` the column value
- `{PropertyX}` the value of `PropertyX` in the item (extended format string syntax)

To show the column value with one digit, use the format string `"{2:0.0}"`.

The default format string for `ColumnSeries` is `"{0}\n{1}:  {2}"`

Example

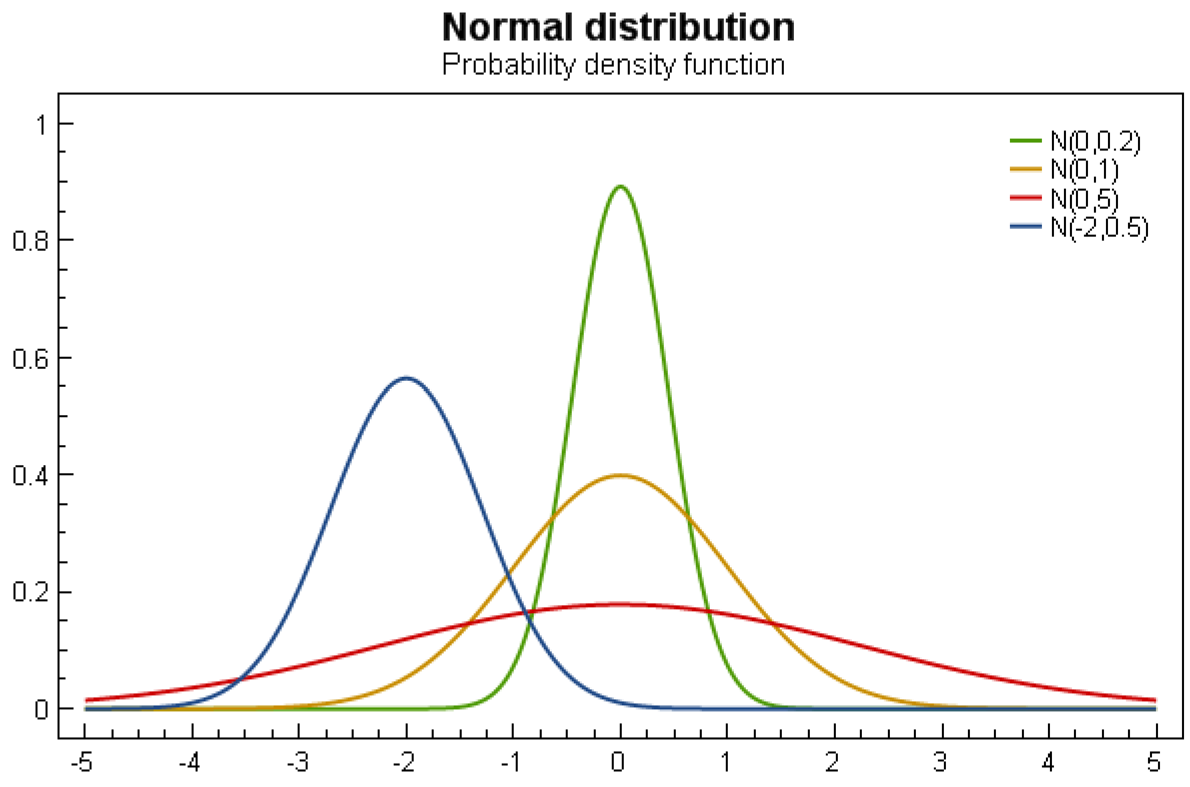
```
var model = new PlotModel { Title = "ColumnSeries" };
// A ColumnSeries requires a CategoryAxis on the x-axis.
model.Axes.Add(new CategoryAxis());
var series = new ColumnSeries();
```

```
model.Series.Add(series);
series.Items.Add(new ColumnItem(100));
```

ContourSeries

Note: This section is under construction. Please contribute!

A `ContourSeries` renders a 2D array of values as contours.



Axes

Data

Tracker The format string may use the following arguments:

- `{0}` the title of the series
- `{1}` the title of the x-axis
- `{2}` the x-value
- `{3}` the title of the y-axis
- `{4}` the y-value
- `{5}` the title of the contour level axis

- `{6}` the contour level
- `{PropertyX}` the value of `PropertyX` in the item (extended format string syntax)

To show the contour level with one digit, use the format string `"{6:0.0}"`.

The default format string for `ContourSeries` is `"{0}\n{1}: {2}\n{3}: {4}\n{5}: {6}"`

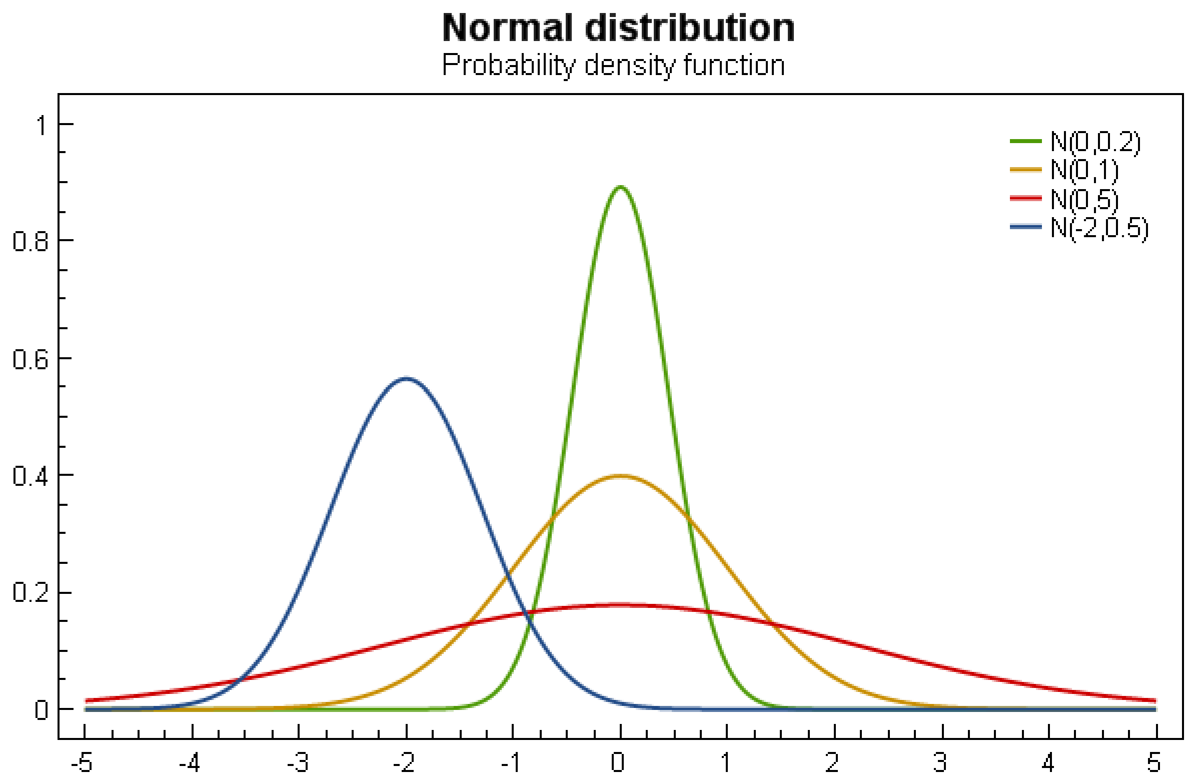
Example

```
var model = new PlotModel { Title = "ContourSeries" };
```

ErrorColumnSeries

Note: This section is under construction. Please contribute!

An `ErrorColumnSeries` shows a column series with error indicators.



Tracker The format string may use the following arguments:

- `{0}` the title of the series
- `{1}` the title of the x-axis
- `{2}` the value
- `{Error}` the error value
- `{PropertyX}` the value of `PropertyX` in the item (extended format string syntax)

To show the error value with one digit, use the format string `"{Error:0.0}"`.

The default format string for `ErrorColumnSeries` is `"{0}\n{1}: {2}, Error: {Error:0.###}"`

Example

```
var model = new PlotModel { Title = "ErrorColumnSeries" };
```

FunctionSeries

Note: This section is under construction. Please contribute!

A `FunctionSeries` shows a function plotted as a line.

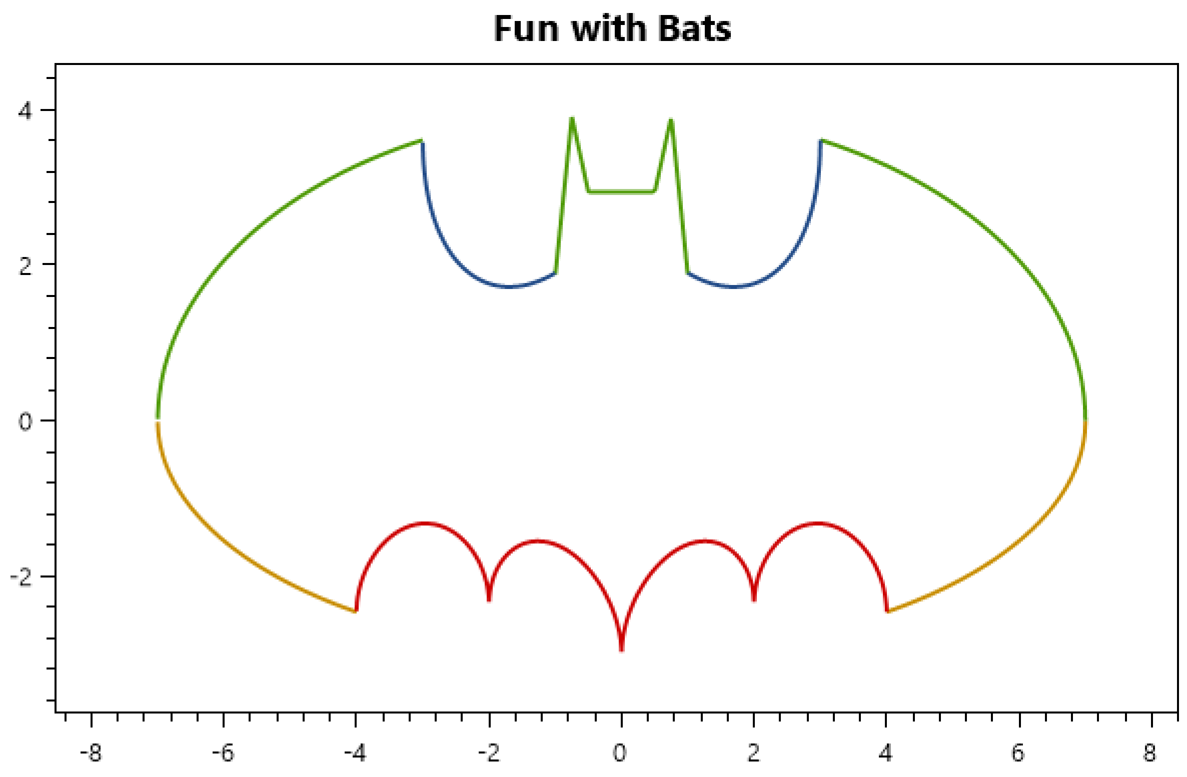
Tracker The format string may use the following arguments:

- `{0}` the title of the series
- `{1}` the title of the x-axis
- `{2}` the x-value
- `{3}` the title of the y-axis
- `{4}` the y-value
- `{PropertyX}` the value of `PropertyX` in the item (extended format string syntax)

To show the x and y values with one digit, use the format string `"{2:0.0},{4:0.0}"`.

The default format string for `FunctionSeries` is `"{0}\n{1}: {2:0.###}\n{3}: {4:0.###}"`

Example This example shows multiple functions combined by one `PlotModel`, displaying the well-known `Batman-Curve`.



```
var model = new PlotModel{ Title = "Fun with Bats" };

Func<double, double> batFn1 = (x) => 2 * Math.Sqrt(-Math.Abs(Math.Abs(x) - 1) * Math.Abs(3 - Math.Abs
Func<double, double> batFn2 = (x) => -3 * Math.Sqrt(1 - Math.Pow((x / 7), 2)) * Math.Sqrt(Math.Abs(Ma
Func<double, double> batFn3 = (x) => Math.Abs(x / 2) - 0.0913722 * (Math.Pow(x, 2)) - 3 + Math.Sqrt(1
Func<double, double> batFn4 = (x) => (2.71052 + (1.5 - .5 * Math.Abs(x)) - 1.35526 * Math.Sqrt(4 - Ma

model.Series.Add(new FunctionSeries(batFn1, -8, 8, 0.0001));
model.Series.Add(new FunctionSeries(batFn2, -8, 8, 0.0001));
model.Series.Add(new FunctionSeries(batFn3, -8, 8, 0.0001));
model.Series.Add(new FunctionSeries(batFn4, -8, 8, 0.0001));

model.Axes.Add(new LinearAxis{ Position = AxisPosition.Bottom, MaximumPadding = 0.1, MinimumPadding =
model.Axes.Add(new LinearAxis{ Position = AxisPosition.Left, MaximumPadding = 0.1, MinimumPadding = 0

return model;
```

HeatMapSeries

Note: This section is under construction. Please contribute!

A `HeatMapSeries` shows a 2D array of values as a heat map.

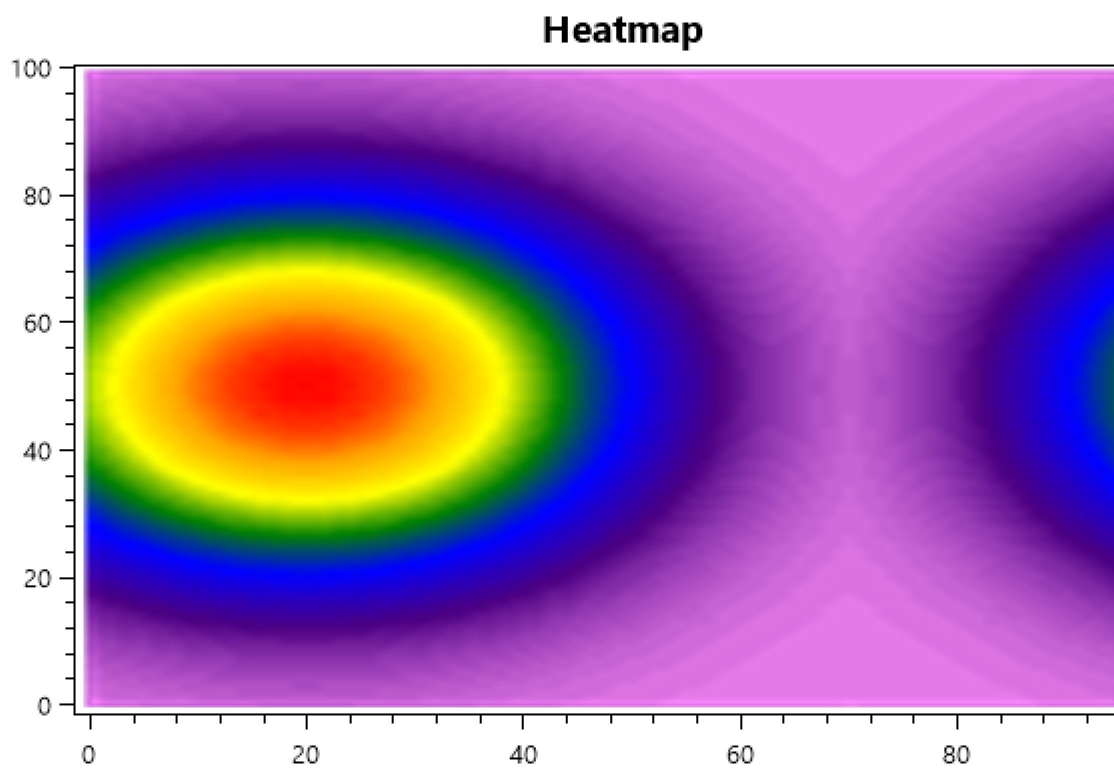
Tracker The format string may use the following arguments:

- `{0}` the title of the series

- `{1}` the title of the x-axis
- `{2}` the x-value
- `{3}` the title of the y-axis
- `{4}` the y-value
- `{5}` the title of the value/color-axis
- `{6}` the color-value
- `{PropertyX}` the value of `PropertyX` in the item (extended format string syntax)

To show the x and y values with one digit, use the format string `"{2:0.0},{4:0.0}"`.

The default format string for `HeatMapSeries` is `"{0}\n{1}:  {2}\n{3}:  {4}\n{5}:  {6}"`



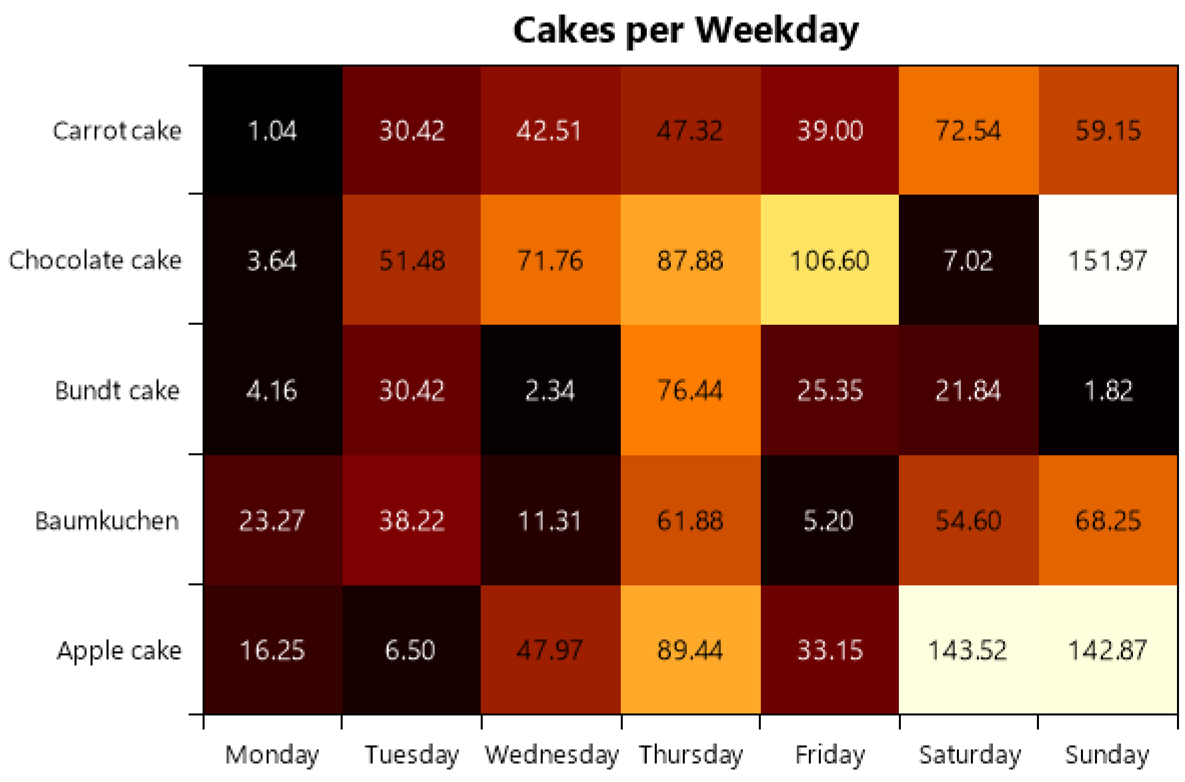
Example (Linear, Bitmap)

```
var model = new PlotModel { Title = "Heatmap" };

// Color axis (the X and Y axes are generated automatically)
model.Axes.Add(new LinearColorAxis
{
    Palette = OxyPalettes.Rainbow(100)
});

// generate 1d normal distribution
var singleData = new double[100];
for (int x = 0; x < 100; ++x)
{
    singleData[x] = Math.Exp((-1.0 / 2.0) * Math.Pow(((double)x - 50.0) / 20.0, 2.0));
}
```

```
// generate 2d normal distribution
var data = new double[100, 100];
for (int x = 0; x < 100; ++x)
{
    for (int y = 0; y < 100; ++y)
    {
        data[y, x] = singleData[x] * singleData[(y + 30) % 100] * 100;
    }
}

var heatMapSeries = new HeatMapSeries
{
    X0 = 0,
    X1 = 99,
    Y0 = 0,
    Y1 = 99,
    Interpolate = true,
    RenderMethod = HeatMapRenderMethod.Bitmap,
    Data = data
};

model.Series.Add(heatMapSeries);
```

Example (Categorized, Rectangle) The following diagram has categories on both, the x-axis and the y-axis.

It visualizes the amount of cake (y-axis) consumed on the specific day of week (x-axis). As can be seen in the diagram, the amount of cake rises from Monday to Sunday.



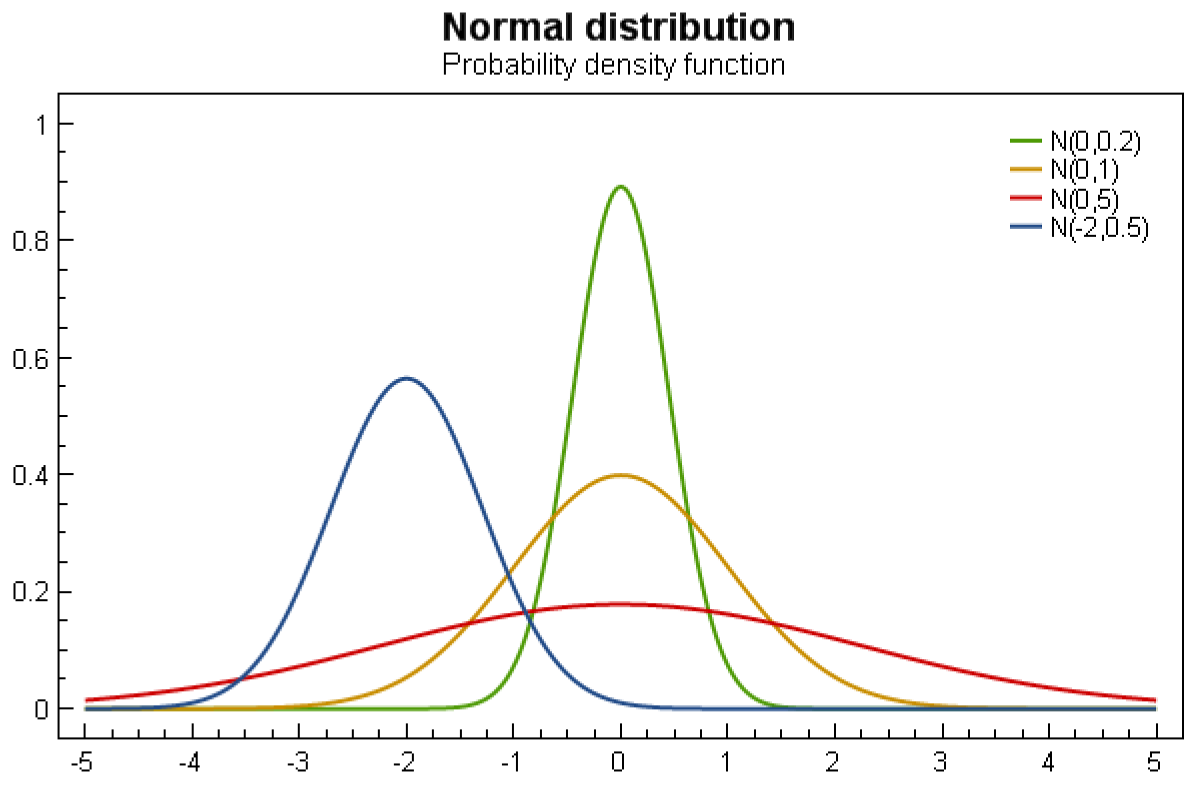
	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday	Sunday
Carrot cake	1.04	30.42	42.51	47.32	39.00	72.54	59.15
Chocolate cake	3.64	51.48	71.76	87.88	106.60	7.02	151.97
Bundt cake	4.16	30.42	2.34	76.44	25.35	21.84	1.82
Baumkuchen	23.27	38.22	11.31	61.88	5.20	54.60	68.25
Apple cake	16.25	6.50	47.97	89.44	33.15	143.52	142.87

```
var model = new PlotModel { Title = "Cakes per Weekday" };

// Weekday axis (horizontal)
model.Axes.Add(new CategoryAxis
{
    Position = AxisPosition.Bottom,

    // Key used for specifying this axis in the HeatMapSeries
    Key = "WeekdayAxis",

    // Array of Categories (see above), mapped to one of the coordinates of the 2D-data array
    ItemsSource = new[]
    {
            "Monday",
            "Tuesday",
            "Wednesday",
            "Thursday",
            "Friday",
            "Saturday",
            "Sunday"
    }
});

// Cake type axis (vertical)
model.Axes.Add(new CategoryAxis
{
    Position = AxisPosition.Left,
    Key = "CakeAxis",
    ItemsSource = new[]
    {
            "Apple cake",
            "Baumkuchen",
            "Bundt cake",
            "Chocolate cake",
            "Carrot cake"
    }
});

// Color axis
model.Axes.Add(new LinearColorAxis
{
    Palette = OxyPalettes.Hot(200)
});

var rand = new Random();
var data = new double[7, 5];
for (int x = 0; x < 5; ++x)
{
    for (int y = 0; y < 7; ++y)
    {
        data[y, x] = rand.Next(0, 200) * (0.13 * (y + 1));
    }
}

var heatMapSeries = new HeatMapSeries
{
    X0 = 0,
    X1 = 6,
    Y0 = 0,
    Y1 = 4,
    XAxisKey = "WeekdayAxis",
    YAxisKey = "CakeAxis",
    RenderMethod = HeatMapRenderMethod.Rectangles,
    LabelFontSize = 0.2, // neccessary to display the label
    Data = data
};

model.Series.Add(heatMapSeries);
```

HighLowSeries

Note: This section is under construction. Please contribute!

A `HighLowSeries` shows a set of points. The points can also have a size and color value.



Tracker The format string may use the following arguments:

- `{0}` the title of the series
- `{1}` the title of the x-axis
- `{2}` the x-value
- `{3}` the high value
- `{4}` the low value

- `{5}` the open value
- `{6}` the close value
- `{PropertyX}` the value of `PropertyX` in the item (extended format string syntax)

To show the close value with one digit, use the format string `"{6:0.0}"`.

The default format string for `HighLowSeries` is `"{0}\n{1}: {2}\nHigh: {3:0.###}\nLow: {4:0.###}\nOpen: {5:0.###}\nClose: {6:0.###}"`

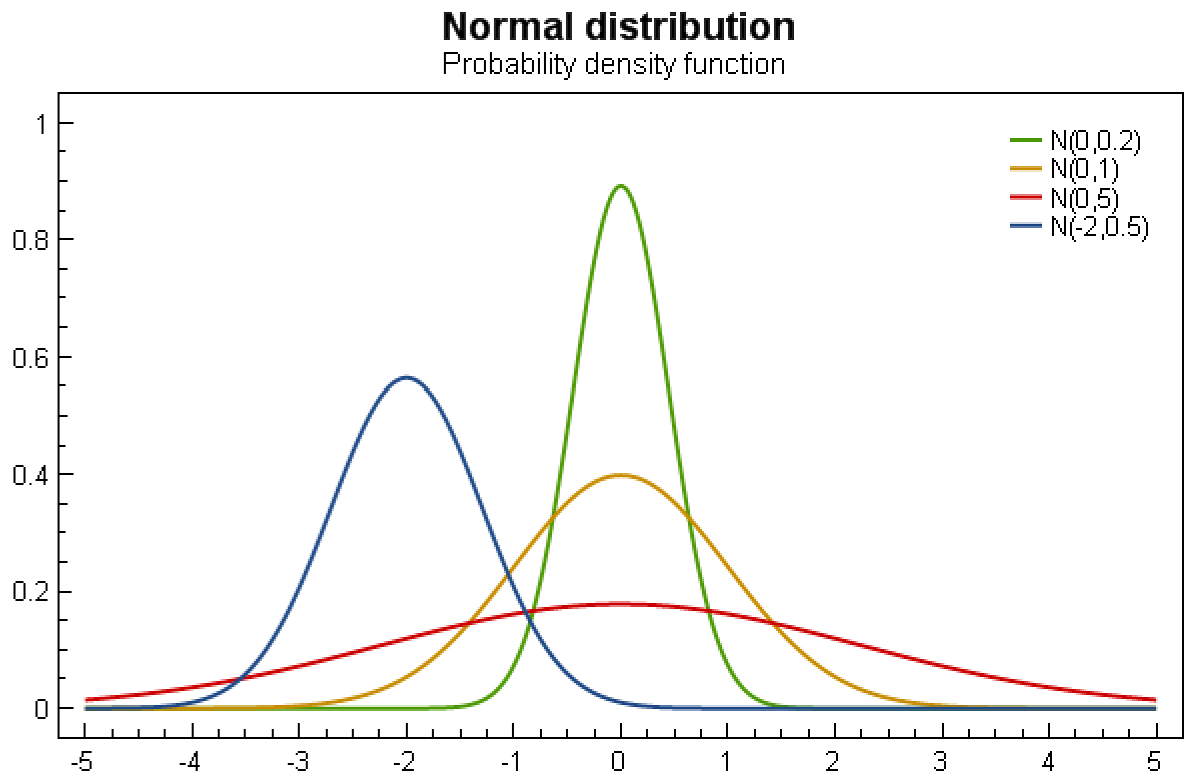
Example

```
var model = new PlotModel { Title = "HighLowSeries" };
```

IntervalBarSeries

Note: This section is under construction. Please contribute!

A `IntervalBarSeries` shows columns defined by start and end values.



Tracker The format string may use the following arguments:

- `{0}` the title of the series
- `{1}` the title of the category axis
- `{2}` the category

- `{3}` the title of the value axis
- `{4}` the start value
- `{5}` the end value
- `{6}` the item title
- `{PropertyX}` the value of `PropertyX` in the item (extended format string syntax)

To show the start and end values with one digit, use the format string `"{4:0.0},{5:0.0}"`.

The default format string for `IntervalBarSeries` is `"{0}\n{1}:  {2}\n{3}:  {4}"`

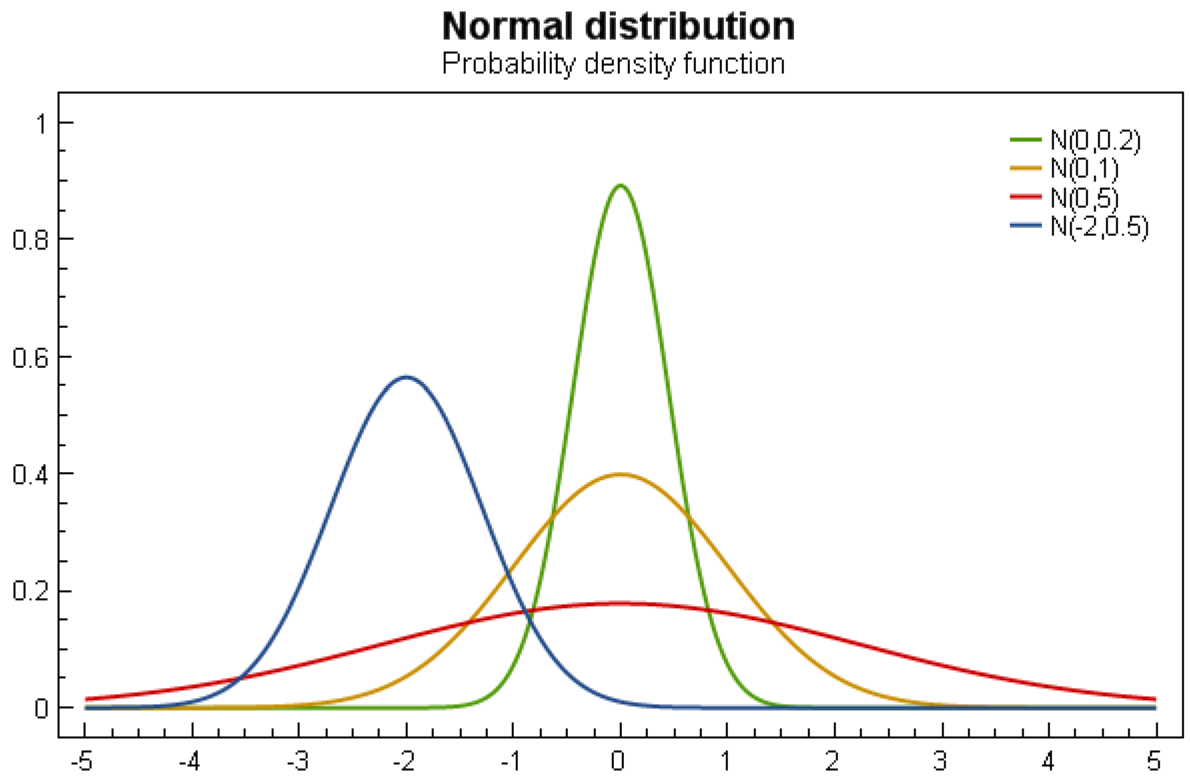
Example

```
var model = new PlotModel { Title = "IntervalBarSeries" };
```

LineSeries

Note: This section is under construction. Please contribute!

A `LineSeries` is used to render data as a polyline in the plot. It is also possible to render markers at each point of the polyline.



Axes A `LineSeries` requires a horizontal and a vertical axis.

By default, the `LineSeries` will use the default horizontal and vertical axes in the parent `PlotModel`. If there are more than one horizontal/vertical axis, the axes can be specified by the `XAxisKey` and `YAxisKey` properties. This requires the `Key` property to be set on the desired axes.

Data Use the `Points` collection to add data to the `LineSeries`:

```
lineSeries1.Points.Add(new DataPoint(0, 0));
lineSeries1.Points.Add(new DataPoint(100, 40));
```

Alternatively, you can specify a collection in the `ItemsSource` property.

- If the `Mapping` property is set, each element in the collection will be transformed
- If the collection is a list of `DataPoint`, or a type that implements `IDataPointProvider`, it will be used with no mapping
- If the `DataFieldX` and `DataFieldY` properties are set, each element of the collection will be reflected to create a data point

Tracker The `TrackerFormatString` property is used to format the string shown in the tracker. The format string may use the following arguments:

- `{0}` the title of the current series
- `{1}` the title of the x-axis
- `{2}` the x-value
- `{3}` the title of the y-axis
- `{4}` the y-value
- `{PropertyX}` the value of `PropertyX` in the nearest item (extended format string syntax)

To show the x and y values with one digit, use the format string `"{2:0.0},{4:0.0}"`.

If an item was hit, it is also possible to use the extended format string syntax, e.g. `{PropertyX:0.##}`, where the value of `PropertyX` will be found by reflection of the item.

The default format string for `LineSeries` is `"{0}\n{1}:  {2:0.###}\n{3}:  {4:0.###}"`

See MSDN for more information about format strings.

The `CanTrackerInterpolatePoints` property should be set to `false` if the tracker should not interpolate values between the points. The default value is `true`.

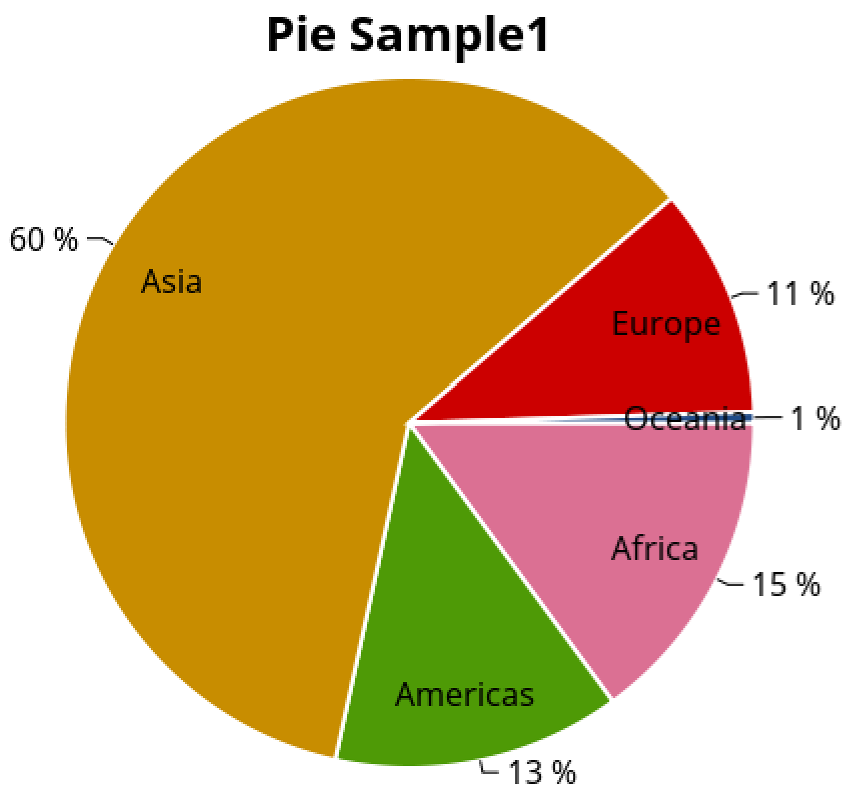
The `TrackerKey` property may be used to specify a custom tracker. This makes it possible to use different trackers for each series.

Color The `Color` defines the color of the line. The default value is `Automatic`. In this case the color will be set automatically from the colors specified in the `DefaultColors``property of the parent ``PlotModel`.

PieSeries

Note: This section is under construction. Please contribute!

A `PieSeries` renders a pie chart. Only one pie series is supported per `PlotModel`.



Tracker The format string may use the following arguments:

- `{0}` the title of the series
- `{1}` the label of the pie slice
- `{2}` the value of the pie slice
- `{3}` the percentage of the pie slice
- `{PropertyX}` the value of `PropertyX` in the item (extended format string syntax)

To show the values with one digit, use the format string `"{2:0.0}"`.

The default format string for `PieSeries` is `"{1}:  {2:0.###}  ({3:P1})"`

Labels The format string may use the following arguments:

- `{0}` the value of the pie slice
- `{1}` the label of the pie slice
- `{2}` the percentage of the pie slice
- `{PropertyX}` the value of `PropertyX` in the item (extended format string syntax)

To show the values with one digit, use the format string `"{2:0.0}"`.

The default format string for `PieSeries` is `"{1}:  {2:0.###}  ({3:P1})"`

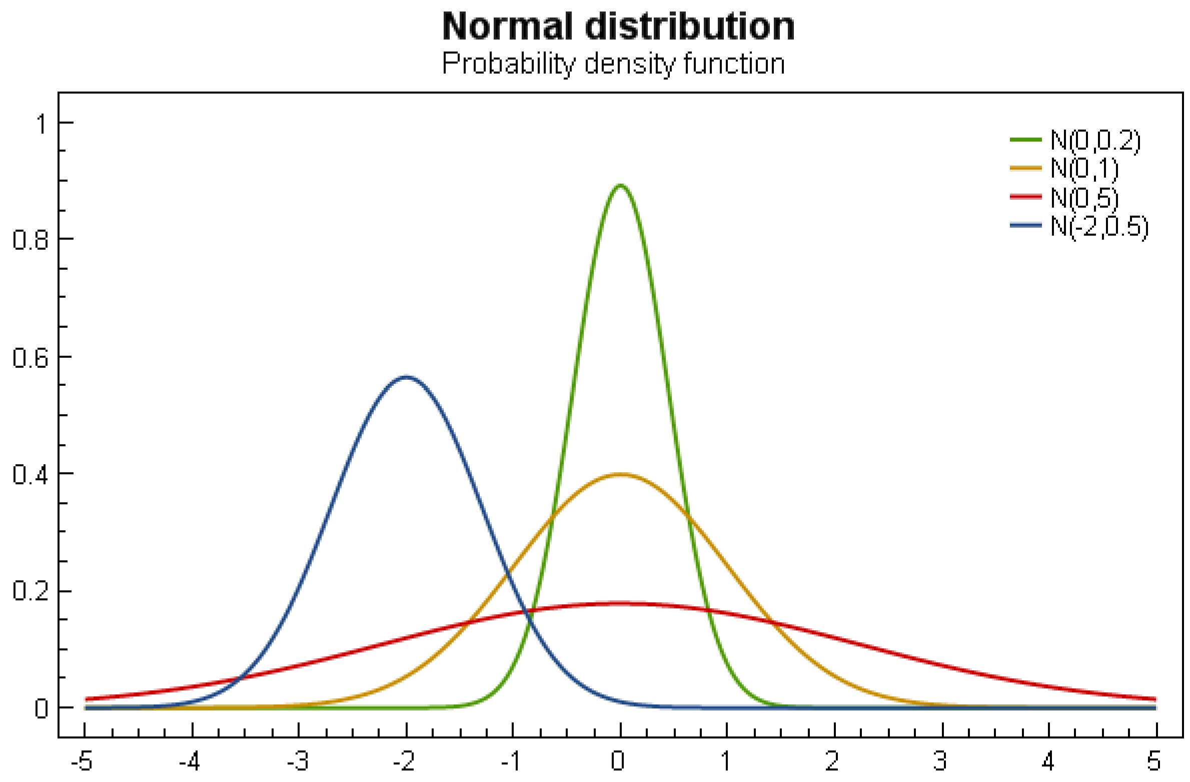
Example

```
using OxyPlot;
using OxyPlot.Series;

namespace ExampleLibrary
{

    public class PieViewModel
    {
        private PlotModel modelP1;
        public PieViewModel()
        {
            modelP1 = new PlotModel { Title = "Pie Sample1" };

            dynamic seriesP1 = new PieSeries { StrokeThickness = 2.0, InsideLabelPosition = 0.8, Angl

            seriesP1.Slices.Add(new PieSlice("Africa", 1030) { IsExploded = false, Fill = OxyColors.P
            seriesP1.Slices.Add(new PieSlice("Americas", 929) { IsExploded = true });
            seriesP1.Slices.Add(new PieSlice("Asia", 4157) { IsExploded = true });
            seriesP1.Slices.Add(new PieSlice("Europe", 739) { IsExploded = true });
            seriesP1.Slices.Add(new PieSlice("Oceania", 35) { IsExploded = true });

            modelP1.Series.Add(seriesP1);

        }

        public PlotModel Model1
        {
            get { return modelP1; }
            set { modelP1 = value; }
        }

    }

}
```

RectangleBarSeries

Note: This section is under construction. Please contribute!

A `RectangleBarSeries` shows rectangles defined by minimum and maximum x and y values.



Tracker The format string may use the following arguments:

- `{0}` the title of the series
- `{1}` the title of the x-axis
- `{2}` the minimum x-value (X0)
- `{3}` the maximum x-value (X1)
- `{4}` the title of the y-axis
- `{5}` the minimum y-value (Y0)
- `{6}` the maximum y-value (Y1)
- `{PropertyX}` the value of `PropertyX` in the item (extended format string syntax)

To show the minimum x and y values with one digit, use the format string `"{2:0.0},{5:0.0}"`.

The default format string for `RectangleBarSeries` is `"{0}\n{1}:  {2} {3}\n{4}:  {5} {6}"`

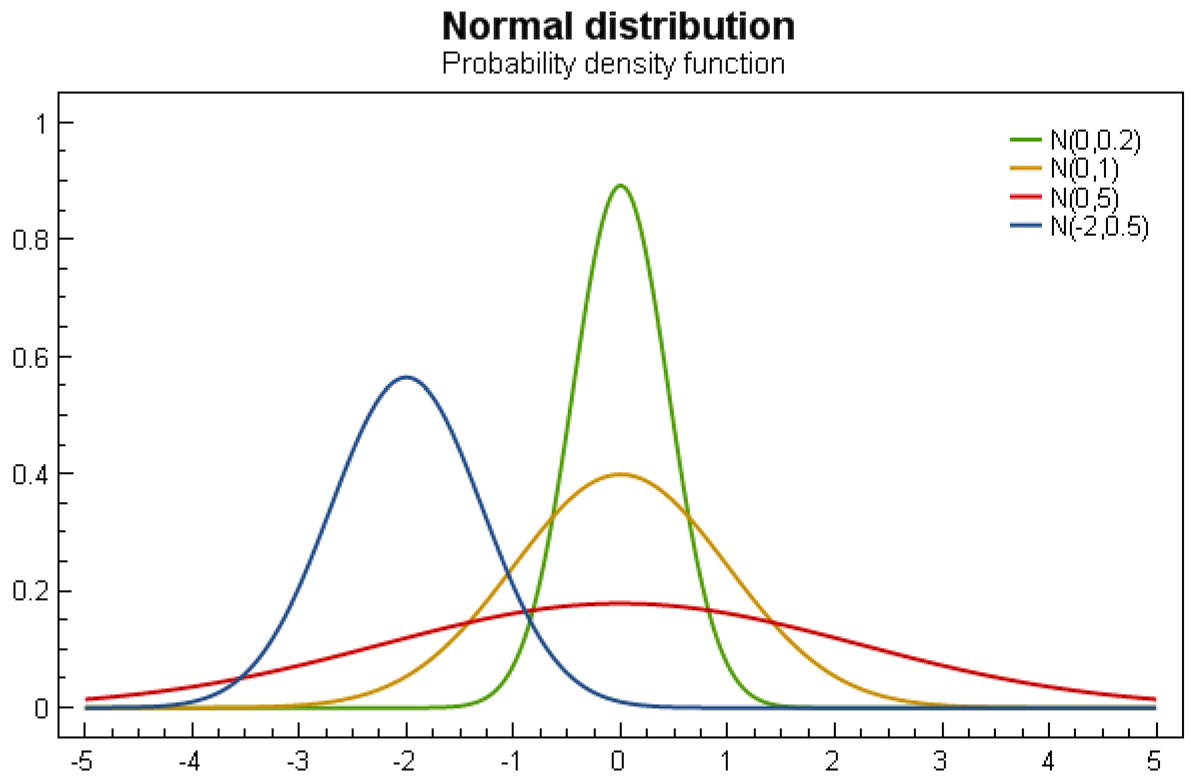
Example

```
var model = new PlotModel { Title = "RectangleBarSeries" };
```

ScatterSeries

Note: This section is under construction. Please contribute!

A `ScatterSeries` shows a set of points. The points can also have a size and color value.



Axes

Data

Tracker The format string may use the following arguments:

- `{0}` the title of the series
- `{1}` the title of the x-axis
- `{2}` the x-value
- `{3}` the title of the y-axis
- `{4}` the y-value
- `{5}` the title of the value/color-axis
- `{6}` the color-value
- `{PropertyX}` the value of `PropertyX` in the item (extended format string syntax)

To show the x and y values with one digit, use the format string `"{2:0.0},{4:0.0}"`.

The default format string for `ScatterSeries` is `"{0}\n{1}: {2:0.###}\n{3}: {4:0.###}"`

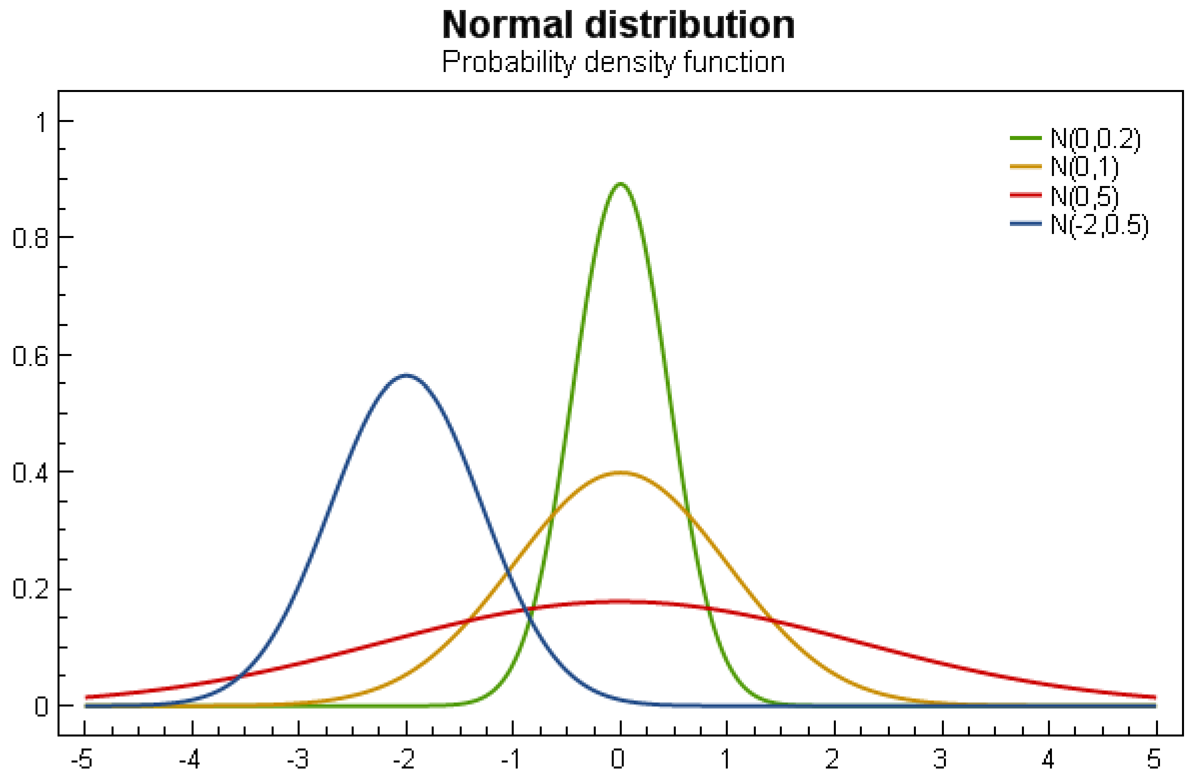
Example

```
var model = new PlotModel { Title = "ScatterSeries" };
var scatterSeries = new ScatterSeries { MarkerType = MarkerType.Circle };
var r = new Random(314);
for (int i = 0; i < 100; i++)
{
    var x = r.NextDouble();
    var y = r.NextDouble();
    var size = r.Next(5, 15);
    var colorValue = r.Next(100, 1000);
    scatterSeries.Points.Add(new ScatterPoint(x, y, size, colorValue));
}

model.Series.Add(scatterSeries);
model.Axes.Add(new LinearColorAxis { Position = AxisPosition.Right, Palette = OxyPalettes.Jet(200) })
```

StairStepSeries

Note: This section is under construction. Please contribute!

A `StairStepSeries` shows a stairstep curve at each data point.



Tracker The format string may use the following arguments:

- `{0}` the title of the series

- `{1}` the title of the x-axis
- `{2}` the x-value
- `{3}` the title of the y-axis
- `{4}` the y-value
- `{PropertyX}` the value of `PropertyX` in the item (extended format string syntax)

To show the x and y values with one digit, use the format string `"{2:0.0},{4:0.0}"`.

The default format string for `StairStepSeries` is `"{0}\n{1}: {2:0.###}\n{3}: {4:0.###}"`

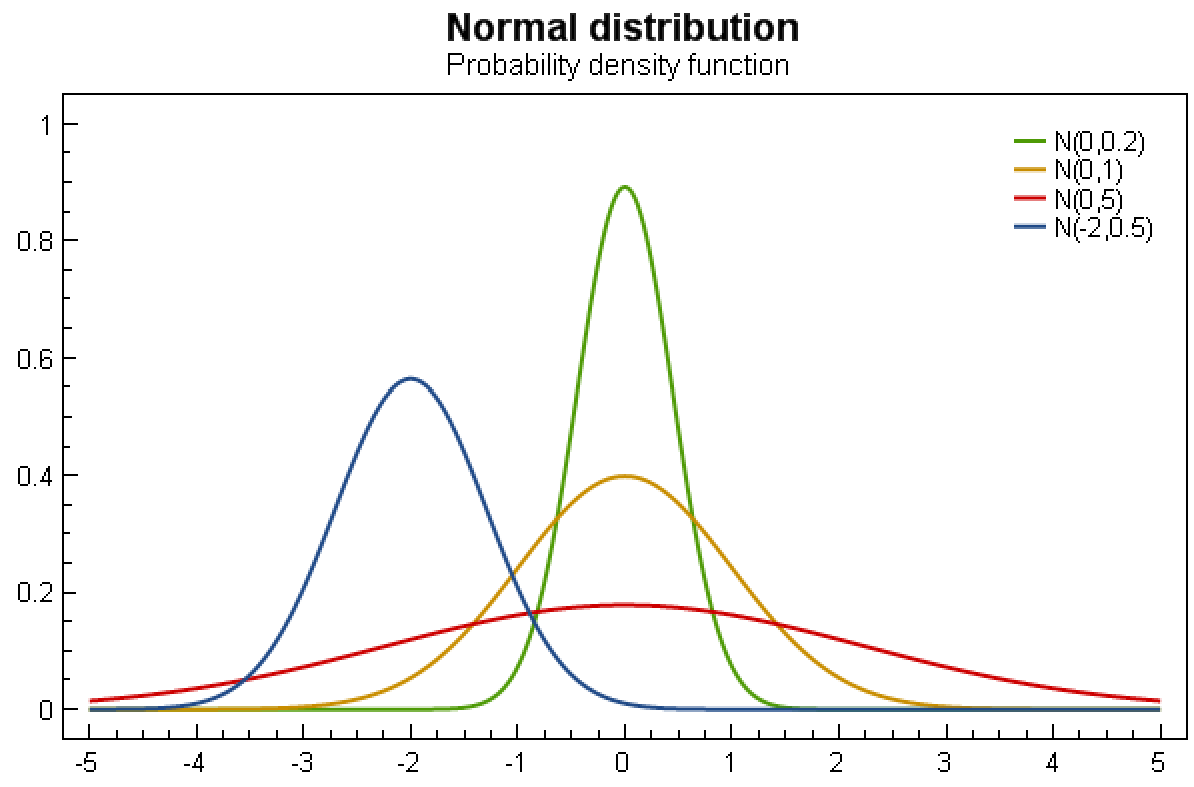
Example

```
var model = new PlotModel { Title = "StairStepSeries" };
```

StemSeries

Note: This section is under construction. Please contribute!

A `StemSeries` shows stems to each data point.



Tracker The format string may use the following arguments:

- `{0}` the title of the series
- `{1}` the title of the x-axis

- `{2}` the x-value
- `{3}` the title of the y-axis
- `{4}` the y-value
- `{PropertyX}` the value of `PropertyX` in the item (extended format string syntax)

To show the x and y values with one digit, use the format string `"{2:0.0},{4:0.0}"`.

The default format string for `StemSeries` is `"{0}\n{1}:  {2:0.###}\n{3}:  {4:0.###}"`

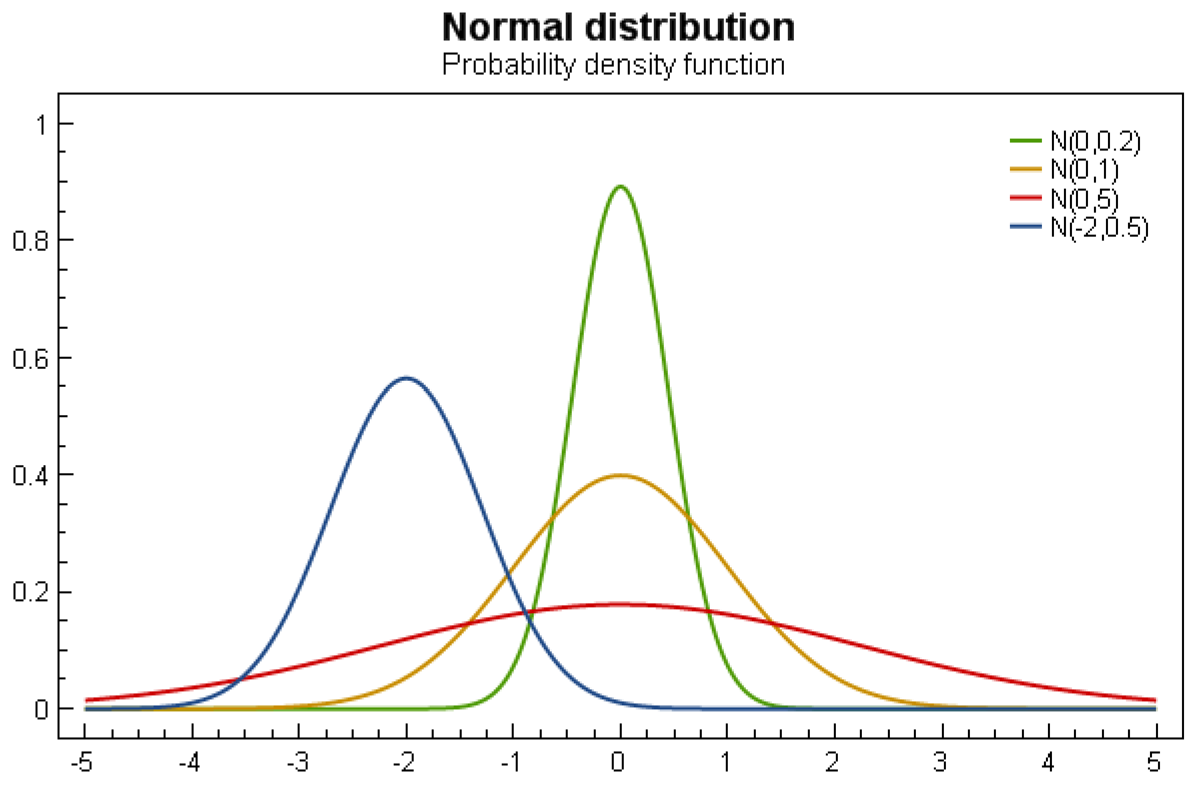
Example

```
var model = new PlotModel { Title = "StemSeries" };
```

TornadoBarSeries

Note: This section is under construction. Please contribute!

A `TornadoBarSeries` shows columns defined by a minimum, maximum and base value.



Tracker The format string may use the following arguments:

- `{0}` the title of the series
- `{1}` the title of the category axis
- `{2}` the category

- `{3}` the title of the value axis
- `{4}` the minimum or maximum value
- `{PropertyX}` the value of `PropertyX` in the item (extended format string syntax)

To show the value with one digit, use the format string `"{4:0.0}"`.

The default format string for `TornadoBarSeries` is `"{0}\n{1}:  {2}\n{3}:  {4}"`

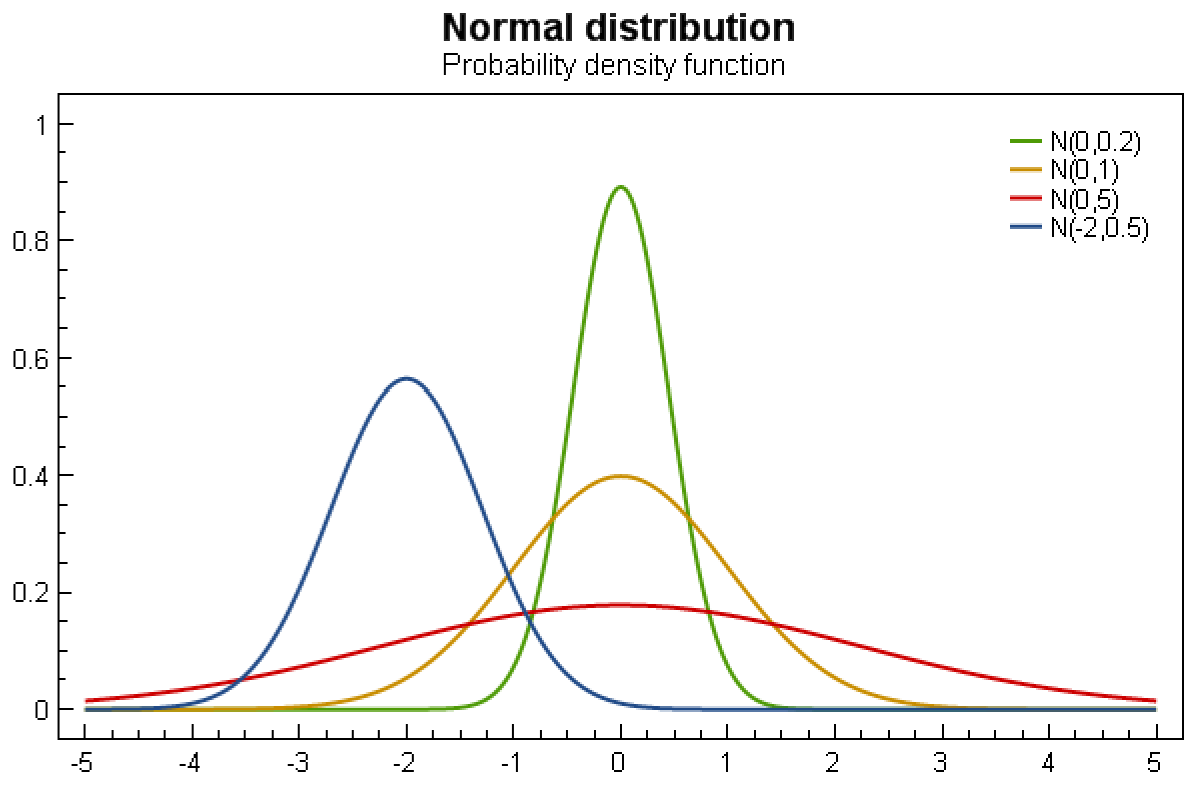
Example

```
var model = new PlotModel { Title = "TornadoBarSeries" };
```

TwoColorAreaSeries

Note: This section is under construction. Please contribute!

A `TwoColorAreaSeries` shows an area series where the area has different color of each side of a specified limit.



Tracker The format string may use the following arguments:

- `{0}` the title of the series
- `{1}` the title of the x-axis
- `{2}` the x-value
- `{3}` the title of the y-axis

- `{4}` the y-value
- `{PropertyX}` the value of `PropertyX` in the item (extended format string syntax)

To show the x and y values with one digit, use the format string `"{2:0.0},{4:0.0}"`.

The default format string for `TwoColorAreaSeries` is `"{0}\n{1}: {2:0.###}\n{3}: {4:0.###}"`

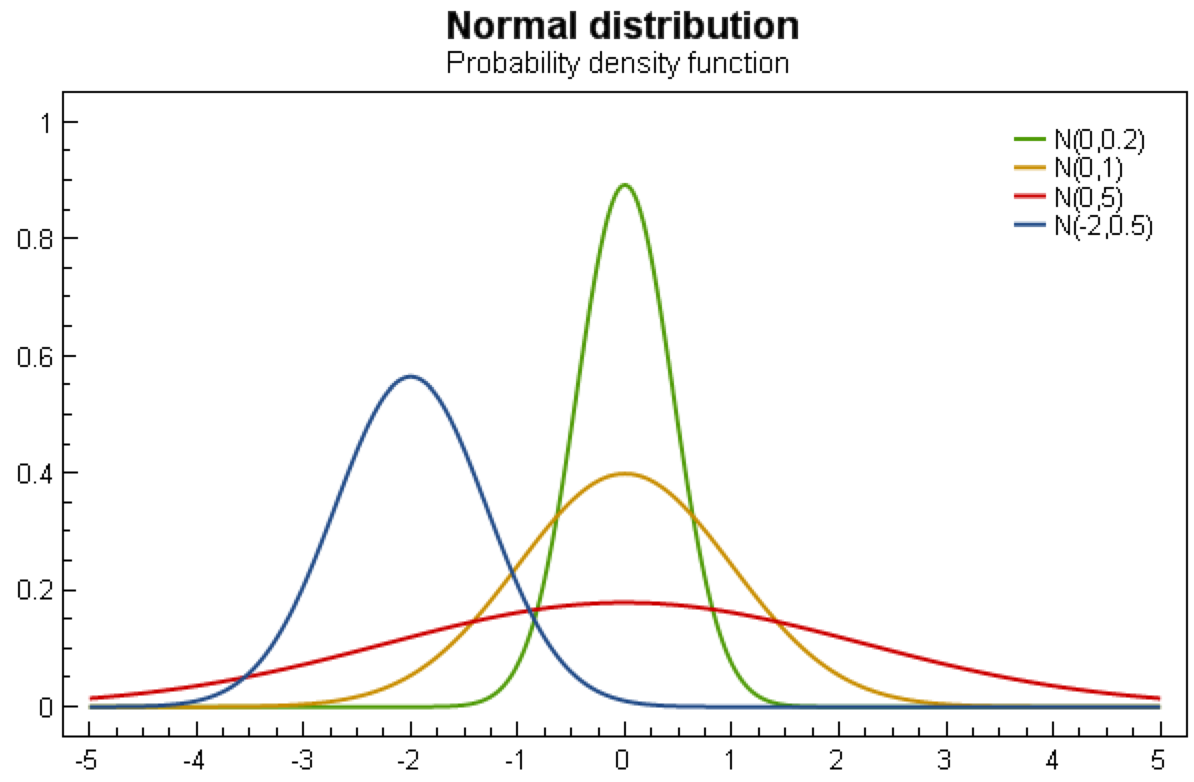
Example

```
var model = new PlotModel { Title = "TwoColorAreaSeries" };
```

TwoColorLineSeries

Note: This section is under construction. Please contribute!

A `TwoColorLineSeries` shows a line where the the color is different above and below a specified limit.



Tracker The format string may use the following arguments:

- `{0}` the title of the series
- `{1}` the title of the x-axis
- `{2}` the x-value
- `{3}` the title of the y-axis
- `{4}` the y-value

To show the x and y values with one digit, use the format string `"{2:0.0},{4:0.0}"`.

The default format string for `TwoColorLineSeries` is `"{0}\n{1}: {2:0.###}\n{3}: {4:0.###}"`

Example

```
var model = new PlotModel { Title = "TwoColorLineSeries" };
```

1.3.5 Annotations

Note: This section is under construction. Please contribute!

Annotations are elements of the plot that shows information that is not part of the series collection. This also means the annotations will not show up in the series legends. Arrows, text labels, lines and areas are examples of annotations.

Annotation types

ArrowAnnotation

Note: This section is under construction. Please contribute!

To add an arrow to your *PlotModel*:

```
var arrowAnnotation = new ArrowAnnotation {
    StartPoint = new DataPoint(0, 0),
    EndPoint = new DataPoint(10, 10)
};
myPlotModel.Annotations.Add(arrowAnnotation);
```

- The `ArrowDirection` property can be used to create an arrow that has a fixed size in screen coordinates. If this property is set, the `StartPoint` property is not used.
- `HeadLength`, `HeadWidth` and `Veeness` controls the shape of the arrow head. The values are relative to the thickness.
- The `Text` property can be used to set text that is shown at the start point of the arrow. The alignment is automatic and related to the direction of the arrow.

EllipseAnnotation

Note: This section is under construction. Please contribute!

FunctionAnnotation

Note: This section is under construction. Please contribute!

ImageAnnotation

Note: This section is under construction. Please contribute!

LineAnnotation

Note: This section is under construction. Please contribute!

PointAnnotation

Note: This section is under construction. Please contribute!

PolygonAnnotation

Note: This section is under construction. Please contribute!

PolyLineAnnotation

Note: This section is under construction. Please contribute!

RectangleAnnotation

Note: This section is under construction. Please contribute!

TextAnnotation

Note: This section is under construction. Please contribute!

1.4 Controller

Note: This section is under construction. Please contribute!

1.4.1 PlotController

Explain...

1.4.2 Default input bindings

The default input bindings in the *PlotController* are:

Action	Gesture
Pan*	Right mouse button
Zoom*	Mouse wheel
Zoom by rectangle	Ctrl+Right mouse button
Reset*	Right mouse button double-click
Show 'tracker'	Left mouse button
Reset axes	'A'
Copy bitmap	Ctrl+C
Copy code	Ctrl+Alt+C
Copy properties	Ctrl+Alt+R

You can zoom/pan/reset a single axis by positioning the mouse cursor over the axis before starting the zoom/pan.

1.4.3 Customizing the bindings

Create a new PlotController:

```
var myController = new PlotController();
```

Set the controller in the *PlotView* control

```
plotView.Controller = myController;
```

Bind an input gesture to a command

```
myController.BindMouseDown(OxyMouseButton.Left, PlotCommands.Pan);
```

Unbind an input gesture

```
myController.UnbindMouseDown(OxyMouseButton.Right);
```

Unbind all commands

```
myController.UnbindAll();
```

- Creating new commands
- Creating new manipulators

1.5 View

1.5.1 Plot views

Note: This section is under construction. Please contribute!

A `PlotView` control is implemented for each `supported-platforms`.

// TODO: features, properties, events etc.

1.5.2 Tracker

Note: This section is under construction. Please contribute!

The "tracker" is shown when you press down the left mouse button over some data in the plot. By default the tracker shows the values of the current point.

Format string

The easiest way to modify the information shown in the tracker, is to modify the *TrackerFormatString* of the series.

The arguments are specific to each class.

|| Class || ~{0} || ~{1} || Examples | | LineSeries | X | Y | *"{0:0.00} {1:0.00}"* |

If you need to include extra information in the tracker beyond the standard parameters (e.g. series name, x and y values), you can use the `Series.ItemSource` property to populate the data points (see e.g. LineSeries for more details). The format string can also contain formatting codes for properties in the `ItemSource` list item that defines the data point closest to the tracker hit. Use the property name inside the curly braces: *{MyProperty:0.00}*.

The culture of the tracker format string can be set in the *Culture* property in the *PlotModel*. The default culture is the current UI culture.

Template

The WPF/Silverlight *Plot* controls contain a dependency property *DefaultTrackerTemplate* where a custom *ControlTemplate* can be defined. When the tracker is shown, the *DataContext* will be set to a *TrackerHitResult*. You can bind *Position* and data properties from the hit results.

> TODO: example template

See the *SourceExamplesWPFCustomTrackerDemo* application for examples on how to make custom trackers.

Tracker control

The WPF and Silverlight projects contain a *TrackerControl* that can be used in tracker templates. You should bind the *Position*, *LineExtents* and *Content* properties to data in the *TrackerHitResult*. Example

> TODO: template using TrackerControl

TrackerHitResult

This class contains data about the current hit of the tracker:

Position | the XY screen coordinates of the tracker.|
DataPoint | the data point if the tracker shows an item from a DataPointSeries.|
Series | the current series.|

PlotModel | the current plot model.|
LineExtents | the rectangle that should be used to draw the horizontal and vertical lines.|
XAxis | the current x axis.|
YAxis | the current y axis.|
Item | the current item if an *ItemsSource* was used to generate the data.|
Text | the text to be shown in the tracker. If this property is set, the series' *TrackerFormatString* will not be used.|

The TrackerHitResult for the current tracker state can be obtained by handling the PlotModel.TrackerChanged event. However, this event occurs after the text has been set on the tracker UI, so you cannot use this event handler to set custom tracker text.

Tracker definitions

If you want to specify different trackers for the series, you should define *TrackerDefinition's in the 'Plot* control. Set the *TrackerKey* of the series to select which *TrackerDefinition* to use.

1.6 Common tasks

1.6.1 Data binding

Note: This section is under construction. Please contribute!

- explain different ways to bind to data
- include code examples

Updating the plot after data has changed

OxyPlot is not observing property or collection changes. This means that the client application is responsible for refreshing the plot when the content has changed. See the Refreshing a plot section for information about how to do this.

1.6.2 Refreshing a plot

Note: This section is under construction. Please contribute!

The OxyPlot views do not automatically refresh when you change the `PlotModel` (e.g. changing series, axes, annotations or properties).

To update the plot, you must do one of the following alternatives:

- Change the `Model` property of the `PlotView` control
- Call `Invalidate` on the `PlotView` control
- Call `Invalidate` on the `PlotModel`

Examples

1.6.3 Text formatting

Note: This section is under construction. Please contribute!

Superscript and subscript

Titles, axis titles and series legends can contain sub- and superscript.

Use the following notation (like Tex):

Superscript: `x^{2}` Subscript: `H_{2}O`

Note: the sub- or superscript must always be enclosed in curly brackets (you cannot use `x^2`).

1.7 Guidelines

1.7.1 Performance guidelines

Note: This section is under construction. Please contribute!

Data binding

How you add data to your model is important for the performance. For series based on the DataPointSeries you have the following options, from fast to slow:

1. Add instances based on `IDataPoint` directly to the Points collection
2. Set ItemsSource to a collection of IDataPoints
3. Set ItemsSource and use the *Mapping* delegate
4. Set ItemsSource and use the data field properties (this uses reflection - slow!)

Style

The following style properties are important for the performance of the rendering:

- Solid lines are much faster than dashed/dotted lines
- Grid lines are slow to draw (be careful not creating too many of them)
- Unfilled markers (Plus, Cross, Star) are faster than the filled markers (Circle, Square, Diamond, Triangle)
- Square markers are faster than circles

1.7.2 Design guidelines

Note: This section is under construction. Please contribute!

The following guidelines are picked up mostly from the books by Edward Tufte and Stephen Few.

General

- 3D projection is not recommended (particularly for bar series, ribbons, pie charts)
- Pie charts are not recommended (difficult to compare values, bar charts are usually better)
- Minimize the ratio ink/data
- Don't use a border around legends
- Use only one font
- Don't use a border (box) around the plot

Grid lines

- Avoid dark/heavy grid lines

Ticks

- Use TickStyle = Outside
- Don't use minor tick marks

Category axes

- Don't use tick marks
- Don't use axis line on vertical category axes
- Limit the number of subdivisions (not more than 5-6 bar series)

Bar series

- Use a bar width around 50%
- For histograms you could use a bar width of 100%

Recommended reading

- The Visual Display of Quantitative Information (Edward Tufte) google amazon tutorial
- Perceptual edge (Stephen Few)
- Show me the numbers (Stephen Few) amazon google

1.8 Export

1.8.1 Export to PDF

Note: This section is under construction. Please contribute!

You have two options when you want to export your plots to PDF. You can use the `OxyPlot.PdfExporter` that is included in the portable OxyPlot core library, or the `OxyPlot.Pdf.PdfExporter` included in the OxyPlot.Pdf library. The latter depends on PdfSharp/SilverPdf and is not portable.

OxyPlot Pdf writer

- Simple PDF export is included in the OxyPlot core library
- There are limitations on text encoding, fonts and images

```
using (var stream = File.Create(fileName))
{
    var pdfExporter = new PdfExporter { Width = 600, Height = 400 };
    pdfExporter.Export(plotModel, stream);
}
```

See also the section Creating PDF files about the underlying Pdf writer.

OxyPlot.Pdf

- based on the PdfSharp/SilverPdf open-source projects
- better handling of fonts and images

1.8.2 Export to PNG

The PNG export functionality is located in the platform specific libraries (OxyPlot.Wpf, OxyPlot.WindowsForms). Note that PNG export is not yet supported on all platforms.

Write to a stream

```
var stream = new MemoryStream();
var pngExporter = new PngExporter { Width = 600, Height = 400, Background = OxyColors.White };
pngExporter.Export(plotModel, stream);
```

Write to a file

```
var pngExporter = new PngExporter { Width = 600, Height = 400, Background = OxyColors.White };
pngExporter.ExportToFile(plotModel, fileName);
```

Copy to the clipboard

```
var pngExporter = new PngExporter { Width = 600, Height = 400, Background = OxyColors.White };
var bitmap = pngExporter.ExportToBitmap(plotModel);
Clipboard.SetImage(bitmap);
```

1.8.3 Export to SVG

Note: This section is under construction. Please contribute!

The plots can be exported to SVG by the *SvgExporter* in the OxyPlot core library.

```
using (var stream = File.Create(fileName))
{
    var exporter = new SvgExporter { Width = 600, Height = 400 };
    exporter.Export(plotModel, stream);
}
```

- width/height units
- document svg option

Note that SVG can be exported to a standalone document (.svg file) or a HTML5 `<svg\>` element.

Text measuring

The SVG output requires an `ITextMeasurer` to measure string sizes (rendered width and height). If a text measurer is not specified, the text measurer of the `PdfRenderContext` will be used, which supports simple Type-1 fonts only (Helvetica/Arial, Roman, Courier) limited to WinAnsi encoding. To get better text measurements, use one of the render contexts from the platform specific libraries.

1.9 Extra features

1.9.1 Creating PDF files

The OxyPlot core library contains a class `PortableDocument` that can be used to create PDF files:

```
var doc = new PortableDocument();
doc.Title = "Hello world";
doc.Author = "objo";
doc.AddPage(PageSize.A4);
doc.SetFont("Arial", 96);
doc.DrawText(50, 400, "Hello world!");
doc.Save("HelloWorld.pdf");
```

Note that the coordinate system origin is at the bottom left corner of the page and the unit is point (1/72 inch).

The `PortableDocument` class supports

- document properties (title, author, subject etc.)
- multiple pages (specify size and orientation)
- text drawing

- Type 1 fonts (Helvetica, Roman, Courier) in WinAnsi encoding
- text size measuring
- circles
- ellipses
- lines
- polygons
- rectangles
- images
- clipping rectangle
- transforms
- transparency

More examples can be found in the unit tests in `Source\OxyPlot.Tests\Pdf\PortableDocumentTests.cs`

1.9.2 Report model

Note: This section is under construction. Please contribute!

The report model can be used to generate simple reports. The objective is to create a very simple object model that can be used for basic reports containing headers, paragraphs, simple tables and figures.

The supported output formats are HTML, Latex and plain text. Support for Word (OpenXML) and PDF is under consideration, but this will add dependencies to other libraries.

Item	Description	HTML	Latex	Plain text	Word	PDF
Header	Headers	Yes	Yes	Yes		
Paragraph	Text	Yes	Yes	Yes		
Table	Tables	Yes	Yes	Yes		
Image	Bitmap graphics	Yes	Yes	No		
Plot	Plot model	Yes	Yes	No		
Drawing	Vector graphics			No		
Equation	Using Tex syntax			No		

Example

```
var report = new Report();
// TODO
```

CHAPTER 2

Indices and tables

- genindex
- modindex
- search