# WPF – LiveCharts

Petr Novák / novakpe@fel.cvut.cz / 2021-07-31

## Obsah

1 Úvod ..... 1
2 Základní použití ..... 1
3 Základní typy grafů ..... 2
4 CartesianMapper ..... 5
5 Osy ..... 6
6 Data ve vlastní třídě ..... 7
7 Vytvoření Time-OnLine zobrazení ..... 8
8 Vodorovná čára v grafu ..... 8
9 Testovací aplikace ..... 8

## 1 Úvod

Existuje určitě několik různých knihoven pro tvorbu grafů ve WPF. Jedna zcela volně dostupná je **LiveCharts**. Poskytuje mnoho dostatečných schopností pro běžné použití. Tento dokument obsahuje popis jak tuto knihovnu použít. Její oficiální WWW stránka je **https://lvcharts.net/**. Pro použití **LiveCharts** je potřeba instalovat Nuget s názvem **LiveCharts** podle typu vytvořené aplikace:

.NET Core + WPF –> LiveCharts.WPF.NetCore3

Jeho celkem dobrá výhoda spočívá v možnosti zobrazování i časově OnLine dat.

## 2 Základní použití

Na začátek XAML přidat (případně obdobu podle použitého projektu):

```
xmlns:lvc="clr-namespace:LiveCharts.Wpf;assembly=LiveCharts.Wpf"
```

Do XAML na místo pro zobrazení grafu vložit například:

```
<lvc:CartesianChart Name="myLiveCharts" Zoom="None" Hoverable="False"
      DataTooltip="{x:Null}" DisableAnimations="True">
  <!-- pro zrychleni vykreslovani -->
  <lvc:CartesianChart.CacheMode>
    <BitmapCache EnableClearType="False" RenderAtScale="1"
          SnapsToDevicePixels="False" />
  </lvc:CartesianChart.CacheMode>
  <!-- X - sirka / horizontalni (pocet bodu na sirku) -->
  <lvc:CartesianChart.AxisX>
    <lvc:Axis Title="Time (X)" DisableAnimations="True" FontSize="14"
          Unit="1" MaxValue="20" MinValue="0" Foreground="PaleVioletRed" />
  </lvc:CartesianChart.AxisX>
```

```
    <!-- Y - vyska / vertikalni (velikost / rozsah hodnoty) -->
    <lvc:CartesianChart.AxisY>
      <lvc:Axis Title="Value (Y)" DisableAnimations="True" FontSize="14"
                Unit="1" MaxValue="100" MinValue="0" Foreground="PaleVioletRed" />
    </lvc:CartesianChart.AxisY>
  </lvc:CartesianChart>
```

Pro vykreslení pouze množiny bodů lze použít:

```
// jedna vykresovana posloupnost bodu
LineSeries mySeries = new LineSeries()
{
  // zadana posloupnost bod;
  Values = new ChartValues<int>() { 1, 2, 3, …, X },
  // parametry vykreseni
  Fill = Brushes.Transparent, LineSmoothness = 0, Stroke = Brushes.Red
};
// pridani posloupnosti bodu do grafu
myLiveCharts.Series.Add(mySeries);
```

Bude vykreslen graf složený ze zadaných bodů. Každý bod bude vodorovně umístěn na pozici dané jeho pořadím v seznamu. Podle ukázky v XAML by měl seznam obsahovat 20 bodů typu INT o rozsahu 0 až 100.

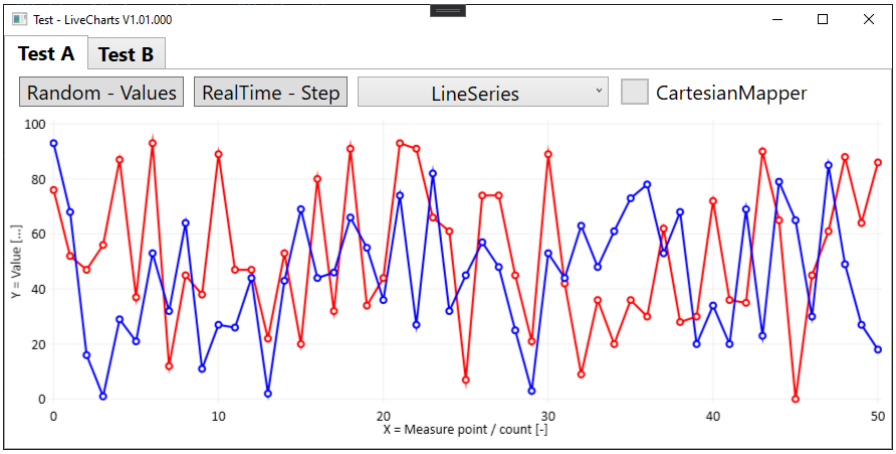
# 3 Základní typy grafů

Pokud se použije **CartesianChart** tak jsou dostupné tyto způsoby zobrazení.

**LineSeries** – Pouhý seznam bodů proložený čárou / křivkou. Jde v podstatě o běžný graf XY. Každý bod bude vodorovně umístěn na pozici dané jeho pořadím v seznamu. Podle ukázky v XAML by měl seznam obsahovat 20 bodů typu INT o rozsahu 0 až 100.

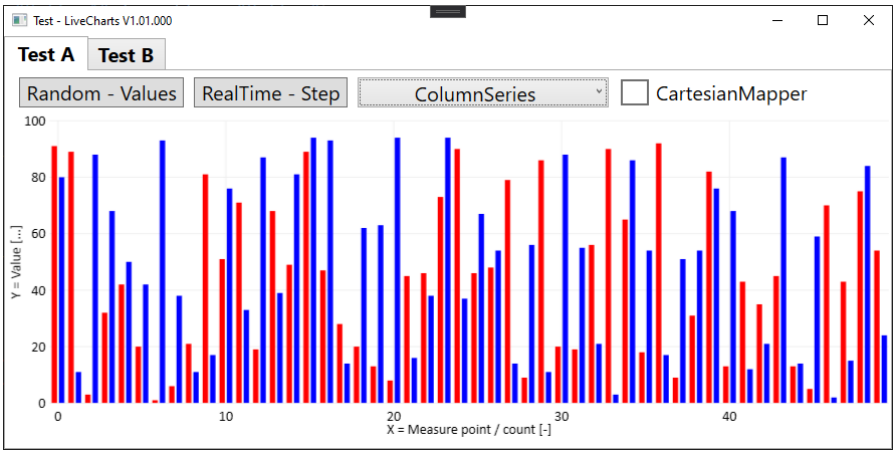
```
// jedna vykresovana posloupnost bodu
LineSeries mySeries = new LineSeries()
{
  // zadana posloupnost bod;
  Values = new ChartValues<int>() { 1, 2, 3, …, X },
  // parametry vykreseni
  Fill = Brushes.Transparent, LineSmoothness = 0, Stroke = Brushes.Red
};
// pridani posloupnosti bodu do grafu
myLiveCharts.Series.Add(mySeries);
```

Graf může obsahovat libovolný počet takto zadaných průběhů pomocí „myLiveCharts.Series.Add(…)“



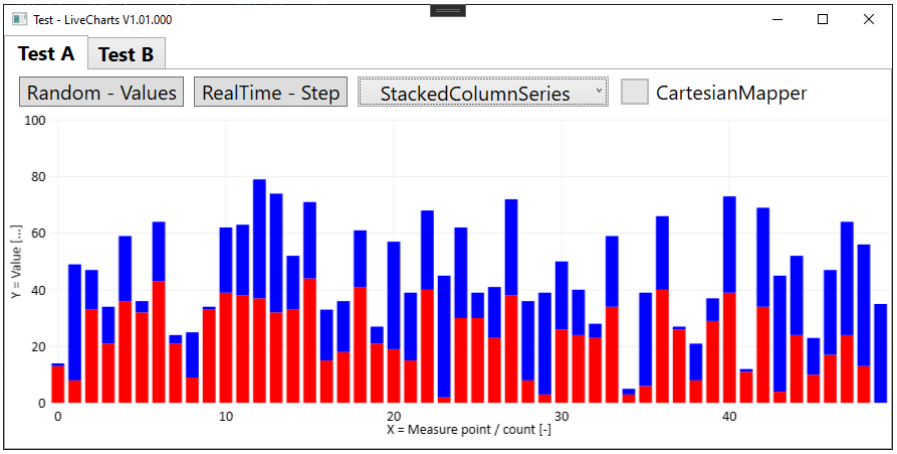

**ColumnSeries** – Hodnoty jsou zobrazeny jako sloupce (vždy od 0 do zadané hodnoty). Každý sloupec bude vodorovně umístěn na pozici dané jeho pořadím v seznamu. Podle ukázky v XAML by měl seznam obsahovat 20 bodů typu INT o rozsahu 0 až 100.

```
// jedna vykresovana posloupnost bodu
ColumnSeries mySeries = new ColumnSeries()
{
  // zadana posloupnost bod;
  Values = new ChartValues<int>() { 1, 2, 3, …, X },
  // parametry vykreseni
  Fill = Brushes.Red, Stroke = Brushes.Red
};
// pridani posloupnosti bodu do grafu
myLiveCharts.Series.Add(mySeries);
```

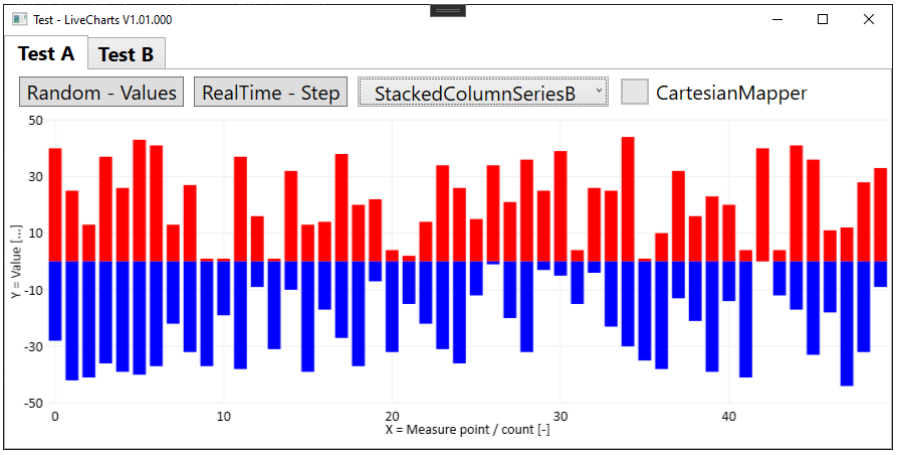



Pokud bude v grafu zadáno více „ColumnSeries“ (samozřejmě je vhodné je odlišnou barvou), tak budou sloupce jednotlivých barev umístěn za sebou (vedle sebe).

**StackedColumnSeries** – Jedná se o obdobu **ColumnSeries**. Hodnoty z jednotlivých číselných řad (tedy z StackedColumnSeries) nejsou umístěny vedle sebe (za sebou), ale nad sebou (pro příslušnou pozici X).




Podle obrázku. Hodnoty z první řady „StackedColumnSeries“ jsou červené a jsou umístěny dole. Hodnoty z druhé řady „StackedColumnSeries“ jsou modré a jsou umístěny nahoře. Tedy sedí nad hodnotami z první řady „StackedColumnSeries“. Pokud budou další řady „StackedColumnSeries“, tak budou umístěny ještě dále nad. Celková výška sloupce je tedy součtem hodnot v jednotlivých „StackedColumnSeries“ na odpovídající pozici v seznamu.
Pokud je „0“ grafu na ose Y uprostřed, tak lze do prvního „StackedColumnSeries“ vložit kladná čísla a ta vytvoří sloupce směrem nahoru (od „0“ do +). Do druhého „StackedColumnSeries“ vložit záporná čísla a ta vytvoří sloupce směrem dolu (od „0“ do –).




Pozor však na skutečnost, že sloupce pro druhou „StackedColumnSeries“ (modré) nejsou ve skutečnosti pouze od „0“ do „požadované záporné hodnoty“, ale jsou (zřejmě) od „konce / vrcholu kladného sloupce“ do „požadované záporné hodnoty“.
Tedy podle obrázku, červené hodnoty jsou od „0“ do „požadované kladné hodnoty“ a modré sloupce jsou od „konce / vrcholu tohoto sloupce“ (zadem) do „požadované záporné hodnoty“. Nová barva je vždy vzadu a proto nepřekrývá tu předchozí. (Ale toto nevím přesně / skutečně / určitě.)

Parametry vykresleni dat:

- **ChartValues** – Seznam bodů pro zobrazení. První bod je vlevo a poslední vpravo.
- **Stroke** – Barva křivky zobrazené v grafu.
- **Fill** – Jakou barvou bude vyplněna plocha pod křivkou grafu. Pokud je nastaveno „Transparent“, tak pod křivkou grafu nebude žádná barva.
- **LineSmoothness** – Zda jsou pouze čárové spojnice zobrazených bodů (0) nebo je křivka mezi zobrazenými body (nějak) interpolována / proložena (1). Míru interpolace udává velikost čísla od 0 do 1.
- **PointGeometrySize** – Velikost bodu. Pokud je „0“, tak nejsou body na čáře zobrazeny.

# 4 CartesianMapper

**CartesianMapper** slouží pro (nějakou) změnu zobrazení vstupních hodnot od výchozího nastavení. Zde je ukázka pro **ColumnSeries** (Je však zřejmě použitelný i pro jiné typy zobrazeni). Činnost je následující:

- Vytvořit **ColumnSeries** například:

```
// jedna vykresovana posloupnost bodu
ColumnSeries mySeries = new ColumnSeries()
{
  // zadana posloupnost bod;
  Values = new ChartValues<int>() { 1, 2, 3, …, X },
  // parametry vykreseni
  Fill = Brushes.Red, Stroke = Brushes.Red
};
// pridani posloupnosti bodu do grafu
myLiveCharts.Series.Add(mySeries);
```

- Jde o běžné zobrazení sloupců (jedna hodnota jeden sloupec). Nyní je požadavek zobrazit sloupce tak více červeně jak jsou vysoké. K tomuto se vytvoří **CartesianMapper**:

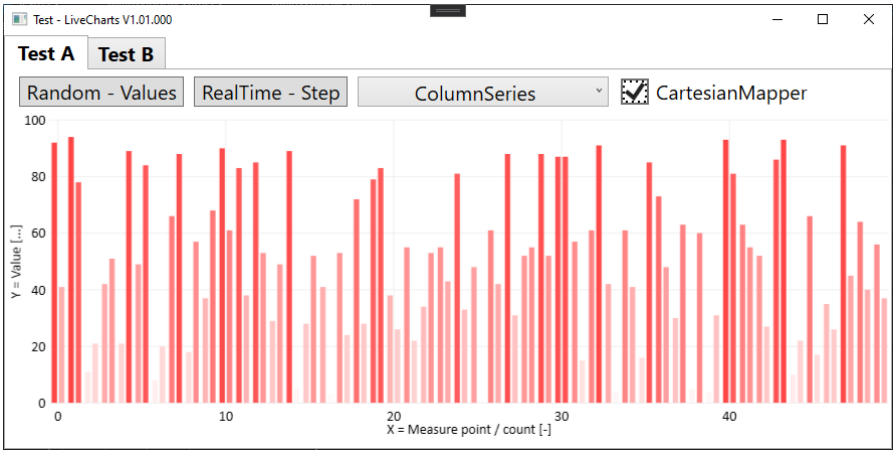
```
CartesianMapper<int> cartesianMapper = new CartesianMapper<int>()
       // pozice bude index
       .X((value, index) => index)
       // data jsou hodnota
       .Y((value) => value)
       // barva pro každý zobrazovany sloupec
       // cim vyssi hodnoty, tim vice cervene ('value' je pouze 0-100)
       .Fill((value, index) =>
       {
           return new SolidColorBrush(Color.FromRgb(
              (byte)255, (byte)(255 - value * 2), (byte)(255 - value * 2)));
       });
```

- Nakonec je potřeba **CartesianMapper** přiřadit k **ColumnSeries** a to následovně:

```
// spojení CartesianMapper přiřadit k ColumnSeries
mySeries.Configuration = cartesianMapper;
```

Případně přímo v konstruktoru pro **ColumnSeries**:

```
ColumnSeries mySeries = new ColumnSeries()
{
  // …
  Configuration = cartesianMapper
};
```




# 5 Osy

Graf může mít (zřejmě) i několik X a několik Y os. Pro jednoduchost je vhodné aspoň jednu osu X a Y definovat v XAML (pokud to lze). Například:

```
<!-- graf -->
<lvc:CartesianChart … >
	<!-- X - sirka / horizontalni (pocet bodu na sirka) -->
	<lvc:CartesianChart.AxisX>
		<lvc:Axis Title="X = Measure point / count [-]"
			DisableAnimations="True" FontSize="14" Unit="1"
			MaxValue="20" MinValue="0" Foreground="PaleVioletRed" />
	</lvc:CartesianChart.AxisX>
	<!-- Y - vyska / vertikalni (velikost hodnoty) -->
	<lvc:CartesianChart.AxisY>
		<lvc:Axis Title="Y = Weight [g]" FontSize="14" Unit="1"
			MaxValue="50" MinValue="-50"
			DisableAnimations="True" Foreground="PaleVioletRed" />
	</lvc:CartesianChart.AxisY>
</lvc:CartesianChart>
```

Pokud je potřeba rozsah některé osy kdykoli změnit v programovém kódu lze použít následující (nově zadané hodnoty se projeví okamžitě):

```
// nove hodnoty pro rozsah Y
myLiveCharts.AxisY[0].MinValue = 10; myLiveCharts.AxisY[0].MaxValue = 100;
```

Pokud je potřeba zadat vlastní popis osy X. Lze postupovat takto:

- Definovat vlastní popis pomocí pole řetězců, například:

  ```
  // popisky na osu grafu (null – pradne popisky)
  string[] myLabels = new [] { null, null, "1",null, null, null, "2",
  	null, null, null, "3", null, null, null, "4", null, null, null, "5" };
  ```

  Položky kde je **null**, tak nejsou jako popisky zobrazeny. Takto lze zobrazit pouze popisky na požadovaných místech a ostatní snadno skrýt. Přiřadit soubor popisků do grafu:

  ```
  // prirazeni souboru popisku k ose grafu
  myLiveCharts.AxisX[0].Labels = myLabels;
  ```

Pozor na skutečnost, že graf zobrazuje pouze tolik popisků (zejména na vodorovné ose X), aby byly čitelné, tedy aby se nepřekrývaly. Lze, však stanovit s jakým krokem se budou popisky zobrazovat a to pomocí:

```
// bude se zobrazit pouze každý paty popisek
myLiveCharts.AxisX[0].Separator.Step = 5;
```

Tedy zobrazí každý pátý popisek. Bohužel, i pokud jsou nějaké popisky zadány jako **null**, tak se jsou v podstatě platné. Pokud jsou zadané popisky dostatečně daleko od sebe a proloženy několika **null**, může být požadováno zobrazení každého platného popisku. V tomto případě je nutno nastavit:

```
// bude se vždy zobrazovat každý popisek
myLiveCharts.AxisX[0].Separator.Step = 1;
```

# 6 Data ve vlastní třídě

Pro zobrazení lze použít i data umístěná přímo ve vlastní třídě. Není je tedy nutno přepisovat / vkládat do jiné. Pro tuto skutečnost je potřeba využít **CartesianMapper**. Postup je následující:

- Mám vlastní třídu pro data, například:

```
public class MeasureData
{
 public int Pozice { get; set; } = 0;
 public int Hodnota { get; set; } = 0;
}
```

- Definici pro **LineSeries** vytvoříme běžně takto:

```
// jedna vykresovana posloupnost bodu
LineSeries mySeries = new LineSeries()
{
  // zadana posloupnost bod;
  Values = new ChartValues<MeasureData>(),
  // parametry vykreseni
  Fill = Brushes.Transparent, LineSmoothness = 0, Stroke = Brushes.Red
};
// pridani posloupnosti bodu do grafu
myLiveCharts.Series.Add(mySeries);
```

- **LineSeries** však není schopna vyzvedávat data přímo z **MeasureData** (neví které položky ze třídy). Z tohoto důvodu je potřeba vytvořit **CartesianMapper**. Například následovně:

```
  var mapper = Mappers.Xy<MeasureData>()
     .X(item => item.Pozice)      // toto je index pro osu X
     .Y(item => item.Hodnota);    // toto je hodnota pro osu Y
```

- Samozřejmě je nutno přiřadit **mapper** k **LineSeries**:

```
mySeries.Configuration = mapper;
```

Poznámky:

- Nevím proč je zde **Mappers.Xy<…>** místo **CartesianMapper<…>** (možná je to stejné). Pokud někdo zjistí, ať dá vědět (ale pouze nějaké celistvé vysvětlení).

https://stackoverflow.com/questions/43727680/using-livecharts-constant-changes-example-to-plot-points-based-on-time-and-read

# 7 Vytvoření Time-OnLine zobrazení

Pokud je potřeba snadno zobrazovat data OnLine, tedy tak jak odněkud přicházejí, zle použít velmi jednoduché řešení:

```
// zacatek aktualizace prubehu
mySeries.OnSeriesUpdateStart();
// odstraneni prvních tolika hodnot aby jich zbylo „počet do grafu – 1“
while(mySeries.Values.Count > (50-1)) { mySeries.Values.RemoveAt(0); }
// pridani nove hodnoty na konec prubehu (vpravo)
mySeries.Values.Add(value);
// konec aktualizace prubehu
mySeries.OnSeriesUpdatedFinish();
```

Tímto stylem se vždy odstraní první hodnota v grafu (ta vlevo), ostatní se v podstatě sesunou doleva, tedy k začátku grafu a nakonec (vpravo) se přidá nová hodnota. Graf působí dojmem posuvu zprava doleva a je tedy časově ubíhající. Pořadová čísla na ose X neubíhají, jsou stále stejná, udávají tedy kolik je právě zobrazeno (posledních) vzorků.
(Nezkoušel jsem žádnou velkou rychlost, takže graf samozřejmě nemusí desítky vzorků za vteřinu vůbec zvládat.)

# 8 Vodorovná čára v grafu

Občas je potřeba vytvořit v grafu vodorovnou (jako osa X) čárku signalizující nějakou mezní hodnotu. Jako nejjednodušší řešení je vytvořit **LineSeries** a naplnit ji body o požadované hodnotě přes celou šířku grafu. Tohoto lze však dosáhnout i jinak:

- Vytvořit **LineSeries** obsahující pouze dvě (v podstatě většinou stejné hodnoty) hodnoty:
  - Hodnoty pro začátek této čáry
  - Hodnotu pro konec této čáry
- Vytvořit **Mapper**, který první hodnotu umístí na pozici 0 a druhou hodnotu umístí na pozici konce grafu v ose X. Mapper bude například následovný:

```
// 'Mapper' pro vodorovne cary pres celou sirku grafu
ownA_HorLineMapper = new CartesianMapper<int>()
       // (cara obsahuje pouze dva body / hodnoty s 'index = 0' a 'index = 1')
       // (prvni na pozici '0' (0 * index), druhy na konci grafu (max * index))
       .X((value, index) => index * (int)myLiveCharts.AxisX[0].MaxValue)
       // první a druha hodnota je beze zmeny
       .Y((value) => value);
```

- Nyní spojit **Mapper** s **LineSeries** a naplnit do **LineSeries** dvě hodnoty pro začátek a konec čára.

# 9 Testovací aplikace

K dispozici je testovací aplikace s názvem **TestLiveCharts V01.01.000**:

- Záložky **TestA**,**TestB**, … jsou pro zde uvedené příklady.
- Záložky **OwnA**, **OwnB**, … jsou pro (nějaké) vlastní testy.

**TestA** – Některé vzorové příklady z toho dokumentu.

**OwnA** – Zkouška vytvoření grafu obsahující tři barevné (červený, zelený, modrý) sloupce v rozsahu -/+ jako jednu sadu dat a mezi nimi vždy mezera. Například pro zobrazení rozsahu nějakých tří hodnot kolem nuly a toto opakovaně (s mezerou mezi zobrazeními skupinami). Stručný popis:

- Vytvořeno šest **StackedComunSeries**. Vždy dvojice pro každý sloupec, tedy horní (od nuly nahoru) a dolní (od nuly dolu) část.
- Vytvořeny tři **Mapper** zajišťující vložení hodnoty z příslušné **StackedComunSeries** na požadovanou pozici v grafu. Vždy jeden **Mapper** pro dvě **StackedComunSeries** umístěné na jedné pozici v grafu (v podstatě nad sebou, jedna nahoru a druhá dolu). Parametry jsou:
    - Data se zobrazují od pozice „1“ (nikoli „0“ na které graf ve skutečnosti začíná)
    - Každá dvojice **StackedComunSeries** je vždy o jednu pozici v ose X posunuta vpravo
    - Krok pro zobrazení je „x4“ (tři sloupce + mezera)
- Jedna sada dat pro zobrazení (6 hodnot, 3 dvojice hodnot nahoru / dolu) se vždy vloží jako jedna sada dat do šesti **StackedComunSeries**. Tedy vždy jedna hodnota do každé z **StackedComunSeries** (pro každou část sloupce v grafu, graf má v podstatě 6 sloupců, tedy vždy 3 dvojice sloupců nad sebou).