KAPITOLA

17

Tisk a dialogová okna

Prohlídka oboru názvů *System.Printing* je jen pro otrlé. Naleznete zde třídy související s ovladači tiskáren, tiskovými frontami, tiskovými servery a tiskovými úlohami. V souvislosti s tiskem naštěstí platí, že většina aplikací může převážnou část tříd v oboru názvů *System.Printing* bezpečně ignorovat. Logika tisku vašich programů se pravděpodobně bude soustřeďovat kolem třídy *PrintDialog*, která je definována v oboru názvů *System.Windows.Controls*.

Hlavní třída, kterou budete potřebovat z oboru názvů *System.Printing*, se nazývá *PrintTicket*. Kvůli použití této třídy se ve svých projektech také budete odkazovat na sestavení ReachFramework.dll. Třída je kromě toho užitečná i pro programy, které tisknou nebo spravují proměnnou typu *PrintQueue*. Tuto třídu naleznete také v oboru názvů *System.Printing*, kde však pochází ze sestavení System.Printing.dll. Další třídy týkající se tisku jsou definovány v oboru názvů *System.Windows.Documents*.

Třída *PrintDialog* samozřejmě zobrazuje dialogové okno, ale obsahuje také metody umožňující tisk jedné stránky nebo vícestránkového dokumentu. Obsah tištěné stránky představuje v obou případech objekt typu *Visual*. Jak již víte, mezi důležité potomky třídy *Visual* patří třída *UIElement*. To znamená, že můžete tisknout instanci libovolné třídy, která se odvozuje od třídy *FrameworkElement*, včetně panelů, ovládacích prvků a jiných prvků uživatelského rozhraní. Lze například vytvořit panel *Canvas* nebo jiný panel, umístit na něj různé ovládací či jiné prvky nebo tvary a potom celý panel vytisknout.

Tisk panelu sice zdánlivě poskytuje značnou pružnost, ale jednodušší přístup k tisku využívá třídu *DrawingVisual*, která také pochází od třídy *Visual*. Třídu *DrawingVisual* jsme si představili ve třídě *ColorCell*, která je součástí projektu *SelectColor* z kapitoly 11. Třída *DrawingVisual* obsahuje metodu s názvem *RenderOpen*, která vrací objekt typu *DrawingContext*. Volání metod třídy *DrawingContext* (která jsou zakončena voláním metody *Close*) umožňují uložit grafiku v objektu *DrawingVisual*.

Následující program PrintEllipse patří k nejjednodušším programům pro tisk, které si lze představit. Program s možností tisku by měl obsahovat nějakou součást, která tisk inicializuje. V tomto případě se jedná o tlačítko. Když na tlačítko klepnete, program vytvoří objekt typu *PrintDialog* a zobrazí jej. V tomto dialogovém okně můžete vybrat tiskárnu (máte-li více než jednu) a případně změnit některá nastavení tiskárny. Když potom dialogové okno *PrintDialog* zavřete klepnutím na tlačítko Tisk, začne program připravovat vlastní tisk.

PrintEllipse.cs

```
//-----------------------------------------
// PrintEllipse.cs (c) 2006 by Charles Petzold
//-----------------------------------------
using System;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Input;
using System.Windows.Media;

namespace Petzold.PrintEllipse
{
    public class PrintEllipse : Window
    {
        [STAThread]
        public static void Main()
        {
            Application app = new Application();
            app.Run(new PrintEllipse());
        }
        public PrintEllipse()
        {
            Title = "Tisk elipsy";
            FontSize = 24;

            // Vytvoří panel StackPanel jako obsah okna.
            StackPanel stack = new StackPanel();
            Content = stack;

            // Vytvoří objekt Button pro tisk.
            Button btn = new Button();
            btn.Content = "_Tisk...";
            btn.HorizontalAlignment = HorizontalAlignment.Center;
            btn.Margin = new Thickness(24);
            btn.Click += PrintOnClick;
            stack.Children.Add(btn);
        }
        void PrintOnClick(object sender, RoutedEventArgs args)
        {
            PrintDialog dlg = new PrintDialog();

            if (dlg.ShowDialog().GetValueOrDefault())
            {
                // Vytvoří objekt DrawingVisual a otevře DrawingContext.
                DrawingVisual vis = new DrawingVisual();
                DrawingContext dc = vis.RenderOpen();

                // Vykreslí elipsu.
                dc.DrawEllipse(Brushes.LightGray, new Pen(Brushes.Black, 3),
```

```
                                    new Point(dlg.PrintableAreaWidth / 2,
                                              dlg.PrintableAreaHeight / 2),
                                    dlg.PrintableAreaWidth / 2,
                                    dlg.PrintableAreaHeight / 2);

                        // Zavře DrawingContext.
                        dc.Close();

                        // Nakonec vytiskne stránku.
                        dlg.PrintVisual(vis, "Moje první tisková úloha");
                    }
                }
            }
        }
```

Třída *ShowDialog* ze své definice vrací nulovatelný typ *bool*. V případě tohoto dialogového okna vrací třída *ShowDialog* hodnotu *true*, pokud uživatel klepne na tlačítko Tisk, hodnotu *false*, jestliže klepne na tlačítko Storno, a hodnotu *null* v případě, že uživatel dialogové okno zavře klepnutím na červené tlačítko Zavřít v pravém horním rohu v záhlaví okna. Volání *GetValueOrDefault* převede vrácenou hodnotu *null* na hodnotu *false*, takže je výsledkem příkazu *if* vždy hodnota typu *bool*.

Pokud metoda *ShowDialog* vrátí hodnotu *true*, program vytvoří objekt typu *DrawingVisual*, zavolá metodu *RenderOpen*, která vrátí objekt *DrawingContext*, a potom pro objekt *DrawingContext* zavolá metody *DrawEllipse* i *Close*. Nakonec je objekt *DrawingVisual* předán metodě *PrintVisual* okna *PrintDialog*. Tisková úloha je přitom v tiskové frontě identifikována textovým řetězcem.

Všimněte si, že argumenty metody *DrawEllipse* odkazují na dvě vlastnosti, které jsou definovány třídou *PrintDialog*. Nazývají se (ne zcela přesně) *PrintableAreaWidth* a *PrintableAreaHeight*. Když stránku vytištěnou programem PrintEllipse prozkoumáte, pravděpodobně zjistíte, že části elipsy byly oříznuty, protože sahají do netisknutelných částí stránky. Vlastnosti *PrintableAreaWidth* a *PrintableAreaHeight* se nevztahují k tisknutelné oblasti stránky, ale určují celkovou fyzickou velikost stránky v jednotkách nezávislých na zařízení (1/96 palce, tj. 0,26 mm).

V dolní části dialogové okna Tisk naleznete čtyři přepínače označené Vše, Výběr, Aktuální stránka a Stránky. Tyto přepínače umožňují nastavit, která část dokumentu se vytiskne: celý dokument, aktuální výběr, aktuální stránka nebo rozsah stránek zadaný uživatelem. Všechny přepínače kromě prvního jsou standardně zobrazeny šedě. Přepínače Výběr a Aktuální stránka nelze v první verzi grafického subsystému Windows Presentation Foundation povolit. Přepínač Stránky lze povolit nastavením vlastnosti *UserPageRangeEnabled* na hodnotu *true*.

Můžete inicializovat výchozí přepínač a později získat uživatelskou volbu pomocí vlastnosti *PageRangeSelection*, která se rovná jednomu z členů výčtu *PageRangeSelection*: *AllPages* nebo *UserPages*. Jestliže vlastnost *PageRangeSelection* nabývá hodnoty *PageRangeSelection.UserPages* (odpovídá přepínači s názvem Stránky), obsahuje vlastnost *PageRanges* dialogového okna *PrintDialog* kolekci objektů typu *PageRange*, což je struktura s vlastnostmi *PageFrom* a *PageTo*.

Součástí dialogového okna Tisk je také pole Počet kopií. Zadáte-li do tohoto pole číslo větší než 1, vytiskne metoda *PrintVisual* více kopií.

Dialogové okno Tisk dále obsahuje tlačítko s názvem Předvolby tisku, které dovoluje vyvolat okno se stránkami vlastností pro konkrétní tiskárnu. Vyberete-li na těchto stránkách jinou velikost tiskové stránky, můžete si na tiskovém výstupu z programu všimnout, že se velikost stránky

odráží ve vlastnostech *PrintableAreaWidth* a *PrintableAreaHeight*. Je také možné změnit orientaci stránky na šířku, ale v tomto programu se uvedené nastavení příliš neprojeví.

Uživatelem požadovaná velikost stránky, počet kopií a mnoho dalších informací se společně ukládají do vlastnosti třídy *PrintDialog*, která se nazývá *PrintTicket* a má typ *PrintTicket*. Tato třída je definována v oboru názvů *System.Printing* a nachází se v sestavení ReachFramework. Chcete-li pracovat se třídou *PrintTicket*, musíte uvést odkaz na sestavení ReachFramework.dll.

Pokud vyvoláte dialogové okno *PrintDialog* v programu PrintEllipse, změníte nastavení (např. orientaci stránky), spustíte tisk a potom dialogové okno *PrintDialog* otevřete znovu, zjistíte, že byla obnovena výchozí nastavení. Slušné moderní programy obvykle uchovávají uživatelské volby (např. orientaci stránky) pro další tiskové úlohy. Chcete-li této vlastnosti dosáhnout ve svém programu, můžete definovat proměnnou typu *PrintTicket*.

```
PrintTicket prntkt;
```

Složka *prntkt* má samozřejmě po vytvoření hodnotu *null*. Když v rámci přípravy na zobrazí dialogového okna vytvoříte objekt *PrintDialog*, nastavíte z této proměnné jeho vlastnost *PrintTicket* – avšak pouze za předpokladu, že se proměnná nerovná *null*:

```
PrintDialog dlg = new PrintDialog();

if (prntkt != null)
dlg.PrintTicket = prntkt;
```

Jestliže uživatel v tomto dialogu klepne na tlačítko Tisk, je nutné do proměnné uložit objekt *PrintTicket* založený na nastavení okna:

```
if (dlg.ShowDialog().GetValueOrDefault())
{
    prntkt = dlg.PrintTicket;
    ...
}
```

Díky tomuto nastavování a načítání objektu *PrintTicket* je zajištěno, že uživatelské volby zůstanou zachovány. Pokud uživatel otevře dialogové okno Tisk, nastaví orientaci stránky na hodnotu Na šířku, klepne na tlačítko OK a potom otevře dialogové okno Tisk znovu, bude i nadále nastavena orientace Na šířku.

Jestliže má uživatel nainstalováno více tiskáren, zobrazují se v poli se seznamem *ComboBox* v horní části dialogového okna Tisk. Vybere-li uživatel v dialogu Tisk jinou než výchozí tiskárnu, pravděpodobně budete chtít tuto změnu rovněž uložit a zobrazit stejnou tiskárnu i při příštím zobrazení dialogového okna Tisk. Můžete toho dosáhnout tak, že uložíte vlastnost *PrintQueue* dialogového okna jako proměnnou typu *PrintQueue* a při zobrazení dialogového okna nastavíte danou vlastnost z této proměnné. Kvůli použití třídy *PrintQueue* je nutné uvést odkaz na sestavení System.Printing.

Třída *PrintDialog* zatím neposkytuje možnost uživatelského nastavení okrajů stránky. Nyní tedy k tomuto účelu vytvoříme speciální dialogové okno. Třídy dialogových oken se odvozují od třídy *Window* a velmi připomínají jiné třídy založené na třídě *Window*. Liší se však několika rozdíly.

Konstruktor dialogového okna by měl nastavit vlastnost *ShowInTaskbar* na hodnotu *false*. Velikost dialogového okna je často určena podle jeho obsahu a vlastnost *ResizeMode* je nastavena

na hodnotu *NoResize*. Vlastnost *WindowStyle* lze podle vlastního uvážení nastavit na hodnotu *SingleBorderWindow* či *ToolWindow*. V současnosti se již považuje za chybu, pokud program zobrazí dialogové okno bez záhlaví.

Dialogové okno lze snadno umístit relativně k jeho vlastníkovi nastavením vlastnosti *WindowStartupLocation* na hodnotu *CenterOwner*. I když ponecháte výchozí nastavení vlastnosti *WindowStartupLocation* (které ponechá umístění dialogového okna na systému Windows), musíte nastavit vlastnost *Owner* dialogového okna na objekt *Window*, který dialog vyvolal. Kód obvykle vypadá takto:

```
MyDialog dlg = new MyDialog();
dlg.Owner = this;
```

Pokud zobrazíte dialogové okno s vlastníkem *null* a uživatel se přepne do jiné aplikace, může se dialogové okno nakonec ocitnout *za* oknem, které jej vyvolalo. Okno aplikace by však ani nadále nemohlo reagovat na uživatelský vstup. Tato situace rozhodně není v pořádku.

Třída *CommonDialog* definovaná v oboru názvů *Microsoft.Win32* (od které se odvozují třídy *OpenFileDialog* a *SaveFileDialog*) obsahuje definici metody *ShowDialog*, která vyžaduje argument typu *Owner*. To je jedno možné řešení. Případně lze argument *Owner* vyžadovat v konstruktoru třídy dialogového okna. Zvolíte-li tento postup, získáte více možností při umisťování dialogového okna. Předpokládejme například, že chcete umístit dialogové okno s posunem jednoho palce oproti levému hornímu rohu okna aplikace. Kód konstruktoru bude mít jednoduchý tvar

```
public MyDialog(Window owner)
{
    Left = 96 + owner.Left;
    Top = 96 + owner.Top;
    ...
}
```

Dialogové okno téměř vždy definuje alespoň jednu veřejnou vlastnost pro informace, které okno získává od uživatele a předává aplikaci. Můžete dokonce definovat speciální třídu či strukturu, která bude obsahovat všechny údaje poskytované příslušným dialogovým oknem. V definici této vlastnosti inicializuje přístupová metoda *set* ovládací prvky dialogového okna na základě hodnoty vlastnosti. Přístupová metoda *get* získá vlastnosti z ovládacích prvků.

Dialogové okno obsahuje tlačítka OK a Storno, i když může být tlačítko OK ve skutečnosti označeno jako Otevřít, Uložit nebo Tisk. Vlastnost *IsDefault* tlačítka OK je nastavena na hodnotu *true*. Uživatel tedy může tlačítko stisknout, když v aktivním okně stiskne klávesu Enter. Tlačítko Storno má svou vlastnost *IsCancel* nastavenu na hodnotu *true*, takže lze místo klepnutí na toto tlačítko stisknout klávesu Escape.

Pro tlačítko Storno není nutné poskytovat obslužnou rutinu události *Click*. Dialogové okno je zavřeno automaticky na základě nastavení vlastnosti *IsCancel* na hodnotu *true*. Obslužná rutina *Click* tlačítka OK nemusí provést jinou akci než nastavit vlastnost *DialogResult* na hodnotu *true*. Tím se dialogové okno uzavře a metoda *ShowDialog* vrátí hodnotu *true*.

Třída *PageMarginsDialog* obsahuje čtyři ovládací prvky *TextBox*, které uživateli umožňují nastavit levý, pravý, horní a dolní okraj stránky v palcích. Třída definuje jedinou veřejnou vlastnost s názvem *PageMargins* typu *Thickness*, která převádí hodnoty mezi palci a jednotkami nezávislými na zařízení. Přístupová metoda *get* vlastnosti *PageMargins* může voláním metody *Double*.

Parse bez rizika převést obsah čtyř ovládacích prvků *TextBox* na čísla, protože třída deaktivuje tlačítko OK, dokud všechna pole neobsahují platné hodnoty.

PageMarginsDialog.cs

```
//------------------------------------------------
// PageMarginsDialog.cs (c) 2006 by Charles Petzold
//------------------------------------------------
using System;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Controls.Primitives;

namespace Petzold.PrintWithMargins
{
    class PageMarginsDialog : Window
    {
        // Interní výčet odkazuje na strany papíru.
        enum Side
        {
            Left, Right, Top, Bottom
        }

        // Čtyři ovládací prvky TextBox pro vstup čísel.
        TextBox[] txtbox = new TextBox[4];
        Button btnOk;

        // Veřejná vlastnost typu Thickness pro okraje stránky.
        public Thickness PageMargins
        {
            set
            {
                txtbox[(int)Side.Left].Text = (value.Left / 96).ToString("F3");
                txtbox[(int)Side.Right].Text =
                                        (value.Right / 96).ToString("F3");
                txtbox[(int)Side.Top].Text = (value.Top / 96).ToString("F3");
                txtbox[(int)Side.Bottom].Text =
                                        (value.Bottom / 96).ToString("F3");
            }
            get
            {
                return new Thickness(
                            Double.Parse(txtbox[(int)Side.Left].Text) * 96,
                            Double.Parse(txtbox[(int)Side.Top].Text) * 96,
                            Double.Parse(txtbox[(int)Side.Right].Text) * 96,
                            Double.Parse(txtbox[(int)Side.Bottom].Text) * 96);
            }
        }

        // Konstruktor.
        public PageMarginsDialog()
        {
            // Standardní nastavení pro dialogová okna.
            Title = "Vzhled stránky";
            ShowInTaskbar = false;
            WindowStyle = WindowStyle.ToolWindow;
            WindowStartupLocation = WindowStartupLocation.CenterOwner;
            SizeToContent = SizeToContent.WidthAndHeight;
```

```
ResizeMode = ResizeMode.NoResize;

// Nastaví panel StackPanel jako obsah okna.
StackPanel stack = new StackPanel();
Content = stack;

// Nastaví objekt GroupBox jako podřízenou položku panelu
// StackPanel.
GroupBox grpbox = new GroupBox();
grpbox.Header = "Okraje (v palcích)";
grpbox.Margin = new Thickness(12);
stack.Children.Add(grpbox);

// Nastaví panel Grid jako obsah objektu GroupBox.
Grid grid = new Grid();
grid.Margin = new Thickness(6);
grpbox.Content = grid;

// Dva řádky a čtyři sloupce.
for (int i = 0; i < 2; i++)
{
    RowDefinition rowdef = new RowDefinition();
    rowdef.Height = GridLength.Auto;
    grid.RowDefinitions.Add(rowdef);
}

for (int i = 0; i < 4; i++)
{
    ColumnDefinition coldef = new ColumnDefinition();
    coldef.Width = GridLength.Auto;
    grid.ColumnDefinitions.Add(coldef);
}

// Umístí na panel Grid ovládací prvky Label a TextBox.
for (int i = 0; i < 4; i++)
{
    Label lbl = new Label();
    lbl.Content = "_" + Enum.GetName(typeof(Side), i) + ":";
    lbl.Margin = new Thickness(6);
    lbl.VerticalAlignment = VerticalAlignment.Center;
    grid.Children.Add(lbl);
    Grid.SetRow(lbl, i / 2);
    Grid.SetColumn(lbl, 2 * (i % 2));

    txtbox[i] = new TextBox();
    txtbox[i].TextChanged += TextBoxOnTextChanged;
    txtbox[i].MinWidth = 48;
    txtbox[i].Margin = new Thickness(6);
    grid.Children.Add(txtbox[i]);
    Grid.SetRow(txtbox[i], i / 2);
    Grid.SetColumn(txtbox[i], 2 * (i % 2) + 1);
}

// Použije panel UniformGrid pro tlačítka OK a Storno.
UniformGrid unigrid = new UniformGrid();
unigrid.Rows = 1;
unigrid.Columns = 2;
stack.Children.Add(unigrid);
```

```
            btnOk = new Button();
            btnOk.Content = "OK";
            btnOk.IsDefault = true;
            btnOk.IsEnabled = false;
            btnOk.MinWidth = 60;
            btnOk.Margin = new Thickness(12);
            btnOk.HorizontalAlignment = HorizontalAlignment.Center;
            btnOk.Click += OkButtonOnClick;
            unigrid.Children.Add(btnOk);

            Button btnCancel = new Button();
            btnCancel.Content = "Storno";
            btnCancel.IsCancel = true;
            btnCancel.MinWidth = 60;
            btnCancel.Margin = new Thickness(12);
            btnCancel.HorizontalAlignment = HorizontalAlignment.Center;
            unigrid.Children.Add(btnCancel);
        }
        // Povolí tlačítko OK, pouze pokud ovládací prvky TextBox obsahují
        // číselné hodnoty.
        void TextBoxOnTextChanged(object sender, TextChangedEventArgs args)
        {
            double result;

            btnOk.IsEnabled =
                Double.TryParse(txtbox[(int)Side.Left].Text, out result) &&
                Double.TryParse(txtbox[(int)Side.Right].Text, out result) &&
                Double.TryParse(txtbox[(int)Side.Top].Text, out result) &&
                Double.TryParse(txtbox[(int)Side.Bottom].Text, out result);
        }
        // Zavře dialogové okno po klepnutí na tlačítko OK.
        void OkButtonOnClick(object sender, RoutedEventArgs args)
        {
            DialogResult = true;
        }
    }
}
```

Třída *PageMarginsDialog* je součástí projektu PrintWithMargins, který obsahuje i následující soubor. Tento program ukazuje, jak lze pomocí objektů *PrintQueue* a *PrintTicket* ukládat nastavení dialogového okna a znovu je použít při jeho dalším zobrazení. Dialog *PageMarginsDialog* se zobrazí po klepnutí na tlačítko s popisem Vzhled stránky.

PrintWithMargins.cs

```
//-----------------------------------------------
// PrintWithMargins.cs (c) 2006 by Charles Petzold
//-----------------------------------------------
using System;
using System.Globalization;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Input;
using System.Windows.Media;
using System.Printing;

namespace Petzold.PrintWithMargins
{
```

```
public class PrintWithMargins : Window
{
    // Soukromé proměnné k uložení informací z dialogu PrintDialog.
    PrintQueue prnqueue;
    PrintTicket prntkt;
    Thickness marginPage = new Thickness(96);

    [STAThread]
    public static void Main()
    {
        Application app = new Application();
        app.Run(new PrintWithMargins());
    }
    public PrintWithMargins()
    {
        Title = "Tisk s okraji";
        FontSize = 24;

        // Vytvoří panel StackPanel jako obsah okna.
        StackPanel stack = new StackPanel();
        Content = stack;

        // Vytvoří objekt Button pro Vzhled stránky.
        Button btn = new Button();
        btn.Content = "Vzhled _stránky...";
        btn.HorizontalAlignment = HorizontalAlignment.Center;
        btn.Margin = new Thickness(24);
        btn.Click += SetupOnClick;
        stack.Children.Add(btn);

        // Vytvoří tlačítko Tisk.
        btn = new Button();
        btn.Content = "_Tisk...";
        btn.HorizontalAlignment = HorizontalAlignment.Center;
        btn.Margin = new Thickness(24);
        btn.Click += PrintOnClick;
        stack.Children.Add(btn);
    }
    // Tlačítko Vzhled stránky: vyvolá dialog PageMarginsDialog.
    void SetupOnClick(object sender, RoutedEventArgs args)
    {
        // Vytvoří dialog a inicializuje vlastnost PageMargins.
        PageMarginsDialog dlg = new PageMarginsDialog();
        dlg.Owner = this;
        dlg.PageMargins = marginPage;

        if (dlg.ShowDialog().GetValueOrDefault())
        {
            // Uloží okraje stránky z dialogového okna.
            marginPage = dlg.PageMargins;
        }
    }
    // Tlačítko Tisk: vyvolá dialog PrintDialog.
    void PrintOnClick(object sender, RoutedEventArgs args)
    {
        PrintDialog dlg = new PrintDialog();

        // Nastaví objekty PrintQueue a PrintTicket podle proměnných.
        if (prnqueue != null)
            dlg.PrintQueue = prnqueue;
```

```
            if (prntkt != null)
                dlg.PrintTicket = prntkt;

            if (dlg.ShowDialog().GetValueOrDefault())
            {
                // Uloží objekty PrintQueue a PrintTicket z dialogového okna.
                prnqueue = dlg.PrintQueue;
                prntkt = dlg.PrintTicket;

                // Vytvoří objekt DrawingVisual a otevře DrawingContext.
                DrawingVisual vis = new DrawingVisual();
                DrawingContext dc = vis.RenderOpen();
                Pen pn = new Pen(Brushes.Black, 1);

                // Obdélník definuje stránku bez okrajů.
                Rect rectPage = new Rect(marginPage.Left, marginPage.Top,
                                    dlg.PrintableAreaWidth -
                                        (marginPage.Left + marginPage.Right),
                                    dlg.PrintableAreaHeight -
                                        (marginPage.Top + marginPage.Bottom));

                // Vykreslí obdélník, který určuje okraje nastavené
                // uživatelem.
                dc.DrawRectangle(null, pn, rectPage);

                // Vytvoří formátovaný textový objekt, který zobrazí
                // vlastnosti PrintableArea.
                FormattedText formtxt = new FormattedText(
                    String.Format("Ahoj, tiskárno! {0} x {1}",
                                  dlg.PrintableAreaWidth / 96,
                                  dlg.PrintableAreaHeight / 96),
                    CultureInfo.CurrentCulture,
                    FlowDirection.LeftToRight,
                    new Typeface(new FontFamily("Times New Roman"),
                                 FontStyles.Italic, FontWeights.Normal,
                                 FontStretches.Normal),
                    48, Brushes.Black);

                // Získá fyzickou velikost formátovaného textového řetězce.
                Size sizeText = new Size(formtxt.Width, formtxt.Height);

                // Vypočítá bod k zarovnání textu na střed v rámci okrajů.
                Point ptText =
                    new Point(rectPage.Left +
                                    (rectPage.Width - formtxt.Width) / 2,
                              rectPage.Top +
                                    (rectPage.Height - formtxt.Height) / 2);

                // Vykreslí text a okolní obdélník.
                dc.DrawText(formtxt, ptText);
                dc.DrawRectangle(null, pn, new Rect(ptText, sizeText));

                // Zavře DrawingContext.
                dc.Close();

                // Nakonec vytiskne stránky (stránky).
                dlg.PrintVisual(vis, Title);
            }
        }
    }
}
```

Tento program vytiskne obdélník, který je dán uživatelsky nastavenými okraji stránky, a v rámci těchto okrajů vystředí text, který je také umístěn do obdélníka. Text obsahuje slova „Ahoj, tiskárno!“, po kterých následují rozměry stránky v palcích.

Vzhledem k tomu, že třída *PrintVisual* vytiskne objekt typu *Visual*, můžete vytvořit stránku uspořádáním prvků a ovládacích prvků na panel, např. panel *Canvas*. Tento postup můžete aplikovat i na grafiku z knihovny Shapes a text přitom zobrazit v poli *TextBlock*.

Následující program s názvem PrintaBunchaButtons vytiskne panel *Grid* s pozadím gradientního štětce, na kterém je uspořádáno 25 tlačítek různých velikostí.

PrintaBunchaButtons.cs

```
//-----------------------------------------------------
// PrintaBunchaButtons.cs (c) 2006 by Charles Petzold
//-----------------------------------------------------
using System;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Input;
using System.Windows.Media;

namespace Petzold.PrintaBunchaButtons
{
    public class PrintaBunchaButtons : Window
    {
        [STAThread]
        public static void Main()
        {
            Application app = new Application();
            app.Run(new PrintaBunchaButtons());
        }
        public PrintaBunchaButtons()
        {
            Title = "Tisk spousty tlačítek";
            SizeToContent = SizeToContent.WidthAndHeight;
            ResizeMode = ResizeMode.CanMinimize;

            // Vytvoří tlačítko 'Tisk'.
            Button btn = new Button();
            btn.FontSize = 24;
            btn.Content = "Tisk ...";
            btn.Padding = new Thickness(12);
            btn.Margin = new Thickness(96);
            btn.Click += PrintOnClick;
            Content = btn;
        }
        void PrintOnClick(object sender, RoutedEventArgs args)
        {
            PrintDialog dlg = new PrintDialog();

            if ((bool)dlg.ShowDialog().GetValueOrDefault())
            {
                // Vytvoří panel Grid.
                Grid grid = new Grid();

                // Definuje pět řádků a sloupců s automatickou velikostí.
                for (int i = 0; i < 5; i++)
```

```
                {
                    ColumnDefinition coldef = new ColumnDefinition();
                    coldef.Width = GridLength.Auto;
                    grid.ColumnDefinitions.Add(coldef);

                    RowDefinition rowdef = new RowDefinition();
                    rowdef.Height = GridLength.Auto;
                    grid.RowDefinitions.Add(rowdef);
                }

                // Nastaví pro panel Grid gradientní štětec.
                grid.Background =
                    new LinearGradientBrush(Colors.Gray, Colors.White,
                                            new Point(0, 0), new Point(1, 1));

                // Každý program potřebuje trochu náhodnosti.
                Random rand = new Random();

                // Umístí na panel Grid 25 tlačítek.
                for (int i = 0; i < 25; i++)
                {
                    Button btn = new Button();
                    btn.FontSize = 12 + rand.Next(8);
                    btn.Content = "Tlačítko č. " + (i + 1);
                    btn.HorizontalAlignment = HorizontalAlignment.Center;
                    btn.VerticalAlignment = VerticalAlignment.Center;
                    btn.Margin = new Thickness(6);
                    grid.Children.Add(btn);
                    Grid.SetRow(btn, i % 5);
                    Grid.SetColumn(btn, i / 5);
                }

                // Nastaví velikost panelu Grid.
                grid.Measure(new Size(Double.PositiveInfinity,
                                      Double.PositiveInfinity));

                Size sizeGrid = grid.DesiredSize;

                // Určí bod pro vystředění panelu Grid na stránce.
                Point ptGrid =
                    new Point((dlg.PrintableAreaWidth - sizeGrid.Width) / 2,
                              (dlg.PrintableAreaHeight - sizeGrid.Height) / 2);

                // Rozložení je hotovo.
                grid.Arrange(new Rect(ptGrid, sizeGrid));

                // Nyní se vytiskne.
                dlg.PrintVisual(grid, Title);
            }
        }
    }
}
```

Kvůli lepší názornosti program neobsahuje některé funkce předchozího programu – např. definici okrajů nebo uložení nastavení pomocí vlastností *PrintTicket* a *PrintQueue*.

Když program tiskne instanci třídy odvozené od třídy *UIElement*, nelze vynechat jeden klíčový krok: prvek musí projít úpravou rozložení. To znamená, že je nutné na objekt zavolat metody *Measure* a *Arrange*. V opačném případě bude mít objekt nulové rozměry a na stránce se nezob-

razí. Program PrintaBunchaButtons používá při volání metody *Measure* pro objekt *Grid* nekonečné rozměry. Případně lze rozměry odvodit z velikosti stránky. Při volání metody *Arrange* program získá velikost panelu *Grid* z jeho vlastnosti *DesiredSize*, vypočítá bod, který umožní umístit panel *Grid* na střed stránky, a potom zavolá metodu *Arrange* s daným bodem a velikostí. Panel *Grid* lze nyní předat metodě *PrintVisual*.

Vícestránkový dokument můžete vytisknout pomocí více volání metody *PrintVisual*, ale v tomto případě bude každá stránka považována za odlišnou tiskovou úlohu. Při tisku vícestránkového dokumentu je vhodnější, když program zavolá metodu *PrintDocument*, která je definována ve třídě *PrintDialog*. Argumenty metody *PrintDocument* jsou instance třídy odvozené od třídy *DocumentPaginator* a textový řetězec pro tiskovou frontu, který popisuje dokument.

DocumentPaginator je abstraktní třída, která pochází z oboru názvů *System.Windows.Documents*. Je nutné vytvořit třídu, která dědí od třídy *DocumentPaginator*, a přetížit několik vlastností a metod, které třída *DocumentPaginator* definuje jako abstraktní. Jednou z nich je metoda *GetPage*, která vrací objekt typu *DocumentPage*. Objekt *DocumentPage* lze snadno vytvořit z objektu *Visual*. To znamená, že tisk více stránek se příliš neliší od tisku jedné stránky. Každá stránka dokumentu tvoří samostatný objekt *Visual*.

Vaše třída odvozená od třídy *DocumentPaginator* musí přetížit logickou vlastnost *IsPageCountValid* pouze pro čtení, vlastnost *PageCount* pouze pro čtení, vlastnost *PageSize* pro čtení i zápis, metodu *GetPage* (která má argument počtu stránek počítaný od nuly a vrací objekt typu *DocumentPage*) a vlastnost *Source*, která může vracet hodnotu *null*.

Třída pocházející od třídy *DocumentPaginator* bude nejspíše definovat i některé nové vlastnosti. Předpokládejme například, že chcete napsat program, který bude tisknout nápisy. Programy na tisk nápisů kdysi tiskly na souvislý skládaný papír, ale v současnosti tyto programy tisknou velká písmena na samostatné stránky. Třída, která je následníkem třídy *DocumentPaginator*, pravděpodobně potřebuje vlastnost typu *Text*. Pro tuto vlastnost program nastaví například hodnotu „Vše nejlepší k narozeninám pro nejlepší mámu na Zemi a možná i na jiných planetách". Třída odvozená od třídy *DocumentPaginator* by také měla obsahovat vlastnost k nastavení písma. Počet potřebných vlastností lze omezit tím, že bude většina informací o písmu sdružena do objektu typu *Typeface*.

Pokud však chcete svou třídu typu *DocumentPaginator* trochu zjednodušit, můžete se bez uživatelských vlastností pravděpodobně obejít a místo toho prostě definovat konstruktor, který tyto informace bude přijímat. Objekt *DocumentPaginator* zpravidla neexistuje tak dlouho, aby záleželo na tom, který přístup zvolíte. Objekt budete pravděpodobně vytvářet v reakci na klepnutí na tlačítko Tisk v dialogu *PrintDialog* a zrušíte jej po zavolání metody *PrintDocument*.

Následující třída *BannerDocumentPaginator* definuje dvě vlastnosti nazvané *Text* a *Typeface*.

BannerDocumentPaginator.cs

```
//-------------------------------------------------------
// BannerDocumentPaginator.cs (c) 2006 by Charles Petzold
//-------------------------------------------------------
using System;
using System.Globalization;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Documents;
```

```
using System.Windows.Media;

namespace Petzold.PrintBanner
{
    public class BannerDocumentPaginator : DocumentPaginator
    {
        string txt = "";
        Typeface face = new Typeface("");
        Size sizePage;
        Size sizeMax = new Size(0, 0);

        // Veřejné vlastnosti specifické pro tento objekt DocumentPaginator.
        public string Text
        {
            set { txt = value; }
            get { return txt; }
        }
        public Typeface Typeface
        {
            set { face = value; }
            get { return face; }
        }

        // Soukromá funkce vytvoří objekt FormattedText.
        FormattedText GetFormattedText(char ch, Typeface face, double em)
        {
            return new FormattedText(ch.ToString(), CultureInfo.CurrentCulture,
                           FlowDirection.LeftToRight, face, em, Brushes.Black);
        }

        // Potřebné přetížené funkce.
        public override bool IsPageCountValid
        {
            get
            {
                // Určí maximální velikost znaků na základě velikosti em 100.
                foreach (char ch in txt)
                {
                    FormattedText formtxt = GetFormattedText(ch, face, 100);
                    sizeMax.Width = Math.Max(sizeMax.Width, formtxt.Width);
                    sizeMax.Height = Math.Max(sizeMax.Height, formtxt.Height);
                }
                return true;
            }
        }
        public override int PageCount
        {
            get { return txt == null ? 0 : txt.Length; }
        }
        public override Size PageSize
        {
            set { sizePage = value; }
            get { return sizePage; }
        }
        public override DocumentPage GetPage(int numPage)
        {
            DrawingVisual vis = new DrawingVisual();
            DrawingContext dc = vis.RenderOpen();
```

```
            // Při výpočtu faktoru velikosti em předpokládá okraje velikosti
            // půl palce (12,7 mm).
            double factor = Math.Min((PageSize.Width - 96) / sizeMax.Width,
                                     (PageSize.Height - 96) / sizeMax.Height);

            FormattedText formtxt = GetFormattedText(txt[numPage], face,
                                                     factor * 100);

            // Vyhledá bod k zarovnání znaku na střed stránky.
            Point ptText = new Point((PageSize.Width - formtxt.Width) / 2,
                                     (PageSize.Height - formtxt.Height) / 2);

            dc.DrawText(formtxt, ptText);
            dc.Close();

            return new DocumentPage(vis);
        }
        public override IDocumentPaginatorSource Source
        {
            get { return null; }
        }
    }
}
```

Jak vyplývá z názvu *DocumentPaginator*, potřebuje třída rozdělit dokument na stránky. Musí určit, kolik stránek dokument obsahuje a co bude na jednotlivých stránkách. Tento objekt paginator zjistí počet stránek jednoduše: jedná se o počet znaků v textovém řetězci. Objekt paginator však musí také určit, jak velké mají tyto znaky být. Tento úkol vyžaduje konstrukci objektů typu *FormattedText* pro každé písmeno, aby bylo možné zjistit výšku písmen a maximální šířku znaků.

Pokud definujete konstruktor, jehož argumenty nesou veškeré informace, které třída potřebuje pro stránkování, bude tento konstruktor také obsahovat logiku stránkování. Jestliže však definujete vlastnosti, musíte stránkování provést na jiném místě kódu. Podle mého názoru představuje vhodné vlastnost *IsPageCountValid*, protože tato vlastnost je volána jako první. Objekt *BannerDocumentPaginator* tedy ve své vlastnosti *IsPageCountValid* vytvoří objekty *FormattedText* pro všechny znaky v textovém řetězci na základě vlastnosti *Typeface* a velikosti em 100 a uloží největší zjištěnou velikost.

Metoda *GetPage* odpovídá za vrácení objektů typu *DocumentPage*, které lze vytvořit z objektu typu *Visual*. Metoda vytvoří objekt *DrawingVisual*, otevře *DrawingContext* a vypočítá násobící faktor, který je určen velikostí největšího znaku a velikostí stránky po odečtení půlpalcových okrajů. Metoda vytvoří nový objekt *FormattedText* a vystředí jej na stránce pomocí volání metody *DrawText*. Objekt *DrawingVisual* je předán konstruktoru třídy *DocumentPage* a metoda *GetPage* vrátí výsledný objekt.

Uživatelské rozhraní programu na tisk nápisů je poměrně jednoduché:

PrintBanner.cs

```
//-----------------------------------------
// PrintBanner.cs (c) 2006 by Charles Petzold
//-----------------------------------------
using System;
using System.Printing;
```

```
using System.Windows;
using System.Windows.Controls;
using System.Windows.Input;
using System.Windows.Media;

namespace Petzold.PrintBanner
{
    public class PrintBanner : Window
    {
        TextBox txtbox;

        [STAThread]
        public static void Main()
        {
            Application app = new Application();
            app.Run(new PrintBanner());
        }
        public PrintBanner()
        {
            Title = "Tisk nápisu";
            SizeToContent = SizeToContent.WidthAndHeight;

            // Nastaví panel StackPanel jako obsah okna.
            StackPanel stack = new StackPanel();
            Content = stack;

            // Vytvoří ovládací prvek TextBox.
            txtbox = new TextBox();
            txtbox.Width = 250;
            txtbox.Margin = new Thickness(12);
            stack.Children.Add(txtbox);

            // Vytvoří objekt Button.
            Button btn = new Button();
            btn.Content = "_Tisk...";
            btn.Margin = new Thickness(12);
            btn.Click += PrintOnClick;
            btn.HorizontalAlignment = HorizontalAlignment.Center;
            stack.Children.Add(btn);

            txtbox.Focus();
        }
        void PrintOnClick(object sender, RoutedEventArgs args)
        {
            PrintDialog dlg = new PrintDialog();

            if (dlg.ShowDialog().GetValueOrDefault())
            {
                // Zkontroluje, zda je nastavena orientace Na výšku.
                PrintTicket prntkt = dlg.PrintTicket;
                prntkt.PageOrientation = PageOrientation.Portrait;
                dlg.PrintTicket = prntkt;

                // Vytvoří nový objekt BannerDocumentPaginator.
                BannerDocumentPaginator paginator = new
                                        BannerDocumentPaginator();

                // Nastaví vlastnost Text z pole TextBox.
                paginator.Text = txtbox.Text;
```

```
                // Nastaví vlastnost PageSize založenou na rozměrech papíru.
                paginator.PageSize = new Size(dlg.PrintableAreaWidth,
                                              dlg.PrintableAreaHeight);

                // Zavolá objekt PrintDocument, který dokument vytiskne.
                dlg.PrintDocument(paginator, "Nápis: " + txtbox.Text);
            }
        }
    }
}
```

Program na tisk nápisů poskytuje nejlepší výsledky při orientaci stránky na výšku. I když tedy uživatel zvolí možnost Na šířku, změní metoda *PrintOnClick* toto nastavení zpět na hodnotu Na výšku. Metoda pak vytvoří objekt typu *BannerDocumentPaginator* a nastaví vlastnosti *Text* a *PageSize*. Metoda závěrem předá inicializovaný objekt *BannerDocumentPaginator* objektu *PrintDocument*, který provede zbývající akce.

Všimněte si, že program PrintBanner neposkytuje objektu *BannerDocumentPaginator* vlastní objekt *Typeface*. Tento program rozhodně potřebuje dialogové okno pro nastavení písma, ale první verze grafického subsystému Windows Presentation Foundation bohužel žádný takový dialog neobsahuje. Proto jsem jej musel vytvořit sám. Vůbec se mi do psaní dialogu pro nastavení písma nechtělo, ale neměl jsem jinou možnost.

Dialogy písma obvykle zahrnují pole se seznamem, u nichž je vždy zobrazena část seznamu. Třída *ComboBox* v první verzi grafického subsystému Windows Presentation Foundation bohužel tento režim nepodporuje a můj první pokus o napodobení tohoto typu pole se seznamem pomocí ovládacích prvků *TextBox* a *ListBox* nebyl příliš úspěšný. Chtěl jsem dosáhnout toho, aby bylo zaměření pro vstup trvale nastaveno do pole *TextBox*, ale nedařilo se mi zabránit tomu, aby zaměření pro vstup získával seznam *ListBox*. Potom jsem si uvědomil, že program musí vycházet od jednoduchého ovládacího prvku bez možnosti nastavit zaměření pro vstup, který by vypadal a fungoval jako seznam *ListBox*. Tento vlastní prvek jsem nazval *Lister*. Následuje první soubor z projektu ChooseFont.

Lister.cs

```
//---------------------------------------
// Lister.cs (c) 2006 by Charles Petzold
//---------------------------------------
using System;
using System.Collections;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Input;
using System.Windows.Media;

namespace Petzold.ChooseFont
{
    class Lister : ContentControl
    {
        ScrollViewer scroll;
        StackPanel stack;
        ArrayList list = new ArrayList();
        int indexSelected = -1;
```

```
// Veřejná událost.
public event EventHandler SelectionChanged;

// Konstruktor.
public Lister()
{
    Focusable = false;

    // Nastaví Border jako obsah objektu ContentControl.
    Border bord = new Border();
    bord.BorderThickness = new Thickness(1);
    bord.BorderBrush = SystemColors.ActiveBorderBrush;
    bord.Background = SystemColors.WindowBrush;
    Content = bord;

    // Nastaví objekt ScrollViewer jako podřízenou položku okraje.
    scroll = new ScrollViewer();
    scroll.Focusable = false;
    scroll.Padding = new Thickness(2, 0, 0, 0);
    bord.Child = scroll;

    // Nastaví panel StackPanel jako obsah objektu ScrollViewer.
    stack = new StackPanel();
    scroll.Content = stack;

    // Instaluje obslužnou rutinu pro událost klepnutí levého tlačítka
    // myši.
    AddHandler(TextBlock.MouseLeftButtonDownEvent,
        new MouseButtonEventHandler(TextBlockOnMouseLeftButtonDown));

    Loaded += OnLoaded;
}
void OnLoaded(object sender, RoutedEventArgs args)
{
    // Při prvním zobrazení ovládacího prvku Lister posune vybranou
    // položku tak, aby byla zobrazena.
    ScrollIntoView();
}

// Veřejné metody pro přidání, vložení atd. položek do ovládacího
// prvku Lister.
public void Add(object obj)
{
    list.Add(obj);
    TextBlock txtblk = new TextBlock();
    txtblk.Text = obj.ToString();
    stack.Children.Add(txtblk);
}
public void Insert(int index, object obj)
{
    list.Insert(index, obj);
    TextBlock txtblk = new TextBlock();
    txtblk.Text = obj.ToString();
    stack.Children.Insert(index, txtblk);
}
public void Clear()
{
    SelectedIndex = -1;
    stack.Children.Clear();
    list.Clear();
```

```
}
public bool Contains(object obj)
{
    return list.Contains(obj);
}
public int Count
{
    get { return list.Count; }
}

// Volání této metody umožní vybrat položku na základě stisknuté
// klávesy.
public void GoToLetter(char ch)
{
    int offset = SelectedIndex + 1;

    for (int i = 0; i < Count; i++)
    {
        int index = (i + offset) % Count;

        if (Char.ToUpper(ch) == Char.ToUpper(list[index].ToString()[0]))
        {
            SelectedIndex = index;
            break;
        }
    }
}

// Vlastnost SelectedIndex odpovídá za zobrazení panelu pro výběr.
public int SelectedIndex
{
    set
    {
        if (value < -1 || value >= Count)
            throw new ArgumentOutOfRangeException("SelectedIndex");

        if (value == indexSelected)
            return;

        if (indexSelected != -1)
        {
            TextBlock txtblk = stack.Children[indexSelected]
                                        as TextBlock;
            txtblk.Background = SystemColors.WindowBrush;
            txtblk.Foreground = SystemColors.WindowTextBrush;
        }

        indexSelected = value;

        if (indexSelected > -1)
        {
            TextBlock txtblk = stack.Children[indexSelected]
                                        as TextBlock;
            txtblk.Background = SystemColors.HighlightBrush;
            txtblk.Foreground = SystemColors.HighlightTextBrush;
        }
        ScrollIntoView();

        // Aktivuje událost SelectionChanged.
        OnSelectionChanged(EventArgs.Empty);
```

```
        }
        get
        {
            return indexSelected;
        }
    }

    // Vlastnost SelectedItem použije hodnotu vlastnosti SelectedIndex.
    public object SelectedItem
    {
        set
        {
            SelectedIndex = list.IndexOf(value);
        }
        get
        {
            if (SelectedIndex > -1)
                return list[SelectedIndex];

            return null;
        }
    }

    // Veřejné metody pro posun seznamu o stránku nahoru a dolů.
    public void PageUp()
    {
        if (SelectedIndex == -1 || Count == 0)
            return;

        int index = SelectedIndex - (int)(Count *
                            scroll.ViewportHeight / scroll.ExtentHeight);
        if (index < 0)
            index = 0;

        SelectedIndex = index;
    }
    public void PageDown()
    {
        if (SelectedIndex == -1 || Count == 0)
            return;

        int index = SelectedIndex + (int)(Count *
                            scroll.ViewportHeight / scroll.ExtentHeight);
        if (index > Count - 1)
            index = Count - 1;

        SelectedIndex = index;
    }

    // Soukromá metoda pro posun vybrané položky tak, aby byla viditelná.
    void ScrollIntoView()
    {
        if (Count == 0 || SelectedIndex == -1 ||
                            scroll.ViewportHeight > scroll.ExtentHeight)
            return;

        double heightPerItem = scroll.ExtentHeight / Count;
        double offsetItemTop = SelectedIndex * heightPerItem;
        double offsetItemBot = (SelectedIndex + 1) * heightPerItem;
```

```
            if (offsetItemTop < scroll.VerticalOffset)
                scroll.ScrollToVerticalOffset(offsetItemTop);

            else if (offsetItemBot > scroll.VerticalOffset +
                                     scroll.ViewportHeight)
                scroll.ScrollToVerticalOffset(scroll.VerticalOffset +
                                     offsetItemBot - scroll.VerticalOffset -
                                     scroll.ViewportHeight);
        }

        // Obslužná rutina a aktivace události.
        void TextBlockOnMouseLeftButtonDown(object sender,
                                            MouseButtonEventArgs args)
        {
            if (args.Source is TextBlock)
                SelectedIndex = stack.Children.IndexOf(args.Source as
                                            TextBlock);
        }

        protected virtual void OnSelectionChanged(EventArgs args)
        {
            if (SelectionChanged != null)
                SelectionChanged(this, args);
        }
    }
}
```

Objekt *Lister* je ovládací prvek typu *ContentControl* s objektem *Border*, který obsahuje objekt *ScrollViewer*. Ten obsahuje panel *StackPanel*, na kterém je umístěno více položek typu *TextBlock*. Ke konci souboru se nachází metoda *TextBlockOnMouseLeftButtonDown*, která nastaví vlastnost *SelectedIndex* indexu podle pole *TextBlock*, na které uživatel klepl. Vlastnost *SelectedIndex* udržuje barvu popředí a pozadí položek *TextBlock*, aby bylo zřejmé, která položka je aktuálně vybrána, a aktivuje událost *SelectionChanged* voláním metody *OnSelectionChanged*.

Rozhraní klávesnice pro změnu vybrané položky se obsluhuje mimo třídu *Lister* v následující třídě. Třída *TextBoxWithLister* se také odvozuje od třídy *ContentControl* a obsahuje panel *DockPanel* s ovládacím prvkem *TextBox* a prvkem *Lister*. Třída mění definici metody *OnPreviewKeyDown* tak, aby bylo možné změnit vybranou položku pomocí kláves se šipkami.

TextBoxWithLister.cs

```
//------------------------------------------------
// TextBoxWithLister.cs (c) 2006 by Charles Petzold
//------------------------------------------------
using System;
using System.Collections;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Input;
using System.Windows.Media;

namespace Petzold.ChooseFont
{
    class TextBoxWithLister : ContentControl
    {
        TextBox txtbox;
```

```
Lister lister;
bool isReadOnly;

// Veřejné události.
public event EventHandler SelectionChanged;
public event TextChangedEventHandler TextChanged;

// Konstruktor.
public TextBoxWithLister()
{
    // Vytvoří panel DockPanel jako obsah ovládacího prvku.
    DockPanel dock = new DockPanel();
    Content = dock;

    // Ovládací prvek TextBox je ukotven nahoře.
    txtbox = new TextBox();
    txtbox.TextChanged += TextBoxOnTextChanged;
    dock.Children.Add(txtbox);
    DockPanel.SetDock(txtbox, Dock.Top);

    // Lister vyplní zbývající část panelu DockPanel.
    lister = new Lister();
    lister.SelectionChanged += ListerOnSelectionChanged;
    dock.Children.Add(lister);
}

// Veřejné vlastnosti, které souvisejí s položkou TextBox.
public string Text
{
    get { return txtbox.Text; }
    set { txtbox.Text = value; }
}
public bool IsReadOnly
{
    set { isReadOnly = value; }
    get { return isReadOnly; }
}

// Další veřejné vlastnosti, které komunikují s prvkem Lister.
public object SelectedItem
{
    set
    {
        lister.SelectedItem = value;

        if (lister.SelectedItem != null)
            txtbox.Text = lister.SelectedItem.ToString();
        else
            txtbox.Text = "";
    }
    get
    {
        return lister.SelectedItem;
    }
}
public int SelectedIndex
{
    set
    {
        lister.SelectedIndex = value;
```

```
            if (lister.SelectedIndex == -1)
                txtbox.Text = "";
            else
                txtbox.Text = lister.SelectedItem.ToString();
        }
        get
        {
            return lister.SelectedIndex;
        }
    }
    public void Add(object obj)
    {
        lister.Add(obj);
    }
    public void Insert(int index, object obj)
    {
        lister.Insert(index, obj);
    }
    public void Clear()
    {
        lister.Clear();
    }
    public bool Contains(object obj)
    {
        return lister.Contains(obj);
    }

    // Při klepnutím myši nastaví zaměření pro vstup z klávesnice.
    protected override void OnMouseDown(MouseButtonEventArgs args)
    {
        base.OnMouseDown(args);
        Focus();
    }

    // Pokud je nastaveno zaměření pro vstup z klávesnice, je předáno
    // ovládacímu prvku TextBox.
    protected override void OnGotKeyboardFocus(
                                    KeyboardFocusChangedEventArgs args)
    {
        base.OnGotKeyboardFocus(args);

        if (args.NewFocus == this)
        {
            txtbox.Focus();
            if (SelectedIndex == -1 && lister.Count > 0)
                SelectedIndex = 0;
        }
    }

    // Stisknutá klávesa je předána metodě GoToLetter třídy Lister.
    protected override void OnPreviewTextInput(TextCompositionEventArgs args)
    {
        base.OnPreviewTextInput(args);

        if (IsReadOnly)
        {
            lister.GoToLetter(args.Text[0]);
            args.Handled = true;
        }
    }
```

```
// Obsluha kláves se šipkami pro změnu vybrané položky.
protected override void OnPreviewKeyDown(KeyEventArgs args)
{
    base.OnKeyDown(args);

    if (SelectedIndex == -1)
        return;

    switch (args.Key)
    {
        case Key.Home:
            if (lister.Count > 0)
                SelectedIndex = 0;
            break;

        case Key.End:
            if (lister.Count > 0)
                SelectedIndex = lister.Count - 1;
            break;

        case Key.Up:
            if (SelectedIndex > 0)
                SelectedIndex--;
            break;

        case Key.Down:
            if (SelectedIndex < lister.Count - 1)
                SelectedIndex++;
            break;

        case Key.PageUp:
            lister.PageUp();
            break;

        case Key.PageDown:
            lister.PageDown();
            break;

        default:
            return;
    }
    args.Handled = true;
}
// Obslužné rutiny a aktivace událostí.
void ListerOnSelectionChanged(object sender, EventArgs args)
{
    if (SelectedIndex == -1)
        txtbox.Text = "";
    else
        txtbox.Text = lister.SelectedItem.ToString();

    OnSelectionChanged(args);
}
void TextBoxOnTextChanged(object sender, TextChangedEventArgs args)
{
    if (TextChanged != null)
        TextChanged(this, args);
}
protected virtual void OnSelectionChanged(EventArgs args)
{
```

```
                if (SelectionChanged != null)
                    SelectionChanged(this, args);
            }
        }
    }
```

Tato třída má dva režimy, které jsou řízeny vlastností *IsReadOnly*. Pokud je nastavena hodnota *false*, může uživatel do pole *TextBox* zadat libovolnou hodnotu. Program, který s tímto ovládacím prvkem pracuje, pak získá uživatelský vstup z daného pole *TextBox*. Tento režim se v dialogovém okně *FontDialog* používá k nastavení velikosti písma. Část ovládacího prvku se seznamem by měla nabízet několik běžných velikostí písma, ale uživatel by měl mít možnost zadat libovolnou jinou hodnotu.

Jestliže má vlastnost *IsReadOnly* hodnotu *true*, pole *TextBox* vždy zobrazí vybranou položku a ručně nelze zadávat žádné hodnoty. Všechny stisknuté klávesy jsou předány metodě *GoToLetter* třídy *Lister*, která se pokusí vybrat následující položku začínající daným písmenem. Tento režim je vhodný pro pole dialogového okna *FontDialog*, která umožňují nastavit řadu písem, styl písma, řez a deformaci písma. Aby zůstalo dialogové okno co nejjednodušší, rozhodl jsem se, že by uživatel neměl mít možnost vybrat ani zadat žádnou nedostupnou hodnotu.

Navzdory nastavení vlastnosti *IsReadOnly* si pole *TextBox* stále udržuje zaměření pro vstup z klávesnice. Ovládací prvek získá z obslužné rutiny události oznámení, pokud uživatel klepne na položku v seznamu, ale seznam pro tuto akci nedostane zaměření pro vstup. Přetížená metoda *OnPreviewKeyDown* poskytuje větší část rozhraní klávesnice pro ovládací prvek, protože obsluhuje všechny klávesy se šipkami.

Samotná třída *FontDialog* obsahuje pět ovládacích prvků *TextBoxWithLister* (pro zobrazení řad písem, stylů, řezů, deformací a velikostí), pět popisů těchto ovládacích prvků, další prvek *Label* pro zobrazení ukázkového textu a tlačítka OK a Storno. Složitý je však teprve dynamický obsah ovládacích prvků.

První ovládací prvek *TextBoxWithLister* zobrazuje všechny dostupné řady písem. Tyto řady lze získat ze statické metody *Fonts.SystemFontFamilies*. K umístění názvů řad písem do tohoto ovládacího prvku dochází na konci konstruktoru. Jednotlivé řady písem však poskytují různé styly písem, řezy a deformace. Pokaždé, když uživatel vybere odlišnou řadu písem, musí obslužná rutina *FamilyOnSelectionChanged* třídy *FontDialog* vymazat tři ovládací prvky *TextBoxWithLister* a znovu je naplnit na základě objektů *FamilyTypeface*, které jsou k dispozici z vlastnosti *FamilyTypefaces* vybraného objektu *FontFamily*.

Třída *FontDialog* definuje pouze dvě veřejné vlastnosti: vlastnost *Typeface* zapouzdřuje vlastnosti *FontFamily*, *FontStyle*, *FontWeight* a *FontStretch* a vlastnost *FaceSize* (typu *double*) je určena k nastavení velikosti písma. (Poslední vlastnost jsem nemohl nazvat *FontSize*, protože samotná třída *FontDialog* dědí stejnojmennou vlastnost *FontSize*, která řídí velikost písma použitého v ovládacích prvcích dialogového okna!)

FontDialog.cs

```
//-----------------------------------------
// FontDialog.cs (c) 2006 by Charles Petzold
//-----------------------------------------
using System;
using System.Collections.Generic;
using System.Windows;
```

```
using System.Windows.Controls;
using System.Windows.Input;
using System.Windows.Media;

namespace Petzold.ChooseFont
{
    public class FontDialog : Window
    {
        TextBoxWithLister boxFamily, boxStyle, boxWeight, boxStretch, boxSize;
        Label lblDisplay;
        bool isUpdateSuppressed = true;

        // Veřejné vlastnosti.
        public Typeface Typeface
        {
            set
            {
                if (boxFamily.Contains(value.FontFamily))
                    boxFamily.SelectedItem = value.FontFamily;
                else
                    boxFamily.SelectedIndex = 0;

                if (boxStyle.Contains(value.Style))
                    boxStyle.SelectedItem = value.Style;
                else
                    boxStyle.SelectedIndex = 0;

                if (boxWeight.Contains(value.Weight))
                    boxWeight.SelectedItem = value.Weight;
                else
                    boxWeight.SelectedIndex = 0;

                if (boxStretch.Contains(value.Stretch))
                    boxStretch.SelectedItem = value.Stretch;
                else
                    boxStretch.SelectedIndex = 0;
            }
            get
            {
                return new Typeface((FontFamily)boxFamily.SelectedItem,
                                    (FontStyle)boxStyle.SelectedItem,
                                    (FontWeight)boxWeight.SelectedItem,
                                    (FontStretch)boxStretch.SelectedItem);
            }
        }
        public double FaceSize
        {
            set
            {
                double size = 0.75 * value;
                boxSize.Text = size.ToString();

                if (!boxSize.Contains(size))
                    boxSize.Insert(0, size);

                boxSize.SelectedItem = size;
            }
            get
            {
                double size;
```

```
            if (!Double.TryParse(boxSize.Text, out size))
                size = 8.25;

            return size / 0.75;
        }
    }

    // Konstruktor.
    public FontDialog()
    {
        Title = "Písmo";
        ShowInTaskbar = false;
        WindowStyle = WindowStyle.ToolWindow;
        WindowStartupLocation = WindowStartupLocation.CenterOwner;
        SizeToContent = SizeToContent.WidthAndHeight;
        ResizeMode = ResizeMode.NoResize;

        // Vytvoří panel Grid se třemi řádky jako obsah okna.
        Grid gridMain = new Grid();
        Content = gridMain;

        // Tento řádek slouží pro ovládací prvky TextBoxWithLister.
        RowDefinition rowdef = new RowDefinition();
        rowdef.Height = new GridLength(200, GridUnitType.Pixel);
        gridMain.RowDefinitions.Add(rowdef);

        // Tento řádek je určen pro ukázkový text.
        rowdef = new RowDefinition();
        rowdef.Height = new GridLength(150, GridUnitType.Pixel);
        gridMain.RowDefinitions.Add(rowdef);

        // Na tento řádek jsou umístěna tlačítka.
        rowdef = new RowDefinition();
        rowdef.Height = GridLength.Auto;
        gridMain.RowDefinitions.Add(rowdef);

        // Jeden sloupec na hlavním panelu Grid.
        ColumnDefinition coldef = new ColumnDefinition();
        coldef.Width = new GridLength(650, GridUnitType.Pixel);
        gridMain.ColumnDefinitions.Add(coldef);

        // Vytvoří panel Grid se dvěma řádky a pěti sloupci pro ovládací
        // prvky TextBoxWithLister.
        Grid gridBoxes = new Grid();
        gridMain.Children.Add(gridBoxes);

        // Tento řádek je určen pro popisy.
        rowdef = new RowDefinition();
        rowdef.Height = GridLength.Auto;
        gridBoxes.RowDefinitions.Add(rowdef);

        // Na tomto řádku se nacházejí ovládací prvky EditBoxWithLister.
        rowdef = new RowDefinition();
        rowdef.Height = new GridLength(100, GridUnitType.Star);
        gridBoxes.RowDefinitions.Add(rowdef);

        // První sloupec je FontFamily.
        coldef = new ColumnDefinition();
        coldef.Width = new GridLength(175, GridUnitType.Star);
        gridBoxes.ColumnDefinitions.Add(coldef);
```

```
// Druhý sloupec je FontStyle.
coldef = new ColumnDefinition();
coldef.Width = new GridLength(100, GridUnitType.Star);
gridBoxes.ColumnDefinitions.Add(coldef);

// Třetí sloupec je FontWeight.
coldef = new ColumnDefinition();
coldef.Width = new GridLength(100, GridUnitType.Star);
gridBoxes.ColumnDefinitions.Add(coldef);

// Čtvrtý sloupec je FontStretch.
coldef = new ColumnDefinition();
coldef.Width = new GridLength(100, GridUnitType.Star);
gridBoxes.ColumnDefinitions.Add(coldef);

// Pátý sloupec je Size.
coldef = new ColumnDefinition();
coldef.Width = new GridLength(75, GridUnitType.Star);
gridBoxes.ColumnDefinitions.Add(coldef);

// Vytvoří popisy FontFamily a ovládací prvky TextBoxWithLister.
Label lbl = new Label();
lbl.Content = "Řada písem";
lbl.Margin = new Thickness(12, 12, 12, 0);
gridBoxes.Children.Add(lbl);
Grid.SetRow(lbl, 0);
Grid.SetColumn(lbl, 0);

boxFamily = new TextBoxWithLister();
boxFamily.IsReadOnly = true;
boxFamily.Margin = new Thickness(12, 0, 12, 12);
gridBoxes.Children.Add(boxFamily);
Grid.SetRow(boxFamily, 1);
Grid.SetColumn(boxFamily, 0);

// Vytvoří popisy FontStyle a ovládací prvky TextBoxWithLister.
lbl = new Label();
lbl.Content = "Styl";
lbl.Margin = new Thickness(12, 12, 12, 0);
gridBoxes.Children.Add(lbl);
Grid.SetRow(lbl, 0);
Grid.SetColumn(lbl, 1);

boxStyle = new TextBoxWithLister();
boxStyle.IsReadOnly = true;
boxStyle.Margin = new Thickness(12, 0, 12, 12);
gridBoxes.Children.Add(boxStyle);
Grid.SetRow(boxStyle, 1);
Grid.SetColumn(boxStyle, 1);

// Vytvoří popisy FontWeight a ovládací prvky TextBoxWithLister.
lbl = new Label();
lbl.Content = "Řez";
lbl.Margin = new Thickness(12, 12, 12, 0);
gridBoxes.Children.Add(lbl);
Grid.SetRow(lbl, 0);
Grid.SetColumn(lbl, 2);

boxWeight = new TextBoxWithLister();
boxWeight.IsReadOnly = true;
```

```
boxWeight.Margin = new Thickness(12, 0, 12, 12);
gridBoxes.Children.Add(boxWeight);
Grid.SetRow(boxWeight, 1);
Grid.SetColumn(boxWeight, 2);

// Vytvoří popisy FontStretch a ovládací prvky TextBoxWithLister.
lbl = new Label();
lbl.Content = "Deformace";
lbl.Margin = new Thickness(12, 12, 12, 0);
gridBoxes.Children.Add(lbl);
Grid.SetRow(lbl, 0);
Grid.SetColumn(lbl, 3);

boxStretch = new TextBoxWithLister();
boxStretch.IsReadOnly = true;
boxStretch.Margin = new Thickness(12, 0, 12, 12);
gridBoxes.Children.Add(boxStretch);
Grid.SetRow(boxStretch, 1);
Grid.SetColumn(boxStretch, 3);

// Vytvoří popisy Size a ovládací prvky TextBoxWithLister.
lbl = new Label();
lbl.Content = "Velikost";
lbl.Margin = new Thickness(12, 12, 12, 0);
gridBoxes.Children.Add(lbl);
Grid.SetRow(lbl, 0);
Grid.SetColumn(lbl, 4);

boxSize = new TextBoxWithLister();
boxSize.Margin = new Thickness(12, 0, 12, 12);
gridBoxes.Children.Add(boxSize);
Grid.SetRow(boxSize, 1);
Grid.SetColumn(boxSize, 4);

// Vytvoří ovládací prvek pro zobrazení ukázkového textu.
lblDisplay = new Label();
lblDisplay.Content = "AaBbCc XxYzZz 012345";
lblDisplay.HorizontalContentAlignment = HorizontalAlignment.Center;
lblDisplay.VerticalContentAlignment = VerticalAlignment.Center;
gridMain.Children.Add(lblDisplay);
Grid.SetRow(lblDisplay, 1);

// Vytvoří panel Grid s pěti sloupci pro objekty Buttons.
Grid gridButtons = new Grid();
gridMain.Children.Add(gridButtons);
Grid.SetRow(gridButtons, 2);

for (int i = 0; i < 5; i++)
    gridButtons.ColumnDefinitions.Add(new ColumnDefinition());

// Tlačítko OK.
Button btn = new Button();
btn.Content = "OK";
btn.IsDefault = true;
btn.HorizontalAlignment = HorizontalAlignment.Center;
btn.MinWidth = 60;
btn.Margin = new Thickness(12);
btn.Click += OkOnClick;
gridButtons.Children.Add(btn);
Grid.SetColumn(btn, 1);
```

```
        // Tlačítko Storno.
        btn = new Button();
        btn.Content = "Storno";
        btn.IsCancel = true;
        btn.HorizontalAlignment = HorizontalAlignment.Center;
        btn.MinWidth = 60;
        btn.Margin = new Thickness(12);
        gridButtons.Children.Add(btn);
        Grid.SetColumn(btn, 3);

        // Inicializuje pole FontFamily pomocí systémových řad písem.
        foreach (FontFamily fam in Fonts.SystemFontFamilies)
               boxFamily.Add(fam);

        // Inicializuje pole FontSize.
        double[] ptsizes = new double[] { 8, 9, 10, 11, 12, 14, 16, 18,
                                          20, 22, 24, 26, 28, 36, 48, 72 };
        foreach (double ptsize in ptsizes)
            boxSize.Add(ptsize);

        // Nastaví obslužné rutiny událostí.
        boxFamily.SelectionChanged += FamilyOnSelectionChanged;
        boxStyle.SelectionChanged += StyleOnSelectionChanged;
        boxWeight.SelectionChanged += StyleOnSelectionChanged;
        boxStretch.SelectionChanged += StyleOnSelectionChanged;
        boxSize.TextChanged += SizeOnTextChanged;

        // Inicializuje vybrané hodnoty podle vlastností systému Window.
        // (Při nastavení vlastností budou tyto hodnoty pravděpodobně
        // změněny.)
        Typeface = new Typeface(FontFamily, FontStyle,
                                FontWeight, FontStretch);
        FaceSize = FontSize;

        // Nastaví zaměření pro vstup z klávesnice.
        boxFamily.Focus();

        // Umožní aktualizovat ukázkový text.
        isUpdateSuppressed = false;
        UpdateSample();
    }

    // Obslužná rutina pro událost SelectionChanged v poli FontFamily.
    void FamilyOnSelectionChanged(object sender, EventArgs args)
    {
        // Získá vybranou hodnotu FontFamily.
        FontFamily fntfam = (FontFamily)boxFamily.SelectedItem;

        // Uloží předchozí nastavení Style, Weight, Stretch.
        // Tyto hodnoty by měly být null pouze při prvním volání
        //  této metody.
        FontStyle? fntstyPrevious = (FontStyle?)boxStyle.SelectedItem;
        FontWeight? fntwtPrevious = (FontWeight?) boxWeight.SelectedItem;
        FontStretch? fntstrPrevious = (FontStretch?)boxStretch.SelectedItem;

        // Vypne zobrazení ukázkového textu.
        isUpdateSuppressed = true;

        // Vymaže pole pro styl, řez a deformaci.
        boxStyle.Clear();
```

```
        boxWeight.Clear();
        boxStretch.Clear();

        // Cyklicky prochází řezy písem ve vybrané řadě FontFamily.
        foreach (FamilyTypeface ftf in fntfam.FamilyTypefaces)
        {
            // Vloží hodnotu Style do proměnné boxStyle (hodnota Normal je
            // vždy navrchu).
            if (!boxStyle.Contains(ftf.Style))
            {
                if (ftf.Style == FontStyles.Normal)
                    boxStyle.Insert(0, ftf.Style);
                else
                    boxStyle.Add(ftf.Style);
            }
            // Vloží hodnotu Weight do proměnné boxWeight (hodnota Normal
            // je vždy navrchu).
            if (!boxWeight.Contains(ftf.Weight))
            {
                if (ftf.Weight == FontWeights.Normal)
                    boxWeight.Insert(0, ftf.Weight);
                else
                    boxWeight.Add(ftf.Weight);
            }
            // Vloží hodnotu Stretch do proměnné boxStretch (hodnota
            // Normal je vždy navrchu).
            if (!boxStretch.Contains(ftf.Stretch))
            {
                if (ftf.Stretch == FontStretches.Normal)
                    boxStretch.Insert(0, ftf.Stretch);
                else
                    boxStretch.Add(ftf.Stretch);
            }
        }

        // Nastaví vybranou položku v proměnné boxStyle.
        if (boxStyle.Contains(fntstyPrevious))
            boxStyle.SelectedItem = fntstyPrevious;
        else
            boxStyle.SelectedIndex = 0;

        // Nastaví vybranou položku v proměnné boxWeight.
        if (boxWeight.Contains(fntwtPrevious))
            boxWeight.SelectedItem = fntwtPrevious;
        else
            boxWeight.SelectedIndex = 0;

        // Nastaví vybranou položku v proměnné boxStretch.
        if (boxStretch.Contains(fntstrPrevious))
            boxStretch.SelectedItem = fntstrPrevious;
        else
            boxStretch.SelectedIndex = 0;

        // Obnoví aktualizaci ukázkového textu a aktualizuje jej.
        isUpdateSuppressed = false;
        UpdateSample();
    }
```

```
        // Obslužná rutina události SelectionChanged v polích pro styl, řez
        // a deformaci.
        void StyleOnSelectionChanged(object sender, EventArgs args)
        {
            UpdateSample();
        }

        // Obslužná rutina pro událost TextChanged v poli Size.
        void SizeOnTextChanged(object sender, TextChangedEventArgs args)
        {
            UpdateSample();
        }

        // Aktualizuje ukázkový text.
        void UpdateSample()
        {
            if (isUpdateSuppressed)
                return;

            lblDisplay.FontFamily = (FontFamily)boxFamily.SelectedItem;
            lblDisplay.FontStyle = (FontStyle)boxStyle.SelectedItem;
            lblDisplay.FontWeight = (FontWeight)boxWeight.SelectedItem;
            lblDisplay.FontStretch = (FontStretch)boxStretch.SelectedItem;

            double size;

            if (!Double.TryParse(boxSize.Text, out size))
                size = 8.25;

            lblDisplay.FontSize = size / 0.75;
        }

        // Tlačítko OK uzavře dialogové okno.
        void OkOnClick(object sender, RoutedEventArgs args)
        {
            DialogResult = true;
        }
    }
}
```

Za účelem vyzkoušení třídy *FontDialog* vytvoří následující program *ChooseFont* objekt *Button* uprostřed své klientské oblasti a vyvolá dialog při každém klepnutí na tlačítko. Pokud uživatel klepne na tlačítko OK, nastaví program vlastnosti písma okna podle hodnot zadaných do dialogového okna. Tlačítko tyto vlastnosti samozřejmě zdědí.

ChooseFont.cs

```
//-----------------------------------------
// ChooseFont.cs (c) 2006 by Charles Petzold
//-----------------------------------------
using System;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Input;
using System.Windows.Media;

namespace Petzold.ChooseFont
{
```

```
    public class ChooseFont : Window
    {
        [STAThread]
        public static void Main()
        {
            Application app = new Application();
            app.Run(new ChooseFont());
        }
        public ChooseFont()
        {
            Title = "Výběr písma";

            Button btn = new Button();
            btn.Content = Title;
            btn.HorizontalAlignment = HorizontalAlignment.Center;
            btn.VerticalAlignment = VerticalAlignment.Center;
            btn.Click += ButtonOnClick;
            Content = btn;
        }
        void ButtonOnClick(object sender, RoutedEventArgs args)
        {
            FontDialog dlg = new FontDialog();
            dlg.Owner = this;

            // Nastaví vlastnosti třídy FontDialog ze třídy Window.
            dlg.Typeface = new Typeface(FontFamily, FontStyle,
                                        FontWeight, FontStretch);
            dlg.FaceSize = FontSize;

            if (dlg.ShowDialog().GetValueOrDefault())
            {
                // Nastaví vlastnosti třídy Window ze třídy FontDialog.
                FontFamily = dlg.Typeface.FontFamily;
                FontStyle = dlg.Typeface.Style;
                FontWeight = dlg.Typeface.Weight;
                FontStretch = dlg.Typeface.Stretch;
                FontSize = dlg.FaceSize;
            }
        }
    }
}
```

Bez dalšího vysvětlování hned přejdeme k lepší verzi programu na tisk nápisů. Tento projekt vyžaduje odkazy na soubor BannerDocumentPaginator.cs a tři soubory použité ve třídě *FontDialog*: Lister.cs, TextBoxWithLister.cs a FontDialog.cs. Tato nová verze programu obsahuje tlačítko Písmo, které zobrazí dialogové okno *FontDialog*. Vybraný řez písma je jednoduše předán do objektu *BannerDocumentPaginator*. Požadovaná velikost se samozřejmě ignoruje, protože třída *BannerDocumentPaginator* počítá svou vlastní velikost písma v závislosti na rozměrech stránky.

PrintBetterBanner.cs

```
//-----------------------------------------------
// PrintBetterBanner.cs (c) 2006 by Charles Petzold
//-----------------------------------------------
using Petzold.ChooseFont;
using Petzold.PrintBanner;
```

```
using System;
using System.Printing;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Input;
using System.Windows.Media;

namespace Petzold.PrintBetterBanner
{
    public class PrintBetterBanner : Window
    {
        TextBox txtbox;
        Typeface face;

        [STAThread]
        public static void Main()
        {
            Application app = new Application();
            app.Run(new PrintBetterBanner());
        }
        public PrintBetterBanner()
        {
            Title = "Tisk lepšího nápisu";
            SizeToContent = SizeToContent.WidthAndHeight;

            // Nastaví panel StackPanel jako obsah okna.
            StackPanel stack = new StackPanel();
            Content = stack;

            // Vytvoří ovládací prvek TextBox.
            txtbox = new TextBox();
            txtbox.Width = 250;
            txtbox.Margin = new Thickness(12);
            stack.Children.Add(txtbox);

            // Vytvoří tlačítko Písmo.
            Button btn = new Button();
            btn.Content = "_Písmo...";
            btn.Margin = new Thickness(12);
            btn.Click += FontOnClick;
            btn.HorizontalAlignment = HorizontalAlignment.Center;
            stack.Children.Add(btn);

            // Vytvoří tlačítko Tisk.
            btn = new Button();
            btn.Content = "_Tisk...";
            btn.Margin = new Thickness(12);
            btn.Click += PrintOnClick;
            btn.HorizontalAlignment = HorizontalAlignment.Center;
            stack.Children.Add(btn);

            // Inicializuej proměnnou Facename.
            face = new Typeface(FontFamily, FontStyle, FontWeight, FontStretch);

            txtbox.Focus();
        }
        void FontOnClick(object sender, RoutedEventArgs args)
        {
            FontDialog dlg = new FontDialog();
            dlg.Owner = this;
```

```
            dlg.Typeface = face;

            if (dlg.ShowDialog().GetValueOrDefault())
            {
                face = dlg.Typeface;
            }
        }
        void PrintOnClick(object sender, RoutedEventArgs args)
        {
            PrintDialog dlg = new PrintDialog();

            if (dlg.ShowDialog().GetValueOrDefault())
            {
                // Zkontroluje, zda je nastavena orientace Na výšku.
                PrintTicket prntkt = dlg.PrintTicket;
                prntkt.PageOrientation = PageOrientation.Portrait;
                dlg.PrintTicket = prntkt;

                // Vytvoří nový objekt DocumentPaginator.
                BannerDocumentPaginator paginator =
                                        new BannerDocumentPaginator();

                // Nastaví vlastnost Text z pole TextBox.
                paginator.Text = txtbox.Text;

                // Nastaví vlastnost Typeface z proměnné.
                paginator.Typeface = face;

                // Nastaví vlastnost PageSize založenou na rozměrech papíru.
                paginator.PageSize = new Size(dlg.PrintableAreaWidth,
                                              dlg.PrintableAreaHeight);

                // Zavolá objekt PrintDocument, který dokument vytiskne.
                dlg.PrintDocument(paginator, "Nápis: " + txtbox.Text);
            }
        }
    }

}
```

Třída typu paginator pro nápisy by mohla s výhodou dokumentovat základy tisku více stránek. Ani zdaleka není tak užitečná jako program, který umístí text na stránky. Pokud se používá více řezů písem, může být třída paginator tohoto typu poměrně komplexní. Triviální však není ani programování třídy paginator, která pracuje s prostým textem s jednotnou velikostí.

Bez ohledu na potíže vyžaduje program Notepad Clone z následující kapitoly třídu paginator, která bude zpracovávat neformátovaný text a zajišťovat zalamování slov. Významnou část tohoto projektu tvoří třída *PlainTextDocumentPaginator* (i tuto třídu najdete v následující kapitole).