

Ezen a ponton a Visual Studio tervézésének valahogy úgy kell kinéznie, mint a 29.27. ábrán látható képnek.

```
<!-- Helyezzük a lútra egy Statusbar tipust -->
<Statusbar DocName="Bottom" BackGround="Beige" >
<StatusbarItem Name="statusbarItem" Ready=</TextLock>
<TextLock Name="statusbarItem" Ready=</TextLock>
<StatusbarItem>
```

A Statusbar típus a `DocPanele` lásd részére kerül, és egyptelen `TextLock` típusú kódzettelennel az elölöző Toolbar definíció után:

```
megegyező az állapotsavának. Adjuk a kodhoz a következő markupot,
olyan kis fułszövegekhez optimalizált, mint amilyenek a felhasználói felület
lág nincs szüksége ílyen támogatásra, a TextLock típus másik előnye az, hogy
alaphozzott szövegét, a sortöréseset stb. Míg a Statusbar típusunknak gyakorlati-
mogafülkék többfajta szöveges jelölés használatát, például a felkover szöveget, az
hasonlóan a TextLock típus is szöveget tölöl. A TextLock típusok emellett ta-
tarlaimaz, amelyet a fejezet ezen pontjáig még nem használtunk. A Textboxok
A Statusbar típus a DocPanele lásd részére kerül, és egyptelen TextLock típusú
```

A Statusbar típus készítése

Megjegyzés A Toolbar típus tisztes szereint becsomagoltak `Toolbar` elembe, amelyet a További részleteket forduljunk a .NET Framework 3.5 SDK dokumentációjához.

A `Toolbar` típusunk két button típuszt tartalmaz, amelyek történetesen úgyan-
azokat az eseményeket kezelik, és a kodfájunk ügyanazon metodusai kezelik
azokat az eseményeket kezelik, és a metoduszokat használhatók, hogy ki-
érkezik. A metodusz nevekkel összefügg az eseménykezelőkkel, hogy ki-
szolgálják minden elemtől, minden az eszközökkel. Noha ez az esz-
köztár típus nyomogombokat használ, tudunk kérni, hogy a Toolbar típus "az-
tebe (leggyakrabban lista, kepekre, grafikákkal stb.). Az egyszerűebb érdekes pont az,
egy" ContentControl, ezért nyugodtan beagyazhatunk bármilyen típus a felüle-
tbe (leggyakrabban lista, kepekre, grafikákkal stb.). Az egyszerűebb érdekes pont az,
kötötött nyomogombokat használ, tudunk kérni, hogy a Toolbar típus "az-
szolgáltatásokkal minden eseménytől minden eseménnyel, hogy ki-
érkezik. A metodusz nevekkel összefügg az eseménykezelőkkel, hogy ki-
érkezik. A metodusz nevekkel összefügg az eseménykezelőkkel, hogy ki-
érkezik. A metodusz nevekkel összefügg az eseménykezelőkkel, hogy ki-

Abban kérteink készítéséhez bágyazott panelnek használataval

hogy befejezzük az ábólak felhasználói felületeinek definíjóját: <DockPanel> bal oldalához kerül. Adjuk hozzá a következő XAML-márkupot, sort támogat, és engedélyezte a helyesírás-ellenőrzést. A teljes <grid> a szűle lyesztási játékok listáját. A jobb oldalon lesz egy TextBox típus, amely több Expander típus, amely <stackPanel> típusba csomagolva megjelenít a he grid típus definíciója, amely két osztópot határoz meg. A bal oldalon lesz az Felhasználó felületünk tervezetének utolsó aspektusa egy olyan osztátható

A felhasználói felület végezettsége

29.27. ábra: Helyesírásellenőrző alkalmazásunk alkalmi felhasználói felülete



29. fejezet: Programozás WPF-vézetőfelülemeikkel

```

        txtData.GetSpellLiningError(txtData.CaretIndex);
    SpellLiningError error =
    // Hetylene.
    // Probabilunk helyesfárat hibát keresni a körzor aktuális
    string spellLiningHints = string.Empty;
}

protected void ToolsSpellLiningHints_Click(object sender,
    RoutedEventArgs args)
{
    ...
}

public partial class MainWindow : System.Windows.Window
{
    A felhasználói felület imári készen van, még implementációit kell biztosítanak a megmaradt eseménykezelők számára. Itt van a szoban forgó releváns kód, amely a jelezet ezek pontján már nem igényel sok magyarázatot:
}

```

A megvalósítás véglegesítése

```

<!-- Ez az a terület, ahova lehet majd írni -->
<Label Name="lblSpellLiningHints" FontSize="12"/>
<!-- Ez a rész programozott módon töltjük fel -->
<Expander Name="expanderSpellLining">
    <Label>
        SpellLining Hints
        FontSize="14" Margin="10,0,0,0">
        <Label Name="lblSpellLiningInstructions">
            StackPanel Grid.Column="0" VerticalAlignment="Top" Background="Gray" />
            <Grid.ColumnDefinitions>
                <ColumnDefinition Width="5" Background="Gray" />
                <ColumnDefinition />
            </Grid.ColumnDefinitions>
            <Grid DockPanel.Dock="Left" Background="AliceBlue">
                <!-- A sorok és az oszlopok definíciója -->
                <Grid.ColumnDefinitions>
                    <ColumnDefinition />
                    <ColumnDefinition />
                </Grid.ColumnDefinitions>
                <StackPanel Grid.Column="0" VerticalAlignment="Top" Background="Gray" />
                <Expander Name="expanderName" Margin="10,10,10,10" Header="Try these!">
                    <TextBlock Grid.Column="1" SpellCheck.IsEnabled="True" Text="A megvalósítás véglegesítése" />
                    <TextBlock Grid.Column="1" SpellCheck.IsEnabled="True" Text="A megvalósítás véglegesítése" />
                </Expander>
            </StackPanel>
        </Label>
    </Expander>
    <Label Name="lblLining" Grid.Column="1" SpellCheck.IsEnabled="True" Text="A megvalósítás véglegesítése" />
    <Label Name="lblLining" Grid.Column="1" SpellCheck.IsEnabled="True" Text="A megvalósítás véglegesítése" />
</Label>

```

talyhoz, amelyben definíálták őket. Ezneki, a normális.NET-sesemények szorosan kötődnek ahol az oszthatáros meg, és csak az osztály vagy annak egy részármaotthona használhat nek" tekintethetünk. Mint tudjuk, egy tipikus.NET-seseményt adott alaposztályban támogatja azt a jelenéget, amelyet "vezérlőelem-függelék eseményeket val foglalkozik. A Windows Presentation Foundation a vezérlőutastások reál fejezet következő főbb egysége a vezérlőutastások temakörének vizsgálata.

A WPF vezérlőutastásai

Hogy még erdekesebb tegyük, még kevésbé ismerteketnünk a vezérlő utastásokkal. Úgy is elkezdtük egy mulódó szövegszerkesztő alkalmát. Ahhoz, hogy minden vagyonunk Windows Presentation Foundationnak a vezérlő utastásai,

```

    {
        staticBarText.Text = "Ready";
    }
    protected void MouseLeaveArea(object sender, RoutedEventArgs args)
    {
        staticBarText.Text = "Show Spelling Suggestions";
    }
    protected void MouseEnterToolShintArea(object sender,
        RoutedEventArgs args,
        RoutedEventArgs args2)
    {
        staticBarText.Text = "Exit the Application";
    }
    protected void MouseEnterExitArea(object sender,
        RoutedEventArgs args)
    {
        staticBarText.Text = "Exit the Application";
    }
    protected void ExpandCollapse();
    {
        if (error != null)
        {
            foreach (string s in error.Suggestions)
            {
                spellingHints += string.Format("{0}\n", s);
            }
            label.Content = spellingHints;
        }
    }
    // Mutassuk meg a javaStatokat a Label es az Expander
    // tipuson belül.
    // Készítünk el a helyesírást javaslatok sztringjét.
    // Készítünk el a helyesírást javaslatok sztringjét.
    if (error != null)
    {
        foreach (string s in error.Suggestions)
        {
            spellingHints += string.Format("{0}\n", s);
        }
        label.Content = spellingHints;
    }
}

```


29.4. táblázat: A belső WPF-vezérlőelem parancsobjektumok

A WPF-vezérlőelem	Parancsobjektum	Jelentés
Applikációs parancsok	Close, Copy, Cut, Find, Open, Paste, Save, SaveAll	Olyan tulajdonságokat, amelyek az alkalmazásban elérhetők.
Componenti parancsok	Movedown, MoveFocusBack, Azokat a tulajdonságokat, amelyek a határozott megegyeznek a részlegben.	Olyan tulajdonságokat, amelyek a részlegben elérhetők.
Media parancsok	Boostbase, ChannelUp, Olyan tulajdonságokat, amelyek a részlegben elérhetők.	Olyan tulajdonságokat, amelyek a részlegben elérhetők.
Navigációs parancsok	BrowsesBack, Topb olyan tulajdonságokat, amelyek a WPF előreirányítóban elérhetők.	Több olyan tulajdonságokat, amelyek a WPF előreirányítóban elérhetők.
Editing parancsok	Aligncenter, CorrectSpell, Initial, AmendText, DecreaseFontSize, WPF-dokumentum API alapján történő programozására esetben használhatók.	Olyan tulajdonságokat, amelyeket általában a WPF-dokumentum API alapján történő programozására használhatók.

A 29.4. táblázat az egyes belső Parancsobjektumok által kijátszott alapértelmezett tulajdonságok közül ismertetnékanyát (további részletekért tekintse meg a .NET Framework 3.5 SDK dokumentációját).

modon azonnal használhatók az új menülelemeket. Alkalmaszt, és kijelölések valamennyi szövegeit, akkor a 29.28. ábrán látható képes „masolás, kivágás és beillesztés” műveletekre. Ezért, ha futtatnák az utólagműveletet a masolásnak megfelelő helyen, így az alkalmazás már impozitív kód nekül is megtekinthető. Ezáltal a menülelemek automatikusan megkapják a megfelelő rendelt értékkel. Ezáltal a menülelemek automatikusan megkapják a sajátos rendszerekkel. Hogyan rendelkezik egy, a command tulajdon-ságban rendelt minden részlegben, hogy minden részlegben rendelkezik egy, a command tulajdon-

```

</MenuItem>
</MenuItem>
<MenuItem Header="Tools" ->
  <MenuItem Header="Spelling hints" ->
    <MenuItem Command="Paste" ->
      <MenuItem Command="Cut" ->
        <MenuItem Command="Copy" ->
          <MenuItem Command="Delete" ->
            <MenuItem Header="Edit" ->
              <MenuItem Header="MouseLeaveArea" ->
                <MenuItem Click="MouseLeaveArea" ->
                  <MenuItem Header="MouseEnterToolSHintArea" ->
                    <MenuItem Click="MouseEnterToolSHintArea" ->
                      <MenuItem Header="MouseLeaveToolSHintArea" ->
                        <MenuItem Click="MouseLeaveToolSHintArea" ->
                          <MenuItem Header="MouseLeave" ->
                            <MenuItem Click="MouseLeave" ->
                              <MenuItem Header="MouseLeave" ->
                                <MenuItem Click="MouseLeave" ->
                                  <MenuItem Header="MouseLeave" ->
                                    <MenuItem Click="MouseLeave" ->
                                      <MenuItem Header="MouseLeave" ->
                                        <MenuItem Click="MouseLeave" ->
                                          <MenuItem Header="MouseLeave" ->
                                            <MenuItem Click="MouseLeave" ->
                                              <MenuItem Header="MouseLeave" ->
                                                <MenuItem Click="MouseLeave" ->
                                                  <MenuItem Header="MouseLeave" ->
                                                    <MenuItem Click="MouseLeave" ->
                                                      <MenuItem Header="MouseLeave" ->
                                                        <MenuItem Click="MouseLeave" ->
                                                          <MenuItem Header="MouseLeave" ->
                                                            <MenuItem Click="MouseLeave" ->
                                                              <MenuItem Header="MouseLeave" ->
                                                                <MenuItem Click="MouseLeave" ->
                                                                  <MenuItem Header="MouseLeave" ->
                                                                    <MenuItem Click="MouseLeave" ->
                                                                      <MenuItem Header="MouseLeave" ->
                                                                        <MenuItem Click="MouseLeave" ->
                                                                          <MenuItem Header="MouseLeave" ->
                                                                            <MenuItem Click="MouseLeave" ->
                                                                              <MenuItem Header="MouseLeave" ->
                                                                                <MenuItem Click="MouseLeave" ->
                                                                                  <MenuItem Header="MouseLeave" ->
                                                                                    <MenuItem Click="MouseLeave" ->
                                                                                      <MenuItem Header="MouseLeave" ->
                                                                                        <MenuItem Click="MouseLeave" ->
                                                                                          <MenuItem Header="MouseLeave" ->
                                                                                            <MenuItem Click="MouseLeave" ->
                                                                                              <MenuItem Header="MouseLeave" ->
                                                                                                <MenuItem Click="MouseLeave" ->
                                                                                                  <MenuItem Header="MouseLeave" ->
                                                                                                    <MenuItem Click="MouseLeave" ->
                                                                                                     <MenuItem Header="MouseLeave" ->
                                                                                                       <MenuItem Click="MouseLeave" ->
                                                                                                         <MenuItem Header="MouseLeave" ->
                                                                                                           <MenuItem Click="MouseLeave" ->
                                                                                                             <MenuItem Header="MouseLeave" ->
                                                                                                               <MenuItem Click="MouseLeave" ->
                                                                                                                 <MenuItem Header="MouseLeave" ->
                                                                                                                   <MenuItem Click="MouseLeave" ->
                                                                                                                     <MenuItem Header="MouseLeave" ->
                                                                                                                       <MenuItem Click="MouseLeave" ->
                                                                                                                         <MenuItem Header="MouseLeave" ->
                                                                                                                           <MenuItem Click="MouseLeave" ->
                                                                                                                             <MenuItem Header="MouseLeave" ->
                                                                                                                               <MenuItem Click="MouseLeave" ->
                                                                                                                               <MenuItem Header="MouseLeave" ->
                                                                                                                               <MenuItem Click="MouseLeave" ->
                                                                                                                               <MenuItem Header="MouseLeave" ->
                                                                                                                               <MenuItem Click="MouseLeave" ->
                                                                                                                               <MenuItem Header="MouseLeave" ->
                                                                                                                               <MenuItem Click="MouseLeave" ->
                                                                                                                               <MenuItem Header="MouseLeave" ->
                                                                                                                               <MenuItem Click="MouseLeave" ->
                                                                                                                               <MenuItem Header="MouseLeave" ->
                                                                                                                               <MenuItem Click="MouseLeave" ->
                                                                                                                               <MenuItem Header="MouseLeave" ->
................................................................

```

Há ezek parancstulajdonságok barátjai között szerepelnek hozzákapcsolni egy olyan felhasználófejlesztő-alkalmazáson, amely többféle menülelemet tartalmaz. Aholhol, hogy lassuk, hogyan történik minden, módosítva a jelenlegi menürendszerét úgy, hogy többféle menülelemet, és harom részleget adtak masolásához, beillesztéséhez és kivágásához:

Ütasítások kapcsolása a command tulajdon-sághoz

```

    }

    commandbinding.CommandBindings.Add(new CommandBinding(CommandNameCommands.Help,
        HelpCommandExecute, HelpCommandCanExecute));
}

private void SetHelpCommandBinding()
{
    HelpCommandBinding helpBinding = new CommandBinding(CommandNameCommands.Help,
        HelpCommandExecute, HelpCommandCanExecute);
}

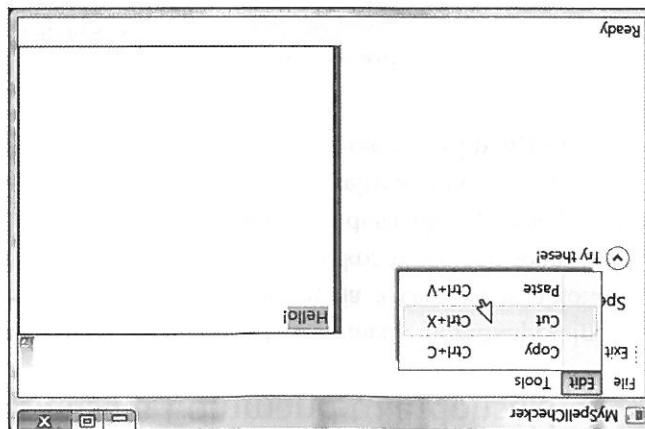
```

Help lehetőséggel mulkodjunk, amely automatikusan rögzít az F1-re: binding objektum, amelyet úgy konfigurálunk, hogy az ApplicationCommands. megnyásá után. Ez az új metódus programozott módon letrehoz egy új Command-nevű metódust, amelyet a konstruktorban hívunk meg az initialComponent() Tetelezzük fel, hogy a fülek Kodájára definiál egy új, SetHelpCommandBinding() rendszere?

felhasználó megnyomja ezt a gombot, akkor aktiválhatja a társtolt menüt-reménk, hogy az egész ablak reagáljon az F1 billentyűre, így amikor a vég-nyel, mit amennyit a XML-ben láthatunk. Például mi történik, ha azt szerekti Kodra. Nem tilásosan boncolult dolog, de egy kicsit több logikát igényel, mint amelyet a kommand tulajdonosa fog, akkor szükségesünk lesz impementálni nem támogatja a kommand tulajdonosát, amely olyan elemhez szerepelnek utasításokat kapcsolni.

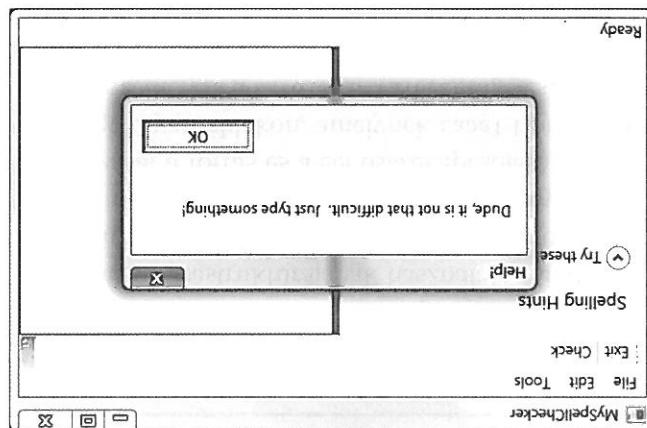
Utasítások kapcsolása a felhasználói felület

29.28. ábra: A parancsobjektumok gazdag funkciionalitását biztosítanak ingyen



29. fejezet: Programozás WF-Vezérlélemekekkel

29.29. ábra: Az egyszerű rendszerszintű



rendszerűbbet (amely nem tul hasznos, és kicsit talán sejte is). A megtekintethetők a 29.29. ábrán látható, felhasználóknak kezelt tímumentős Most futtathatók az alkalmazásunkat. Ha megnömajuk az F1 billentyűt, metódusban megjelenő „sugorendszertík” alig több, mint egy üzenteiben. hatunk, és szüksége esetén false értékkel adhatunk vissza. A HelpExecute() helyzetekben a sugorendszermek nem kellene megjeleníteni, erről gondoskod az F1 sugár megjelentetését a true érték visszadással. Azonban, ha bizonyos Itt implementáltuk a CanHelpExecute() metódust, és minden lehetővé tesszük

```
{
    private void HelpExecute(object sender, ExecutedRoutedEventArgs e)
    {
        // megakkadályoznák, hogy az utastáts végrehajtása megtörjenjen.
        // Itt beállítjuk a CanExecute false értékét, ha szükség esetén
        // megakadályoznák, hogy a CanExecute false értékét, ha szükség esetén
        // megakadályoznák, hogy az utastáts végrehajtása megtörjenjen.
        CanExecute = false;
    }
}
```

metódusok milyen formátumát követhetik meg): tipusunkhoz (folyékony meg, hogy a részről metódusreferenciák az egyes program működése alapján végrehajtsa-e vagy sem) és az Execute eseményt (amelyben meghatározható az utastáts végrehajtásakor megjelenített tartalom). Adunk hozzá a következő eseménykezelőt a window-lezármazott (amelyben meghatározható az utastáts végrehajtásakor megjelenített tartalmat). A legtöbb kommandobjektum kezelheti akárja majd a CanExecute menyt (amely lehetővé teszi, hogy az utastáts a rendszer a

Igazság szerint a WPF adatkötési infrastruktúrájának használata minden op-cionális. Ha egy féllesztő saját adatkötési logikáját alkalmazza, a forrás és a célok közötti kapcsolat általában magaban foglalja különöző események kezelését és impozitív kód letrehozását a forrás es a cél összekapcsolásához. Például, ha van egy scrollbar vagy olyan ablakon, amelynek label típuson két részletes megjelenítése eretkezik, akkor kezelhetők a scrollbar valószínűleg eseményt, amihez csak a scrollbar értékét módosítva a label tartalmát.

Megjegyzés Egy adatkötési művelet cél tulajdonoságának egy felhasználófejlesztő-elem frissítése. Segítségével mindenekkel lehet lenni.

A belső WPF adatkötési motor használata során hisztabán kell lenniuk a következőket. Amely az adattartalmat használja majd (egy checkbox, egy TextBox stb.). Míg a cella (vagy CellPoint) az a felhasználói felület vezérlőelem-tulajdonosa, így a műveletek forrása az adat (egy Boolean tulajdonoság, relations adat stb.), melyet forrásnak kell tekintenünk. Ahogy van variánsa, egy adatkötési művelet forrásnak a cella közötti különbségekkel. Ahol a műveletet a számított művelettel köti össze, a műveletet a számított művelethez köti össze.

- checkbox vezérlelőelem bejelölését az adott objektum Boolean tulajdon-számának állapota. Például:
- Adatok megjelenítését TextBox tipusokban egy relations adatbazis-tablezatból.
- Egész számhoz kapcsolt Labelt, amely a feljelzés számát mutatja egy mapban.

Vállalozó állapotát Például: Ezáltal a felhasználói interfejsz eleme megjelenítheti a kódunkban elmagyarázott adatokat, amelyek vállalzhatnak az alkalmazás eléről törökítését. Az adatkötés tulajdonképpen vezérlőelem-tulajdonoságok kapcsolásának műveletei mindenekkel gyakran célpontjai különöző adatkötési műveleteknek. A vezérlőelemek minden részét a vezérlőelemeket tartalmazza. A forrásoknak nyilvántartói lásd a Bevezetés XLV. oldalát.

A WPF adatkötési modell

Források A MySpellCheck kódjálok a forrásoknak nyilvántartja a Bevezetés XLV. fejezetének alkonytvátra tartalmazza. A forrásoknak nyilvántartói lásd a Bevezetés XLV. oldalát.

Ezen a ponton a Visual Studio tervézésének valahogy úgy kell kinezinie, mint a 29.27. ábrán látható képnek.

```
<!-- Helyezzük alulra egy Statusbar tipust -->
<Statusbar DockPanel.Dock="Bottom" Background="Beige" >
<TextBlock Name="statusBarText" Ready=</TextBlock>
<StatusbarItem>
<TextBlock Name="statusBarText" Ready=</TextBlock>
</Statusbar>
```

Közvetlenül az előző Toolbar definíció után: megegyezései az állapotávapanelekn. Adjuk a kodhoz a következő markupot: olyan kis részletekhez optimalizált, mint amilyenek a felhasználói felület lág nincs szüksége minden támogatásra, a TextBlock típus másik előnye az, hogy alaphozzott szövegét, a sortöréséket stb. Míg a Statusbar típusunknak gyakorlatilag minden TextBlock típus is szöveget tölöl. A TextBlock típusok emellett tartalmaz, amelyet a fejezetben pontítag meg nem használtunk. A Textboxok hasonlóan a TextBlock típus is szöveget tölöl. A TextBlock típusok emellett minden TextBlock típus a DockPanel-t is használja meg.

A Statusbar típus készítése

Megjegyzés A Toolbar típus tetszés szerint becsomagoltak **ToolBar** elembe, amelyetet. További részletekért forduljunk a .NET Framework 3,5 SDK dokumentációjához.

A Toolbar típus két Button típuszt tartalmaz, amelyek történetesen úgyanazokat az eseményeket kezeli, és a kodjunk úgyanazon metodusaikat kezelik az eseményeket. A modoszer segítségével összefűrészhetjük az eseménykezelőket, hogy minden eseményt kezeljük. A kezelők minden eseményt lemekezt, minden eseményt kezeljük. Hogyan kezeljük minden eseményt? Kérdezzük meg a C#-ban használt Cursor tulajdonság rövén.

hogy befejezzük az általunk felhasználói felületeink definícióját: <DocPane1> bal oldalához kerül. Adjuk hozzá a következő XAML-márkupot, sort támogat, és engedélyezte a helyesírás-ellenőrzést. A teljes <Grid> a szűlő lyerstráj javaslatok listáját. A jobb oldalon lesz egy TextBox típus, amelyet több Expander típus, amely <StackPanel> típusba csomagolva megjelenít a hégrid típus definíálása, amely két osztópot határoz meg. A bal oldalon lesz az Felhasználó felületünk tervezetének utolsó aspektusa egy olyan osztatható

A felhasználói felület véglegesítése

29.27. ábra: Helyismerellemez - alkalmazásunk aktuális felhasználói felülete



29. Fejezet: Programozás WPF-vézetőlemekekkel

```

        txtData.GetSpellLiningError(txtData, CaretIndex);
    SpellLiningError error =
    // Helyenél.

    // Probálunk helyesírást hibát keresni a kurzor aktuális
    string spellLiningHints = string.Empty;
    }

    protected void ToolsSpellLiningHints_Click(object sender,
        ...
    }

    public partial class MainWindow : System.Windows.Window
    kod, amely a fejzett ezek pontján már nem igényel sok magyarázatot:
    csak a megmaradt eseménykezelők számára. Itt van a szoban forgó releváns
    - A felhasználói felület imára készén van, míg implementációit kell biztosítja

```

A megvalósítás véglegesítése

```

        </grid>
    </textbox>
    BorderBrush = "Blue">
    Name = "txtData" FontSize = "14"
    AcceptsReturn = "True"
    SpellCheck.IsEnabled = "True"
    <textbox Grid.Column = "1" />
    <!-- Ez az a terület, ahova lehet majd írni -->
    <!-- Ez programozott módon töltjük fel -->
    Header = "Try these!" Margin = "10,10,10,10" />
    <expander Name = "expanderSpell">
        <!-- Ez a rész minden részben elérhető -->
        <label>
            spellLiningHints
            <label Name = "lblSpellLining" FontSize = "14" Margin = "10,0,0,0" />
            <stackpanel Name = "lblSpellLining" VerticalAlignment = "Stretch" />
            <grid DockPanel.Dock = "Left" Background = "AliceBlue" />
            <!-- A sorok ezek osztópont definiálása -->
            <grid.ColumnDefinitions>
                <columnDefinition />
                <columnDefinition />
                <columnDefinition />
            </grid.ColumnDefinitions>
            <grid.RowDefinitions>
                <rowDefinition Height = "5" Background = "Gray" />
                <rowDefinition Height = "5" Background = "Gray" />
                <rowDefinition Height = "5" Background = "Gray" />
            </grid.RowDefinitions>
            <gridSplitter Grid.Column = "0" Width = "5" Background = "Gray" />
            <grid>
                <grid.ColumnDefinitions>
                    <columnDefinition />
                    <columnDefinition />
                </grid.ColumnDefinitions>
                <grid>
                    <grid.RowDefinitions>
                        <rowDefinition Height = "12" />
                    </grid.RowDefinitions>
                    <label Name = "lblSpellLining" FontSize = "12" />
                </grid>
            </grid>
        </expander>
    </expander>
    <!-- Ez a rész minden részben elérhető -->
    <label Name = "lblSpellLining" FontSize = "12" />

```

taljhoz, amelyben definíálták őket. Ia. Eznelelül, a normális.NET-események szorosan kötődnek ahol az oszthatároz meg, és csak az osztály vagy annak egy leszármazottá használhat nek” tekintethetünk. Mint tudjuk, egy tipikus.NET-eseményt adott alapoztatáven támogatja azt a jelenéget, amelyet „vezérlőelem-függeléken eseményeket val foglalkozik. A Windows Presentation Foundation a vezérlőutastáskat meghosszítva során importálva kódral (és egy egészére) Kézzen vagyunk! Mindössze néhány sornyi imperektiv kódral (és egy egészére) hogy megérdekesesse tegyük, meg kell ismerkednünk a vezérlő utastáskakkal.

Kézzen vagyunk! Mindössze néhány sornyi imperektiv kódral (és egy egészére) ges adag XAML-lel) elkezültek egy mulkodó szövegszerkezettel. Ahhoz,

```

        }
        {
        startBarText.Text = "Show Spelling suggestions";
        {
        protected void MouseLeaveArea(object sender, RoutedEventArgs args)
        {
        startBarText.Text = "Exit the Application";
        {
        protected void MouseEnterArea(object sender,
        RoutedEventArgs args)
        {
        expanderspellling.IsEnabled = true;
        // Boncsuk ki a bővítfület.
        // Mutasuk meg a javaslatokat a Label es az Expander
        {
        foreach (string s in error.Suggestions)
        {
        spellingHints += string.Format("{0}\n", s);
        }
        spellingHints.Content = spellingHints;
        // tipuson belül.
        // Készítsek el a helyesírást javaslatok sztringjét.
        if (error != null)
    
```

Bár a saját interfészimplacíonkát is megalakothatunk egy vezérlőtől, amely támogatja a WPF tövábbított esemény modellet. Felületi kommand típusát, amelyet a statikus osztályok több olyan tulajdonoságot definiálnak, amelyek dést. Ezek a statikus osztályok minden tulajdonoságot biztosítják ezet a működéshez, kicsi annak az esetére, hogy erre szüksége van, kiszönbetűen az azon-

```
{
    void Execute(object parameter);
}

// Definíciója a paramécs indításákor megfizetendő metódust.

bool CanExecute(object parameter);
// Végrehajtható-e aktuális állapotban.

// Definíciója azt a metódust, amely meghatározza, hogy a paramécs event EventHanlder CanExecuteChange;
// a paramécs végrehajtásra kerüljön-e vagy sem.

// Akkor fordul elő, amikor a módszertől bekövetkezik, hogy
}

public interface ICommand
{
    ICommand interfész objektumot
    olván tulajdonoságot (gyakran Command néven), amely viszazzád egy írt látható, szemponthatól egy WPF-vezérlőtől (vagy más beviteli lehetőséget). Programozási bonyolultsága miatt a WPF-vezérlőtől csak a Windows Form (mivel a Windows Form) szabványos eseményeket biztosítanak ilyen célra, a végéredmény általában más felhasználói felületek eseményeit (ezeket a Windows Form)
```

A WPF számos bővíthető utasításai között, amelyek mindenekikhez közelítethetők. A végéredmény általában kisebb esetleges kommand interfész objektumot olván tulajdonoságot (gyakran Command néven), amely viszazzád egy írt látható, szemponthatól egy WPF-vezérlőtől (vagy más beviteli lehetőséget). Programozási bonyolultsága miatt a WPF-vezérlőtől csak a Windows Form (mivel a Windows Form)

A belső vezérlőelem paramécsök

Míg más felhasználói felületek eszközrendszerek (mint a Windows Form) szabványos eseményeket biztosítanak ilyen célra, a végéredmény általában más felhasználói felületek eseményeit (ezeket a Windows Form) közelítve meg, eszközökkel, egyszerű gombokkal, valamint gyorsbillentyűkkel (Ctrl + C, Ctrl + V stb.). Ies köreiben lehet alkalmazni kiilónéle felhasználói felületek elemekre (menülemekek), amelyeket szélelűen támogatja a műsorok, a beillesztes és kivágás paramécsökkel. A WPF például támogatja a műsorok, a beillesztes és kivágás paramécsökkel. A WPF például támogatja a műsorok, a beillesztes és kivágás paramécsökkel. A WPF például támogatja a műsorok, a beillesztes és kivágás paramécsökkel. A WPF például támogatja a műsorok, a beillesztes és kivágás paramécsökkel.

29.4. táblázat: A belső WPF-vezérlőelem parancsobjektumok

A WPF-vezérlőelem	Parancsobjektum	Jelentés
Applikációjának parancsai	CLOSE, COPY, CUT, PASTE, FIND, OPEN, MOVEDOWN, MOVEUP, CHANNELDOWN, CHANNELUP, BOOSTBASE, BROWSEBACK, BROWSEFORWARD, NAVIGATIONCOMMANDS, EDITINGCOMMANDS	.NET Framework 3.5 SDK dokumentációját).
Componentek parancsai	AZOKAT A TULAJDONSAGGOKAT, MOVEDOWN, MOVEUP, CHANNELDOWN, CHANNELUP, BOOSTBASE, BROWSEBACK, BROWSEFORWARD, NAVIGATIONCOMMANDS, EDITINGCOMMANDS	Tárol olyan tulajdonsságokat, hogy hozzájárhatók legyenek, amelyek a törzszel, vagy különöző végrehajtott közzök után adjának ki.
Media parancsai	OLYAN TULAJDONSSÁGGOKAT HÁRÍTÓK, CHANNELDOWN, CHANNELUP, BOOSTBASE, CHANNELUP, CHANNELDOWN, BROWSEBACK, BROWSEFORWARD, NAVIGATIONCOMMANDS, EDITINGCOMMANDS	Elérhetők lesznek a WPF definíciók módjára, ha mindenek által használt alkalmazások esetében használhatók.
Navigációs parancsai	TÖBB OLYAN TULAJDONSSÁGGOKAT HÁRÍTÓK, FAVORITES, LASTPAGE, NEXTPAGE, ZOOM, NAVIGATIONCOMMANDS, EDITINGCOMMANDS	Definiál, amelyek a WPF használói felület ki-
Edzőparancsai	OLYAN TULAJDONSSÁGGOKAT DEFINÍL, AMELYEKET ÁLLÍTALAKBAN CORRECTSPELLINGERROR, DECREASESFONTSIZE, INCREASESFONTSIZE, WPFDOKUMENTUM API ALATT, ENTERRAPARAGRAPHBREAK, TALFELETRÍNEBREAK, WPFDOKUMENTUM API ALATT, MOVEWORD, MOVEBOLD, MOVEUNDERLINE, MOVEPARAGRAPHBREAK, ENTERRAPARAGRAPHBREAK, TALFELETRÍNEBREAK, WPFDOKUMENTUM API ALATT, MOVERIGHTBYWORD, TÉNHASZNÁLUNK.	Finnál, amelyeket állítalakban minden használata csak.

A 29.4. táblázat az egyes belső parancsobjektumok által kijátszott alapfüjediobjektumokat ismerteti neheányat (további részletekért tekintse el a .NET Framework 3.5 SDK dokumentációját).

Függyelük meg, hogy minden részletet rendelkezik egypt, a Command tulajdon-módon azonnal használhatók az új menülemekek. A kalkmázás, es kijelölések valamennyi szövege, akkor a 29.28. ábrán látható képes „masolás, kivágás és bővítesztés” műveletekre. Ezért, ha futtathat az intuitív grafikus felülethez, így az alkalmazás már impozáns kód nekül is megfelelő névet és gyorsbillentyűt (pl. Ctrl + C a masolás művelezhető) a meghoz rendeltetettségekkel. Ezáltal a menülemelek automatikusan megkapják a sajátos rendeltetettségeket.

```

</Menu>
</MenuItem>
<Click = "ToolsSpellingHintsClick"/>
<MouseLeave = "MouseLeaveArea"/>
<MouseEnter = "MouseEnterToolSHintsArea"/>
<Menuitem Header = "Spelling Hints">
<MenuItem Header = "Tools">
</MenuItem>
<MenuItem Command = "ApplicationCommands.Paste"/>
<MenuItem Command = "ApplicationCommands.Cut"/>
<MenuItem Command = "ApplicationCommands.Copy"/>
<MenuItem Header = "Edit">
<!-- Új menületet parancsokkal! -->
</MenuItem>
<Separator/>
<MenuItem Header = "File" Click = "FileExitClick">
<BorderBrush = "Black">
<HorizontalAlignment="Left" Background="White">
<Menu DockPanel.Dock = "Top">

```

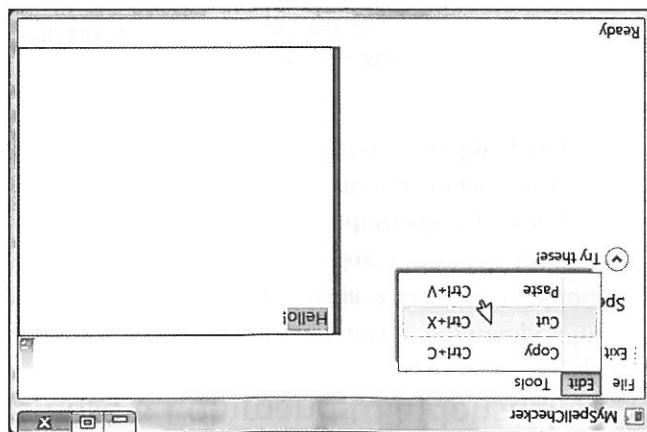
Azaz, hogy lassuk, hogy támogasson egypt Edit névű új felső menülemet, és hárrom rendszert ügy, hogy minden részletet a jelenlegi menüben elhelyezzék a szöveges adatok masolásához, bővíteszéhez és kivágásához: (ilyen például egy button vagy egy menütem), akkor nincs sok területünk, amely támogatja a Command tulajdonnal. Ha ezek parancs tulajdonai között a baromeiyiket szeretnénk hozzákapcsolni egypt részleteket a szöveges adatok masolásához, bővíteszéhez és kivágásához:

Utasiások kapcsolása a Command tulajdonához

Teljesen más a felület, mint a Windows alkalmazásoké. Ez az új felületet úgy konfigurálunk, hogy az AppliicationCommands megnyássá után. Ez az új metódus programozott módon leterhözésre így új Command rendszert? rendszert, amelyet a kodolási általában hivatalos meg az IntializeComponent O meghívásával, amelyet a konstruktorban definiált érte. Azt az AppliicationCommands-t, hogy a főablak Kodolája definíciója így új, setApplicationCommandBinding() meghívásával meghívásával meghívja ezt a gombot, akkor aktiválhatja a társtort menti- nek, mivel mindenkit a XML-been láthatunk. Például mi történik, ha azt szerepelni, hogy az egész ablak reagáljon az F1 billentyűre, így mindenki a vég- rövidítéshez elérhető. Nem tiltságosan bonyolult dollog, de egy kicsit több logikát igényel, mint amennyit a XAML-been szeretnénk. Ha azonban a felületet a Kódra. Amely nem támogatja a Command tulajdonaságot, akkor szükségesünk lesz impementálni a fehér felületet olyan elemehez szerelemeket, utasítást kapcsolni.

Utasítások kapcsolása a fehér felület

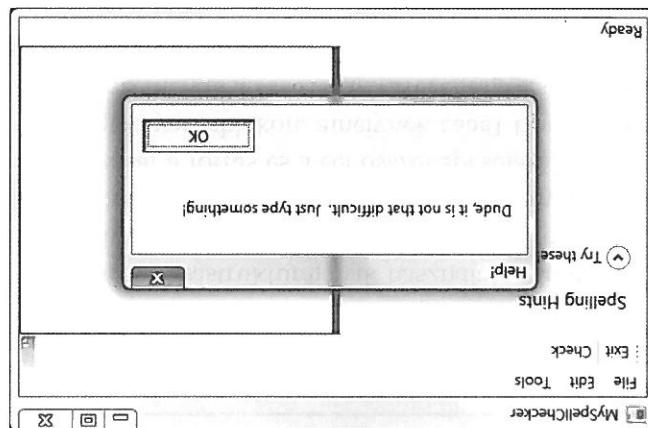
29.28. ábra: A parancsobjektumok közötti funkcionális biztosításnak ingyen



29. fejezet: Programozás WPF-vézetőlemekek

```
{
    commandbinding.Add(helpbinding);
    helpbinding.Executed += CanHelpExecute;
    helpbinding.Canceled += CanHelpExecute;
    new CommandBinding(ApplicationCommands.Help) {
        CommandBinding.CanExecute += CanHelpExecute;
        CommandBinding.Execute += Help();
    };
}
```

29.29. ábra: Az egységi sugorakészítők



rendszertípusokat (amely nem tul hasznos, es kicsit talán sejtő is). Még többet megtudhatunk a 29.29. ábrán Látható, felhasználóknak készült önművekkel. Most futtathatók az alkalmazásunkat. Ha megnyomjuk az F1 billentyűt, metódusban megjelenő „sugorakészítők” alig több, mint egy szövegben. hatunk, és szüksége esetén false értékkel adhatunk vissza. A HelpExecute() hellyeztetében a sugorakészítmények nem kellene megjelenítenie, erről gondoskod az F1 sugár megjelenítését a true érték visszadássaval. Azonban, ha bizonysos

itt implementálunk a CanHelpExecute() metódust, és minden lehetővé tesszi k

```
{
    private void HelpExecute(object sender, ExecutedRoutedEventArgs e)
    {
        // Itt beállítjuk a CanExecute false értékét, ha szüksége esetén
        // megakadályozunk, hogy az utasítás végrehajtása megtörjenjen.
        e.CanExecute = true;
        // megakadályozunk, hogy a szüksége esetén
        // a messagebox.Show("Dude, it is not that difficult.",
        // "Help!");, "Help!");
    }
}
```

metódusok milyen formátumát követhetik meg:

(amelyben meghatározhatunk az utasítás végrehajtásakor megjelenített tartalomat). Adunk hozzá a következő eseménykezelőt a window-lemezről:

```

private void CanHelpExecute(object sender,
                           CanExecuteRoutedEventArgs e)
{
    // Itt beállítjuk a CanExecute false értékét
    // megakadályozva az utasítás végrehajtását-e vagy sem)
    e.CanExecute = true;
    // megakadályozva az utasítás végrehajtását-e vagy sem)
    // a program működése alapján végrehajtsa-e vagy sem) es az Execute eseményt
    // menyt (amely lehetővé teszi, hogy az utasítást a rendszer a
    // rendszerű kommandobjektum kezelni akarja majd a CanExecute esete-
```

Igazság szerint a WPF adatkötési infrastruktúrájának használata minden op-
es megtelően frissítésekkel a label tartalmát.
megeléntenie erreket, akkor kezeltetik a scroll bar valószínűleg eseményét,
dául, ha van egy scroll bar olyan ablakon, amelynek label típuson kér-
lését es impáratív kod letrehozását a forrás es a cél összekapcsolásához. Pé-
cel közötti kapcsolat általában magabán foglalja ki többöző események keze-
cionalis. Ha egy fejlesztő a saját adatkötési logikáját alkalmazna, a forrás es a

ségi tulajdonosának kell lennie.

Megjegyzés Egy adatkötési művelet cél tulajdonosának egy felhasználói felület-elem függő-

amely az adattartalmat használja majd (egy checkbox, egy TextBox stb.).
míg a cél (vagy célpont) az a felhasználói felület vezérlőelem-tulajdonosa,
si művelet forrása az adat (egy Boolean tulajdonoság, relations adat stb.),
művelet forrása es célja között többeseggel. Ahogyan várhatóuk, egy adatköté-
A belső WPF adatkötési motor használata során tisztaban kell lenniük a kötési
mappában.

- Egész számhoz kapcsolt labelt, amely a folyók számát mutatja egy tablázatból.
- Adatok megeléntését TextBox típusokban egy relations adatbázis-
ságának alapján.
- Checkbox vezérlőelem bejelölését az adott objektum Boolean tulajdon-
valtozó állapotát, például:

A vezérlőelemek gyakran célpontjai többöző adatkötési műveleteknek. Az adatkötés tulajdonkeppen vezérlőelem-tulajdonoságok kapcsolásának művelete
olyan adattertekhez, amelyek valtozhatnak az alkalmazás ellettartama so-
rán. Ezáltal a felhasználói interfejs elemre megeléntetheti a kodunkban egy
adatkötés tulajdonkeppen vezérlőelem-tulajdonosai közötti kölcsönnytár-
tartalmazzák. A források közötti kölcsönnytárrol lásd a Bevezetés xl. oldalat.

A WPF adatkötési modell

Források A MySpellCheckek kódjaikat a források közötti kölcsönnytár 29. fejezetének alkonyvtára tartalmazza. A források közötti kölcsönnytárrol lásd a Bevezetés xl. oldalat.

Ezen a Ponton a Visual Studio tervézésének valahogyan így kell kímezni, mint a 29.27. ábrán látható képnek.

```
<!-- Helyezzük alulra egy statusbar tipust -->
<statusbar dockpanel="Bottom" BackGround="Beige" >
  <textblock DockPanelName="StatusBarText">Ready</textblock>
  <statusbaritem>
    <textblock Name="StatusBarItemText">TextBlock</textblock>
  </statusbaritem>
</statusbar>
```

A statusbar típus a `DocKPanel` tipus a részére kerül, és egyptelen `TextBlock` típusát köszönteni az eljöző toolbar definíció után:

megjegyzés az állapotstáppanellen. Adjuk a kodhoz a következő markupot, olyan kis fülszövegekhez optimalizált, mint amilyenek a felhasználói felület lág rövid szakaszébe ilyen formagátásra, a textblock típus másik előnye az, hogy csaláthozott szövegét, a sortöréssek stb. Míg a statusbar típusunknak gyakorlatára megfelelő többfajta szöveges jelölés használata, például a felkörver szöveget, az hasonlóan a textblock típus is szövegeset tölöl. A textblock típusok emellett tartalmaz, amelyet a fejezet ezen pontjáig még nem használtunk. A textboxok szolgáltak többfajta szöveges jelölés használata, amelyet a fejezetben írtam, mindenhol a textblock típus is szövegeset tölöl. A textblock típusok emellett támogatják többfajta szöveges jelölés használatait, például a felkörver szöveget, az amely a textblock típus a `DocKPanel` tipus a részére kerül, és egyptelen `TextBlock` típusát köszönti a felhasználónak.

A Statusbar típus készítése

Megjegyzés A Toolbar típus részében szerint becsomagolt a `ToolBar` objektumok esetében szabalyozza az elrendezést, a dokkolását és a húzás- és-dobás műveit. További részleteket forduljunk a .NET Framework 3.5 SDK dokumentációjához.

A `ToolBar` típus részében minden részére készítünk, és a kezeléshez minden részhez a `ContentControl` típusot használunk, tudunkuk kell, hogy a Toolbar típus "az egész" kontentcontrollal, ezért nyugodtan beágyazhatunk bármilyen típus a felületre (leggyakrabban listákat, képeket, grafikákat stb.). Az egyptelen érdekes Pont az, hogy a Check gomb egyptelen egérkúzzot támogat a cursor tulajdonoság révén.

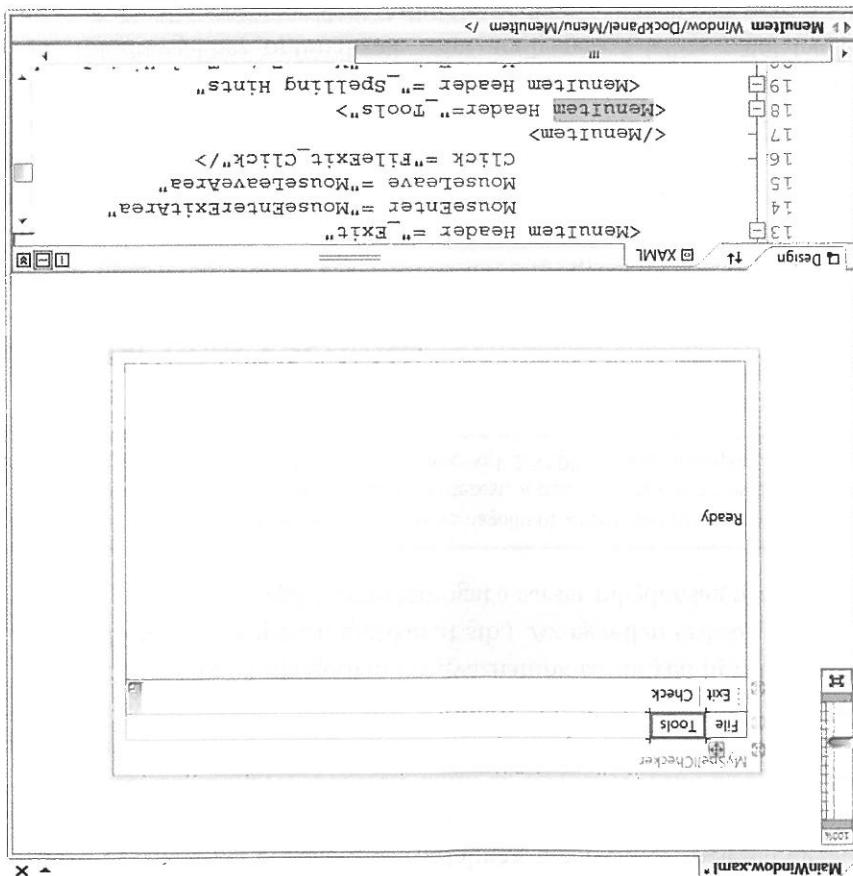
szolgáltak minden részére készítünk, és a kezeléshez minden részhez a `ContentControl` típusot használunk, tudunkuk kell, hogy a Toolbar típus "az egész" kontentcontrollal, ezért nyugodtan beágyazhatunk bármilyen típus a felületre (leggyakrabban listákat, képeket, grafikákat stb.). Az egyptelen érdekes Pont az, hogy a Check gomb egyptelen egérkúzzot támogat a cursor tulajdonoság révén.

szolgáltak minden részére készítünk, és a kezeléshez minden részhez a `ContentControl` típusot használunk, tudunkuk kell, hogy a Toolbar típus "az egész" kontentcontrollal, ezért nyugodtan beágyazhatunk bármilyen típus a felületre (leggyakrabban listákat, képeket, grafikákat stb.). Az egyptelen érdekes Pont az, hogy a Check gomb egyptelen egérkúzzot támogat a cursor tulajdonoság révén.

hogy befejezzük az ablak felhasználói felületenek definíját: <DockPanel>, bal oldalához kerül. Adjuk hozzá a következő XAML-márkupot, sort támogat, és engedélyezte a helyesírás-ellenőrzést. A teljes <Grid> szűrő létreiasztások listáját. A jobb oldalon lesz egy TextBox típus, amely több Expander típus, amely <StackPanel> típusba csomagolva megjelenít a helyesírás, amely a <MenuItem> típus definíálása, amely két osztópot határoz meg. A bal oldalon lesz az Grid típus definíciója, amely a <MouseLeave> eseménytől kivételével minden más eseménytől elszabadítja a felületet.

A felhasználói felület véglegesítése

29.27. ábra: Helyesírásellenőrző alkalmazásunk akutális felhasználói felülete



29. fejezet: Programozás WPF-vezetőelemekkel

```

        txtdata.GetSpelllingError(error);
        SpelltingError error =
        // helyenel.

        // Probabilunk helyesírást hiábát keresni a kurzor aktualis
        string spelltingHints = string.Empty;
    }

    protected void ToolsSpelllingHints_Click(object sender,
        RoutedEventArgs args)
    {
        ...
    }
}

public partial class Mainwindow : System.Windows.Window
{
    A felhasználói felület imájáról készén van, mely implementációit kell biztosítja
    // csak a meghatározott módon töltjük fel -->

    // Ez az a terület, ahol a Lehet majd írni -->

    // Ez a sorok és az oszlopok definíciója -->
    <Grid DockPanel.Dock="Left" Background="AliceBlue">
        <Grid.ColumnDefinitions>
            <ColumnDefinition />
            <ColumnDefinition />
        </Grid.ColumnDefinitions>
        <Grid.RowDefinitions>
            <RowDefinition Height="5" Background="Gray" />
            <RowDefinition Height="5" Background="Gray" />
            <RowDefinition Height="5" Background="Gray" />
            <RowDefinition Height="5" Background="Gray" />
            <RowDefinition Height="5" Background="Gray" />
        </Grid.RowDefinitions>
        <GridSplitter Grid.Column="0" Width="5" Background="Gray" />
        <StackPanel Name="tblSpelllingHints" FontSize="12" />
        <Label Name="lblSpelllingHints" Margin="10,10,10,10" />
        <Header Header="Try these!" Margin="10,10,10,10" />
        <Expander Name="expanderSpellling">
            <Label>
                <StackPanel Name="tblSpellHints" FontSize="14" Margin="10,0,0,0" />
                <Label Name="lblSpellHints" Margin="0" VerticalAlignment="Top" />
                <GridSplitter Grid.Column="0" Width="5" Background="Gray" />
                <Grid.ColumnDefinitions>
                    <ColumnDefinition />
                    <ColumnDefinition />
                </Grid.ColumnDefinitions>
                <Grid DockPanel.Dock="Left" Background="AliceBlue">
                    <Grid.ColumnDefinitions>
                        <ColumnDefinition />
                        <ColumnDefinition />
                    </Grid.ColumnDefinitions>
                    <StackPanel Name="tblStackPanel" FontSize="14" Margin="10,0,0,0" />
                    <Label Name="lblStackPanel" Margin="0" VerticalAlignment="Top" />
                    <GridSplitter Grid.Column="0" Width="5" Background="Gray" />
                    <Grid.ColumnDefinitions>
                        <ColumnDefinition />
                        <ColumnDefinition />
                    </Grid.ColumnDefinitions>
                    <Grid DockPanel.Dock="Left" Background="AliceBlue">
                        <Grid.ColumnDefinitions>
                            <ColumnDefinition />
                            <ColumnDefinition />
                        </Grid.ColumnDefinitions>
                        <StackPanel Name="tblGrid" FontSize="14" Margin="10,0,0,0" />
                        <Label Name="lblGrid" Margin="0" VerticalAlignment="Top" />
                    </Grid>
                </Grid>
            </StackPanel>
        </Grid>
    </Grid>
}

```

A megvalósítás véglegesítése

```

</Grid>
</Textbox>
        BorderBrush="Blue">
        Name="txtdata" FontSize="14"
        AcceptsReturn="True"
        SpellCheck.IsEnabled="True"
        <Textbox Grid.Column="1" />
    <!-- Ez az a terület, ahol a Lehet majd írni -->

    <!-- Ez a sorok és az oszlopok definíciója -->
    <Grid DockPanel.Dock="Left" Background="AliceBlue">
        <Grid.ColumnDefinitions>
            <ColumnDefinition />
            <ColumnDefinition />
        </Grid.ColumnDefinitions>
        <StackPanel Name="tblStackPanel" FontSize="14" Margin="10,0,0,0" />
        <Label Name="lblStackPanel" Margin="0" VerticalAlignment="Top" />
        <GridSplitter Grid.Column="0" Width="5" Background="Gray" />
        <Grid.ColumnDefinitions>
            <ColumnDefinition />
            <ColumnDefinition />
        </Grid.ColumnDefinitions>
        <Grid DockPanel.Dock="Left" Background="AliceBlue">
            <Grid.ColumnDefinitions>
                <ColumnDefinition />
                <ColumnDefinition />
            </Grid.ColumnDefinitions>
            <StackPanel Name="tblGrid" FontSize="14" Margin="10,0,0,0" />
            <Label Name="lblGrid" Margin="0" VerticalAlignment="Top" />
        </Grid>
    </Grid>
}

```

talyhoz, amelyben definíálták őket. Ja. Ezrejelül, a normális.NET-események szorosan kötődnek ahol az oszthatáros műg, és csak az osztály vagy annak részéről mázott a használhatókban. Mint tudjuk, egy tipikus.NET-eseményt adott alapszinten elnevezésben támogatja azt a jelenések, amelyet „vezérlőelem-függelék események” val foglalkozik. A Windows Presentation Foundation a vezérlőutastások révén támogatja ezeket a vezérlőutastások temakörének vizsgálatát. A fejezet következő része egyesége a vezérlőutastásokról.

A WPF vezérlőutastásai

Kézzel vagyunk! Mindössze néhány sornyi ímperiatív kódval (és egy egész széges adag XAML-jel) elérhetők egy mulkodó szövegszerkesztő alkja! Ahhoz, hogy megérdekesse tegyük, meg kell ismerkedniuk a vezérlő utasításokkal.

```

        }
    }
}

protected void MouseLeaveArea(object sender, RoutedEventArgs args)
{
    startBarText.Text = "Show Spelling suggestions";
}

protected void MouseEnterToolShintArea(object sender,
                                         RoutedEventArgs args)
{
    startBarText.Text = "Exit the Application";
}

protected void MouseEnterExitArea(object sender,
                                  RoutedEventArgs args)
{
    startBarText.Text = "Exit the Application";
}

private void Bonustuk_ki_a_bovitot()
{
    expanderspelling.IsExpanded = true;
}

// Mutassuk meg a javaslatokat a Label es az Expander
// tipuson belül.
// Készítünk el a helyesírást javaslatok sztringjét.
// Készítünk el a helyesírást javaslatok sztringjét.
if (error != null)
{
    spelling += string.Format("{0}\n", s);
}
foreach (string s in error.Suggestions)
{
    tb1SpellingHints.Content = spelling;
}
// Bonustuk ki a bovitót.
// Mutassuk meg a javaslatokat a Label es az Expander
// tipuson belül.
// Készítünk el a helyesírást javaslatok sztringjét.
// Készítünk el a helyesírást javaslatok sztringjét.
if (error != null)
{
    spelling += string.Format("{0}\n", s);
}
foreach (string s in error.Suggestions)
{
    tb1SpellingHints.Content = spelling;
}

```

RoutedeICommmand típuszt, amely támogatja a WPF vezérlőt a WPF esemény modellel. Ezzel a statikus osztályok több olyan tulajdonosától objektumokat – általában a WPF-parancsobjektumnak, amelyek biztosítják ezt a működését. Ezek a saját interfészimplimentációkat is megalakíthatnak egy vezérlőtől.

```
{
    void Execute(object parameter);
    // Definiálja a parancs indításához meghívandó metódust.

    bool CanExecute(object parameter);
    // Végrejátszható-e aktuális állapotban.

    // Definiálja azt a metódust, amely meghatározza, hogy a parancs
    // akkor fordul el, amikor a módszertásk befordítják, hogy
    // a parancs végehezjárásra kerüljön-e vagy sem.

    event EventHandler CanExecuteChanged;
    // A parancs végehezjárásra kerüljön-e vagy sem.

    public interface ICommand
    {
        // Akkor fordul el, amikor a módszertásk befordítják, hogy
        // a parancs végehezjárásra kerüljön-e vagy sem.
        // Végrehajtás előtt a módszertásk elutasítja a parancsot.
        // Ezután a parancs végehezjárásra kerüljön-e vagy sem.
        // A parancs végehezjárásra kerüljön-e vagy sem.
        void Execute(object parameter);
        bool CanExecute(object parameter);
    }
}
```

A WPF számos bővíthető vezérlő utasítással erkezik, amelyek mindenekikhez szabványos eseményeket biztosítanak (ilyen cél(ok)ról, a WPF-parancsaihoz közelítőkön kívül). A WPF-parancsok redundáns, és a kodolt nézet karbantartani. A WPF-modell alatt a parancsok általában szemponthoz kötött (gyakran kommand néven), amely viszazzad egy írt látatható, olajan tulajdonosát (gyakran kommand néven), amely támogat egy szemponthoz kötött (gyorsbillentyűt) (vagy más beviteli lehetőségeket). Programozási beállításainak gyorsbillentyűt (vagy más beviteli lehetőségeket). Programozási szempontból egy WPF-vezérlőtartásról (gyakran command néven), amely támogat egy szemponthoz kötött (gyakran kommand néven), amely támogat egy írt látatható, szemponthoz kötött (gyorsbillentyűt) (vagy más beviteli lehetőségeket).

A belső vezérlőelem parancsobjektumok

Míg más felhasználófelületek-szkozrendszerek (mint a Windows Forms) szabványos eseményeket biztosítanak (ilyen cél(ok)ról, a WPF-parancsaihoz közelítőkön kívül). A WPF számos bővíthető vezérlő utasítással erkezik, amelyek mindenekikhez szabványos eseményeket biztosítanak (ilyen cél(ok)ról, a WPF-parancsaihoz közelítőkön kívül). A WPF-parancsok redundáns, és a kodolt nézet karbantartani. A WPF-modell alatt a parancsok általában szemponthoz kötött (gyakran kommand néven), amely viszazzad egy írt látatható, olajan tulajdonosát (gyakran kommand néven), amely támogat egy szemponthoz kötött (gyorsbillentyűt) (vagy más beviteli lehetőségeket). Programozási beállításainak gyorsbillentyűt (vagy más beviteli lehetőségeket). Programozási szempontból egy WPF-vezérlőtartásról (gyakran command néven), amely támogat egy szemponthoz kötött (gyorsbillentyűt) (vagy más beviteli lehetőségeket).

Míg más felhasználófelületek-szkozrendszerek (mint a Windows Forms) szabványos eseményeket biztosítanak (ilyen cél(ok)ról, a WPF-parancsaihoz közelítőkön kívül). A WPF számos bővíthető vezérlő utasítással erkezik, amelyek mindenekikhez szabványos eseményeket biztosítanak (ilyen cél(ok)ról, a WPF-parancsaihoz közelítőkön kívül). A WPF-parancsok redundáns, és a kodolt nézet karbantartani. A WPF-modell alatt a parancsok általában szemponthoz kötött (gyakran kommand néven), amely viszazzad egy írt látatható, olajan tulajdonosát (gyakran kommand néven), amely támogat egy szemponthoz kötött (gyorsbillentyűt) (vagy más beviteli lehetőségeket). Programozási beállításainak gyorsbillentyűt (vagy más beviteli lehetőségeket). Programozási szempontból egy WPF-vezérlőtartásról (gyakran command néven), amely támogat egy szemponthoz kötött (gyorsbillentyűt) (vagy más beviteli lehetőségeket).

Míg más felhasználófelületek-szkozrendszerek (mint a Windows Forms) szabványos eseményeket biztosítanak (ilyen cél(ok)ról, a WPF-parancsaihoz közelítőkön kívül). A WPF számos bővíthető vezérlő utasítással erkezik, amelyek mindenekikhez szabványos eseményeket biztosítanak (ilyen cél(ok)ról, a WPF-parancsaihoz közelítőkön kívül). A WPF-parancsok redundáns, és a kodolt nézet karbantartani. A WPF-modell alatt a parancsok általában szemponthoz kötött (gyakran kommand néven), amely viszazzad egy írt látatható, olajan tulajdonosát (gyakran kommand néven), amely támogat egy szemponthoz kötött (gyorsbillentyűt) (vagy más beviteli lehetőségeket). Programozási beállításainak gyorsbillentyűt (vagy más beviteli lehetőségeket). Programozási szempontból egy WPF-vezérlőtartásról (gyakran command néven), amely támogat egy szemponthoz kötött (gyorsbillentyűt) (vagy más beviteli lehetőségeket).

29.4. táblázat: A belső WPF-vezérlőelem parancsobjektumok

A WPF-vezérlőelem	Parancsobjektum	Jelentés
ApplicacionCommands	Close, Copy, Cut, MoveDown, MoveLeft, Paste, Save, Open,	Olyan tulajdonságokat, hogy a hatarozza meg, amelyek a felhasználói felület elemeit sziszintű parancsokat keppen átirányítják.
ComponentCommands	MoveDown, MoveLeft, MoveFocusBack, ChannelDown, FastForward, BrowseForward, BrowseBack,	Olyan tulajdonságokat, amelyek a WPF definíciókat, hogy különöző tárózásokat lehetségesképpen előidézik.
MediaCommands	BoostBase, ChannelUp, BoostBase, ChannelUp,	Olyan tulajdonságokat, amelyek a WPF definíciókat, hogy a tárózásokat lehetségesképpen előidézik.
NavigationCommands	BrooseBack, BrooseForward, TabToBb olyan tulajdonságokat, hogy a tárózásokat lehetségesképpen előidézik.	Több olyan tulajdonságokat, hogy a tárózásokat lehetségesképpen előidézik.
EditingCommands	AlignCenter, CorrectSpellingError, DecreaseFontSize, DeleteText, InsertLineBreak, InsertParagraphBreak, InsertLineBreak, Final, AmendTextAtTheBeginning, Final, AmendTextAtTheEnd, MoveRightByWord, MoveDownByLine,	Olyan tulajdonságokat, amelyeket általában WPF-dokumentum API által implementáló programozás esetén használnak.

A 29.4. táblázat az egyes belső parancsobjektumok által kialakított alapértelmezett attribútumokat ismerteti neheanyát (további részletekért tekintse át a .NET Framework 3.5 SDK dokumentációját).

modon azonnal használhatók az új menülemekek. A kalkamazás, és kijelölések valamennyi szövege, akkor a 29.28. ábrán látható képes „masolás, kivágás és beillesztés” műveletekre. Ezért, ha futtatók az újelém grafikus felületében, így az alkalmazás már impozánsul is megfelelően nevet és gyorsbillentyűt (pl. Ctrl + C a masolás művelezet) a megfelelő rendeltetésekkel. Ezáltal a menülemekek automatikusan megkapják a sajátos rendeltetést. Ezáltal a menülemekek automatikusan megkapják a sajátos rendeltetést.

```

    </Menu>
    <MenuItem>
        Click = "ToolsSpellingHintsClick" />
        MouseLeave = "MouseLeaveArea"
        MouseEnter = "MouseEnterToolSHintsArea"
        <MenuItem Header = "Spelling Hints" />
        <MenuItem Header = "Tools" />
        </MenuItem>
        <MenuItem Command = "ApplicationCommands.Paste" />
        <MenuItem Command = "ApplicationCommands.Cut" />
        <MenuItem Command = "ApplicationCommands.Copy" />
        <MenuItem Header = "Edit" />
        <-- Új menület parancsokkal! -->
        </MenuItem>
        <MenuItem Header = "Exit" MouseEnter = "MouseEnterExitArea"
        Separator />
        <MenuItem Header = "File" Click = "FileExitClick" />
        BorderBrush = "Black"
        HorizontalAlignment = "Left" Background = "White"
        <Menu DockPanel.Dock = "Top" />
    </MenuItem>
    <-- Új menület a szöveges adatok masolásához, beillesztéséhez és kivágásához:
    rendszert így, hogy támogasson egy Edit nevű új felső menülemet, és hárrom Ahhoz, hogy lassuk, hogyan történik minden, módosításuk a jelenlegi menü- (ilyen például egy button vagy menütem), akkor nincs sok terminálonk. Olyan felhasználófejlesztők, amely támogatja a command tulajdonosságot Ha ezek parancsstílusainak baromeiyliket szeretnének hozzákapcsolni egy

```

Utasiások kapcsolása a Command tulajdonssághoz

```

    {
        commandbinding.CommandBinding.Add(helpBinding);
    }

    helpBinding.Executed += new EventHandler(Execute);
    helpBinding.CanExecute += new CanExecuteHandler(CanExecute);
}

private void SetCommandBinding()
{
    commandBinding.HelpBinding = new CommandBinding(ApplicationCommands.Help);
}

```

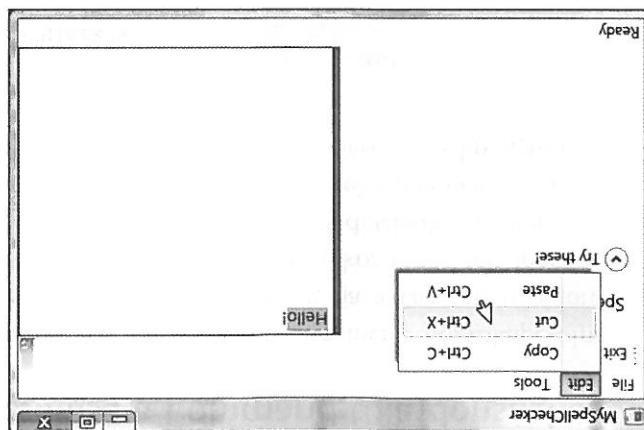
Help lehetőséggel mutatjuk, amely automatikusan reagál az F1-re: Bindings objektumot, amelyet úgy konfigurálunk, hogy az ApplicationCommands. megnyílasa után. Ez az új metódus programozott módon letrehoz egy új Command-nevű metódust, amelyet a konstruktorban hívunk meg az InitializeComponent() Tegellezzük fel, hogy a fölábbiak kodjája definíál egy új, SetCommandBinding() rendszert?

felhasználó megnyomja ezt a gombot, akkor aktiválhatja a társtort mentü-remmek, hogy az egész ablak reagáljon az F1 billentyűre, így mindenki a vég-nyel, mint amennyit a XML-ben láthatunk. Például mi töreink, ha azt szerektől kódra. Nem tiltságosan bonyolult dollog, de egy kicsit több logikát igényel, mint amelyt a command tulajdonsságot, akkor szíksegíink lesz ímpe-

Ha a felhasználó felülte olyan elemhez szeretnénk utasítást kapcsolni,

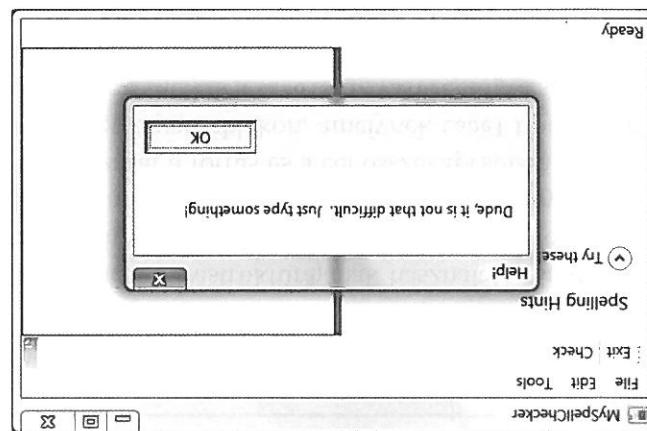
Utasítások kapcsolása a felhasználói felület

29.28. ábra: A parancsobjektumok közötti funkcionális biztosítanak integrációt.



29. fejezet: Programozás WPF-vézetőlemekekkel

29.29. ábra: Az egységes sugárzendőszervítek



rendszerünkön (amely nem tul hasznos, és kicsit talán sejtő is). Most futtathatók a 29.29. ábrán látható, felhasználóknak készült ütműtató-metódusban megjelenő „sugárzendőszervítek” alig több, mint egy üzemetdoboz, hatunk, és szüksége esetén false ertéket adhatunk vissza. A HelpExecute() helyzetekben a sugárzendőszermek nem kellene megjelennie, erről gondoskod-t az F1 sugár megjelentetését a true erték visszadásával. Azonban, ha bizonyos kiegészítőkben a CanHandleExecute() metódust, és minden lehetővé tesziük

```
{
    private void HelpExecute(object sender, ExecutedRoutedEventArgs e)
    {
        // Itt beállítjuk a CanExecute false ertéket, ha szükség esetén
        // megakadályoznánk, hogy az utasítás végrehajtása megtörjenjen.
        e.CanExecute = true;
        // megjelenítjük a CanExecute false ertéket, ha szükség esetén
        // messagebox.Show("Dude, it is not that difficult.");
        MessageBox.Show("Dude, just type something!", "Help!");
    }
}
```

metódusok milyen formátumát követhetik meg: tipusunkhoz (folyékony meg, hogy a részről metodusrérenekik az ilyes talmat). Adunk hozzá a következő eseménykezelőket a window-lezármaztatásra (amelyben meghatározhatunk az utasítás végrehajtásakor megjelenített tar-rogram működését alapján végrehajtsa-e vagy sem) és az Execute eseményt (amely lehetővé teszi, hogy megadjuk, hogy az utasítás a rendszer a menyt (amely a kommandobjektum kezelni akarja majd a CanExecute ese-

Igazság szerint a WPF adatkötési infrastruktúrájának használata minden op-
csionális. Ha egy feljelzés a saját adatkötési logikáját alkalmazna, a forrás es a
cel közötti kapcsolat általában magabán foglalna különöző események keze-
lését és impérativ kód letervezését a forrás es a cél összekapcsolásához. Tel-
dául, ha van egy ScrollBar egy olyan ablakon, amelynek label típuson két
megjelenítendő értéket, akkor kezelhetők a ScrollBar valószínűleg eseményét,
és megfelelően frissíthetők a label tartalmát.

Megjegyzés Egy adatkötési művelet cél tulajdonoságának egy felhasználói felület-elem függő.
ségi tulajdonoságának kell lennie.

A belső WPF adatkötési motor használata során tisztaban kell lenniük a kötési
műveleteknek az *cella* közötti különbségeit. Ahogyan várhatóuk, egy adatkötési
művelet forrása az adat (egy Boolean tulajdonoság, relations adat stb.),
mely a cél (vagy clipboard) az a felhasználói felületi vezetőjellem-tulajdonoság,
ami a forrás maga az adat (egy checkbox, TextBox stb.).

- Egyes számhoz kapcsolt label, amely a feljök számát mutatja egy tablázatból.
- Adatok megjelenítését TextBox típusokban egy relations adatbázis-sággával, a legelső objektum Boolean tulajdon-
- checkbox vezérlőelem bejelölését az adott objektum Boolean tulajdon-
val töző állapotához, például:

A vezérlőelemek gyakran célpontjai különöző adatkötési műveleteknek. Az adatkötés tulajdonkeppen vezérlőelem-tulajdonoságok kapcsolásának művelete
olyan adatertekhez, amelyek valtozhatnak az alkalmazás ellettertama so-
rán. Ezáltal a felhasználói interfejs elemre megjelenítheti a kodunkban egy
adatkötés tulajdonkeppen vezérlőelem-tulajdonoság xlvi. oldalát.

A WPF adatkötési modell

Források A MySpellCheckek kódjáhozkat a forrásokonyvtárat lásd a Bevezetés xlvi. oldalát.

Vagyíuk észre, hogy a <scrollbar> típusú (amelynek a mySB nevet adtuk) a 0 forráslelem megszerezni kívánt tulajdonságát. rendelünk hozzá. Itt az ElementName érték kepviselet az adatkötési művelet teszi, a {binding} jelölő kifejezés, amelyet a label Content tulajdonságához tömítük minden részül az aktuális ertékkel. Az a „ragaszto”, amely ezt lehetővé teszi csiszoláját (vagy rakkintunk a bála vagy jobbra nyílra), a label es 100 közötti tartományal konfigurálunk. Ahogy újrapozicionálunk a görge- és 100 közötti tartományal konfigurálunk. Ahogy újrapozicionálunk a görge-

```

</window>
</stackpanel>
</>
Content = "[Binding ElementName=mySB, Path=Value]"
<label Height="30" BorderBrush="Blue" BorderThickness="2"
      MaxWidth="100" LargeChange="1" SmallChange="1"/>
<!-- A címke tartalomérteke az adatkötés célja -->
<!-- A görgetősáv értéke az adatkötés forrása -->
<label Content="Move the scroll bar to see the current value"/>
<stackpanel Width="250">
</stackpanel>
<window x:Class="SimpleDataBinding.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Title="Simple Data Binding" Height="300" Width="300">
    WindowsStartUpLocation="CenterScreen"
    <!-- A görgetősáv értéke az adatkötés forrása -->
    <!-- A címke tartalomérteke az adatkötés célja -->
    Content = "[Binding ElementName=mySB, Path=Value]"
    <!-- A görgetősáv értéke az adatkötés forrása -->
    <!-- A címke tartalomérteke az adatkötés célja -->
    <label Height="30" BorderBrush="Blue" BorderThickness="2"
          MaxWidth="100" LargeChange="1" SmallChange="1"/>
    <!-- A görgetősáv értéke az adatkötés forrása -->
    <!-- A címke tartalomérteke az adatkötés célja -->
    <label Content="Move the scroll bar to see the current value"/>
</window>

```

amely a következő maradványt hagyotta meg egy window típus számára: hogy van egy új WPF-alkalmazás projektünk (SimpleDataBinding névre), kezdjük el a WPF adatkötési lehetőségeimelő vizsgálatát, és tettezzük fel,

Ismrekedés az adatkötéssel

Azonnalban a WPF-adatkötés használatákor közvetlenül XAML-ben (vagy C#-kod használatával a kodoljban) kapcsolhatók össze a forrás a célt kellekül, hogy kezelniük kellene különöző eseményeket, vagy kódolunk kevésbé a kapcsolatot a forrás és a cél között. Attól függően, hogyán állítottuk fel az adatkötési logikát, biztosíthatók, hogy a forrás es a cél szinkronban marad, ha valamelyik ertékei megváltoznak.

```

<!-- Jegyelők meg, hogy a StackPanel belül tűj a Datacontext
<Label Content="Move the scroll bar to see the current value"/>
<StackPanel Width="250" Datacontext = "{Binding ElementName=mysb}">
    <!-- tulajdonságok -->

```

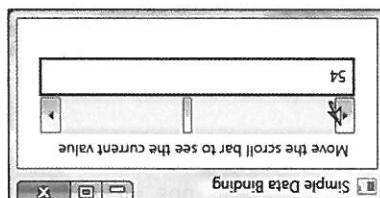
XAML-definíciót a jelentégi `<StackPanel>` tagolnál:

érteket kene ismétlünk több vezetőelemmel. Vizsgájuk meg az alábbi részletet ahelyett, hogy rönteg redundáns „`{Binding ElementName=x, Path=y}`” XAML-don könnyedén beállíthatjuk úgyanazt az adattartast vezetőlemelek családjára. Mert függéségi tulajdonság, ezért az ertereket öröklheti a részleteimetől. Ily módon könnyedén beállíthatjuk úgyanazt az adattartast vezetőlemelek családjára. Ez a tulajdonság nagyon hasznos lehet, hogy mikor lenne szüksegünk a Datacontext-tulajdonság kizárolagos beállításra. Ez a ponton elgondolkozhatunk, hogy mikor lenne szükségünk a Datacontext-es céljának megújításra, amelyek úgyanazt a kiemelést eredményeztek.

A jelentégi példánkban láthatunk, hogy a `Content` tulajdonság a `Label` típusú adatait frissítve, ha a scroll bar-t meglévőkkel szemben a `Path`-kódot kellené irunk (lásd a 29.30. ábrát).

A Datacontext tulajdonság

29.30. ábra: A ScrollBar érték kötése Label tipushoz



Ha futtatunk ezt az alkalmazást, akkor nagy előreminőségekkel fogunk (lásd a 29.30. ábrát).

```

<!-- Szeretnél az objektumot/ertereket a Datacontexten
    Keresztl -->
<Label Height="30" BorderBrush="Blue" BorderThickness="2">
    <!-- Content = "{Binding Path=Value}" -->
    <Datacontext = "{Binding ElementName=mysb}">
        <!-- tulajdonság -->

```

Alternatív formátumként kibövíthetjük a `{Binding}` jelelő kiterjesztés által megadott ertereket úgy, hogy kizárolagosan beállíthatunk a Datacontext tulajdonságát kötési művelet forrásaként, az alábbiak szerint:

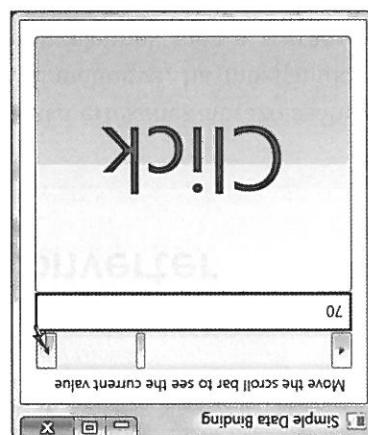
Megjegyzés Az `ElementName` és a `Path` elnevezések furcsának tűnhet, ha olyan intuitív nevet fogunk, mint a „`Source`” és a „`Destination`”. Azonban a fejezet későbbi részében látni fogunk, hogy XML-dokumentumok lehetséges gyárdoktorálásai művelet forrásai (általában az őszintűbbek), amikor a `ElementName` és a `Path` elnevezések nem mindenhol használhatók.

29. fejezet: Programozás WPF-vezetőlemekekkel

A datatkötési művelet leterhezásakor valásztathatunk a Különöző működési jel a forrás. A példánkban a Label Content tulajdonságának megvaltoztatása amely azt határozza meg, hogy a Célon vezető módszertek nem befolyásol-donság eretkezik. Alapértelmezés szerint a Mode tulajdonság eretkezene, módok között, ha a Path eretkezik meghatározása során beállítuk a Mode tulaj-módok között, a Célon vezető módszert használja.

A Mode tulajdonság

29.31. ábra: A ScrollBar eretkezés Label és Button típushoz



A Datacontext tulajdonságot írt kozvetlenül a <StackPanel> típuson állítottuk be. Ezért, ha meggyűlik a Célzásokat, nemcsak az aktuális eretkezet látjuk a Label csökkent ügynemezen eretkezni megfelelően. A 29.31. ábrán láthatunk egy le-élemen, hanem azt is tapasztaljuk, hogy a Button típus merevebb, illetve hetséges kiimehetet.

```
</StackPanel>
<FontSize>={Binding Path=Value}</FontSize>
<Button Content="Click" Height="200">
    <Content>={Binding Path=Value}</Content>
<Label Height="30" BorderBrush="Blue" BorderThickness="2">
    <Content>={Binding Path=Value}</Content>
    <!-- Most a felhasználói felület elemeit egyedi módon használjak
        a görgetőségi eretkezet -->
    <MaxIMUM>=100 LargeChange="1" SmallChange="1"/>
<ScrollBar Orientation="Horizontal" Height="30" Name="MSB">
```

Megjegyzés Noha bárminely adatkötési művelet megalakításához puasztaan impozáns kód használható, a következő példák a XAML segítségével konvertációkat az adattípusokat. Ehhez szükséges latával, a megoldásokat mindenki által elérhetően kell készíteni.

Először is a ValueConverter osztályt kell létrehozni. Ez az implementálja a System.Windows.IValueConverter interfészetet. Ez az interface a ValueChanged eseményt kezeli, amelyet a Valueproperty tulajdonságban megadott típusú érték változásakor. A Valueproperty tulajdonságban megadott típus double típusú értékkel alakítja a Valueproperty tulajdonságban megadott típusú értéket. Ezáltal a Valueproperty tulajdonságban megadott típusú érték a Valueproperty tulajdonságban megadott típusú értékkel megegyezik.

A datatálatkítás az IValueConverter segítségével

Megjegyzés A Mode tulajdonságot OneTime értékre is beállíthatjuk. Ezzel beállíthatjuk a címtáblának értékét minden billentyűzetről, de a többi váltóztatásokat a rendszer nem követheti nyomára.

```
<Textbox Height="30" BorderBrush="Blue" BorderThickness="2" Text = "[Binding Path=Value]"/>
```

A Twoway mod használatakor bemutatásához tetelezzük fel, hogy a görge-váltóztathatók az értékkel.

Az értékkel szinkronizált tartalmának megjelenítése a Label-t az alábbi TextBox-ra cserélhető. A Twoway mod használatakor bemutatásához tetelezzük fel, hogy a görge-váltóztathatók az értékkel.

kat az adatok megjelenítéséhez a TextBoxban:

Most, hogy az osztály a helyére került, vizsgáljuk meg a következő rönbőn, mivel a szövegmezők működésének lebegőpontos számot mutatnak.

Vissza a ConvertTo() metódusba, akkor úgy tűnik, hogy a körtes nincs szinkronban a rendszer még egszer meghívja. Ha egszerűen újjel erősítik adának annak köszönhető, hogy a ConvertTo() metódust a ConvertFrom() meghívása felhasználja a Tab billentyűvel kiépített vezetőelemmel. Ez a "szabad" konverzió Textbox előmbe, és automatikusan konvertálja az egész számot (pl. 99,9) a adújk az erőket. Ezáltal beérhetünk egy lebegőpontos értékkel (pl. 99,9) a (ha engedélyeztük a kettőnyi körtesmódot). Itt egszerűen közvetlenül vissza-

A ConvertTo() metódust írjuk, ha a formák megkapha az erőket a Célbox juk a típusát, és visszadíjk az új számot.

A másikról az erőtől átdíjk a formásból (a ScrollBarból) a célmak (a TextBox

```

        }

        {
            return value;
        }

        // Kiépített TextBox objektumot.

        // A példánkban, amikor a Felhasználó Tab billentyűvel
        // írva paramétert kapunk, ehhez az átalakításba csak a bejövő objektet kell
        // bejövő paramétereit használni, a rendszer a ConvertTo() metódust hívja meg. Míg több
        // Text tulajdonságának), a rendszer a ConvertTo() metódust hívja meg.

        // A bejövő erőtől közvetlenül visszadíjk.

    }

    public object ConvertTo(object value, Type targetType)
    {
        System.Globalization.CultureInfo cultureInfo = CultureInfo.CurrentCulture;
        object result = null;

        if (targetType == typeof(double))
        {
            double resultDouble = 0.0;
            resultDouble = Convert.ToDouble(value);
            result = resultDouble;
        }
        else if (targetType == typeof(int))
        {
            int resultInt = 0;
            resultInt = Convert.ToInt32(value);
            result = resultInt;
        }
    }
}

```

Tegyük fel, hogy egsz számokat szeretnék megjeleníteni a TextBox vezetőelemben, így az alábbi osztálytípusú hozzájuk letre (mindekképpen im-re).

Portájuk a System.Windows.Data névtérrel a TextBox vezetőelemekben, így az adottakról származók szerepelnek megjeleníténi a TextBox vezetőelemekben, így az alábbi osztálytípusú hozzájuk letre (mindekképpen im-re).

statičesource nevű jelek kiterjesztéshez.

Pusunkat használja, hozzárendeltük a converter tulajdonosát egy másik, réven. A TextBox text tulajdonosát modositottuk, hogy a MyDoubleConverter-t amelyet később meg szerelhetünk a XML-fájlban a DoubleConverter tulajdonosát, dows típus erőforrászatához a MyDoubleConverter tulajdonosát, hogy példánya át, hétfő a projektünk gyökérénélre (lásd a 28. fejezetet), hozzáadtuk a Windows projekt meghatározottának olyan egyedi XML-nevére, amely leképezi

```

</Window>
</StackPanel>

FontSize = "[Binding Path=Value"]"/>
<Button Content="Click" Height="200">

    Converter={StaticResource DoubleConverter}"/>
    Text = "[Binding Path=Value",
    BorderThickness="2" Name="TextThumbValue"
    <TextBox Height="30" BorderBrush="Blue"
        a Converter tulajdonosát -->
<!-- Fügjeljük meg, hogy a {Binding} kiterjesztés most bállítja
    Maxium = "100" LargeChange="1" SmallChange="1"/>
<ScrollBar Orientation="Horizontal" Height="30" Name="MySB">

    Content="Move the scroll bar to see the
    current value"/>
<Label Content="Move the scroll bar to see the
    current value"/>
    DataContent = "[Binding ElementName=MySB]">
    StackPanel Width="250"
    objektumra -->
<!-- A panel bállítja az DataContent-et a görgetőszáv->
<Window.Resources>
    <myConverters:MyDoubleConverter x:Key="DoubleConverter"/>
    <Window.Resources>

        További részletek a 30. fejezetben -->
        definíciókat, amelyek a külcsük alapján meg szerelhetők.
        <!-- Az erőforrászatnak lehetővé teszik olyan objektumok
            a tipusunkhoz -->
            xmlns:myConverters="clr-namespace:SimpleDataBinding"
            Title="Simple Data Binding" Height="334" Width="288"
            WindowStartupLocation="CenterScreen">
            <!-- Definílani kell egy CLR-nevéret, hogy hozzáférjünk
                a tipusunkhoz -->
                xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
                <Window x:Class="SimpleDataBinding.MainWindow">
                    <Window x:Class="SimpleDataBinding.MainWindow", MainWindow>
                        <Window x:Class="SimpleDataBinding.MainWindow", MainWindow>
                            <Window x:Class="SimpleDataBinding.MainWindow", MainWindow>

```

```

</Window.Resources>
<myConverters:MyDoubleConverter x:Key="ColorConverter"/>
<myConverters:MyDoubleConverter x:Key="DoubleConverter"/>
<Window.Resources>

```

Ha a következő szerint hozzáadunk egy új tagot az erőforrászsortunkhoz:

```

    {
        return value;
    }
}

public object ConvertBack(object value, Type targetType,
    object parameter, CultureInfo culture)
{
    byte v = (byte)d;
    double d = (double)value;
    // Készítéséhez.
    // A csúszka értéknek használata egy változó zöld eset
    // A számrendszerben a számkörök közötti zöld elválasztás
    // mindenek között a számok jelentések meg a textbox objektumban.
    // Mindeközött, ha futtathunk az alkalmazásunkat, akkor azt látnak,
    // hogy csak egész számok jelentések meg a textbox objektumban.
    // WPF erőforrásrendszereinek tövábbi jellemzőit a 30. fejezetben található-
    // Adattálatkérés az ValueConverter segítségével
}

```

Konvertálás különöző adattípusok között

A WPF erőforrásrendszereinek tövábbi jellemzőit a 30. fejezetben találhatók. Mindeközött, ha futtathunk az alkalmazásunkat, akkor azt látnak, hogy csak egész számok jelentések meg a textbox objektumban.

```

        </Window>
    </Grid>
</GridDefinitions>
<RowDefinition Height="*"/>
<RowDefinition Height="Auto"/>
<GridDefinitions>
<Grid.ColumnDefinitions>
<ColumnDefinition Width="*"/>
<ColumnDefinition Width="200"/>
<Grid>
    <Grid.ColumnDefinitions>
        <Grid.ColumnDefinition Width="200"/>
    <Grid>
        Loaded="MainWindow_Loaded"
        ResizeMode="NoResize" WindowStartupLocation="CenterScreen"
        Title="Car Viemer App" Height="294" Width="502"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:x:Class="CarViemerApp.MainWindow"
    
```

írat, hogy két sort és két oszlopot tartalmazzon:
 kezeljük a MainWindow Loaded eseményt, és módosítsuk a <Grid> definíció-toztatásuk meg a kezdeti Window típusunk nevet MainWindow-ra. Ezután ViewerApp neven), majd a 28. fejezetben felvázolt Lépések segítségével val-csolhatók. Kezdiuk egy új WPF-alakmazás projektet letrehozásával (Car-gyan az egységi objektumok tulajdonságait a felhasználói felületünkhez kap-Az adatkötés következő típusa, amelyet megvizsgálunk, annak mödja, aho-

Kötés egységi objektumokhoz

Forráskód A SimpleBinding kódjához a forráskódoknnyelől lásd a Bevezetés Xvi. oldalát.

Ha meglént futtathat az alkalmazásunkat, akkor a Button színe biztosan vár-befejezzük, nézzük meg, hogy an képezhethetünk le egységi objektumokat és XML-dokumentum adatokat a felhasználófelület-retegünkre.

```

        Content={StaticResource ColorConverter}"/>
Background="{Binding Path=Value",
FontSize = "[Binding Path=Value]"


```

akkor használhatunk a különböző Button típusunk Background tulajdon-sá-gának beállításához:

29.32. ábrán látható elemek.
Ezen a ponton az alakulásnak következők így kellene kinéznie, mint a

```

<!-- A racs jobb oldali ablaktábla -->
<TableGrid GridID="1" Row="2" />
<TableGrid Name="txcarstats" BackGround="LightYellow" />
<TableGrid Column="1" Name="txcarstats" Row="2" />
<!-- A racs bal oldali ablaktábla -->
<TableGrid GridID="0" Row="2" />
<TableGrid Name="allCars" Row="2" />
<TableGrid CellSelectionMode="ListSelect" Row="1" />
<TableGrid Name="txcarstats" Row="1" />
<TableGrid GridID="0" Row="0" />
<TableGrid GridID="1" Row="0" />

```

pusban. Itt van a fennmaradó típusok definíciója:
A `<grid>` fennmaradó bal oldali része egy lista típusú tartalmazó dock-
panelel, míg a jobb oldali rész egyszerűen textblock típusú foglal magaban.
A lista típus lesz végül az egyszerű objektumok gyűjteményét magába fog-
laló adattípusi muvelet céllá, úgyhogy állítsuk be az itemsource tulajdonosa-
got a `{binding}` jelölő kiterjesztésre (a kötés forrását hamarosan kódban adjuk meg). Ahogy a felhasználó kiválaszt egy elemet a lista típusból, rögzítjük
a selectionchanged eseményt, hogy frissíthesseük a tartalmat a textblock-t.
A lista típus lesz végül az egyszerű objektumok gyűjteményét foglal magabán.

```

<!-- Menusav -->
<DockPanel>
  <Grid ColumnSpan="2" Row="0" />
  <Menu>
    <MenuItem Click="ExitApplication" />
    <Separator />
    <MenuItem Click="AddNewCarWizard" Header="New Car" />
    <MenuItem Click="Exit" Header="Exit" InputGestureText="Alt-F4" />
  </Menu>
  <Grid Column="0" Row="1" />
  <Grid Column="1" Row="1" />

```

A `<grid>` sorra tartalmaz egy menürendszer File menüvel, amelyben két
almenü van (Add New Car és Exit). Végül ezekhez, hogy kezeljük minden
exit menühöz, lehetővé teszi a menürendszer elhelyezését az első sor celláiban.
Ait + F4 billentyűkombinációt. Végül figyejük meg, hogy a grid ColumnSpan er-
téké 2, amely lehetővé teszi a menürendszer elhelyezését az első sor celláiban.

```

    public class CarList : observableCollection<Car>
    {
        namespace CarViewerApp
    }

    using System.Collections.ObjectModel;
}

using System;

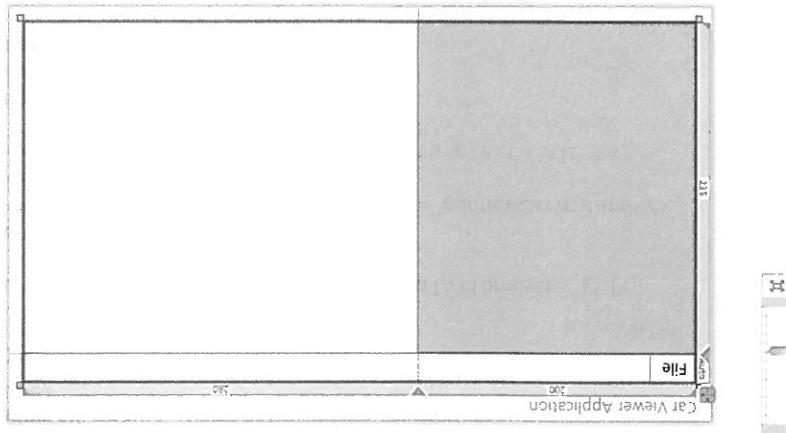
```

mint biztosítja a ToString() metódus megfelelő implementációját: **Car** típusban car típusú listaelemeket (amelyeket egyedi konstruktor használhatnak), valamint a C# automatikusan töltsön meg a lista elemeket (ezeket a típusokat minden típusnak definiálja). A Car típus ezért observableCollection-t írunk, ahol t valójában car típusú. A Car típus ezért alkalmazásunkba, amely a carlist névű osztály definiálja, amelyből bárki az keletkezik, például az adattípusi műveletek céljának. Ezután be a C#-fajlt az előnye, hogy amikor tartalma megváltozik, függetlenül a kód az eredetileg várt módon működik, nem kell átirányítani a típus használata.

A .NET 3.0 bemutatót egy új gyűjtémenytípuszt a System.Collections.ObjectModel.ObjectModel nevűben hozza létre.

Az ObservableCollection<T> típus használata

29.32. ábra: A fájlakunk felhasználói felülete



```

private void ExitApplication(object sender, RoutedEventArgs e)
{
    Application.Current.Shutdown();
}

```

Mielőtt megvalósítanánk az adattípusi logikát, véglegesítünk a File > Exit menü kezelőjét az alábbiak szerint:

29. fejezet: Programozás WPF-vézetőfelületekkel

Most már futtathatók az alkalmazásunkat, és láthatók a Listbox típusú egyszerű observable-típusban (lásd a 29.33. ábrát).

```
private void window_Loaded(object sender, RoutedEventArgs e)
{
    // Az adatkontextus beállítása.
    allCars.DataContext = myCars;
}
```

Most nyissuk meg a Mainwindow osztályunk kodjáját, és definiálunk egy CarList típusú tagvaltozót mycars nevvel. A window típus loaded eseménykézéről is elindítunk a Listbox Datacontext-be való csatlakozást.

```
public class Car
{
    public string Make { get; set; }
    public string Color { get; set; }
    public int Speed { get; set; }
    public string PetName, Color, Make, Speed;
}

public override string ToString()
{
    return string.Format("{0} the {1} {2} is going {3} MPH",
        PetName, Color, Make, Speed);
}

public void window_Loaded(object sender, RoutedEventArgs e)
{
    myCars.ItemsSource = cars;
    myCars.DataContext = myCars;
}
```

```

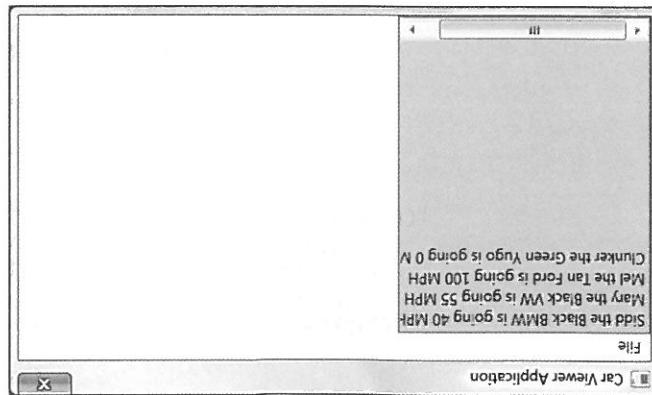
</listbox>
</listbox.ItemTemplate>
</DataTemplate>
</StackPanel>
<Text="Binding Path=PetName"/>
<TextBlock FontStyle="Italic" FontSize="14">
<Ellipse Height="10" Width="10" Fill="Blue"/>
<StackPanel Orientation="Horizontal">
<DataTemplate>
<listbox.ItemTemplate>
Background="LightBlue" ItemSource="{Binding}">
SelectOnChang="ListSelected"
Grid.RowCount="2" Name="AllCars">
<listbox Grid.Column="0">

```

tulajdonságához. Itt van a **Listbox** típus másodikott markupja. A **Listbox** jelenleg minden adattípusnak a carlitszt típusosszás elmeneket PetName-nak. A meleyet hozzákötöttünk a **CarList** típusos százsal mindeneket típusba. Az adatokat a **StackPanel**ban elhelyeztük, amelyben minden elemet felülírni olyan kapcsolt adatokat. A sablonunk a **Listbox** minden elemét meg a hozzá erősítse egy adattípusnak a saját arrol, hogyán jelenítse meg a hozzá ezáltal egyedi adatsablon keszítünk. Az adatsablon használható arra, hogy már megvizsgáltuk, hogyan állapítunk meg egyszerű körösen utvonalakat, szereün a töstríng() meghívásának az eredménye az alobjektumokon. Mivel azonban nem adtunk meg körösi utvonalat, ezért minden listabejegezés egy-szerűen a töstríng() meghívásának az eredménye az eredeti minden listabejelelő.

Egyedi adatsablon készítése

29.33. ábra: A kezdeti adattípus mintájelét

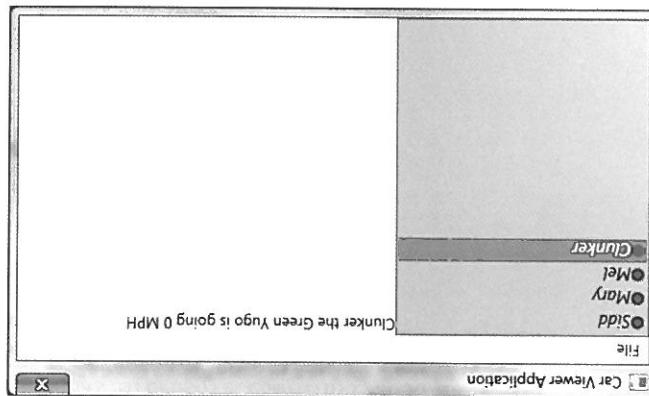


29. fejezet: Programozás WPF-vézetőelemekkel

A következő feladat olyan egyedi párhuzadbalak készítése, amely adatokat tesz objektumban. Először szürijük be a 24. fejezet NavigatiónWithIngridToXML programban egy különböző XML-fájl tartalmának megjelenítéséhez egy stílusát Litsville használ egy különböző XML-fájl tartalmának megjelenítéséhez egy stílusát Litsville pont segítségével. Válasszuk ki az elemet a Solution Explorerben, és a Projektmenü Letrehozott Inventor .xml fájl a Project > Add Existing Item menü-jelek között.

XML-dokumentumokhoz A felhasználi felület eleméinek kódjai

29.34. ábra: Adatokat egyedi adatsablonnal



Ezzel a módszerrrel most már stílusosabb különös kaptott az adatok megjelenítésére (lásd a 29.34. ábrát).

```
{
    txtCarsStats.Text = myCars[allCars.SelectedIndex].ToString();
}

// Majd a ToString() metódus meghívása.
// Gyűjteményből a Listadoobjekt kiválasztott eleme alapján.
// A megfelelő autó beolvásása az observableCollectionban
// privaté void ListitemSelected(object sender,
    SelectionChangedEventArgs e)
{
    foreach (var item in list)
    {
        if (item.ToString() == selectedCar)
        {
            item.Selected = true;
        }
    }
}
```

Itt csoportosítunk a <DataTemplate> sablonokat a Listbox tipushoz a <Listbox> eredményét, implementáljuk a Listbox tipusunk SelectionChanged kezelőjét, hogy megjelenítsük az aktuális kijelölés ToString() metódusát a leginkábbi jobb oldali Textblock tipusban:

```

    <!-- Most készítésük egy adatracsort, képezzük Le az
        attribútumokat/elemekeit az osztókra az xpath kifejezések
        használatával -->
<!-- Az XMLdataprovider használata -->
<grid.Resources>
  <grid.Resources>
    <grid.Resources>
      <grid.Resources>
        <grid.Resources>
          <grid.Resources>
            <grid.Resources>
              <grid.Resources>
                <grid.Resources>
                  <grid.Resources>
                    <grid.Resources>
                      <grid.Resources>
                        <grid.Resources>
                          <grid.Resources>
                            <grid.Resources>
                              <grid.Resources>
                                <grid.Resources>
                                  <grid.Resources>
                                    <grid.Resources>
                                      <grid.Resources>
                                        <grid.Resources>
                                          <grid.Resources>
                                            <grid.Resources>
                                              <grid.Resources>
                                                <grid.Resources>
                                                  <grid.Resources>
                                                    <grid.Resources>
                                                      <grid.Resources>
                                                        <grid.Resources>
                                                          <grid.Resources>
                                                            <grid.Resources>
                                                              <grid.Resources>
                                                                <grid.Resources>
                                                                  <grid.Resources>
                                                                    <grid.Resources>
                                                                      <grid.Resources>
                                                                        <grid.Resources>
                                                                          <grid.Resources>
                                                                            <grid.Resources>
                                                                              <grid.Resources>
                                                                                <grid.Resources>
                                                                                  <grid.Resources>
                                                                                    <grid.Resources>
                                                                                      <grid.Resources>
                                                                                      <grid.Resources>
                                                                                      <grid.Resources>
                                                                                      <grid.Resources>
                                                                                      <grid.Resources>
                                                                                      <grid.Resources>
                                                                                      <grid.Resources>
                                                                                      <grid.Resources>
                                                                                      <grid.Resources>
                                                                                      <grid.Resources>
                                                                                      <grid.Resources>
                                                                                      <grid.Resources>
                                                                                      <grid.Resources>
                                                                                      <grid.Resources>
                                                                                      <grid.Resources>
                                                                                      <grid.Resources>
                                                                                      <grid.Resources>
                                                                                      <grid.Resources>
                                                                                      <grid.Resources>
                                                                                      <grid.Resources>
                                                                                      <grid.Resources>
                                                                                      <grid.Resources>
                                                                                      <grid.Resources>
                                                                                      <grid.Resources>
                                                                                      <grid.Resources>
                                                                                      <grid.Resources>
                                                                                      <grid.Resources>
                                                                                      <grid.Resources>
                                                                                      <grid.Resources>
                                                                                      <grid.Resources>
                                                                                      <grid.Resources>
                                                                                      <grid.Resources>
                                                                                      <grid.Resources>
                                                                                      <grid.Resources>
                                                                                      <grid.Resources>
                                                                                      <grid.Resources>
                                                                                      <grid.Resources>
                                                                                      <grid.Resources>
                                                                                      <grid.Resources>
                                                                                      <grid.Resources>
                                                                                      <grid.Resources>
                                                                                      <grid.Resources>
                                                                                      <grid.Resources>
                                                                                      <grid.Resources>
                                                                                      <grid.Resources>
                                                                                      <grid.Resources>
                                                                                      <grid.Resources>
                                                                                      <grid.Resources>
                                                                                      <grid.Resources>
                                                                                      <grid.Resources>
                                                                                      <grid.Resources>
                                                                                      <grid.Resources>
                                                                                      <grid.Resources>
                                                                                      <grid.Resources>
                                                                                      <grid.Resources>
                                                                                      <grid.Resources>
                                                                                      <grid.Resources>
                                                                                      <grid.Resources>
                                                                                      <grid.Resources>
                                                                                      <grid.Resources>
                                                                                      <grid.Resources>
                                                                                      <grid.Resources>
                                                                                      <grid.Resources>
                                                                                      <grid.Resources>
                                                                                      <grid.Resources>
                                                                                      <grid.Resources>
                                                                                      <grid.Resources>
                                                                                      <grid.Resources>
                                                                                      <grid.Resources>
                                                                                      <grid.Resources>
                                                                                      <grid.Resources>
                                                                                      <grid.Resources>
                                                                                      <grid.Resources>
                                                                                      <grid.Resources>
                                                                                      <grid.Resources>
                                                                                      <grid.Resources>
                                                                                      <grid.Resources>
                                                                                      <grid.Resources>
                                                                                      <grid.Resources>
                                                                                      <grid.Resources>
                                                                                      <grid.Resources>
                                                                                      <grid.Resources>
                                                                                      <grid.Resources>
                                                                                      <grid.Resources>
                                                                                      <grid.Resources>
                                                                                      <grid.Resources>
                                                                                      <grid.Resources>
                                                                                      <grid.Resources>
                                                                                      <grid.Resources>
                                                                                      <grid.Resources>
                                                                                      <grid.Resources>
                                                                                      <grid.Resources>
                                                                                      <grid.Resources>
                                                                                      <grid.Resources>
                                                                                      <grid.Resources>
                                                                                      <grid.Resources>
                                                                                      <grid.Resources>
................................................................

```

Szűrjunk be új WPF Window típusú projektet (AddNewCarDialog néven) a Visual Studio 2008 Project > Add Window menüpontjának használatával. Az új Window jelentő meg az Inventory. xml fájl tartalmát egy testre szabott hagy meghatározunk az új ablak megjelenését és működését. Íme, a teljes markup, az elemzéssel együtt:

Egyedi parbeszédablak készítése

Perthes ablak használatait alkothunk A Copy to Output Directory opciót a Copy Inventory.xml fájlt a rendszer általosítja a \bin\Debug konviktumunkba. Állways érteleke. Ez biztosítja, hogy az alkalmazásunk lefordítása során az in-

Ahhoz, hogy tajekoztatásuk a listview típus front részére front részére rendelkezzen meg, listview típus definíciójaban található. Ez az XML-dokumentumon belül a `<gridview>` elemmel kezdtően a következőként az osztófürosk esetében.

`<gridview id="gridView1" runat="server">`
`<columns>`
`<column field="Name" title="Name" width="200px" type="Text" />`
`<column field="Age" title="Age" width="100px" type="Text" />`
`</columns>`
`<gridview>`

Eloszor is, figyelemre méltó, hogy az itemsource attributum hozzárendelte a listview típus definíciójában található. A markup ígazán nagy része a listview típus definíciójában található.

Azon túl, hogy a `<gridview>` típusunkat adott merre a sorokra boncuk, a kód megvaltoztatását.

```

<gridview id="gridView1" runat="server">
    <columns>
        <column field="Name" title="Name" width="200px" type="Text" />
        <column field="Age" title="Age" width="100px" type="Text" />
    </columns>
</gridview>

```

Mivel a DialogResult alapértelmezett értéke false, ezért nem kell tennünk semmit, ha a felhasználó a Cancel gombra kattint.

```
{
    DialogResult = true;
}
private void button1_Click(object sender, RoutedEventArgs e)
```

eseménykezelőben:

az örökölt DialogResult tulajdonságot true értékre az eseményt gomb kattintási laban egy OK, egy igen vagy egy Ellogad gombra kattintásával jelz), állításuk be nélküli alkalmazna a parbeszédbablakban levő adatokat a programban (amit általában tipusos felismerőlással. Ezért, ha tájékoztatni szeretnék a hívót, hogy a felhasználó tulajdonság egy nullázható logikai értékkel működik, nem pedig egy ellenesen Windows Forms DialogResult tulajdonságával ellentétben, a WPF-modellben ez tulajdonság révén, hogy a felhasználó melegít gombra kattintott. Azonban a 27. fejezetben a WPF-parbeszédbablakok teljeskörűen használjan (lásd OK gomb kattintási eseménykezelőjét. A Windows Formsban használjan (lásd a 27. fejezetet) a DialogResult tulajdonságát a hívott a DialogResult tulajdonság részén, hogy a felhasználó mellyik gombra kattintott. Azonban a WPF-parbeszédbablakok teljeskörűen használjan (lásd a 27. fejezetet).

A DialogResult érték hozzárendelése

Végül figyejük meg, hogy ezek a button típusok meghadnak egy Tabindex elemek körüljel ter bocsztását.

es egy Margin értékkel, az utolsó lehetővé teszi a <wrapPanel> típusban levő megnyomja az Enter, illetve az Esc billentyűket.

gombok közül mellyik reagáljon a kattintási eseményre, ha a felhasználó es az IScanable tulajdonságokat. Ezek állapothák meg, hogy az ablakon levő kezelőjük az OK gomb kattintási eseményét, valamint használjuk az IsDefault tulajdonságát. Az egyetlen erdeklődésre számot tartó pont, hogy minden sorabban található a <wrapPanel> típus, amely két button (es gy leíró label) típus tartalmaz. Az egyetlen erdeklődésre számot tartó pont, hogy minden tulajdonságának címétől eltérően, a felhasználói felület befejezéséhez a <grid> Végül, de nem utolsósorban, a felhasználói felület befejezéséhez a <grid> nösti jogének használataval címétől eltérően, hogy a binding marakkupból mint XML-dokumentumban úgy, hogy a <binding> marakkupból mint XPath mi-

A megmaradó osztók egyszerűen tövább minősítik az elérési tvonalat az specifikus szintaxis használataval az attributum értékek (card) kiszedéséhez. Az ellső <grid item meg a <car> elem id attributumát XPath-