Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar

Automatizálási és Alkalmazott Informatikai Tanszék

Szoftvertechnikák laborgyakorlat

3. mérés

Felhasználói felület kialakítása

Windows Forms, dokkolás, horgonyzás,
menük, TreeView, ListView

A gyakorlatot kidolgozta: Rajacsics Tamás, Kővári Bence, Szabó Gábor
Utolsó módosítás ideje: 2016.02.27.

Tartalom

TARTALOM **2**

A GYAKORLAT CÉLJA **3**

BEVEZETŐ **3**

FELADAT 1 – „HELLO WORLD" WINFORMS TECHNOLÓGIÁVAL **3**

FELADAT 2 – ALAPFOGALMAK MEGISMERÉSE **7**

FORM - FELÜLETTERVEZÉS 7
KONTÉNER-VEZÉRLŐ HIERARCHIA 7
ÜZENETKEZELÉS 7
VISUAL STUDIO DESIGNER 8

FELADAT 3 - MENÜK **8**

FELADAT 4 – MINIEXPLORER **9**

HORGONYZÁS 9
DOKKOLÁS 10
MINIEXPLORER LOGIKA 12
FUTTATÁS 13

A gyakorlat célja

A kapcsolódó előadások:

- 3-4. előadás – Vastagkliens alkalmazások fejlesztése

A gyakorlat célja egy látványos, gyors alkalmazásfejlesztés bemutatása, mely egyben megteremti a lehetőséget a WinForms fejlesztés alapjainak elsajátítására.

Bevezető

A Rapid Application Development (RAD) elve a fejlesztési idő lerövidítését célozza meg azáltal, hogy a fejlesztés során kész komponensekkel dolgozik, integrált fejlesztő környezetet (pl. Visual Studio) és sok automatizmust alkalmaz. Fontos ugyanakkor, hogy az automatizmusok ne szűkítsék be túlzottan a fejlesztő lehetőségeit és kellő rugalmasságot adjanak neki a rendszerek testre szabásában. A következő példákban látni fogjuk, miként alkalmas minderre a Windows Forms környezet.

A Window Forms alkalmazások legfontosabb koncepcióit a tárgy negyedik előadása ismerteti. Egy Windows Forms alkalmazásban az alkalmazásunk minden ablakának egy saját osztályt kell létrehozni, mely a beépített Form osztályból származik. Erre – tipikusan a Visual Studio designerével - vezérlőket helyezünk fel, melyek a form osztályunk tagváltozói lesznek.

Tipp: a következő példákban számos generált (és emiatt hosszú) elnevezéssel fogunk találkozni. Programjaink megvalósításakor használjuk ki az automatikus kódkiegészítés (intellisense) nyújtotta lehetőségeket és ne kézzel gépeljük be az egyes elnevezéseket.

Feladat 1 – „Hello world” WinForms technológiával

A feladat során egy olyan Windows Forms alkalmazást készítünk el, ami egy egyszerű ablakban kiírja a „Hello world!” szöveget.

1. Indítsuk el a Visual Studiot
2. Hozzunk létre egy új projektet (File/New-Project, azon belül Project type: Visual C#, Template: Windows Forms Application), a project neve HelloWorldWF legyen.
3. Kattintsunk duplán a Form1.cs fájlra! Ezt követően a felületen megjelenik egy szürke ablak. Amennyiben a Solution Explorerben a Form1.cs fájl elemet kibontjuk, látni fogjuk, hogy egy Form1.designer.cs nevű fájl is tartozik hozzá.

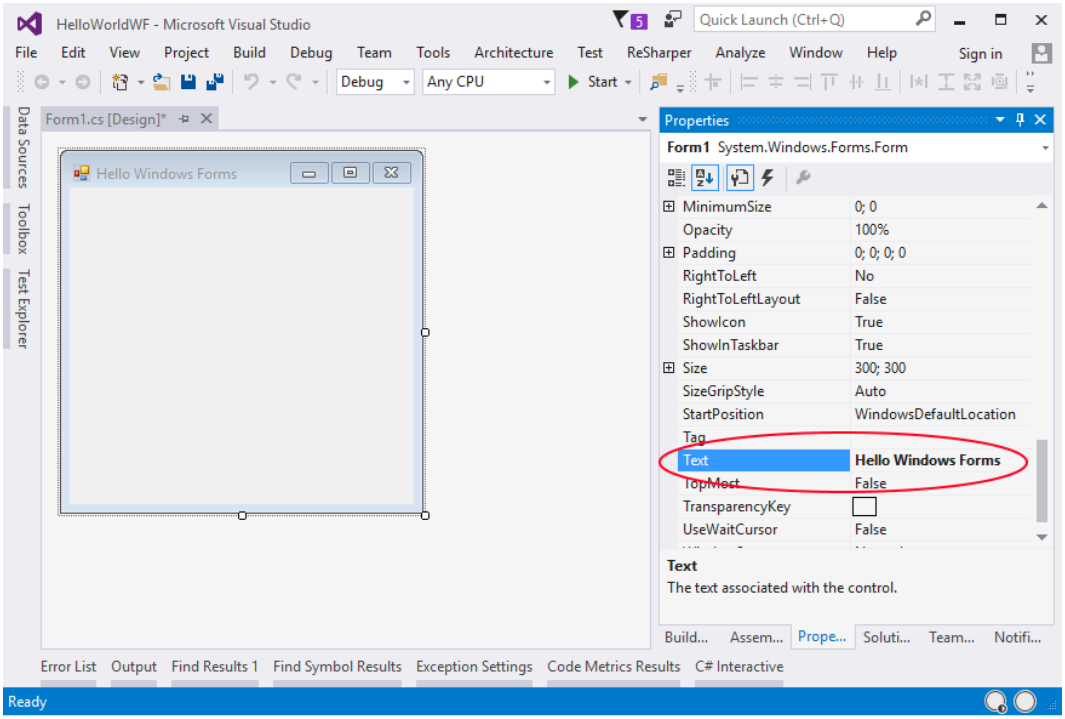
A fenti egyszerű program felépítését könnyen végigkövethetjük korábbi ismereteink alapján. A program belépési pontja itt is a `Program.cs` fájlban található `Main` függvény. A függvény létrehoz egy példányt a Form1 osztályból, majd az `Application.Run` függvény segítségével elindítja az üzenetkezelő ciklust és megjeleníti az ablakot (a Windows Forms világában „Form”-nak hívjuk az ablakokat).

A `Form1` osztály kódja két fájlban van definiálva (ezt a C# `partial` kulcsszava teszi lehetővé). A `Form1.cs` a felhasználó által kezelt kódrészleteket, míg a `Form1.designer.cs` a grafikus „Form designer” által generált kódot tartalmazza. Ez utóbbi mindig teljes szinkronban van a design nézettel, közvetlen módosítására ugyan van lehetőség, de a speciális hibaelhárítási eseteket leszámítva felesleges és kerülendő. Figyeljük meg, hogy a két fájl között alapból kapcsolat van, hiszen a `Form1` konstruktora áthív a másik fájlban definiált `InitializeComponent()` függvénybe.

Tipp: Amennyiben a Form1.cs fájlra duplán kattintunk, alapértelmezésben nem a forráskód, hanem a tervező nézet (Form designer) jelenik meg. Innen a forráskód nézetre a felületen jobb klikkelve, a „view source” menüponttal, vagy az F7 billentyű megnyomásával juthatunk.

Elképzelhető, hogy megjelenik egy további, Form1.resx nevű fájl is. Ez az ablakhoz tartozó erőforrásokat (tipikusan képek, szövegek) tartalmazhatja, de a mi esetünkben most nincs jelentősége.

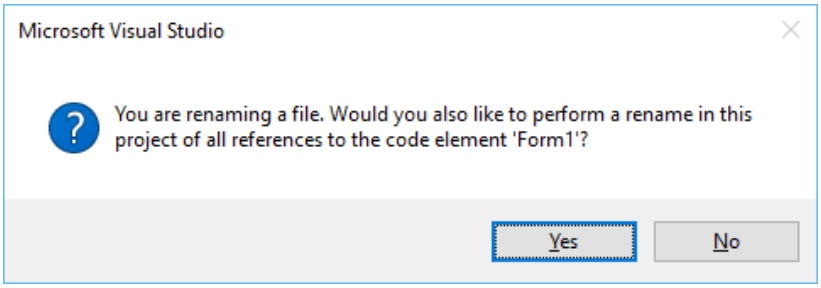
4. Kattintsunk duplán a `Form1.cs` fájlra! Ez alapértelmezésben a tervező nézetet nyitja meg.
5. Kattintsunk az űrlap hátterén, hogy az űrlap legyen kiválasztva. A Visual Studio Properties ablakában láthatjuk az űrlapunk aktuális tulajdonságait. Amennyiben a Properties ablak nem látható, az F4 billentyűvel tudjuk előcsalni (vagy View menü/Properties). A Properties ablakban keressük meg a Text tulajdonságot, és írjuk át „Hello Windows Forms”-ra: ez az űrlapunk fejlécének a szövegét állítja be.



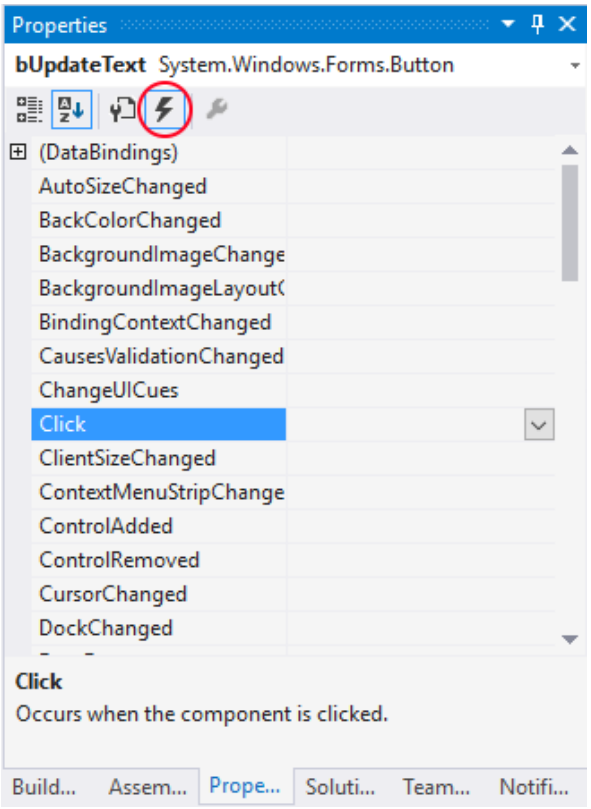
Mint látható, az űrlapunk számos tulajdonsággal rendelkezik, ezek mindegyikét a Properties ablakban be tudjuk állítani az aktuális igényeknek megfelelően.

6. Nyissuk ki a Toolbox-ot (View menü/Toolbox).

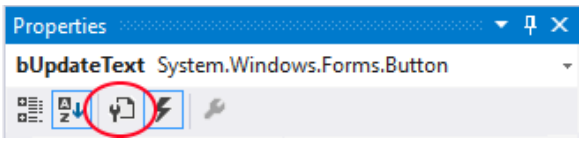
7. Húzzunk rá a form-ra egy `TextBox` és egy `Button` (gomb) vezérlőt tetszőleges helyre! (Ezeket a vezérlőket a Toolbox Common Controls csoportjában találjuk).
8. Kattintsunk egyszer a gomb vezérlőn, hogy biztosan az legyen kiválasztva a designerben. Ekkor a Properties ablakban a gombunk tulajdonságai jelennek meg. Állítsuk be a `Text` tulajdonságát „Beállít”-ra, ez a gombunk szövegét fogja ennek megfelelően beállítani.
9. Ugyanitt a Properties ablakban állítsuk be gombunk nevét, vagyis `Name` tulajdonságát „button1”-ről „bUpdateText”-re. Lényeges, hogy a vezérlőinket a funkciójuknak megfelelő nevekkel lássuk el, ez nagyban segíti a kódunk olvashatóságát. A „b” prefix a vezérlő `Button` típusára utal.
10. Az előző lépés mintájára nevezzük át a textbox vezérlőnket „tbDemoText”-re. A „tb” prefix a vezérlő `TextBox` típusára utal.
11. Az űrlapunk neve jelenleg Form1, ami szintén elég semmitmondó. Nevezzük át MainForm-ra, végül is ez az alkalmazásunk főablaka. Az átnevezést a Solution Explorerben tudjuk megtenni, itt több technikát is használhatunk. Válasszuk ki a Form1 elemet, majd még egyszer kattintsunk rajta bal gombbal: ekkor a név szerkeszthetővé válik (pont úgy dolgozunk, ahogy egy fájlt is átnevezünk Windows Explorerben). Vagy egyszerűen csak megnyomjuk az F2 billentyűt az átnevezés elindításához. Vagy akár használhatjuk a jobb gombos menü Rename funkcióját. Akárhogy is indultunk, írjuk be új névnek a MainForm.cs-t, majd nyomjuk meg az Enter billentyűt. Ekkor a Visual Studio rákérdez egy felugró ablakban, hogy minden kapcsolódó elemet nevezzen-e át ennek megfelelően: itt mindenképpen Yes-t válasszunk:



12. A következő lépésben a gombkattintás eseményt fogjuk lekezelni: ennek hatására a tbDemoText `TextBox` vezérlőbe beírjuk a „Hello” szöveget. Egy űrlap/vezérlő eseményeinek megjelenítésére is a Properties ablak szolgál, csak át kell váltsunk az eseménymegjelenítő nézetére. Ehhez a Properties ablak felső részén található villám ikonon kell kattintanunk:



Megjegyzés: a tulajdonságok megjelenítésére úgy tudunk a későbbiekben majd visszaváltani, ha a villám ikontól balra elhelyezkedő villáskulcs ikonra kattintunk (ezt egyelőre ne tegyük meg):



Az eseménylistában látható, hogy a `Button` osztálynak számos eseménye van. Számunkra most a `Click` esemény az érdekes. Erre kétféleképpen tudunk feliratkozni:

- Az eseménylistában a `Click` elemen duplán kattintunk
- Mivel a `Click` esemény a `Button` osztály alapértelemezett eseménye (ez logikus, hiszen az esetek többségében erre iratkozunk fel), a designer felületen a gombon duplán kattintva.

Válasszuk most a második lehetőséget, kattintsunk duplán a gomb vezérlőn. Ez létrehoz egy eseménykezelő függvényt, a törzsébe pedig írjuk be a következőt:

```
tbDemoText.Text = "Hello";
```

13. Futtassuk az alkalmazást (F5)! Nyomjuk meg a gombot!
14. Nézzünk bele újra a `MainForm.Designer.cs`-be. Megtaláljuk az újonnan generált kódot: az űrlapon elhelyezett vezérlőkből tagváltozók lesznek, melyek az `InitializeComponent` függvényben kerülnek inicializálásra, itt találjuk a tulajdonságaik beállítását, valamint az eseményekre való feliratkozást.

    Láthatjuk, hogy a designer csak olyan kódot generál, amit akár mi is meg tudnánk írni, de persze így egyszerűbb azt elkészíteni.

15. Az általunk beírt kód magyarázata: a `TextBox` osztály `Text` tulajdonsága beállítja a vezérlő szövegét.
16. Figyeljük meg azt is, hogy a gomb lenyomására tulajdonképpen egy eseménykezelővel iratkoztunk fel.

Feladat 2 – Alapfogalmak megismerése

Form - felülettervezés

A Form osztály az ablakot reprezentálja, és egyben a konténer-vezérlő kapcsolatban a legfelső konténer.
A felület kialakítása szempontjából az alkalmazásunk lehet:

- Dialógus alapú: Kizárólag vezérlőket helyezünk el a form-on, mintha egy dialógus ablak lenne. Ha szükséges, új ablakot nyitunk az egyes funkcióknak. Pl. Üzleti alkalmazások.
- SDI, vagy MDI: Dokumentum alapú alkalmazás, amelyben a form a dokumentum megjelenítője, és esetleg szerkesztője, az egyéb vezérlőket/funkciókat a menübe és a toolbar-ra tesszük. Az SDI (Single Document Interface) egy objektumot kezel egy időben, az MDI (Multiple Document Interface) pedig többet. Pl. szövegszerkesztők.
- Vegyes: Az ilyen jellegű alkalmazásokban a dokumentum szerkesztése a cél csakúgy, mint az SDI/MDI változatokban, azonban az ablak egy része fenn van tartva vezérlők számára, ahol könnyen elérhetjük a funkciókat. Pl. CAD alkalmazások.

Konténer-vezérlő hierarchia

Egy ablak hierarchikus (fa) felépítésű, amelyben a gyökérobjektum maga a Form. Alatta újabb konténerek lehetnek egymásba ágyazva, vagy csak egymás mellett. A hierarchia alján vannak az egyszerű vezérlők, de lehetnek vezérlő nélküli konténerek is.
Az egymásba ágyazás azért szükséges, hogy egységként lehessen kezelni a konténereket és a gyerekeiket, így például odébb húzva a konténert mennek vele a vezérlők is. A másik fontos ok a tulajdonságöröklés, amely lehetővé teszi, hogy ha megváltoztatjuk valamelyik konténer örökölhető tulajdonságát (pl. Font), akkor azt a gyerekei is megörököljék. Ez nem a szokásos objektum-orientált öröklés, de a szülő-gyermek viszony azonos elvre épül.

Üzenetkezelés

A WinForms alapú alkalmazások üzenetkezelésre épülnek, amelynek a hátterében az operációs rendszer felépítése áll. Az üzenetkezelő ciklus a Main függvényben van (Application.Run), amely csak akkor lép ki, ha bezárjuk az alkalmazásunkat.

Visual Studio Designer

A designer a felhasználói felület szerkesztője, amelyben lehetőségünk van új elemeket felvenni, a meglévőket módosítani és törölni. A designer fontosabb elemei/kellékei:

- **Toolbox**: Erről lehet a konténereket és vezérlőket ráhúzni a form-ra.
- **Properties ablak/Property Editor**: A kijelölt vezérlő tulajdonságait és eseményeit mutatja, és itt lehet szerkeszteni is őket.
- **Smart tag**: Minden vezérlő jobb felső sarkában található egy kis nyíl, amely előhozza. Ebben vannak a fontosabb tulajdonságok összegyűjtve.
- **Document outline ablak**: A konténer-vezérlő hierarchiát mutatja. Itt ki lehet jelölni az elemeket és mozgatni is lehet őket a hierarchiában, átrendezve ezzel őket a form-on is.

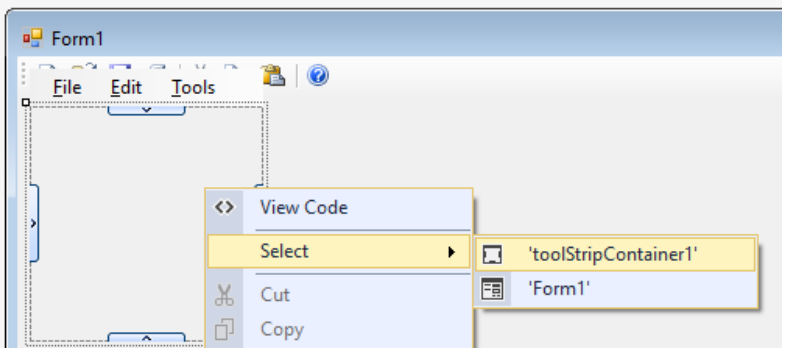
Feladat 3 - Menük

A felülettervezés következő feladata a menük megszerkesztése. Ehhez végezzük el a következő lépéssorozatot.

1. A Toolbox-ról húzzunk rá a Form-ra egy MenuStrip-et (Menus & Toolbars kategóriában van)
2. A `MenuStrip` smart tag-jét kinyitva (kicsi nyíl a jobb felső sarkában) nyomjunk rá az „Insert Standard Items”-re.
3. Ismételjük az első két lépést a `ToolStrip` vezérlővel is.
4. Majd helyezzünk fel alulra egy `StatusStrip` vezérlőt is.

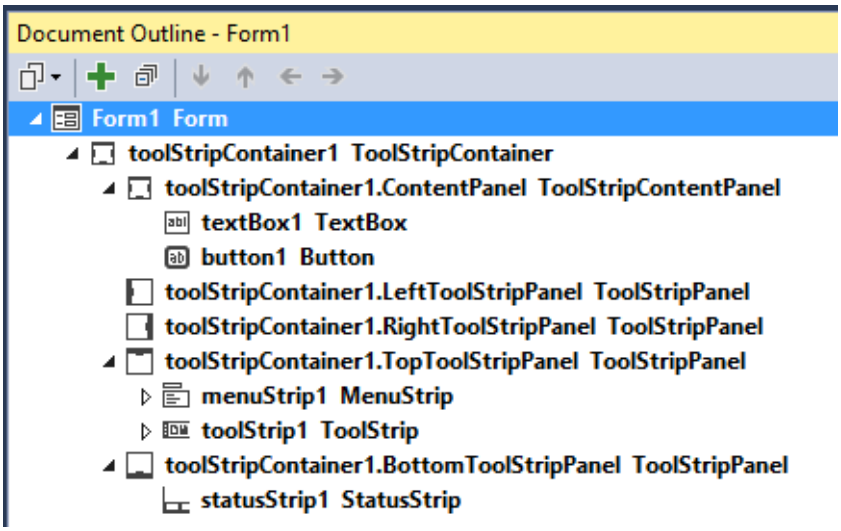
Teszteljük az alkalmazást, vegyük észre, hogy a `ToolStrip`-nek van kis fogantyúja, azonban azt hiába fogjuk meg, nem mozog. Ekkor jön segítségünkre a `ToolStripContainer`, amely egyben jól szemlélteti a konténer-vezérlő felépítést is.

1. A menü smart tag-ében válasszuk ki az „Embed In ToolStripContainer” parancsot, amely feltesz egy ToolStripContainer-t, és a menüt egyből bele is teszi. A form káoszossá válik, mivel a többi strip nem került bele.
2. A `ToolStripContainer` smart tag-jén válasszuk a Dock Fill in Form funkciót. Akkor látszólag minden a helyére kerül, leszámítva, hogy a menü és a toolbar fel vannak cserélve.



Tipp: a ToolStripContainer kijelölése kicsit trükkös a takarási viszonyok miatt. Ehhez használjuk a Document Outline ablakot, vagy használjuk a context menü „Select” menüpontját.

3. Rendezzük a konténer-vezérlő hierarchiát! Nyissuk meg a Document Outline ablakot (View/Other Windows/Document Outline) és korrigáljuk a hierarchiát.
4. Húzzuk át a `ToolStrip`-et és a `StatusStrip`-et a `ToolStripContainer` felső, illetve alsó paneljébe, továbbá a TextBox és Button vezérlőket `ContentPanel`-re. A végeredmény így néz ki:



5. A `MenuStrip` smart tag-jében állítsuk át a Grip Style-t Visible-re, ekkor már a menü is mozgatható.

Próbáljuk ki az alkalmazást, a `ToolStrip` és a `MenuStrip` mozgatható lett (oldalra és alulra is).

Példaként implementáljuk a `File/Exit` menüelemet.

6. Duplán klikkeljünk a menüelemen, majd a kódban adjuk ki a `Close()` parancsot, mely bezárja az ablakot és ezzel leállítja az alkalmazást.

```
Close();
```

7. Futtassuk és teszteljük az alkalmazást.

Feladat 4 – MiniExplorer

Horgonyzás

Anchor: A horgonyzás segítségével elérhető, hogy a vezérlő széle állandó távolságot tartson a szülő konténer megfelelő szélétől. Ez a távolság akkor is megmarad, ha a szülő konténert átméretezik. Alapértelmezésben a vezérlők bal széle és teteje van lehorgonyozva, de ez bármelyik szélre ki- vagy bekapcsolható. Ha két ellentétes szél is le van horgonyozva (például a bal és a jobb oldal), akkor a szülő vízszintes átméretezésekor a vezérlő nőni vagy zsugorodni fog, hogy a két széle megtartsa a távolságot a szülő széleitől.

1. Húzzuk be az első feladatban létrehozott gombot a form közepére.
2. A Property Editor-ban keressük ki és nyissuk le az `Anchor` tulajdonságot.

A tulajdonság értéke valójában egyszerű enum (…) melyhez a Visual Studio az egyszerűség kedvéért egy grafikus nézetet ad. Figyeljük meg, hogy jelenleg a vezérlő bal oldala és teteje van a szülőjéhez kötve.

3. Az `Anchor` szerkesztőjében kattintsunk a jobb oldali horgonyra is. Így már három oldalát rögzítettük a gombnak.
4. Teszteljük a változtatás hatását!

Mivel a horgonyok már tervezési időben is működnek, ehhez az alkalmazást sem kell elindítani. Elég a tervezési nézetben átméretezni a form-ot. Figyeljük meg, hogy immár a gomb jobb széle együtt mozog a form jobb szélével!

5. Módosítsuk a horgonyt úgy, hogy a jobb és az alsó széle legyen rögzítve a gombnak, a teteje és a bal széle nem. Teszteljük a megoldást!

Utóbbi megoldás használható például arra, hogy egy dialógusablak bezáró gombját mindig a jobb alsó sarokban tartsuk.

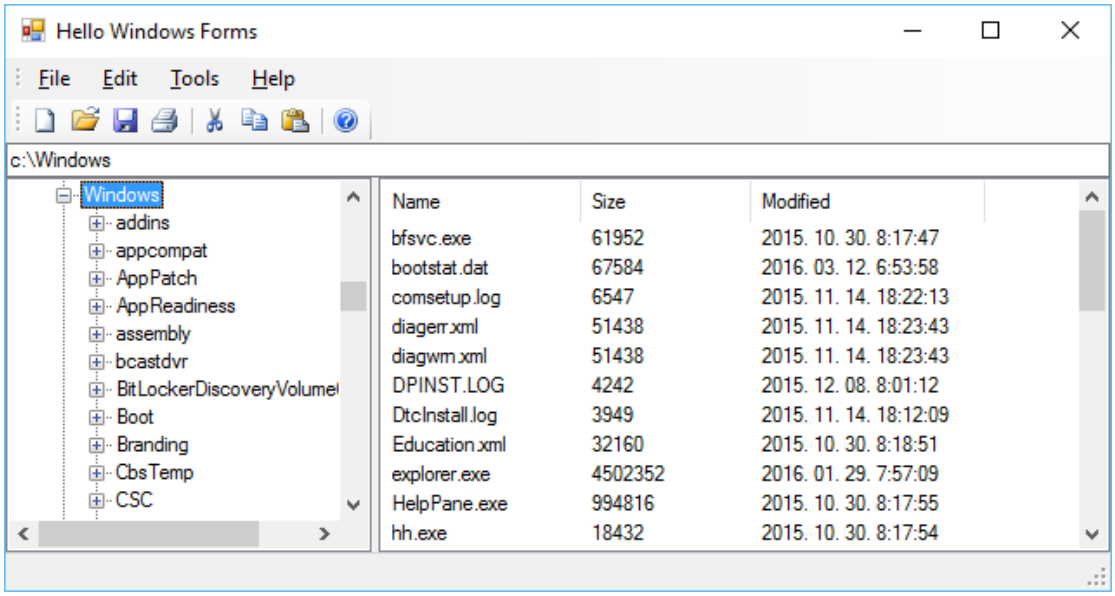
6. A gombra a továbbiakban nem lesz szükségünk, töröljük a felületről.

Dokkolás

Dock: A dokkolás segítségével egy vezérlő hozzácsatolható az őt tartalmazó konténer valamelyik széléhez, vagy beállítható, hogy töltse ki a rendelkezésre álló helyet. Lehetséges értékei: `None`, `Top`, `Left`, `Right`, `Bottom` és `Fill`. Ahhoz, hogy egy vezérlő kitöltse a neki szánt teret a dokkolást Fill-re kell állítani, így a szülő átméretezésekor a vezérlő is automatikusan átméreteződik

SplitContainer: Ez egy olyan konténer típusú vezérlő, mely az őt tartalmazó konténert két panelre osztja függőleges vagy vízszintes irányban. A két panel közé egy splittert helyez el, mellyel akár futásidőben is átméretezhető a két panel. A splitter mozgatása letiltható, és a két panel közül az egyikre beállítható, hogy a szülő konténer méretezésekor a megadott panel mérete ne változzon. (Fixed nevű tulajdonságoknál érdemes keresni.)

A feladat során egy Windows Forms alapú fájlrendszer böngésző (MiniExplorer) alkalmazást kell elkészíteni. A program kinézetét a következő ábra szemlélteti.



Az ablak három részből álljon:

- címsor az ablak tetején (`TextBox`)
- `TreeView` a címsor alatt bal oldalon
- `ListView` a címsor alatt jobb oldalon

A címsorban mindig az aktuálisan kiválasztott mappa teljes elérési útvonalát láthatjuk. Kezdetben legyenek a csomópontok összecsukott állapotban („+” ikon mellettük a fában), lenyitva őket jelenjenek meg a gyerek csomópontok, ha van mappa az adott mappán belül. Elfogadható, hogy először minden csomópont lenyitható, és csak akkor tűnik el a lenyitásra/összecsukásra szolgáló ikon, ha a felhasználó megpróbálta lenyitni és nincs benne mappa. Ha a felhasználó kiválaszt egy mappát a `TreeView`-ban (itt nem a lenyit/összecsuk műveletre kell gondolni), akkor a `ListView`-ban jelenjenek meg a mappában található fájlok. A `ListView` három oszlopban jelenítse meg a fájlok nevét, méretét és az utolsó módosítás dátumát.

1. Válasszuk ki a formon lévő `TextBox`-ot (amit az első példában raktunk ki) és állítsuk a `Dock` tulajdonságát `Top`-ra. Ezzel a címsort az ablak tetejéhez igazítottuk.
2. Tegyünk a formra egy `SplitContainer`-t (ToolBox/Containers). Figyeljük meg, hogy ez egy speciális vezérlő, mely két egymás mellé rendezett panelből áll és lehetőséget ad a panelok közti arány változtatására.
3. Válasszuk ki a „Dock in parent container” opciót a `SplitPanel` smart tag-jében!
4. A bal oldali panel-re rakjunk rá egy `TreeView` vezérlőt. A smart tag-jében válasszuk a „Dock in parent container” funkciót.
5. A jobb oldali panelre rakjunk egy ListView vezérlőt. A smart tag-jében válasszuk itt is a „Dock in parent container” funkciót.

Ezzel el is készült a MiniExplorer, legalábbis a felülete.

MiniExplorer logika

Mivel készen van a felület, a következő feladat azt kitölteni.

1. Duplán klikkeljünk a form fejlécén, ezzel tudjuk implementálni a `Form.Load` eseményét. Itt fogjuk inicializálni a fát:

```
private void MainForm_Load(object sender, EventArgs e)
{
	TreeNode root = treeView1.Nodes.Add("Local Disk (C:)");
	root.Tag = new DirectoryInfo("C:\\");
	root.Nodes.Add("");
}
```

A `TreeView` vezérlő `TreeNode` objektumokat tud megjeleníteni (ezek a fa csomópontjait jelképezik). A tényleges információt (vagyis, hogy melyik könyvtár tartozik hozzá) a `Tag` tulajdonságban tároljuk el. Ez egy `object` típusú tulajdonság, amivel a legtöbb vezérlő rendelkezik és pont azt a célt szolgálják, hogy a fejlesztők tetszőleges, számukra releváns és az adott vezérlőhöz kötődő információt tároljanak benne. A függvény utolsó sorában létrehozunk egy „üres” gyerek csomópontot. Ennek köszönhetően a szülő mellett meg fog jelenni a kibontás jele (+), továbbá tudni fogjuk, ha egy csomópont először került kibontásra és le tudjuk kérdezni a gyerekeit. Következő lépésként ez utóbbi logikát készítjük el.

2. Menjünk vissza a designer-be, válasszuk ki a `TreeView`-t, majd a `Properties` ablakban váltsunk esemény nézetre (). Duplán klikkeljünk a `BeforeExpand` eseményen, hogy implementálhassuk:

```
DirectoryInfo parentDI = (DirectoryInfo)(e.Node.Tag);
e.Node.Nodes.Clear();
try
{
	foreach (DirectoryInfo di in parentDI.GetDirectories())
	{
		TreeNode node = new TreeNode(di.Name);
		node.Tag = di;
		node.Nodes.Add("");
		e.Node.Nodes.Add(node);
	}
}
catch { }
```

3. Teszteljük az alkalmazást.

4. A jobb oldal kitöltéséhez a `TreeView` eseményei között most kezeljük le az `AfterSelect`-et, amelynek az implementációja:

```
DirectoryInfo parentDI = (DirectoryInfo)(e.Node.Tag);
listView1.Items.Clear();
try
```

```
{
      foreach (FileInfo fi in parentDI.GetFiles())
            listView1.Items.Add(fi.Name);
}
catch { }
```

5. Ugyanitt ki tudjuk tölteni az Address részt is:

```
tbDemoText.Text = parentDI.FullName;
```

Vegyük észre, hogy a fenti két függvényben mindkét esetben egy try-catch blokkot használtunk. Ez azért van, mert a laborgépeken átlagos felhasználóként sokszor nincs jogunk egyes mappák/fájlok elérésére, ami a listázó függvények esetében kivételt vált ki. Egy valós alkalmazásban semmiképpen nem hagynánk üresen a `catch` blokkot, mindenképpen naplóznánk, vagy a felhasználó tudomására hoznánk a hibát.

Következő lépésben valósítsuk meg az oszlopos nézetet a jobboldali panelen.

6. A kódot módosítsuk, hogy ne csak a nevét adja meg a fájlnak, hanem egyéb paramétereit is. Az `Add` függvényt paraméterezzük így:

```
listView1.Items.Add(new ListViewItem(new string[] { fi.Name,
   fi.Length.ToString(),
fi.LastWriteTime.ToString(),fi.FullName}));
```

7. Az adatok tehát már megvannak, de még nem jelennek meg. Ehhez a `ListView`-t módosítani kell a designer-ben, hogy mutassa a részleteket is. Hozzuk elő a smart tag-jét, és állítsuk át rajta a `View`-t `Details`-re.
8. Az oszlopokat nekünk kell létrehoznunk, amihez a smart tag-jében válasszuk az „Edit Columns” funkciót, majd a megjelenő listát töltsük fel 3 új elemmel, amelyeknek a Text tulajdonsága legyen: Name, Size, Modified. Rendezzük el szépen az oszlopok szélességét, hogy minden kiférjen majd a feltöltés után is. Ezekben az oszlopokban az adatok pont olyan sorrendben fognak megjelenni, mint ahogy a 6. pontban a listaelemhez hozzárendeltük azokat.

Futtatás

Utolsó érdekességként megoldhatjuk, hogy a jobb oldali nézetben egy fájlon duplán kattintva a rendszer megnyissa/végrehajtsa azt. Ehhez iratkozzunk fel a `ListView DoubleClick` eseményére és valósítsuk meg a következő képpen:

```
if (listView1.SelectedItems.Count != 1) return;
string fullName = listView1.SelectedItems[0].SubItems[3].Text;
if (fullName != null) Process.Start(fullName);
```