Juhász Tibor – Kiss Zsolt:

Programozási ismeretek

(Műszaki Könyvkiadó, 2011, MK–4462-3)

Visual Studio 2008/2010 Express Edition Ismerkedés a fejlesztőrendszerrel

Kiegészítések a tankönyvhöz

Bevezetés

Az *Ismerkedés a fejlesztőrendszerrel* a Programozási ismeretek tankönyv (Műszaki Könyvkiadó, 2011) kiegészítése. A tankönyvben lehetőség szerint a programozási nyelvektől, illetve fejlesztőrendszerektől függetlenítve ismertetjük a programozási tudnivalókat. Az alábbiakban bemutatjuk a Visual Studio 2008/2010 Express Edition használatát és a Visual Basic programok készítésének módját. A fejlesztőrendszer áttekintése csak a tankönyvhöz kapcsolódó elemekre vonatkozik. A tankönyv anyagát fedi le, de kitérünk néhány fontos és hasznos lehetőségre is, amely túlmutat a gyakorlatok vagy feladatok megoldásán.

A fejlesztőrendszer ismertetését egy Windows- és egy konzolalkalmazás készítésének bemutatásával kezdjük. Az egyes lépéseket olyan részletességgel írjuk le, hogy a teljesen kezdők számára is érthetőek legyenek. Ezek után már nem jelenthet gondot a tankönyv gyakorlatainak az értelmezése, elkészítése.

Javasoljuk, hogy a fejlesztőrendszer használatakor tekintsük át a Visual Studio legfontosabb beállításait (lásd a 28. oldalon).

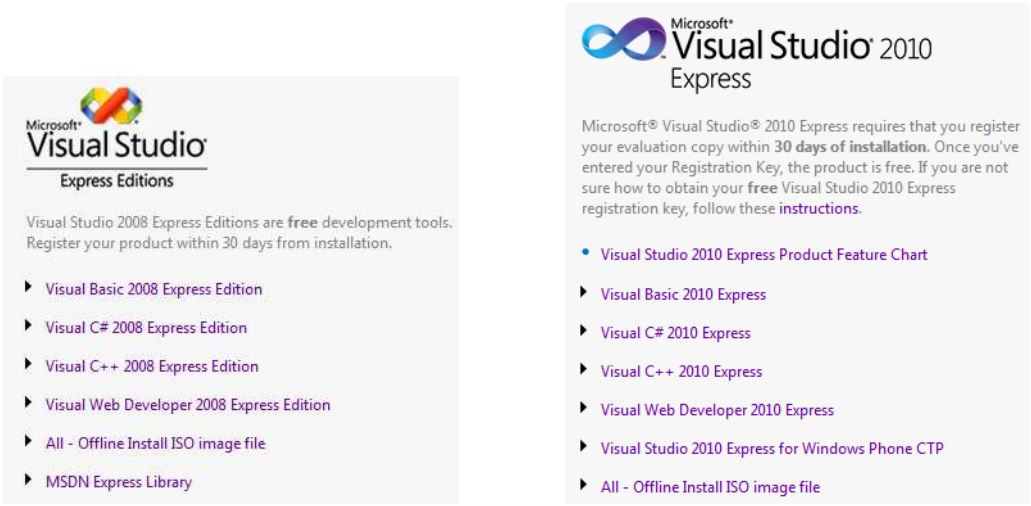
Telepítési útmutató

A Visual Studio letöltése

A *Visual Studio Express Edition* a Microsoft webhelyéről tölthető le:

http://www.microsoft.com/express/Downloads/

A letöltési oldalról válasszuk igény szerint a *Visual Studio 2008 Express* vagy a *Visual Studio 2010 Express* fület, a megjelenő panelen pedig a *Visual Basic 2008/2010 Express* hivatkozást. Az előtűnő legördülő listában módosíthatjuk a nyelvet, majd kattintsunk a *Free Download* parancsgombra! A körülbelül 3 megabájt méretű telepítőfájl letöltése és indítása után kezdődik az on-line telepítés.



A Visual Studio Express Edition letöltési oldalai

Az on-line telepítés helyett célszerűbbnek tartjuk a *Visual Studio Express* teljes változatának a letöltését. Ehhez a letöltési oldalon válasszuk az *All – Offline Install ISO Image* hivatkozást. A körülbelül 700 megabáj-

tos fájl DVD-képfájl, amit szükség esetén lemezre írhatunk, vagy egy képfájlokat olvasó programmal, például a *MagicDisc*-kel csomagolhatunk ki. A freeware program letölthető a MagicISO webhelyéről:

http://www.magiciso.com/tutorials/miso-magicdisc-overview.htm.

A kicsomagolás helyett létrehozhatunk virtuális meghajtót is, amelyről közvetlenül elvégezhetjük a telepítést. Virtuális meghajtót készíthetünk például a *Daemon Tools Lite* programmal, amely letölthető a Disk-Tools webhelyéről (http://www.disk-tools.com/download/daemon).

Telepítés

A Visual Basic telepítéséhez indítsuk el az on-line telepítőfájlt, off-line esetben pedig a *VBExpress* mappa *autorun.exe* fájlját. A telepítés során

- a *Welcome to Setup* ablak jelölőnégyzetével eldönthetjük, hogy küldünk-e információkat a Microsoftnak;
- a *License Terms* ablakban fogadjuk el a felhasználási feltételeket (*I have read and accept the license terms*);
- az *Installation Options* ablakban a *Visual Basic 2008* régebbi változatai esetén hagyjuk meg az *MSDN Express Library* kijelölését. A többi összetevőre nincs szükségünk a tankönyv feldolgozásához. A 2008-as újabb változatainál, illetve a VS 2010-nél töröljük a *Microsoft SQL Server…* jelölőnégyzetének a kiválasztását.

A további folyamatot rábízhatjuk a telepítőkészletre. Előfordulhat, hogy közben – esetleg többször is – újra kell indítani a számítógépet. Erre üzenet figyelmeztet (*You must restart your computer…*). Az újraindítás után a telepítés automatikusan folytatódik.

Megjegyezzük, hogy a telepítés során a fejlesztőrendszer mellett további összetevők (például a .NET megfelelő változata) szintén felkerülnek a számítógépre.

A Microsoft azt javasolja, hogy a VS 2010 használatánál célszerű telepíteni a Windows Automation API 3.0-s változatát (http://support.microsoft.com/kb/981741/hu), amely felgyorsítja az intelligens súgó működését. Az Automation API már része a Vista és a Windows 7 operációs rendszernek, így csak az XP-re célszerű telepíteni.

Regisztrálás

A telepítés után végezzük el a Visual Studio Express ingyenes regisztrálását! Regisztrálás nélkül a programot csak 30 napig használhatjuk.

A regisztráláshoz válasszuk a *Help*/*Register Product* menüparancsot, majd a megjelenő regisztrációs ablakban kattintsunk az *Obtain a registration key online* feliratú parancsgombra! Ha rendelkezünk Windows Live ID-vel (MSN Hotmail, MSN Messenger vagy Passport azonosítóval), akkor ezt felhasználhatjuk a regisztrációs lapon történő bejelentkezéshez. Live ID hiányában kattintsunk a *Sign up now* gombra, majd töltsük ki a megjelenő magyar nyelvű űrlapot! A továbblépés után megjelenő angol nyelvű űrlapon ne hagyjuk üresen a név (*First Name*, *Last Name*), e-mail cím (*E-Mail Address*), ország (*Country*) mezőket. Az elsődleges beosztás (*What is your primary role…*) listájában választhatjuk a *Student* (diák) elemet. A kitöltés után kattintsuk a *Continue* gombra.

A regisztrációs ablak

Hamarosan kapunk egy e-mailt, amelyben találunk egy meglehetősen hosszú hivatkozást. Kattintsunk rá, majd a megjelenő weblapon válasszuk a *Continue* gombot. A következő weblapon megtaláljuk a regisztrációs kódot, amelyet a vágólap segítségével bemásolhatunk a regisztrációs ablak *Registration Key* szövegdobozába. Végül kattintsunk a *Register Now* gombra. A regisztrációs kódot e-mailben is megkapjuk.[1]

[1] Megjegyezzük, hogy a Microsoft időnként változtatja a regisztrálás módját. Lényegében azonban az itt ismertetett lépéseket kell megtennünk.

Visual Basic Power Packs

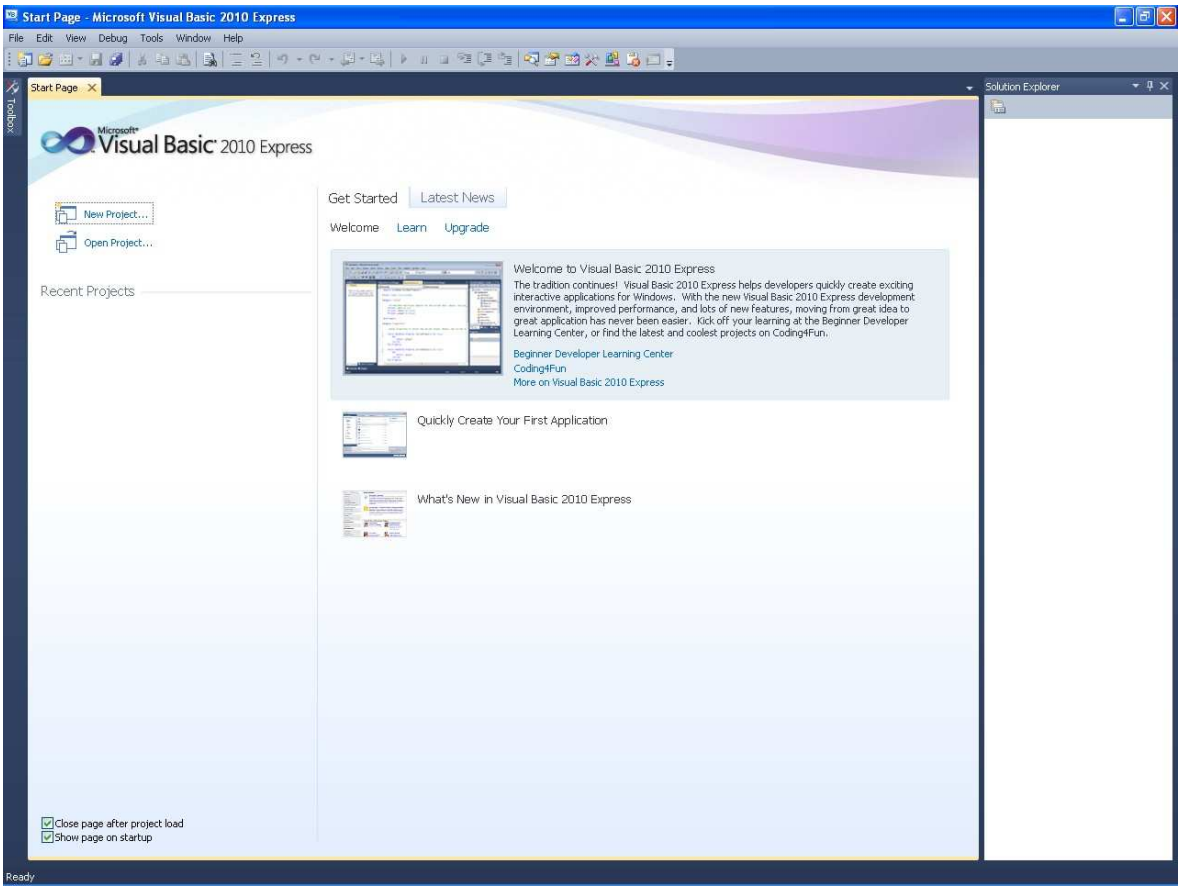
A Visual Basic PowerPacks hasznos elemekkel bővíti a fejlesztőrendszert. A PowerPacks része a Visual Studio 2010-nek és a 2008 újabb változatainak. A 2008-as változat régebbi kiadásainál azonban külön kell telepíteni (http://msdn.microsoft.com/en-us/vbasic/bb735936.aspx). A telepítés után az új vezérlőket az eszköztár *Visual Basic PowerPacks* paneljén találjuk. Tankönyvünkben a *LineShape* vezérlőt használtuk fel a *Műveletek és kifejezések* lecke számológépes feladatának törtvonalaihoz.

Programozás Visual Basicben

Az alábbiakban az integrált fejlesztőrendszerre vonatkozó parancsoknál megadjuk a parancs helyét a menüben, majd zárójelben az eszköztár megfelelő ikonját, illetve a hozzá tartozó billentyűkombinációt.

A Visual Basic indítása és felhasználói felülete

A Visual Basicet a *Start*/*Minden program* menüjének vagy az *Asztal* parancsikonjának a segítségével indíthatjuk el. (Az első indítás egy kicsit hosszabb ideig tarthat.) A megjelenő ablakban láthatjuk a Windowsban megszokott menüsort, eszköztárat, alul pedig az állapotsort.



A Visual Basic 2010 integrált fejlesztőrendszere a startlappal

Az ablak jobb szélén helyezkedik el az egyelőre üres megoldástallózó (*Solution Explorer*), melyet a fájlok kezeléséhez használunk. A bal szélen találjuk az összecsukott eszközkészletet (*Toolbox*), amely akkor nyílik ki/zárul be, ha az egeret a *Toolbox* felirat fölé visszük, illetve elvisszük onnan. Ezt a viselkedést a munkaablak *Auto hide* ikonjával módosíthatjuk. Fekvő gombostű ( ) esetén a nyitás/zárás az egérrel végezhető, míg álló gombostűnél ( ) folyamatosan nyitva marad az ablak. Az ikont megtaláljuk a többi munkaablak címsorában is.

Ha nem látjuk valamelyik munkaablakot, akkor a View menü segítségével nyitható meg.

A Visual Basic indítása után a *Close* ( ) gombbal zárjuk be a startlapot![2]

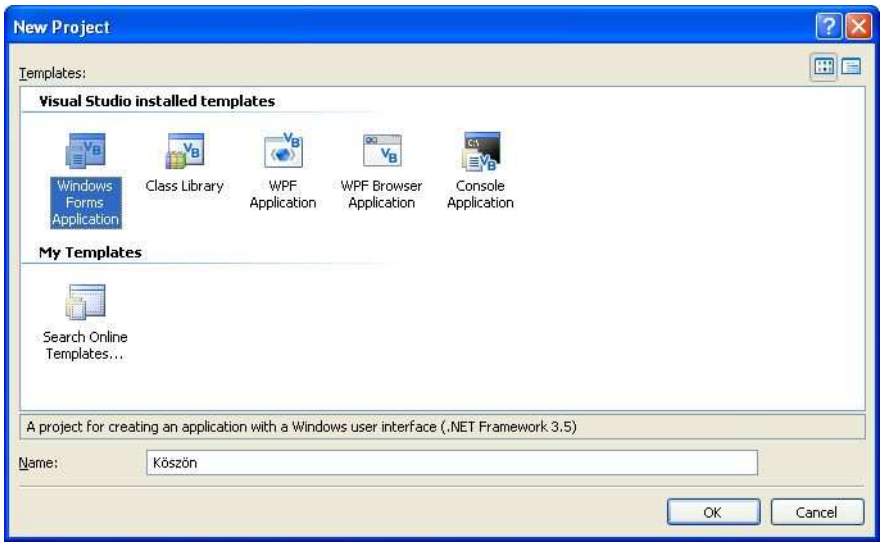
[2] A startlap automatikus megnyitása letiltható a *Tools*/*Options* menü *Environment*/*Startup* csoportjában (*At startup: Show empty environment*). A csoport megjelenítéséhez az *Options* ablakban kapcsoljuk be a *Show all settings* jelölőnégyzetet. A 2010-es változatnál a startlap bal alsó jelölőnégyzetével is letiltható a megjelenítés (*Show page on Startup*).

Visual Basic programok létrehozása

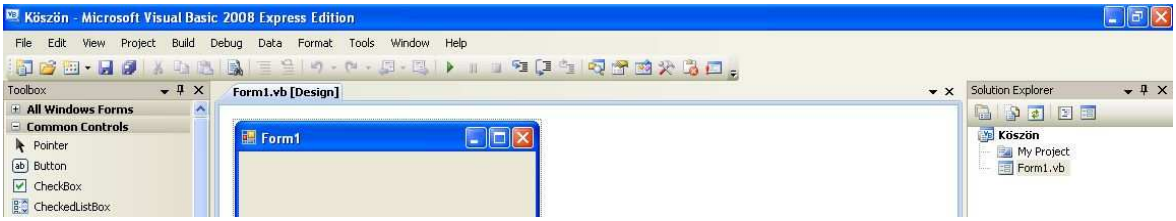
Az új program írását a *File* menü *New project* parancsával kezdjük (, *Ctrl+N*). A Visual Basic több, előre elkészített sablonnal segíti a programozást. A sablon a készülő programot keretbe foglaló utasításokat tartalmazza.

A megjelenő *New Project* ablakban látjuk a rendelkezésre álló sablonokat. A *Windows Forms Application* sablon egy grafikus felületen futó alkalmazás, a *Console Application* pedig egy úgynevezett konzolalkalmazás megírását készíti elő. A konzolalkalmazások nem használják ki a Windows grafikus felhasználói felületét. A felhasználóval a parancssori ablakon[3] keresztül tartják a kapcsolatot. Először Windows-alkalmazást fogunk készíteni. Később megismerkedünk a konzolalkalmazások létrehozásával is.

Válasszuk ki a *Windows Forms Application* sablont, majd a *Name* szövegdobozba írjuk be a nevét (*Köszön*). Ha rákattintunk az OK gombra, létrejön első programunk.



Windows-alkalmazás létrehozása (VS 2008)



Az első program az eszköztárral, az űrlappal és a megoldástallózóval

A képernyőn megjelenik az *Eszköztár* (*Toolbox*), középen a *Tervezőablak* (*Design*), a jobb szélen a *Megoldástallózó* (*Solution Explorer*), alatta pedig a *Tulajdonságok* (*Properties*) munkaablak.

A Visual Basic programok szerkezetét a későbbiekben ismertetjük. Most csak annyit jegyzünk meg, hogy a program elkészítése során eseménykezelő eljárásokat írunk. Az eseménykezelő eljárások szabják meg, hogy a felhasználó által létrehozott események (például egérkattintás) bekövetkezéskor mit kell tenni.

Egy eseménykezelő eljárás utasításait a *Sub … End Sub* utasítások határolják. Maga az eljárás pedig az osztálydefiníció része, amely a *Class … End Class* utasítások között helyezkedik el. Bár a szerkezet bonyolultnak tűnik, a fejlesztőrendszer nagymértékben segíti a kialakítását!

[3] A parancssori ablak a *Start*/*Minden program*/*Kellékek* menüből indítható, de most nem kell megnyitnunk.

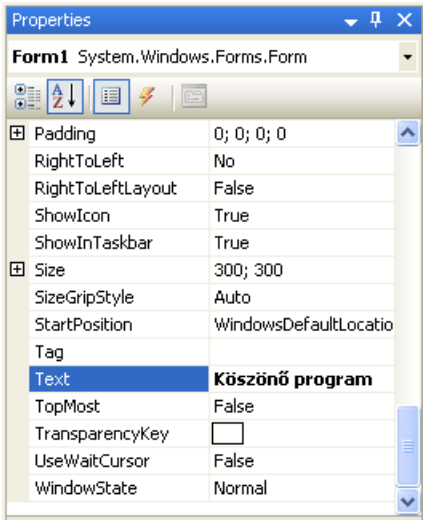
Windows-alkalmazás készítése

Az ablak létrehozása és tulajdonságainak módosítása

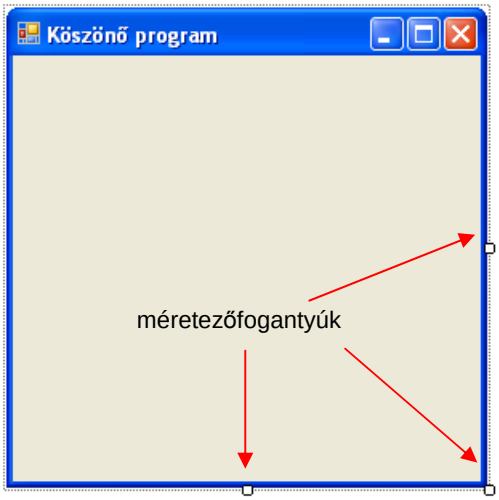
Első programunk egy szövegdobozt és egy parancsgombot fog tartalmazni. Ha a felhasználó beírja a nevét a szövegdobozba, majd rákattint a parancsgombra, akkor a program kiír a képernyőre egy üdvözlő szöveget.

A *Tervezőablakban* a *Form1* nevű űrlapot látjuk, amely a képernyőn megjelenő programablakot jelképezi.[4] Kattintsunk rá az űrlapra. A tulajdonságok munkaablakban megkapjuk az űrlapobjektum tulajdonságainak listáját. Keressük meg a *Text* (szöveg) tulajdonságot, ami az ablak címsorának a szövegét határozza meg. Írjuk be a *Köszönő program* címet. Ha lenyomjuk az Entert vagy az ablak más területére kattintunk, máris megjelenik az űrlap címsorában a begépelt szöveg.

Szükség esetén további tulajdonságokat is módosíthatunk. A *BackColor* például a háttérszínt, a *Size* a méretet jelenti pixelben. Ez utóbbit a méretezőfogantyúk segítségével a tervezőablakban is megváltoztathatjuk. A (*Name*) tulajdonság az űrlapobjektum azonosítóját adja meg. Több ablak létrehozásánál ezt célszerű átírni, az ablak funkciójára utaló nevet választani. Most nem módosítjuk.



Az űrlap tulajdonságai



A készülő űrlap a módosított címmel

A folytatás előtt mentsük el a programot! Ehhez válasszuk a *File* menü *Save all* parancsát (, *Ctrl+Shift+S*). Hagyjuk meg a felajánlott neveket (*Name*, *Solution name*). A *Location* szövegdobozban a *Browse* gomb segítségével válasszuk ki azt a mappát, ahová programjainkat menteni fogjuk. Figyeljünk arra, hogy a programhoz új mappa jöjjön létre (*Create directory for solution* jelölőnégyzet). A mentéshez kattintsunk a *Save* gombra.

Első programunk tulajdonképpen el is készült. Itt az ideje, hogy elindítsuk. A fejlesztői környezeten belül ezt a *Debug*/*Start Debugging* paranccsal (, *F5* funkcióbillentyű) tehetjük meg. Így a programot az úgynevezett hibakereső üzemmódban hajtjuk végre.

A képernyőn megjelenik egy üres ablak, a beállított tulajdonságokkal. Kattintsunk a *Bezárás* gombra. Visszatérünk a fejlesztőrendszerbe. Próbáljuk meg módosítani az ablak tulajdonságait. A módosítás után ismét futtassuk a programot.

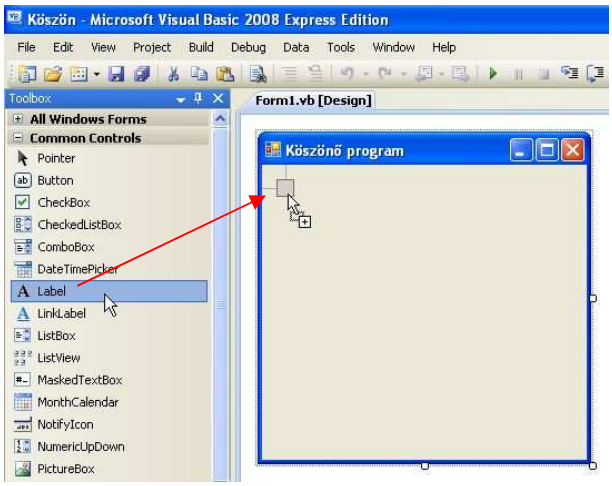
Hálózati környezetben végzett munka esetén előfordulhat, hogy nincs elegendő jogunk a hibakereső üzemmódban történő futtatáshoz. Ekkor használjuk a *Debug*/*Start without Debugging* (*Ctrl+F5*) parancsot (futtatás hibakeresés nélkül).

> A *Start without Debugging* parancs alapbeállítás szerint nem szerepel a *Debug* menüben (a *Ctrl+F5* billentyűparancs ettől függetlenül működik). Szükség esetén a következő módon vehetjük fel. A *Tools*/*Customize* menüparancs ablakában válasszuk a *Commands* panelt! A VS 2010-nél ezután válasszuk ki a *Menu Bar* listájában a *Debug* menüt, majd kattintsunk az *Add Command* gombra (a VS 2008-nál ez a lépés kimarad). A *Categories* listájában kattintsunk a *Debug* parancsra. A *Commands* listájában keressük meg a *Start Without Debugging* parancsot, majd a VS 2008-ban egérrel vigyük a *Debug* felirat fölé a menüben. Erre lenyílik a menü, és a parancsot elhelyezhetjük egy általunk kiválasztott helyre. A VS 2010-ben a mozgatás helyett kattintsunk az *Add Command* ablak *OK* gombjára. Végül zárjuk be a *Customize* ablakot.
> A parancsot a menü helyett az eszköztárban is elhelyezhetjük.

[4] Nem tartjuk szerencsésnek az ablakra vonatkozó *űrlap* elnevezést, de elterjedtsége miatt mi is ezt fogjuk használni.

Objektumok az űrlapon

A *Label* (címke) objektum segítségével írhatunk szöveget az ablakba. Az objektumot az eszközkészletben (*Toolbox*) találjuk a *Common Controls* csoportban. Fogjuk meg az egérrel a *Label* ikonját, és helyezzük el az űrlap bal felső részére. (Hasonló eredményt érhetünk el, ha duplán kattintunk az ikonra.) Megjelenik az űrlapon a címke.



Címke elhelyezése az űrlapon

A készülő ablak

Kattintsunk a címkére, majd a tulajdonságok munkaablakban módosítsuk a *Text* tulajdonságot: *Írd be a neved, majd kattints az OK-gombra!* Hosszú szövegek beírásakor kattintsunk a *Text* tulajdonság jobb szélén lévő, lefelé mutató nyílra! Így nagyobb terület áll rendelkezésünkre a gépeléshez.

Vegyük észre, hogy a tulajdonságok munkaablak mindig a kijelölt objektum jellemzőit mutatja. A létrehozott objektumok között a munkaablak felső részén lévő legördülő lista segítségével is válogathatunk.

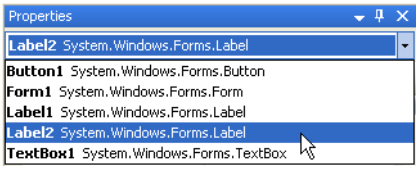
Helyezzünk el az űrlapon egy *TextBox* (szövegdoboz) objektumot, melynek töröljük a *Text* tulajdonságát. A szövegdoboz mellé pedig tegyünk egy *Button* (parancsgomb) objektumot. Írjuk át a parancsgomb feliratát OK-ra (*Text* tulajdonság).

Végezetül tegyünk a szövegdoboz alá még egy címkeobjektumot, melynek töröljük a *Text* tulajdonságát, majd mentsük a projektet! A projekt mentése során a fejlesztői környezet létrehozza az objektumok definícióit tartalmazó szövegfájlt, amit *Form1.Designer.vb* néven találunk meg a projekt mappájában. A szövegfájlt a Jegyzettömbbel is megnyithatjuk, de ne módosítsuk!

Futtassuk a programot! A szövegdobozba írhatunk karaktereket, a parancsgomb azonban még nem működik. El kell készítenünk az egérkattintáshoz tartozó eseménykezelő eljárást.

Mielőtt továbblépnénk, zárjuk be a programot, majd a tervezőablakban kattintsunk az egyes objektumokra! A tulajdonságok ablakban tekintsük meg azonosítóikat (*Name* tulajdonság a lista elején). A szövegdoboz a *TextBox1*, a parancsgomb a *Button1*, a két címkeobjektum pedig a *Label1*, illetve *Label2* nevet kapta. A későbbiekben ezeket az elnevezéseket célszerű olyan azonosítóval helyettesíteni, amely utal az objektum szerepére. Így sokkal könnyebben tudjuk az eseménykezelő eljárásokat áttekinteni.

Az üres címkét jelölő téglalap más objektum kiválasztása esetén eltűnik az ablakból. A legegyszerűbben úgy találhatjuk meg, hogy kiválasztjuk az objektumot a tulajdonságablak legördülő listájából.



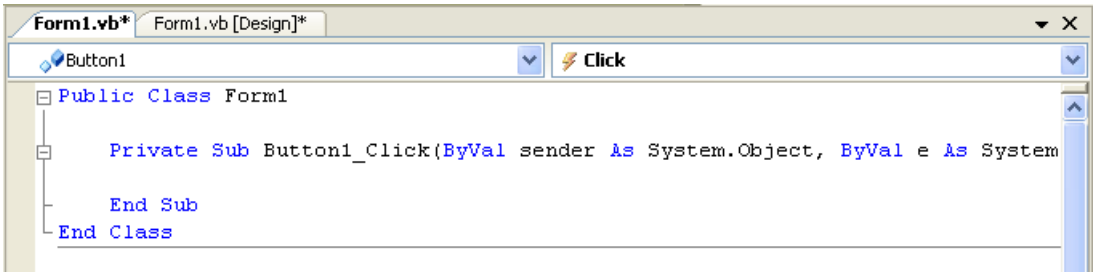
A címkeobjektum kiválasztása a Properties ablak listájában

Megjegyezzük, hogy ha egymást takaró vezérlőelemeket helyezünk az űrlapra, akkor a jobb egérkattintásra megjelenő helyi menü *Bring to Front* parancsával hozhatjuk előre, illetve a *Send to Back* paranccsal küldhetjük hátra.

Eseménykezelés

Az előzőekben a fejlesztői környezet segítségével létrehoztuk a programablakot, és elhelyeztük benne a szükséges elemeket. Most az eseménykezelő eljárás megírása következik. Az eseménykezelő eljárások meghatározott objektumok megadott eseményeihez tartoznak. Az eljárások utasításai az események bekövetkezésekor kerülnek végrehajtásra.

A tervezőablakban kattintsunk duplán az OK-gombra, melyhez hozzárendeljük az eseménykezelő eljárást. Automatikusan megnyílik az úgynevezett kódszerkesztő ablak. A kódszerkesztő és a tervezőablak között az ablak tetején lévő fülek segítségével válthatunk (vagy pedig az *F7*, *Shift+F7* billentyűparanccsal). A fájlnév mellett lévő csillag azt jelzi, hogy az utolsó mentés óta módosult az ablak tartalma.



A kódszerkesztő ablak a fülekkel és a forráskód sablonjával

Megfigyelhetjük, hogy a kódszerkesztő ablakban az utasítások kulcsszavai kék színnel és nagy kezdőbetűvel jelennek meg. **A Visual Basic nem különbözteti meg a kis- és nagybetűket a forráskódban. A kódszerkesztő a kulcsszavak kezdőbetűit átírja nagybetűre még akkor is, ha kisbetűvel gépeljük be.** Ezzel megkönnyíti a szintaktikus hibák keresését. A behúzások alkalmazása pedig áttekinthetővé teszi a forráskódot.

A kódszerkesztő elkészíti a program keretét. Programunk minden eljárása a *Class … End Class* kulcsszavak közé kerül. Az eseménykezelő eljárás utasításait a *Sub … End Sub* kulcsszavak közé írjuk.

Az eljárás neve célszerűen az objektum (*Button1*) és az esemény (*Click*: kattintás) nevéből áll, ezeket aláhúzásjel köti össze.[5] Mint látjuk, a dupla kattintás hatására a kódszerkesztő kiválasztotta az objektumhoz kapcsolódó leggyakoribb, úgynevezett alapértelmezett eseményt (parancsgomb esetén az egérkattintást). Ezt természetesen szükség esetén módosíthatjuk. Az objektumhoz kapcsolódó eseményeket az ablak tetején, a jobb oldalon található legördülő lista tartalmazza.

> Az eseménykezelő eljárás két paraméterrel rendelkezik. Az első paraméter (*sender*) segítségével az eljárást meghívó objektumot, a második paraméterrel (*e*) pedig az esemény tulajdonságait érhetjük el. A paramétereket most nem fogjuk felhasználni. A sor végén álló *Handles* (kezeli) kulcsszó után soroljuk fel azokat az objektumokat és eseményeket, amelyekre az eseménykezelő eljárás vonatkozik. Egy eljárás több objektum több eseményét is kezelheti. A listában az objektum és az esemény közé pontot teszünk. **Az eseménykezelő eljárás keretét a fejlesztői környezet automatikusan elkészíti.**
> A *Sub* előtt álló *Private* kulcsszó azt jelzi, hogy eljárásunk egy másik osztályból/modulból nem hívható meg.

A kódszerkesztő ablakban megfigyelhetjük, hogy az eseménykezelő eljárás a *Form1* osztály egy eljárása. A *Class* után áll az osztály neve. A *Public* kulcsszó arra utal, hogy osztályunkra más kódfájlokból is hivatkozhatunk.

Ha egy projekt megnyitásakor nem látjuk a kódszerkesztő ablakot, akkor a megoldástallózóban kattintsunk a jobb egérgombbal az űrlapra (*Form1*) és válasszuk a *View Code* parancsot! Szükség esetén ugyanígy (vagy dupla kattintással) nyithatjuk meg az űrlapot a tervező nézetben (*View Designer*).

A szöveg másolása a címkére

Mint látjuk, a kódszerkesztő automatikusan elkészíti helyettünk programunk keretét. Nekünk csak az eseményt kezelő utasításokat kell begépelnünk. Feladatunk, hogy a *TextBox1* szövegdobozba írt szöveget az üdvözlettel kiegészítve átmásoljuk a *Label2* címkére. A szövegdoboz szövegét az objektum *Text* tulajdonsága adja meg. Egy objektum tulajdonságára az objektum és a tulajdonság nevével hivatkozunk, amiket ponttal választunk el egymástól:

objektumnév.tulajdonságnév

Például:

```
TextBox1.Text
```

A *Label2* objektum *Text* tulajdonságát a következő utasítással módosítjuk:

```
Label2.Text = TextBox1.Text
```

Ennek hatására a szövegdoboz szövege átkerül a címkeobjektumba.

[5] Az eljárást nem kötelező így elnevezni. Az eseményt és az objektumot az eljárásfej végén (a zárójeles paraméterlista után) álló *Handles Button1.Click* kapcsolja az eljáráshoz.

Az intelligens súgó

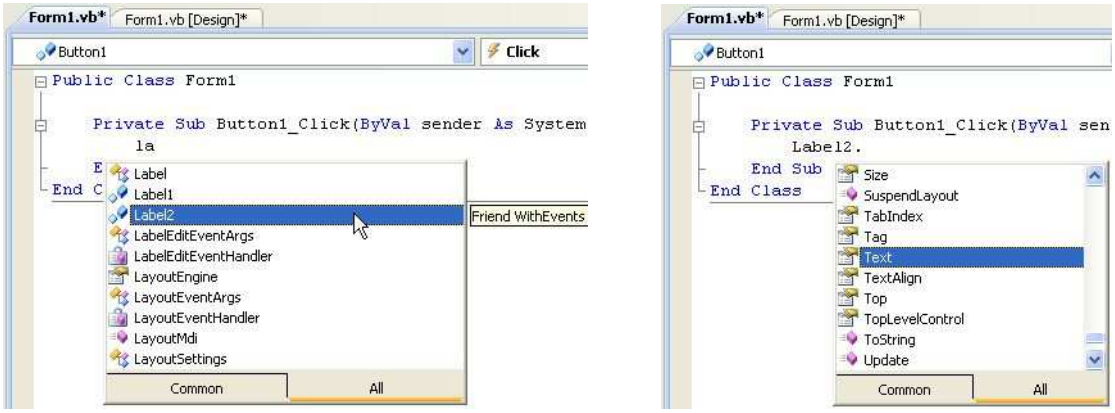
A Visual Studio intelligens súgója nagymértékben megkönnyíti forráskód begépelését. A súgó kilistázza szóba jöhető elemeket. A listából az egérrel vagy a kurzormozgató billentyűkkel választhatunk. Ha kiválasztottunk egy elemet, akkor az utána következő karakter gépelésével folytathatjuk a munkát. A billentyű lenyomásakor a súgó beilleszti a karakter elé a kiválasztott elemet a forráskódba. Ugyanezt az eredményt érjük el, ha a következő karakter helyett a tabulátor billentyűt nyomjuk le, vagy duplán kattintunk az elemre. Ezt a módszert akkor szoktuk alkalmazni, ha már nem folytatjuk a gépelést.

Az eseménykezelő eljárás elkészítéséhez kattintsunk a *Sub* és *End Sub* utasítások közötti üres sorra, majd kezdjük el gépelni a fenti utasítást (kisbetűvel kezdve). Az első karakter (*l* betű) beírásakor megjelenik a lista. Kijelzi az összes lehetséges folytatást. Írjuk be az *a* betűt is! Az egérrel vagy a kurzormozgató billentyűvel válasszuk ki a listából a *Label2*-t, majd nyomjuk le a forráskódban következő pontot! Megjelenik az objektum tulajdonságainak és metódusainak a listája, ahol kijelölhetjük a *Text* tulajdonságot.

A *Text* kiválasztása után a következő karakter, az egyenlőségjel begépelésével folytassuk a forráskód szerkesztését! A kódszerkesztő automatikusan elhelyezi a forráskódban a *Text* szót, és eltűnik a lista. Egészítsük ki a programot az üdvözlés szövegével:

```
Label2.Text = "Szia " & TextBox1.Text & "!"
```

Közben az intelligens súgó folyamatosan megjeleníti az adott utasításra vonatkozó rövid, angol nyelvű magyarázatot. Megjegyezzük, hogy az intelligens súgót utólag a *Ctrl + szóköz* billentyűparanccsal hívhatjuk elő.



Az intelligens súgó a forráskód begépelésénél

Az Enter lenyomásával lépjünk a következő sorba! A létrejövő új sor behúzása megfelel az előző sornak. A kódszerkesztő elvégzi helyettünk a behúzás megfelelő kialakítását. Szóközök beillesztésével pedig tagolja, áttekinthetővé teszi az utasításokat.

Az idézőjelbe tett karaktereket a program módosítás nélkül ki fogja írni az ablakba. A szövegeket az & (*Alt Gr + C*) karakterrel fűzzük egymáshoz.

Mentsük a projektet, majd futtassuk a programot! Próbáljuk ki a működését. Az üdvözlő szöveg megjelenése után írjunk be egy másik nevet, majd ismét kattintsunk az OK-gombra.

A program futtatása

A Visual Basic korszerű vizuális fejlesztőrendszerének a segítségével gyorsan és egyszerűen tervezhetjük meg az ablakokat és írhatjuk meg eseményvezérelt programjainkat.

Futtatható program készítése

Mindeddig csak hibakereső üzemmódban, a Visual Studión belül indítottuk el a programot. Az önállóan futtatható program készítéséhez le kell fordítanunk a forráskódot gépi kódra (illetve a .NET köztes kódjára). A fordítást a *Build*/*Build* parancs segítségével végezhetjük el. A fordítás után az *.exe* fájl a mentésnél megadott *Solution name**Name* mappa *bin**Release* almappájába[6] kerül. A fájlt az Intéző segítségével a szokásos módon futtathatjuk (például dupla kattintással).

A *Release* mappában lévő *.exe* programot akkor is elindíthatjuk, ha a hálózati korlátozások miatt nem használhatjuk a hibakereső üzemmódban történő futtatást.

A *Build* menü *Publish* parancsával telepítőkészletet készíthetünk az alkalmazásunkhoz. Ezzel a lehetőséggel itt nem foglalkozunk.

A programhibák javítása

Hibák a forráskódban

A kódszerkesztő aláhúzással jelöli a forráskódban előforduló, elsősorban szintaktikai hibákat. Az alábbi forráskódban például lehagytuk az első sor elejéről a megjegyzésre utaló aposztrófjelet. Ezért a fordítóprogram nem tudta azonosítani a *Köszönő* szót. A második sorban pedig szóközt hagytunk a *Form* és az 1-es között.

```
Köszönő program
Public Class Form 1
    ' Eseménykezelő eljárás:
    Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button1.Click
        Label2.Text = "Szia " & TextBox1.Text & "!"
    End Sub
End Class
```

Hibák a forráskódban

Figyeljünk arra, hogy egy-egy hibának további következményei lehetnek. A *Button1*, *Label2* és a *TextBox1* például azért minősül hibásnak, mert az előző szintaktikai hiba miatt hibás osztálydefinícióban helyezkedik el.

Ha az aláhúzott rész fölé visszük az egérkurzort, akkor megjelenik a hiba angol nyelvű magyarázata. Kezdő programozóként azonban nem mindig tudunk következtetni a szövegből a hiba okára. A *Comma or ')' expected* valóban a végzárójel hiányát jelzi, de a *Form 1* hibánál megjelenő szöveg nem utal egyértelműen a felesleges szóközre. A *Label2* aláhúzása pedig – mint említettük – egy másik hiba következménye.

```
' Köszönő program   End of statement expected.
Public Class Form 1
    ' Eseménykezelő eljárás:
    Priv  Name 'Label2' is not declared.  ick(ByVal sender As System.Object, ByVal e As System.EventArgs Handles Button1.Click
        Label2.Text = "Szia " & TextBox1.Text & "!"                                  Comma or ')' expected.
    End Sub
End Class
```

A hibák magyarázata

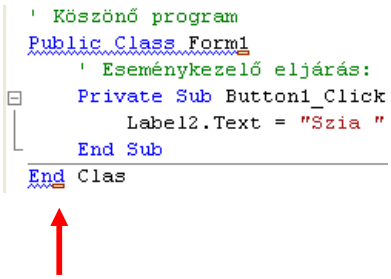
```
' Köszönő program
Public Class Form1
    ' Eseménykezelő eljárás:
    Private Sub Button1_Click(ByVal sender As System.
        Label2.Text = "Szia
    End Sub
End Class
```

String constants must end with a double quote.

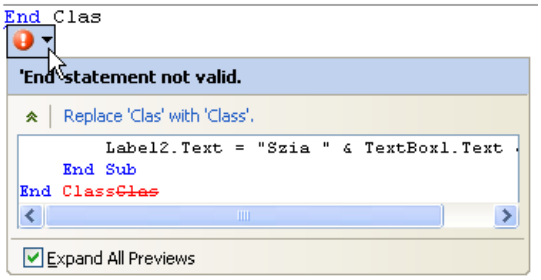
Gyakori hiba a karaktersorozat záró idézőjelének elhagyása
A VS 2010 automatikusan ki is javítja ezt a hibát.

Ha a hiba helye alatt egy kis téglalapot látunk, akkor az egérkurzor föléje helyezésével megjelenik egy intelligens címke, melynek legördülő menüjében részletesebb magyarázatot kapunk (angolul).

[6] A *Release* mappát a *Debug* mappa mellett találjuk. A projekt mappaszerkezetét *Projektek kezelése* fejezet ismerteti.



A hibára utaló téglalap



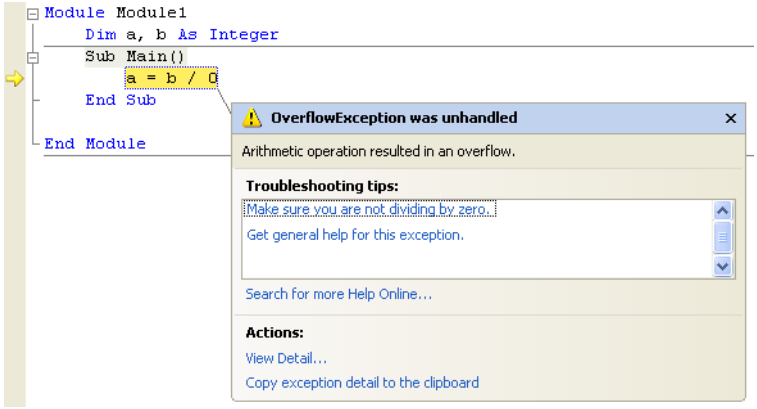
Javaslat a hiba javítására: Replace 'Clas' with 'Class'

Ha rákattintunk a *Replace 'Clas' with 'Class'* javaslatra, akkor a kódszerkesztő elvégzi a hiba kijavítását.

Futási hibák

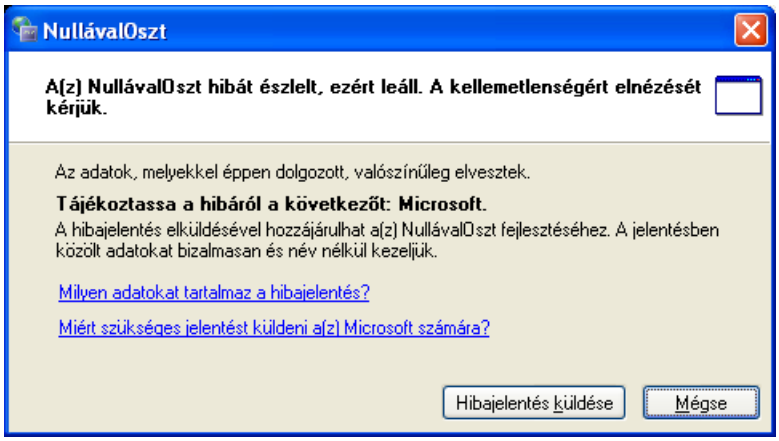
Ha a programot a fejlesztői környezetből futtatjuk, és a futás során következik be valamilyen hiba, akkor a vezérlés visszatér a kódszerkesztő ablakhoz, amelyben megjelenik a hiba helye és magyarázata. **A forráskód javítása előtt a *Debug*/*Stop Debugging* (, *Ctrl+Alt+Break*) paranccsal lépjünk ki ebből az úgynevezett nyomkövető-hibakereső üzemmódból!**

Hibás program futtatása esetén a kódszerkesztő alatt megjelenik a hibák listája (*Error List*). Itt is előfordulhat, hogy egyetlen hiba további hibákat generál.



Futási hiba (nullával való osztás) megjelenítése

Ha nincs elegendő jogunk a hibakereső üzemmódban történő programindításhoz, akkor a *Build*/*Build* parancs segítségével fordítsuk le a forráskódot, majd futtassuk a *bin\Release* mappába kerülő *.exe* fájlt. Ebben az esetben az operációs rendszer jelzi a futási hibát.



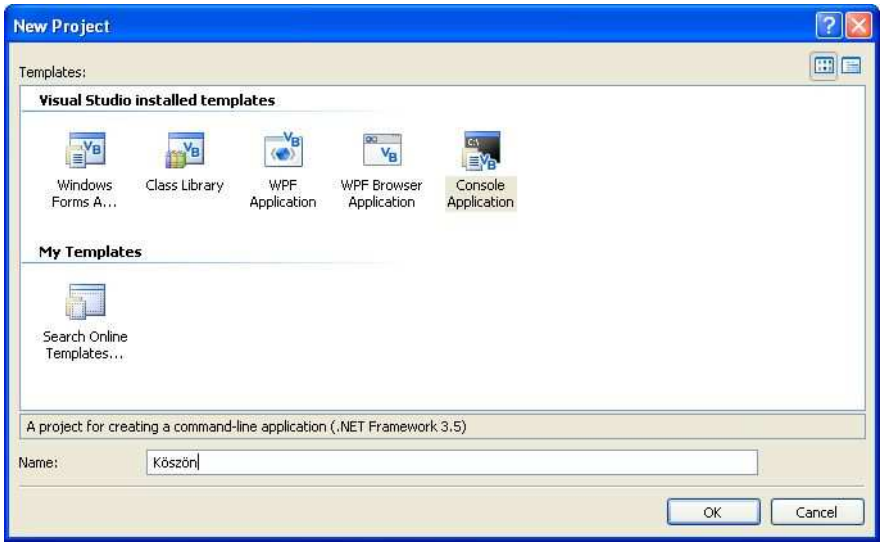
Futási hiba a lefordított program futtatásakor

A munka végén a *File* menü *Close Project* parancsával zárjuk be a projektet.

Konzolalkalmazás készítése

Konzolalkalmazás létrehozása

A konzolalkalmazások a Windows parancssori ablakában futnak. Az ablak karakteres felületén keresztül kommunikálnak a felhasználóval.



Konzolalkalmazás létrehozása

Írjuk meg a fenti, köszönő programot konzolalkalmazásként! Először csak egy üzenetet jelenítünk meg a képernyőn.

Válasszuk ki a *File* menü *New project* parancsát (, *Ctrl+N*). A megjelenő *New Project* ablakban kattintsunk a *Console Application* sablonra. A *Name* szövegdobozba írjuk be a projekt nevét (*Köszön*). Ha rákattintunk az OK gombra, létrejön a konzolalkalmazás váza.

A konzolalkalmazások forráskódját az úgynevezett standard modulok tartalmazzák, melyeket a *Module … End Module* kulcsszavak jelzik. A standard modul mindig tartalmaz egy *Main* nevű eljárást (*Sub Main … End Sub*). Az operációs rendszer ezt az eljárást indítja el a program futtatásakor.

A parancssori ablakba a *Console.WriteLine* utasítással[7] írhatjuk ki az utána szereplő, zárójelben és idézőjelek között álló szöveget. Kattintsunk a *Sub* és *End Sub* utasítások közötti üres sorra, majd a sor elejétől indulva gépeljük be a következő utasítást:

```
Console.WriteLine("Írd be a neved!")
```

Az intelligens súgó most is segíti a forráskód szerkesztését.

A folytatás előtt mentsük el a projektet!

A program futtatása – várakozás az ablak bezárására

Futtassuk a programot! Az indítás után egy pillanatra feltűnik előttünk a fekete hátterű parancssori ablak, de azonnal be is zárul. Egyszerűbb esetben egy újabb utasítás, a *Console.ReadLine*[8] alkalmazásával késleltethetjük az ablak bezárását. A *ReadLine* hatására csak az *Enter* lenyomása után zárul be az ablak. Írjuk be az utasítás a *WriteLine*-t követő sorba!

```
Module Module1

    Sub Main()
        Console.WriteLine("Írd be a neved!")
        Console.ReadLine()
    End Sub

End Module
```

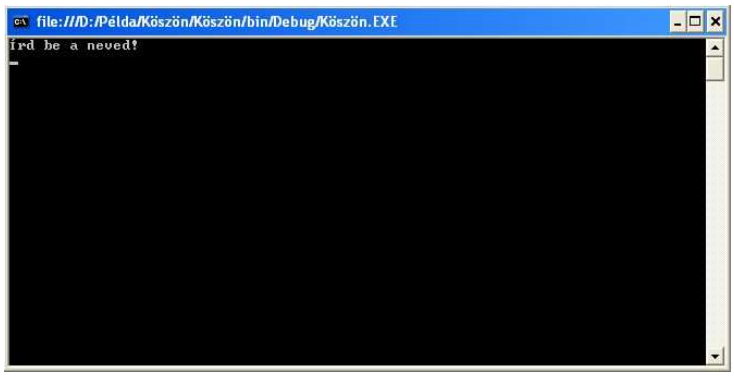
A ReadLine utasítással kiegészített program

Mentsük a megoldást, és futtassuk a programot! Megjelenik előttünk a parancssori ablak az üzenet szövegével. Megfigyelhetjük, hogy a *WriteLine* utasítás a karaktersorozat kiírása után sort emel, a kurzor a következő sorban villog.

[7] Valójában a *Console* objektum *WriteLine* metódusát hívjuk meg.
[8] A *Console* objektum *ReadLine* metódusát hívjuk meg.

Jegyezzük meg az *exe* fájl helyét a háttértáron, ami az ablak címsorában látható! Bár a teljes elérési út függ a mentésnél megadott helytől, a fájl mindig a *Debug* nevű almappában található.

Az *Enter* lenyomásával zárjuk be az ablakot. Így visszatérünk a fejlesztőrendszerhez.



Első programunk a parancssori ablakban

Beolvasás konzolalkalmazásban

A *Console.ReadLine* nem csak az Enter billentyű lenyomására vár, hanem fogadja a felhasználó által beírt karaktereket. Az Enter a gépelés befejezését jelzi.

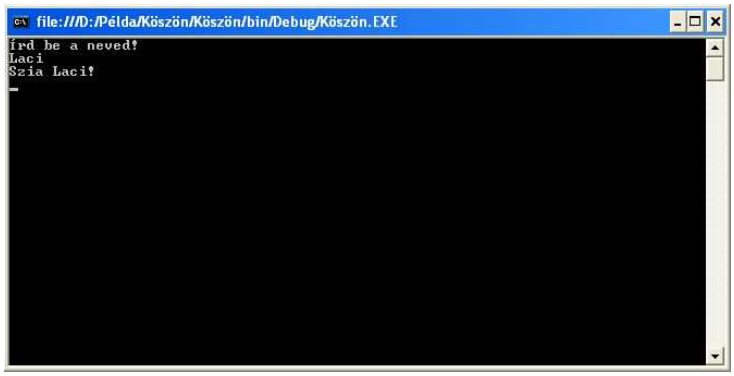
A beolvasott nevet a már megismert módon, az &-jellel fűzzük az üdvözlő szöveghez. Az üzenetet a *Console.WriteLine* utasítással írjuk ki a képernyőre:

```
Console.WriteLine("Szia " & Console.ReadLine() & "!")
```

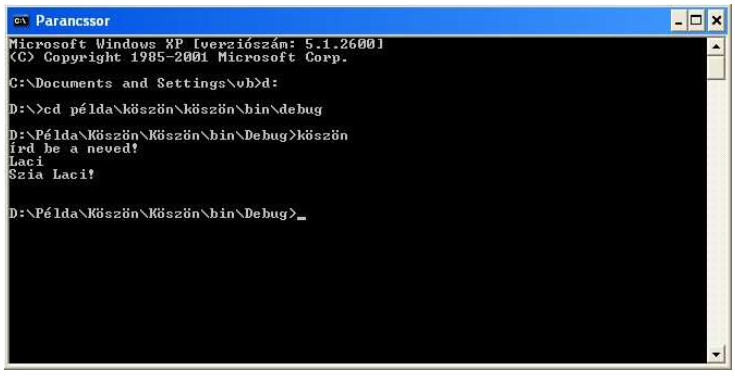
Ügyeljünk arra, hogy az utolsó, *Console.ReadLine()* utasítás megmaradjon a programban! Az esetlegesen előforduló programhibákat a Windows-alkalmazásoknál bemutatott módon javíthatjuk.

Tanulmányaink során megismerkedünk majd a beolvasott adatok tárolásával.

A konzolalkalmazásokat a Windows parancssorából is futtathatjuk. A parancssori ablakot a *Start*/*Minden program*/*Kellékek* csoportban találjuk. A program indításához írjuk be az exe-fájl teljes elérési útját, vagy lépjünk be a fájlt tartalmazó mappába. A parancssori ablak a program futtatása után sem zárul be, így itt nincs szükség a várakozást megvalósító *Console.ReadLine()* utasításra.



A program futtatása



Futtatás a parancssori ablakban

Az eszközkészlet használata

Az eszközkészletet (*Toolbox*) saját igényeinknek megfelelően módosíthatjuk. Az eszközkészlet kezelésének részletes leírását a Visual Studio dokumentációjában találjuk. Itt két hasznos lehetőséget mutatunk be.

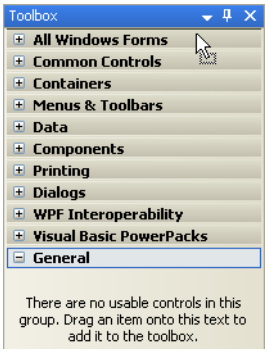
A panelek és vezérlőelemek átrendezése

Gyűjtsük össze a Windows-alkalmazásokban használt leggyakoribb vezérlőelemeket egy külön panelen (*Tab*-on)! Ehhez a *General* panelt fogjuk felhasználni, bár új panelt is létrehozhatunk (jobb egérkattintás egy panelre, majd *Add Tab*).

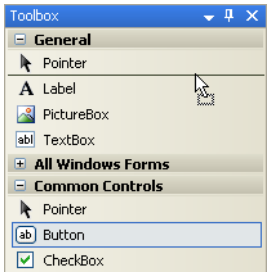
Először fogjuk meg az egérrel a *General* panel címsorát, és helyezzük az *All Windows Forms* panel elé a listában! Majd nyissuk ki a *Common Controls* panelt, és a szükséges elemeket tegyük át a *General* panelre.

A vezérlőelemeket az egyes paneleken belül is mozgathatjuk.

Az eredeti állapotot egy panel címsorára történő jobb egérkattintással, majd a *Reset Toolbox* parancs kiválasztásával állíthatjuk vissza.



A General panel áthelyezése a Toolbox tetejére

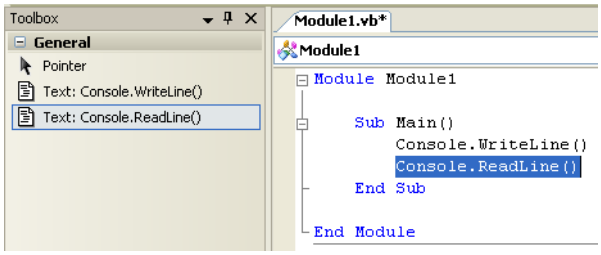


A Button vezérlőelem áthelyezése a General panelre

Kódrészletek tárolása az eszközkészletben

A konzolalkalmazások forráskódjának elkészítése során sokszor be kell írnunk a *Console.WriteLine*, *Console.ReadLine* utasításokat. Bár az intelligens súgó segíti a kód begépelését, a munkát egyszerűsíthetjük az eszközkészletben tárolható kódrészletek segítségével.

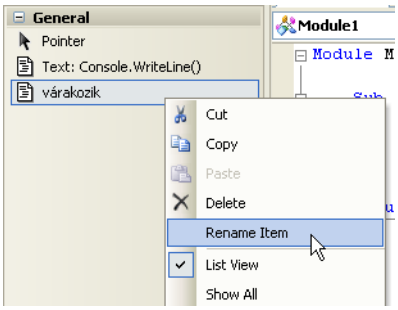
Jelöljük ki a kód egy részletét (például a *Console.ReadLine* utasítást), majd az egér segítségével húzzuk rá az eszközkészlet (*Toolbox*) általános (*General*) paneljére. Ezzel tároltuk a kódrészletet. A továbbiakban a panelről az egér segítségével a forráskód tetszőleges helyére beilleszthetjük. **A beillesztést dupla kattintással szintén elvégezhetjük!**



Kódrészlet elhelyezése az eszközkészletben

A *General* panelen több soros kódrészletet is tárolhatunk, akár konzol-, akár Windows-alkalmazások esetén Ez utóbbiaknál külön *General* panel tartozik a tervező-, illetve a kódszerkesztő nézethez.

Az eszközkészletben elhelyezett kódrészletek listáját áttekinthetőbbé tehetjük, ha funkciójukra utaló nevet adunk az egyes elemeknek. Ezt a jobb egérkattintásra megjelenő helyi menü átnevezés (*Rename Item*) parancsával tehetjük meg.



Kódrészlet átnevezése az eszközkészletben

A forráskód szerkesztése

A forráskód begépelését megkönnyítő intelligens súgót a *Windows-alkalmazás készítése* fejezetben már bemutattuk. A továbbiakban néhány, a szerkesztéssel kapcsolatban előforduló tevékenységet tekintünk át.

Billentyűparancsok

A tervező- és a kódszerkesztő nézet között az ablak tetején megjelenő fülekkel válthatunk. Billentyűparancsok használatával azonban gyorsíthatjuk a munkát.

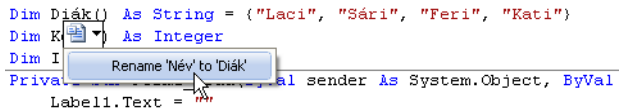
Váltás kódszerkesztő nézetre	*F7*
Váltás tervezőnézetre	*Shift + F7*
Váltás a nézetek között, ha már mindkettőt megnyitottuk (valójában a megnyitott panelek között vált)	*Ctrl + Tab*
Az intelligens súgó (automatikus kódkiegészítés) utólagos előhívása	*Ctrl + szóköz*
Az aktuális projekt futtatása hibakereső (debugger) üzemmódban	*F5*
Az aktuális projekt futtatása hibakeresés nélkül	*Ctrl + F5*

A szerkesztés, futtatás legfontosabb billentyűparancsai

A változónevek módosítása

A progam írása során előfordulhat, hogy egy-egy változó nevét utólag szeretnénk módosítani – természetesen az egész forráskódban. Ehhez a jobb egérgombbal kattintsunk bárhol a változónévre, majd válasszuk a *Rename* parancsot.

Az átnevezés másik módja, ha a változó deklarációjában egyszerűen átírjuk a régi azonosítót az újra. Ekkor megjelenik egy intelligens címke, melynek segítségével az egész forráskódban elvégezhetjük az átnevezést.



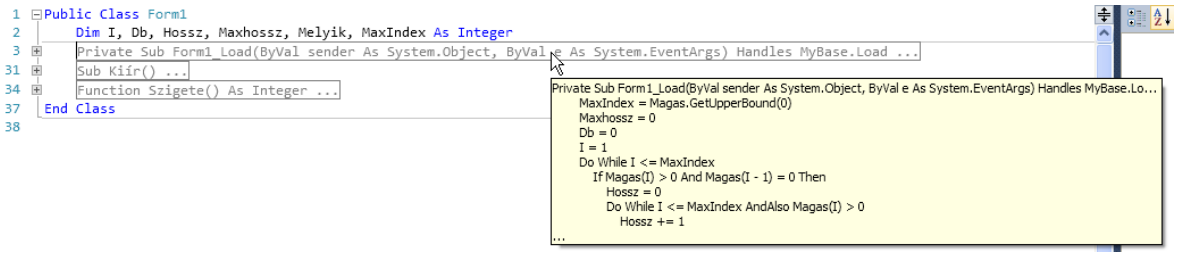
Változó átnevezése a deklarációban

Kódblokkok használata

A Visual Studio a forráskód áttekintését kódblokkok definiálásával könnyíti meg. A kódblokkok bezárhatók, illetve kinyithatók a forráskódban. Megjegyezzük, hogy kódblokkok alkalmazására a Visual Studio 2008 Express változatában nincs lehetőségünk.

A fejlesztőrendszer a forráskódban előforduló definíciókhoz (osztály, eljárás, függvény, struktúra stb.) automatikusan kódblokkot rendel. Ezt a definíció kezdősoránál látható ⊟ jel mutatja. A ⊟ jel fölé mozgatva az egeret a kódszerkesztő sötétebb háttérrel emeli ki a kódblokkhoz tartozó sorokat.

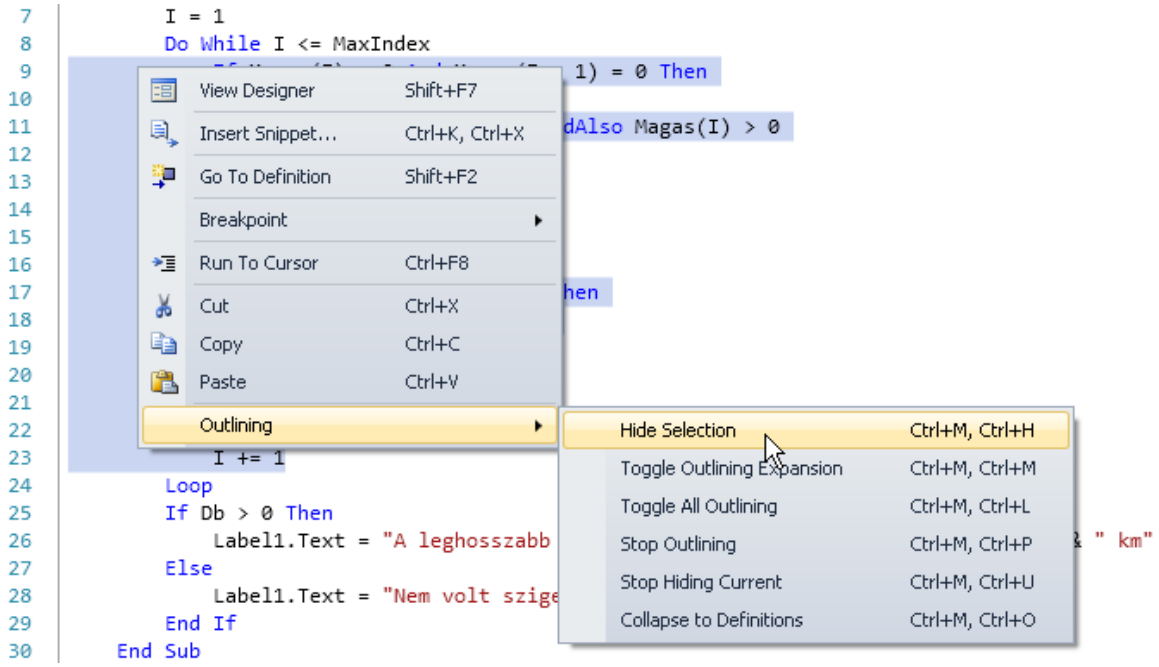
Ha rákattintunk a ⊟ jelre, akkor bezárul a kódblokk. Amikor egy bezárt blokk fölé mozgatjuk az egeret, akkor megjelenik a forráskód néhány sora.



Bezárt kódblokkok (Visual Studio 2010 Express)

A kódblokkot a + jelre kattintással nyithatjuk ki.

Kódblokkot magunk is definiálhatunk. Ehhez jelöljük ki a forráskód sorait, majd a jobb egérkattintásra megjelenő helyi menüben válasszuk az *Oulining/Hide selection* parancsot. Mivel a kijelölt sorok a kódblokk bezárásakor teljesen eltűnnek, ciklusok, feltételes elágazások esetén célszerű az első és utolsó sort kihagyni a kijelölésből. Szemléletesebbé tehetjük a programot az első sorhoz fűzött megjegyzéssel.



Kódblokk létrehozása

A forráskódban tetszőleges számú kódblokkot létrehozhatunk. A kódblokkok egymásba ágyazhatók.

```
        Do While I <= MaxIndex ' végignézzük a méréseket
            If Magas(I) > 0 And Magas(I - 1) = 0 Then ' szigetet találtunk
                Hossz = 0
                Do While I <= MaxIndex AndAlso Magas(I) > 0 ' a sziget végének keresése
...
                Loop
                If I < MaxIndex Then ' maximumkiválasztás
...
                End If
            End If
            I += 1
        Loop
```

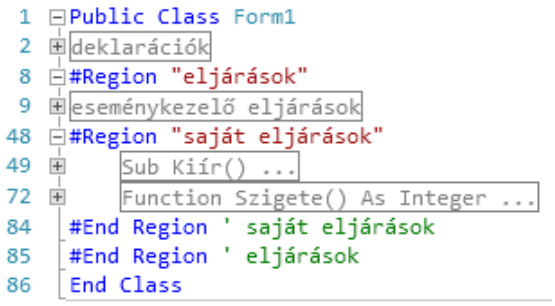
Kódblokkok egymásba ágyazása

Az általunk létrehozott kódblokkot az *Outlining* helyi menü *Stop Hiding Current* parancsával szüntethetjük meg. A kijelöléssel létrehozott kódblokkok így véglegesen megszűnnek.

Definíciókat összefogó kódblokkot a *#Region … #End Region* direktívával is kijelölhetünk:

```
#Region kódblokknév
  ' a kódblokkhoz tartozó utasítások
#End Region
```

ahol a *kódblokknév* egy tetszőleges sztringliterál. Célszerű a kódblokk funkciójára utaló elnevezést alkalmazni. Ha a *Region* direktívával létrehozott kódblokkot bezárjuk, akkor a kódblokk neve jelenik meg a forráskódban.



A Region direktívával létrehozott kódblokkok

A *Region* direktívával jelölt kódblokkokat egymásba ágyazhatjuk.
Eljárások, illetve függvények belsejében nem definiálhatunk kódblokkot a *Region* direktívával.

Osztályok importálása

Az objektumok tulajdonságainak és metódusainak minősített hivatkozása esetenként nagyon hosszú lehet, így csökkenti a forráskód áttekinthetőségét. Ilyenkor célszerű a kódfájl elején importálni a megfelelő osztályt, illetve névteret:[9]

```
Imports névtér.osztály
```

Ha például gyakran használunk matematikai függvényeket, akkor az

```
Imports System.Math
```

következtében elhagyhatjuk a minősítést a matematikai függvények neve elől:

`Math.Sqrt(x)` helyett csak `Sqrt(x)`.

A Visual Basic programok szerkezete

A programok szerkezetét a *Programozási útmutató* ismerteti részletesen. Itt csak az első programok készítéséhez szükséges ismereteket tekintjük át.

A Windows-alkalmazások szerkezete

Windows-alkalmazásaink egyetlen ablak (űrlap, form) osztálydefinícióját tartalmazzák. Az osztálydefiníciót egy kezdő és egy záró utasítás határolja (*Class … End Class*). A *Class* után írjuk az osztály nevét (általában *Form1*):

```
Public Class osztálynév
  …
End Class
```

A *Public* kulcsszó arra utal, hogy az osztálydefinícióra más kódfájlokból is hivatkozhatunk.[10]
Az osztálydefinícióban a változók deklarációit és az eljárások (köztük az eseménykezelő eljárások) definícióit helyezzük el. Az eljárásokon kívül deklarált változók alapértelmezés szerint *Private* hatókörűek, azaz csak az osztályon belül érhetők el. Ezekre a változókra az osztály bármely eljárásában hivatkozhatunk.

```
Public Class osztálynév
  osztályszintű változók deklarációi
  eljárások/eseménykezelő eljárások definíciói
End Class
```

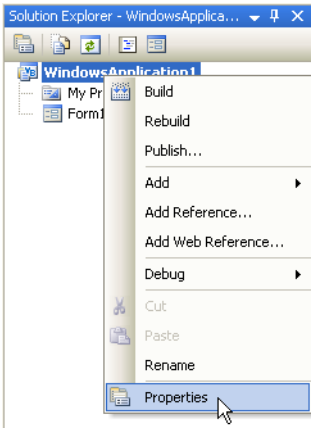
> A *Form1* a .NET keretrendszerben definiált *Form* osztály leszármazottja. Rendelkezik a *Form* osztály tulajdonságaival és metódusaival. A tulajdonságok kezdőértékeit például a *Form* osztályból örökli. Eseménykezelő eljárásaink a leszármazott osztály metódusait bővítik. A fordítási, illetve futási időben megadott tulajdonságok a leszármazott osztály tulajdonságait módosítják.

[9] Lásd még *A programok szerkezete* szakaszt a *Programozási összefoglalóban*.
[10] Pontosabban szólva a *Public* hatókör korlátozás nélküli elérhetőséget jelent.

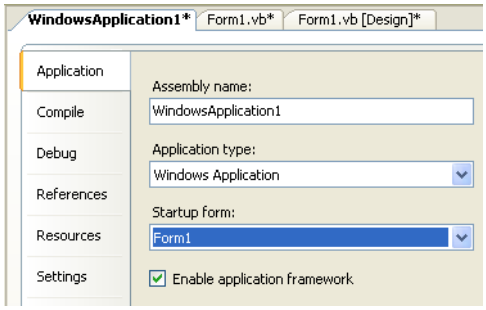
A kezdőablak beállítása

A programablaknál általában meghagyjuk a *Form1* azonosítót, de megadhatunk másik nevet is a forráskódban vagy a tulajdonságablakban (*Properties*).[11]

Ügyeljünk azonban arra, hogy a fordítóprogramnak ismernie kell a programablak (a program futtatásakor megnyíló ablak) azonosítóját. Ezt a Windows-alkalmazás tulajdonságainál állíthatjuk be, melyek a megoldástallózóban a jobb egérkattintásra megnyíló helyi menüből érhetők el (*Properties*). A tulajdonságok között az *Application*/*Startup form* legördülő listájából választhatjuk ki a program indításakor megnyíló ablakot. Ennek általában csak akkor van szerepe, ha több ablakot is létrehozunk a projektben.



Az alkalmazás tulajdonságainak a megnyitása



A Startup form az alkalmazás tulajdonságai között

Ha az űrlap azonosítóját a forráskódban vagy a tulajdonságablakban módosítjuk, akkor automatikusan módosul a *Startup form* értéke. Ha azonban töröljük a *Form1*-et, és egy másik alkalmazásból, esetleg más azonosítóval vesszük át az osztály definícióját, akkor nekünk kell beállítanunk a kezdőablak nevét!

A konzolalkalmazások szerkezete

Konzolalkalmazásaink egyetlen programegységből, úgynevezett modulból állnak. A modult egy kezdő és egy záró utasítás határolja (*Module … End Module*). A kezdő utasítás után írjuk a modul nevét (általában *Module1*):

```
Module modulnév
  …
End Module
```

A modul változódeklarációkat és eljárásokat tartalmazhat. Az eljárásokon kívül deklarált változókra a modul minden eljárásában hivatkozhatunk.

```
Module modulnév
  változók deklarációi
  eljárások definíciói
End Module
```

Egy konzolalkalmazás futtatásakor a *Module1* modul *Main* (fő) nevű eljárása kerül végrehajtásra, ennek kötelezően szerepelnie kell a forráskódban.[12] Egy új konzolalkalmazás létrehozásakor ezek az egységek a szükséges kezdő és záró utasításokkal együtt automatikusan bekerülnek a forráskódba:

```
Module Module1
  Sub Main()
    ' utasítások
  End Sub
End Module
```

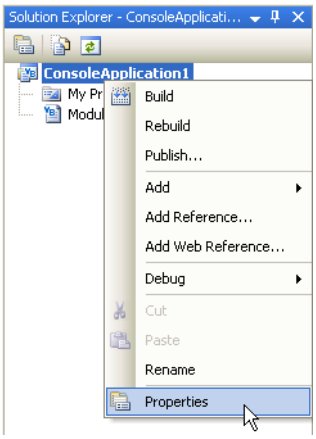
A modulnév módosítása

A modul nevét a forráskódban módosíthatjuk. **A programot indító modulnál azonban az alkalmazás tulajdonságainál meg kell adnunk az új azonosítót!** A tulajdonságok a megoldástallózóban a jobb egérkat-

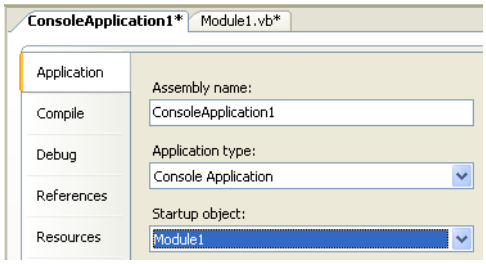
[11] A megoldástallózóban (*Solution Explorer*) a jobb egérkattintásra megnyíló helyi menü *Rename* parancsával az osztálydefiníciót tartalmazó kódfájlt nevezhetjük át.

[12] A Windows-alkalmazások is rendelkeznek *Main* eljárással, de ezt a fordítóprogram automatikusan létrehozza. A *Main* eljárás nyitja meg például a programablakot.

tintásra megnyíló helyi menüből érhetők el (*Properties*). Az *Application/Startup object*[13] legördülő listájából válasszuk ki az új nevet vagy a *Main* eljárást![14]



Az alkalmazás tulajdonságainak a megnyitása



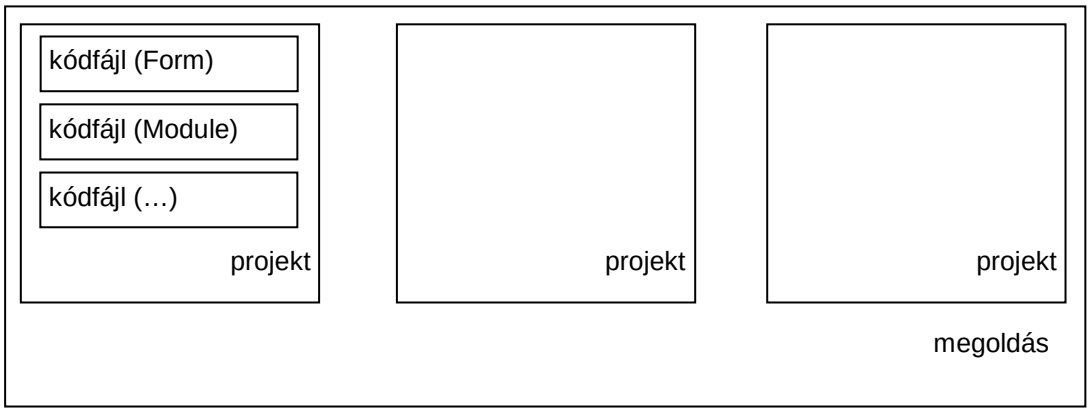
A Startup object az alkalmazás tulajdonságai között

Projektek kezelése

Megoldások és projektek

A Visual Studio összetett feladatok, bonyolult programok készítésére alkalmas. Ezeknél a fejlesztéseknél gyakran előfordul, hogy több programozó dolgozik egy-egy részfeladaton. A program sok részből áll, az egyes részeket egységes rendszerré kell összefogni.

A Visual Studio **megoldásnak** (*solution*) nevezi az összetett feladatot. A megoldás magában foglalja a készülő program összetevőit. Egy összetevő hozhatja létre a képernyőn megjelenő ablakot, vezérelheti a felhasználóval történő kommunikációt. Egy másik összetevő tarthatja a kapcsolatot egy adatbázis- vagy webszerverrel stb.



Egy Visual Studio alkalmazás szerkezete

A megoldás összetevőit projekteknek hívjuk. A megoldás egy vagy több projektből áll. A **projekt** az adott alkalmazáshoz szükséges fájlokat fogja össze. Ilyen fájl írhatja le például a program futtatásakor a képernyőn megjelenő ablak tulajdonságait. Egy másik fájlban helyezhetjük el az eljárások forráskódját vagy a képernyőn megjelenő képeket stb. A megoldáson belül az egyes projektek akár más és más programozási nyelven készülhetnek.

Programjaink forráskódját **kódfájlokban** találjuk. Ezek *.vb* kiterjesztésű szövegfájlok. Kódfájl tartalmazza például az ablakok megjelenését és viselkedését leíró űrlapok (*Form*-ok) definícióit. A standard modulok (*Module*-ok) kódfájljaiban további változókat deklarálhatunk, eljárásokat definiálhatunk.

A Visual Studio adminisztrációs fájlokban tárolja a megoldás és a projektek jellemzőit. Az adminisztrációs fájlok a megoldások és projektek létrehozásakor jönnek létre.

---

13 Windows-alkalmazásnál: *Startup Form*

14 A megoldástallózóban (*Solution Explorer*) a jobb egérkattintásra megnyíló helyi menü *Rename* parancsával a forráskódot tartalmazó fájlt nevezhetjük át.

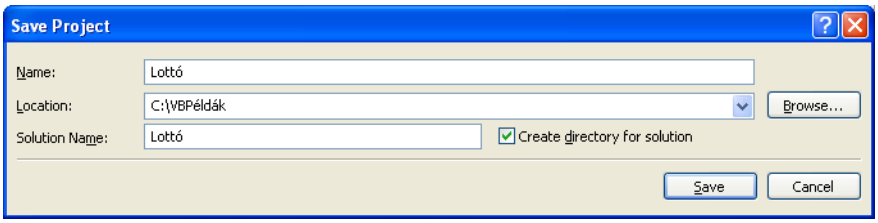
Új projekt létrehozása

Feladataink megoldásához általában elegendő lesz egyetlen projektből álló megoldást készítenünk. Ehhez a *File*/*New Project* parancsot használjuk. Válasszuk ki a megfelelő sablont (*Windows Application*, illetve *Console Application*). A *My Templates* csoportban az általunk előzőleg létrehozott sablonokat láthatjuk (lásd később). A létrehozásnál adjunk beszédes nevet (*Name*) a projektnek!

Amint létrejön a projekt, létrejön a projektet tartalmazó megoldás is. A megoldástallózó azonban csak a projektet mutatja.

A projekt mentése

A munka során ne az egyedi kódfájlokat, hanem a teljes projektet mentsük (*File*/*Save all*, , *Ctrl+Shift+S*)! Az első mentésnél adjunk beszédes nevet mind a projektnek (*Name*), mind a megoldásnak (*Solution Name*)! Figyeljünk oda a mentés helyére (*Location*)! Hozzunk létre saját mappát projektjeinknek, és ne fogadjuk el a Visual Studio által felajánlott elhelyezést! A *Create directory for solution* jelölőnégyzet segítségével készíthetünk saját mappát a projektnek (illetve a megoldásnak)!

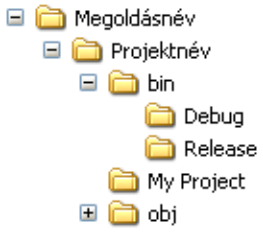


A projekt első mentésénél megjelenő ablak

A fejlesztőrendszerből (debugger üzemmódban) történő futtatáshoz megfelelő hozzáférési jogokra van szükség. Ezért nem célszerű hálózati mappába menteni a projektet. A megfelelő hozzáférési jogok hiánya esetén használhatjuk a *Start Without Debugging* (*Ctrl+F5*) parancsot is (lásd a *Windows alkalmazás készítése* fejezetet).

A projektet a *File*/*Close project* paranccsal zárhatjuk be. A *Discard* feliratú parancsgombbal elvethetjük a módosításokat.

A projekt fájljai



A projekt mappaszerkezete

A Visual Studio a projekt mentésénél összetett mappaszerkezetet alakít ki a megadott elérési úton. A megoldással megegyező nevű mappában jönnek létre a megoldáshoz tartozó egyes projektek mappái.

A konzolalkalmazások forráskódja a *megoldásnév\projektnév* mappában helyezkedik el *.vb* kiterjesztéssel (például: Module1.vb). A Windows-alkalmazások forráskódja ugyanezen mappa két fájljában található. Az *osztálynév*.vb fájl az osztály definícióját tartalmazza a változódeklarációkkal és eljárásokkal együtt, míg az *osztálynév*.Designer.vb fájlban az ablak beállításait, szerkezetét tároljuk (például Form1.vb, illetve Form1.Designer.vb).

A *.vb* kiterjesztésű fájlok (a mappákban lévő sok más fájllal együtt) a Jegyzettömbbel is olvasható szövegfájlok. Módosításukhoz azonban mindig a Visual Studio fejlesztőrendszerét használjuk!

A projekt *bin* mappájában találjuk a futtatható *.exe* fájlokat. A *Debug* mappa *.exe* fájlja akkor jön létre, amikor a fejlesztői környezetben először adjuk ki a *Debug*/*Start Debugging* parancsot. Ez valójában a nyomkövető-hibakereső futtatásnak felel meg.

A *Release* mappába kerülő *.exe* fájlt a *Build*/*Build* parancs kiadása hozza létre. Ez a parancs felel meg a klasszikus programozási nyelvek fordítási (*compile*) parancsának. A *Release* mappa *.exe* fájlja képviseli az elkészült programot. Ezt a fájlt futtathatjuk a fejlesztői környezet telepítése nélkül, ezt a fájlt másolhatjuk más számítógépekre stb.

Összetett alkalmazások esetén a program működéséhez további fájlokra van szükség. A *Build* menü *Publish* parancsával telepítőkészletet készíthetünk az alkalmazásunkhoz. Ezt a lehetőséget itt nem tárgyaljuk.

Projekt megnyitása

Meglévő projektet a *File* menü *Open Project* (*Ctrl+O*) parancsával nyithatunk meg. Ha az Intézőben duplán kattintunk a projekt mappájában lévő *.vbproj* fájlra, akkor a projekttel együtt megnyílik a fejlesztői környezet. [15] Ugyanezt érjük el a megoldás mappájában lévő *.sln* fájllal is. Célszerű mindig az *sln* fájlt megnyitni.

[15] A Visual Basic 2010 dupla kattintásra nem nyitja meg a 2008-as változattal készült fájlokat. A megnyitáshoz a *File*/*Open* parancsot használjuk. Részletesebben lásd a gyakorlatok megoldásánál található *Olvassel.pdf* fájlban.

Egy program szerkesztéséhez, módosításához a projektet (illetve a megoldást) nyissuk meg, ne pedig a modul forráskódját tartalmazó fájlt! A *File* menü *Open File* () parancsa helyett válasszuk az *Open Project* (*Ctrl+O*) parancsot. A *File* menüben megtaláljuk az utoljára megnyitott projektek listáját (*Recent Projects*). Itt se a *Recent Files* listát használjuk a megnyitáshoz!

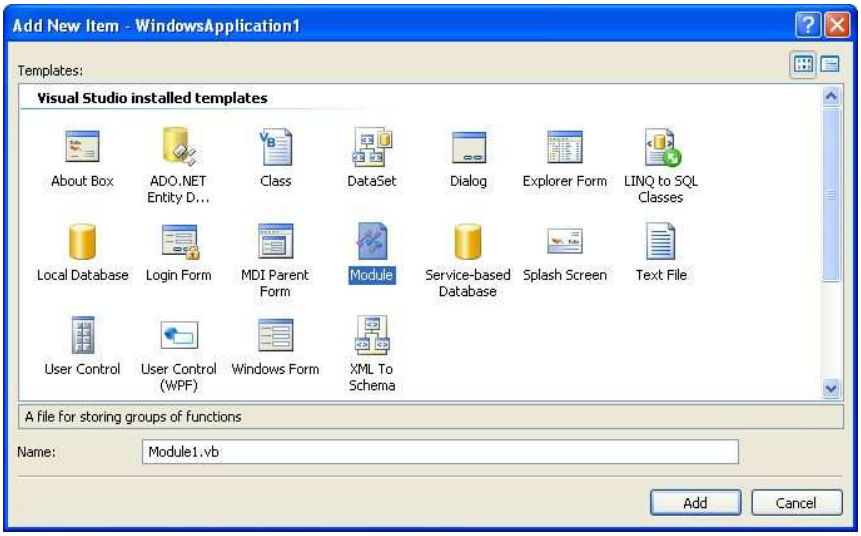
A megoldástallózó megmutatja a megnyitott projekthez tartozó fájlokat. Ha egy projekt megnyitásánál nem látjuk kódszerkesztő vagy a tervezőablakot, akkor a megoldástallózóban a jobb egérgombbal kattintsunk a *.vb* kiterjesztésű fájlra, és válasszuk a *View Code*, illetve a *View Designer* parancsot. Ugyanezt érhetjük el a megoldástallózó eszköztárának ikonjaival (*View Code*:  , *View Designer*: ). A kódszerkesztő ablakra az *F7*, a tervező nézetre pedig a *Shift+F7* funkcióbillentyűvel is átválthatunk.

Dupla kattintással konzolalkalmazásnál a kódszerkesztő ablak, Windows-alkalmazásnál pedig a tervezőablak nyílik meg.

Kódfájlok használata

Egy projekt az űrlap, illetve a *Main* eljárás forráskódját tartalmazó fájl mellett további kódfájlokat tartalmazhat. A tankönyv forrásfájljai között például adatokat tartalmazó kódfájlokat is találunk.

Új kódfájl készítéséhez a megoldástallózóban kattintsunk a jobb egérgombbal a projekt nevére, majd válasszuk az *Add*/*New item* parancsot. A megjelenő ablakban jelöljük ki a kódfájl típusát és adjuk meg a nevét. A leggyakrabban a program adatait és további eljárásait tartalmazó standard modulokat (*Module*) adunk a projekthez.



Új kódfájl hozzáadása

Létező kódfájl felvételéhez a megoldástallózóban kattintsunk a jobb egérgombbal a projekt nevére, majd válasszuk az *Add*/*Existing item* parancsot. A megnyíló ablakban kiválaszthatjuk a fájlt.

Ügyeljünk arra, hogy a kódfájlokban szereplő változókat *Public* minősítéssel lássuk el, különben nem hivatkozhatunk rájuk egy másik modulból.[16]

A kódfájl törléséhez a megoldástallózóban kattintsuk jobb egérgombbal a fájl nevére, majd válasszuk a *Delete* parancsot. **A fejlesztőrendszer nem törli magát a fájlt**, csak kihagyja a projekthez tartozó fájlok listájából.

Mentés másként

A Visual Studio projektjei, megoldásai számos fájlt tartalmaznak, melyeket mappák meghatározott rendszerébe rendez a fejlesztőeszköz. Ezért a projektre vonatkozóan nem létezik az alkalmazásoknál megszokott *Mentés másként* parancs. A *File* menü *Save .. as* parancsa csak az adott kódfájlt menti el, nem pedig az egész projektet.

A projekt/megoldás duplázását az Intézővel (vagy más fájlkezelővel) végezhetjük el. Másoljuk át a mappát a tartalmával együtt egy másik helyre. A megoldás mappáját átnevezhetjük, de a benne lévő fájlok, mappák nevét ne módosítsuk, mert ez hibaüzenetekhez vezet a projekt megnyitásánál!

Létező kódfájlok felülírása

Megtörténhet, hogy egy projektben szeretnénk felülírni a már létező fájlokat. Az egész projekt helyett például csak a *Form1*-et vagy a *Main* eljárást tartalmazó kódfájlt vesszük át egy másik projektből. Ez az eset

[16] Az eljárások és függvények alapértelmezés szerint *Public* elérésűek. Részletesebben lásd a Programozási útmutatóban.

fordul elő például, amikor a tankönyv forrásfájljai között található *.vb* fájlokat emeljük be a projektbe, általában a *Form1* helyére.

Egyszerűbb esetben ezeket a fájlokat egy fájlkezelővel is átmásolhatjuk a másik projektből (vagy a forrásfájlok közül), felülírva a már létező fájlokat. Ha a fájlok felülírását megnyitott projektnél végezzük, akkor a fejlesztői környezet rákérdez a frissítésre.

Figyelmeztető üzenet a fájl felülírásánál. Válasszuk a Yes to All gombot!

Ha olyan projektbe másoljuk át a *.vb* fájlokat, amelyet már előzőleg futtattunk, akkor megtörténhet, hogy az indítás után még a régi program fog megjelenni a képernyőn. A fejlesztőrendszer ugyanis csak akkor fordítja újra a programot, ha megváltoztatjuk a forráskódot. Ehhez elegendő bárhová – akár a forráskód végére – egy üres sort beilleszteni, máris az új programot hozzuk működésbe a futtatásnál.

A fájlokat a megoldástallózóban a jobb egérkattintásra feltűnő helyi menü *Add*/*Existing Item* parancsával is felvehetjük a projektbe. Mindig az osztálydefiníciót tartalmazó *.vb* fájlt illesszük be, ne a tulajdonságokat tartalmazó *Designer* fájlt! A *Designer* fájlt a fejlesztői környezet automatikusan átveszi.

Új fájlok beillesztésekor szükség lehet az alkalmazás *Startup form* tulajdonságának a módosítására (lásd *A modulnév módosítása* szakaszt).

Összetett projekt esetén (például referenciák, további erőforrások alkalmazásánál) természetesen ez az egyszerű módszer nem használható.

Projektcsoportok (megoldások) alkalmazása

A Visual Studio az összetett alkalmazások projektjeit megoldásokba (*Solutions*) rendezi. A projektek mappái a megoldás mappájában helyezkednek el.

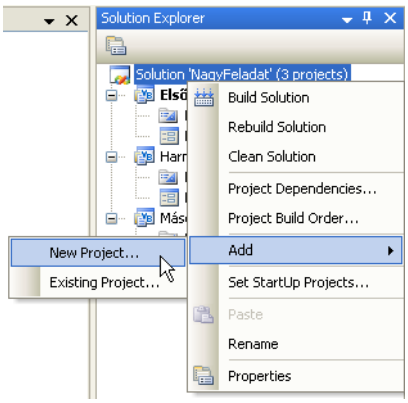
Ha létrehozunk egy új projektet, az önmagában is egy megoldást alkot. A különbség azonban akkor válik láthatóvá, amikor újabb projekteket veszünk fel a megoldásba.

> Bár a megoldás a nagyobb alkalmazások létrehozását segíti elő, a tanulás során felhasználhatjuk egy program különböző változatainak, módosításainak a tárolására, egyszerű áttekintésére.

Több projektből álló megoldás készítése

Az első projekt létrehozásakor létrejön egy megoldás is, amit a projekt mappaszerkezete jelez. A Visual Studio a mentésnél a projekt mellett a megoldás nevére szintén rákérdez (*Solution Name*). Célszerű a projektétől eltérő nevet választani.

A második projektet a *File*/*Add*/*New project* paranccsal vehetjük fel a megoldásba. Ennek hatására kissé átalakul a megoldástallózó munkaablaka. A lista gyökereként megjelenik a megoldás (*Solution*), amelyből kiágaznak az egyes projektek (a nevek ábécé sorrendjében). A továbbiakban a megoldástallózóban a jobb egérkattintásra előtűnő helyi menü *Add*/*New project* (illetve *Existing project*) parancsával is hozzáadhatunk projekteket a megoldáshoz.

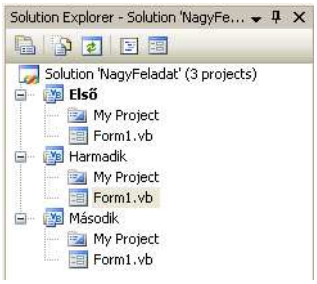


Újabb projekt felvétele a megoldásba

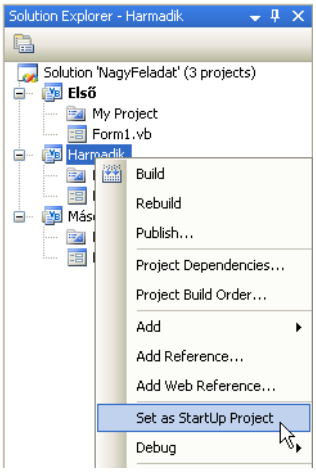
A megoldást a *File*/*Close Solution* paranccsal zárjuk be.

A kezdőprojekt kijelölése

A megoldásnak mindig van egy kezdőprojektje, amelyik elindul, ha a *Debug*/*Start Debugging* parancsot választjuk (▶, F5). A kezdőprojekt neve félkövér betűkkel jelenik meg a megoldástallózóban. Alapértelmezés szerint az elsőként létrehozott projekt lesz a kezdőprojekt.



A kezdőprojekt félkövér névvel látható



A kezdőprojekt beállítása

A kezdőprojektet a megoldástallózóban a jobb egérkattintásra előtűnő helyi menü *Set as Startup Project* parancsával állíthatjuk be.

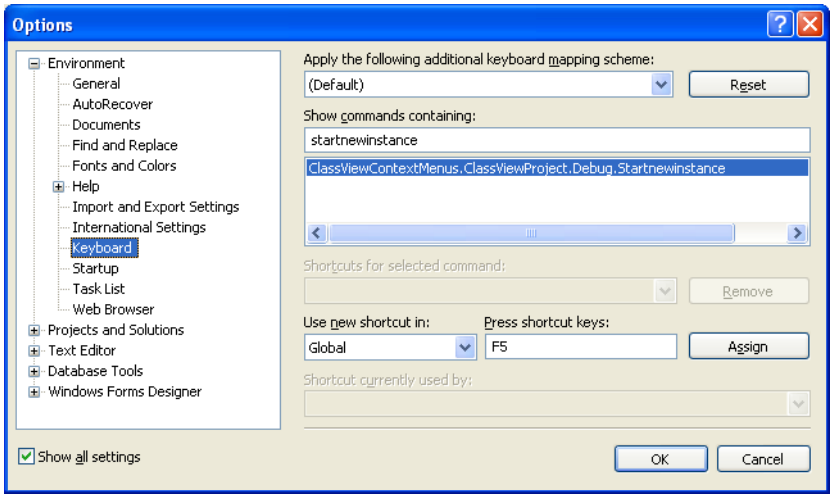
Az aktuális projekt futtatása

Ha nem akarjuk módosítani a kezdőprojektet, akkor a megoldás egy másik projektjét a megoldástallózóban a jobb egérkattintásra előtűnő helyi menü *Debug*/*Start New Instance* parancsával futtathatjuk.

Egyszerűbben érhetjük el ezt a célt, ha módosítjuk az *F5* funkcióbillentyűhöz rendelt parancsot. Nyissuk meg a *Tools*/*Options* ablakot, majd kapcsoljuk be a *Show all settings* jelölőnégyzetet.

Válasszuk ki a listából az *Environment*/*Keyboard* panelt. Gépeljük be a *Show commands containing* szövegdobozba a *startnewinstance* karaktersorozatot (szóközök nélkül!). Kattintsunk a *Press shortcut keys* szövegdobozra, és nyomjuk le az *F5* funkcióbillentyűt. Az *F5* helyett más, általunk kiválasztott funkcióbillentyűt, illetve váltóbillentyű+billentyű parancsot is alkalmazhatunk.[17] Közben a *Shortcut currently used by* szövegdobozban látjuk a jelenleg érvényes hozzárendelést. Az *Assign* gombra kattintással végezzük el a hozzárendelést, majd az OK gombbal zárjuk be az *Options* ablakot.

A hozzárendelés után a megadott billentyűkombináció hatására mindig a megoldástallózóban kiválasztott projekt indul el.



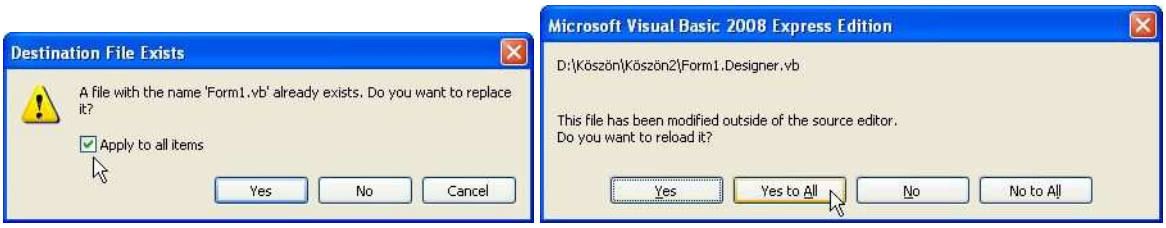
Az F5 hozzárendelése a Start new instance parancshoz

[17] Javasoljuk például az alapértelmezés szerint nem használt *Alt+R* billentyűkombinációt (R: run, futtatás).

Megoldás felhasználása a programváltozatok tárolására

Mint említettük, a megoldás felhasználható a tanulás során egy program különböző változatainak, módosításainak a tárolására. Ehhez hozzunk létre egy projektet, majd készítsük el programunk kezdeti, első változatát.

Adjunk hozzá a megoldáshoz egy újabb projektet, aztán a megoldástallózóban tegyük át az első projekt *Form1* (illetve *Module1*) fájlját az újabb projektbe. Ha nem módosítottuk a fájlneveket, akkor megjelenik egy figyelmeztetés a létező fájlok (*Form1.vb*, illetve *Form1.Designer.vb*) felülírása miatt. Kapcsoljuk be az *Apply to all items* jelölőnégyzetet, majd kattintsunk a *Yes* gombra. Egy újabb figyelmeztetés arra utal, hogy a fájlokat egy másik projektből vettük át. Kattintsunk a *Yes to All* gombra.



Figyelmeztető üzenetek a projekt átírásakor

A program fejlesztését az újabb projektben folytassuk, amely már tartalmazza eddigi fejlesztéseinket. Ezt a folyamatot ismételve tetszőleges számú változatot tárolhatunk.

Mivel a kódszerkesztő ablak tetején lévő fülek a megnyitott fájlok nevét mutatják, könnyebben tájékozódunk, ha bezárjuk a fölösleges fájlokat a szerkesztőablakban. Megtehetjük azt is, hogy a megoldástallózó munkaablakában módosítjuk az azonos, *Form1* fájlneveket. A fájlnév átírása maga után vonja az osztálynév módosítását, ezért célszerű ellenőrizni az újabb projekt *Startup form* tulajdonságát (lásd *A modulnév módosítása* szakaszt).

Ha nem egy *Form1* nevű fájlt teszünk át az új projektbe, akkor megmarad az eredeti fájl (nem írjuk felül). Ekkor kézzel kell törölnünk (például kijelölés után a *Delete* billentyűvel a *Solution Explorer* munkaablakban).

Sablonok készítése

A Visual Studio projektjei számos elemből állnak. Megtaláljuk közöttük az ablakok definícióit tartalmazó fájlokat, a forráskódot és egyéb, a projekt nyilvántartását, adminisztrációját szolgáló fájlokat. Az összetett szerkezet miatt nem létezik a projektre vagy az összetevőkre vonatkozó *Mentés másként* parancs.[18] Ennek hiányát sablonok használatával pótolhatjuk. A projekt egyes összetevőit, vagy magát az egész projektet sablonként menthetjük. A sablonok tartalmazzák a mentésig elvégzett beállításokat, az elkészült forráskódot. Egy új projekt létrehozásánál kiválaszthatjuk az elmentett sablont, így visszakapjuk a sablonban tárolt beállításokat és forráskódot.

Saját sablon készítése

Sablonként a készülő (vagy már kész) projektet, illetve összetevőit menthetjük el. A mentéshez válasszuk ki a *File*/*Export template* parancsot! A sablon létrehozása előtt a Visual Studio szükség esetén rákérdez a projekt mentésére (*Save changes to the following items?*).

A megjelenő varázsló *Choose Template Type* ablakában válasszuk ki a sablon típusát. *Project template* esetén a teljes projektből, *Item template* esetén pedig a következő ablakban kijelölhető elemből készítünk sablont.

Projektsablon készítése

A *Project template* választása estén a következő ablakban megadhatjuk a sablon nevét (*Template name*) és rövid leírását (*Template description*), ami az új projekt létrehozásánál tájékoztatja a felhasználót a sablon tartalmáról. A *Finish* gombbal zárjuk le a sablon létrehozásának folyamatát.

Ablaksablon készítése

Ha valamelyik modult, például az elkészült ablakot és a változódeklarációkat, illetve eljárásokat tartalmazó osztálydefiníciót szeretnék sablonként menteni, akkor a varázsló *Choose Template Type* ablakában jelöljük be az *Item template* választógombot! A megjelenő ablakban válasszuk ki az elemet (például *Form1.vb*). A következő, *Select Item References* ablakban adhatnánk meg a felhasznált hivatkozásokat, de erre általában

[18] A *File*/*Save … As* parancs a projekthez tartozó aktuális fájlt nevezi át, és a munka az új fájllal folytatódik (például ezen a néven jegyzi a program indításakor megnyíló ablakot).

nincs szükségünk, így kattintsunk a *Next* gombra.[19] A folytatás innen kezdve megegyezik a projektsablon készítésével.

Megjegyezzük, hogy ha ékezetes karaktereket használunk a sablon nevében, akkor figyelmeztető üzenetet kapunk a korlátozásokról, de nem kell vele foglalkoznunk.[20]

A Visual Studio a sablon mentése után – hacsak nem tiltjuk le – megnyitja a

C:\Documents and Settings*felhasználó*\Dokumentumok\Visual Studio *évszám*\My Exported Templates

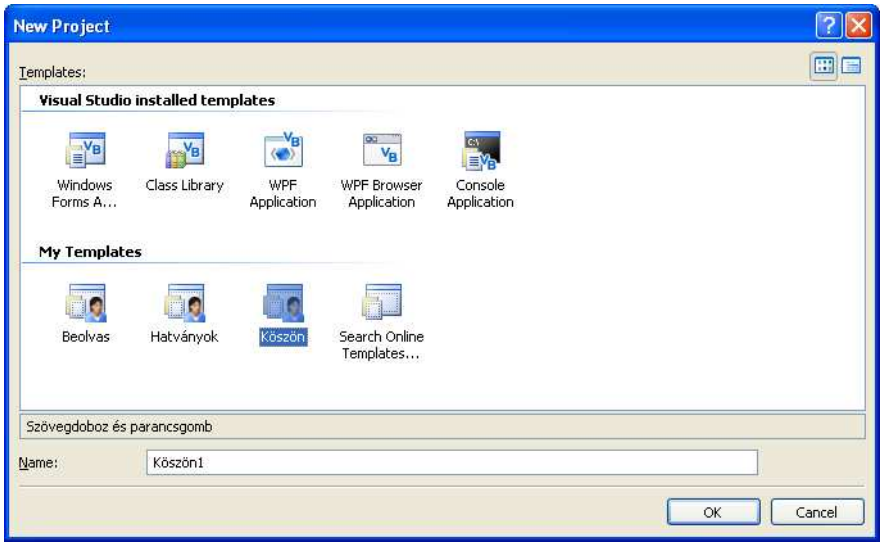
mappát. Itt látjuk az elmentett sablonokat, melyeket *.zip* fájlok tartalmazzák. A fájlokat egy fájlkezelővel (például az Intézővel) a szokásos módon adminisztrálhatjuk (másolás, törlés stb.).

Saját sablon alkalmazása

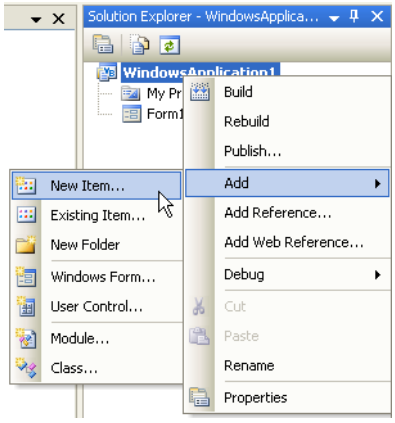
Az elkészült projektsablonokat egy új projekt létrehozásánál használhatjuk fel. A *File*/*New Project* parancsra megnyíló ablakban választhatjuk ki sablonjainkat (VS 2008 esetén a *My Templates* csoportból).

Ablaksablont természetesen csak már létező projektbe illeszthetünk be a megoldástallózóban a jobb egérkattintásra feltűnő helyi menü *Add*/*New Item* parancsával. VS 2008 esetén a megjelenő ablak *My Templates* csoportjában találjuk saját sablonjainkat.

Az ablaksablon beillesztésekor biztonsági figyelmeztetést kapunk. Megbízható forrásból származó sablon esetén kattintsunk a *Trust* gombra.



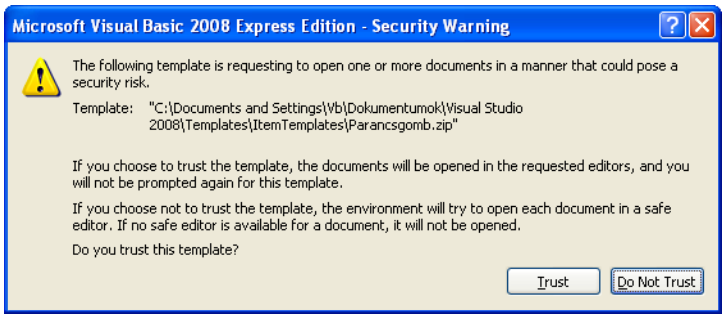
Saját sablon kiválasztása a Visual Studio 2008-ban



Ablaksablon beillesztése

[19] Ezt az opciót csak akkor kell használnunk, ha a projekt készítése során alkalmaztuk az *Add Reference* parancsot.

[20] Az üzenet arra utal, hogy az operációs rendszer angol nyelvű változatának a fájlnév miatt gondjai lehetnek a *.zip* fájl kezelésével.



Biztonsági figyelmeztetés az ablaksablon beillesztésekor

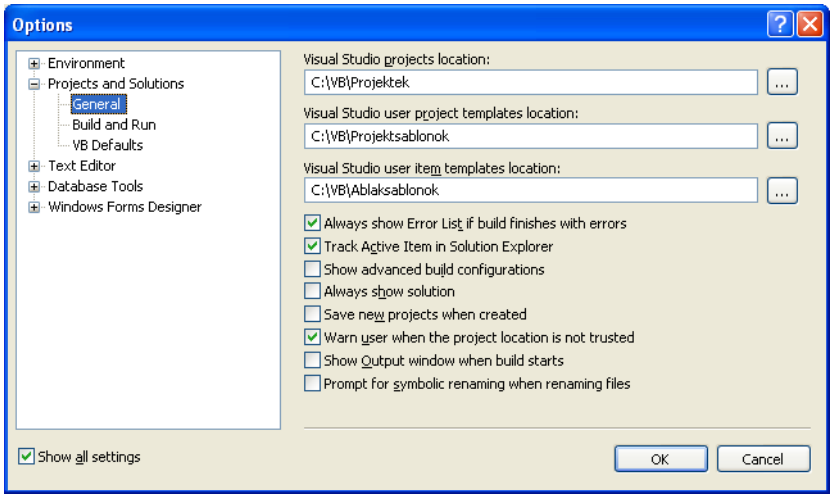
Ablaksablon beillesztésekor legyünk figyelemmel a kezdőablak beállítására (lásd fent). Célszerű először törölni az eredeti *Form1* ablakot, majd átnevezni *Form1*-re a sablonból beillesztett ablakot. Így nincs szükség az alkalmazás *Startup form* tulajdonságának a módosítására.

Megjegyezzük, hogy az ablaksablon tartalmazza az ablak tulajdonságait és az osztálydefiníciót a változókkal és eljárásokkal együtt (továbbá az itt nem tárgyalt beágyazott erőforrásokat is).

A sablonfájlok helyének módosítása

Ha gyakran van szükségünk a sablonok kezelésére, más sablonok átvételére, törlésére, akkor célszerű egy könnyebben elérhető mappát kijelölni a sablonok tárolására. Ezt a *Tools*/*Options* menü *Project and Solutions*/*General* paneljén tehetjük meg. A *User project templates location* a projektsablonok, a *User item templates location* pedig az ablaksablonok helyét adja meg. Az itt elhelyezett sablonfájlok (*.zip* fájlok) listája jelenik meg egy új projekt létrehozásánál a *My Templates* csoportban.

Ugyanitt módosíthatjuk a projektek mentésének alapértelmezett mappáját (*Projects location*).



Saját sablonok helyének a módosítása

Sajátos módon, az új helytől függetlenül a

C:\Dokumentumok*felhasználó*\Visual Studio *évszám*\My Exported Templates

mappába is bekerülnek a sablonok, és egy sablon mentésénél ez a mappa nyílik meg az Intézőben. A mappa megnyitását letilthatjuk a sablonkészítő varázsló *Select Template Options* ablakában a *Display an explorer window on the output files folder* jelölőnégyzetének a kikapcsolásával.

Hibakereső eszközök

Szintaktikus és fordítási hibák

A szintaktikus és fordítási hibák felismerését, kezelését részletesen ismertettük a tankönyv megfelelő leckéjében. A kódszerkesztő a következő színekkel húzza alá az általa felismert hibákat a forráskódban:

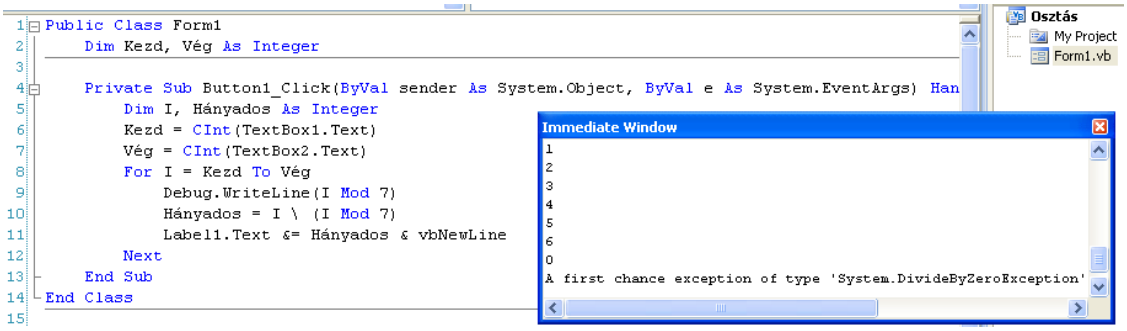
vörös: szintaktikus hiba
kék: fordítási hiba
zöld: figyelmeztető üzenet
lila: egyéb hiba

A hibás utasítás fölé helyezve az egeret megjelenik némi magyarázat és javaslat a hiba kijavítására.

A továbbiakban a hibakereső eszközök kezelésére mutatunk példát. A Visual Studio Express változata nem ismeri a nyomkövető üzemmódot, ezért erre nem térünk ki.

A *Debug* objektumosztály

A *Debug* osztály *WriteLine* osztálymetódával üzeneteket adhatunk a hibakereső üzemmódban történő futtatás során. Az üzenetek az *Immediate* ablakban jelennek meg (megnyitható a *Debug*/*Windows*/*Immediate* paranccsal). A további osztálymetódusokat lást a Visual Studio dokumentációjában.

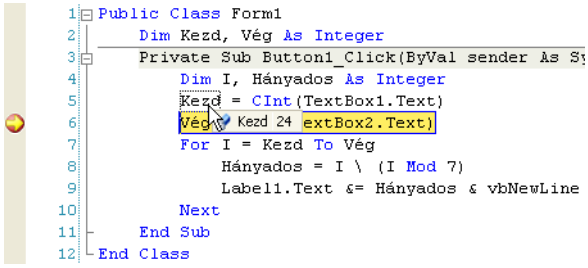


Az Immediate ablak a Debug.WriteLine(I Mod 7) *utasítás hatására*

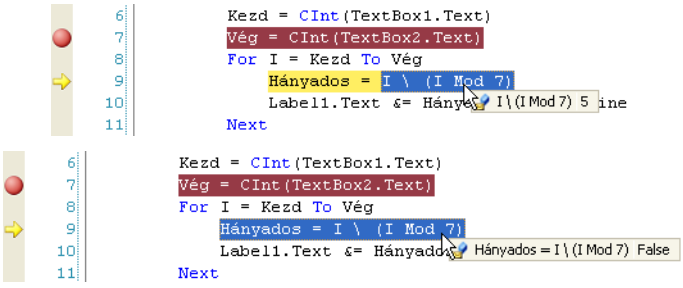
A *Debug* metódushívásai a program végleges, lefordított változatában hatástalanok maradnak.

Töréspontok elhelyezése a programban

Töréspont elhelyezéséhez kattintsunk a kódszerkesztő ablak bal szélén lévő sávra. Megjelenik a töréspontot jelző vörös kör. Eltávolításához kattintsunk a körre.



Változó értékének lekérdezése a program futásának felfüggesztésekor

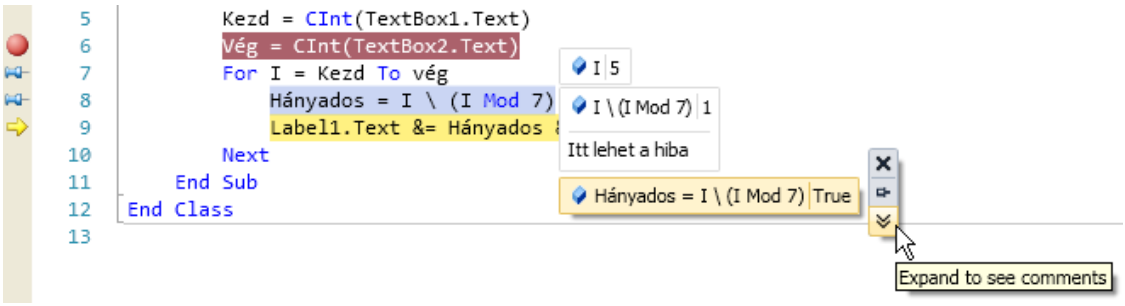


Kifejezés, illetve reláció értékének megjelenítése

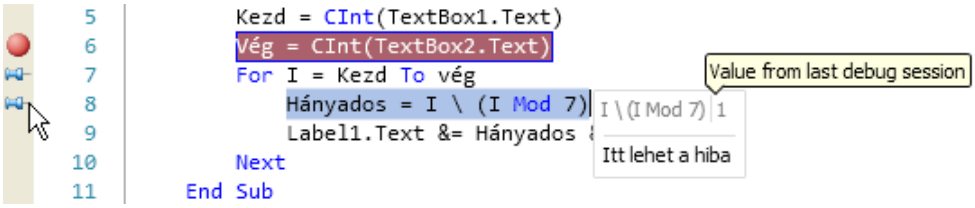
Töréspont esetén a program futása a megjelölt utasításnál leáll. A hibakereső üzemmódot a *Debugging* felirat jelzi az ablak címsorában.

A végrehajtásra kerülő utasítást egy sárga nyíl mutatja. Ha az egeret egy változónév fölé visszük, akkor megjelenik a változó értéke. Az értékre kattintva módosíthatjuk a változót. Ha kijelölünk egy kifejezést, illetve relációt (beleértve az értékadó utasítást, mint egyenlőségvizsgálatot), akkor a fejlesztőrendszer megmutatja az értékét.

A VS 2010-ben a kifejezések értékének kijelzéséhez jelöljük ki a kifejezést, majd a jobb egérkattintás helyi menüjében válasszuk a *Pin to Source* parancsot. Így folyamatossá tesszük a címke megjelenítését. Ezt a beállítást a változóknál is alkalmazhatjuk. Ha a címke fölé visszük az egérkurzort, akkor a megjelenő ikonok segítségével bezárhatjuk a címkét, feloldhatjuk a forráskódhoz való rögzítését, illetve megjegyzést írhatunk a megjelenő szövegdobozba. A rögzített címkék tartalma a hibakereső üzemmódból való kilépés után is elérhető marad. Ehhez vigyük az egeret a sor elején látható gombostű fölé.

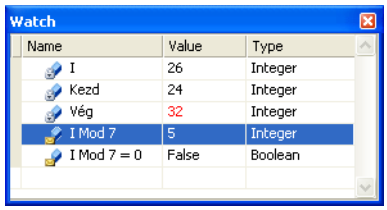


Címke rögzítése, megjegyzés készítése (VS 2010)



Címke tartalmának megjelenítése a hibakeresés után (VS 2010)

Több változó vagy kifejezés értékének megjelenítéséhez használhatjuk a *Watch* ablakot (*Debug*/*Windows*/*Watch* a hibakereső üzemmódban). Írjuk be a *Name* oszlopba a változó nevét, A *Value* oszlopra történő dupla kattintással (vagy a jobb egérgombra előtűnő helyi menü *Edit Value* parancsával) módosíthatjuk is az értéket. A módosítást *Enterrel* zárjuk.



A pirossal jelölt módosított érték a Watch ablakban

A *Watch* ablak *Name* oszlopába kifejezéseket, illetve az objektum nevével minősített tulajdonságokat is beírhatunk. A fejlesztőrendszer megjeleníti a kifejezés értékét. A változókat, kifejezéseket úgy is felvehetjük a *Watch* ablakba, hogy a kijelöljük a forráskód megfelelő részét, majd a jobb egérkattintásra előtűnő menüből kiválasztjuk az *Add Watch* parancsot. A parancs első alkalmazása egyben meg is nyit egy *Watch* ablakot. A kijelölt részt egérrel is áttehetjük a *Watch* ablakba.

A *Watch* ablakból egy sort kijelöléssel, majd a *Delete* billentyűvel törölhetünk.

Az *Autos* ablak a változók és a vezérlőelemek tulajdonságainak értékét jeleníti meg.

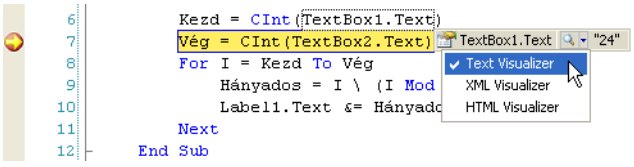
A program normál futása a *Start* (*Continue*) gombra kattintással folytatható.

A programban több töréspontot helyezhetünk el. Az összes töréspontot egyszerre a *Debug* menü *Delete All Breakpoints* parancsával törölhetjük. A fájl mentésénél a Visual Studio menti a nem törölt töréspontokat.

Megjegyezzük, hogy a forráskódba írt *Stop* utasítás a program futásának felfüggesztését és a hibakereső üzemmód bekapcsolását okozza.

Összetett adatszerkezetek megjelenítése

Összetett adatszerkezeteket a VS 2008 hibakereső üzemmódjában a vizualizáló komponensek segítségével jeleníthetünk meg. A komponenst egy kis nagyító jelzi, ha a változó vagy kifejezés fölé visszük az egeret. Sztringek esetén válasszuk a *Text Visualiser*, egyébként pedig a *DataSet*, *DataView* vagy *DataTable* parancsot a helyi menüből. Összetett szerkezetű változók felvétele esetén a nagyító ikon a *Watch* ablakban is megjelenik.



Az intelligens címke Text Visualizer parancsa

Az *Immediate* és a *Command* ablak használata

Az ablakokat a *Debug*/*Windows* menüből nyithatjuk meg.

Az *Immediate* ablakot akkor is felhasználhatjuk parancsaink végrehajtására (például egy változó értékének módosítására), ha a programot nem állítottuk meg töréspont elhelyezésével.

Egy változó értékének módosításához gépeljük be a következő utasítást az *Immediate* ablakba, majd nyomjuk le az Entert:

```
Változónév = újérték
```

Így a változók mellet objektumok (például vezérlőelemek) tulajdonságait szintén módosíthatjuk.

A *Command* ablak segítségével menüparancsokat szintén kiadhatunk a program futása közben. A `File.SaveAll` parancs kiadása például egyenértékű a *File* menü *Save All* menüparancsával.

A *Command* ablakból az *Immediate* ablakba az *immed* parancs kiadásával, visszafelé pedig a *>cmd* paranccsal válthatunk át.

Lépésenkénti végrehajtás

Ha a program futása egy töréspontnál leáll, akkor kezdeményezhetjük a lépésenkénti végrehajtást. A *Debug* menü *Step Into* parancsa ( , *F8*) rátér a következő utasításra. A *Step Over* ( , *Shift+F8*) megállás nélkül végrehajtja a ciklusok és alprogramok utasításait, majd a következő utasításnál függeszti fel a program futtatását. Így elkerülhetjük a hosszadalmas ismétléseket vagy a sok utasításból álló programrészeket.

Ha már beléptünk egy ciklusba vagy alprogramba, akkor a *Step Out* paranccsal ( , *Ctrl+Shift+F8*) kerülhetjük el a blokkon belüli lépésenkénti végrehajtást.

A végrehajtás felfüggesztésénél minden esetben rendelkezésünkre állnak a töréspontok elhelyezésénél ismertetett lehetőségek.

A hibakereső üzemmód parancsainak elérését a *Debug* eszköztár könnyíti meg. Az eszköztárat a *View*/*Toolbars*/*Debug* paranccsal hívhatjuk elő.

A Visual Studio beállításai

Beállítások

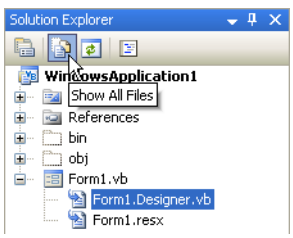
A fejlesztőrendszert a használat előtt célszerű hozzáigazítani az igényeinkhez. A beállításokat a *Tools*/*Options* menüparancsnál találjuk. A módosításokhoz az *Options* párbeszédablakba kapcsoljuk be a *Show all settings* jelölőnégyzetet!

A következő beállításokat javasoljuk.

Environment/Help/Online (csak a VS 2008 régebbi változataiban)	
Try local only, not online	nem használja az online súgót (az érettségi miatt), vagy:
Try local first, then online	először a lokális súgóban keres
Environment/Startup	
At startup: Show empty environment	üres ablakkal indul a Visual Studio megnyitásánál
Environment/Fonts and Colors	
Size	a forráskód betűmérete
Projects and Solutions/General	
Projects location	megadhatjuk a projektek mentésének helyét
Projects and Solutions/VB Defaults	
Default project settings	megadhatjuk az Option utasítások alapértelmezett értékeit (lásd a Programozási összefoglalóban)
Text Editor/All languages/General	
Display: Line numbers	sorok számozásának megjelenítése a forráskódban

Munkaablakok

A szükséges munkaablakok a *View* menü segítésével, vagy az ott látható billentyűparancsokkal jeleníthetők meg. Javasoljuk a *Toolbox*, a *Solution Explorer* és a *Properties* munkaablak megnyitását.



Rejtett fájlok és mappák megjelenítése a Solution Explorerben

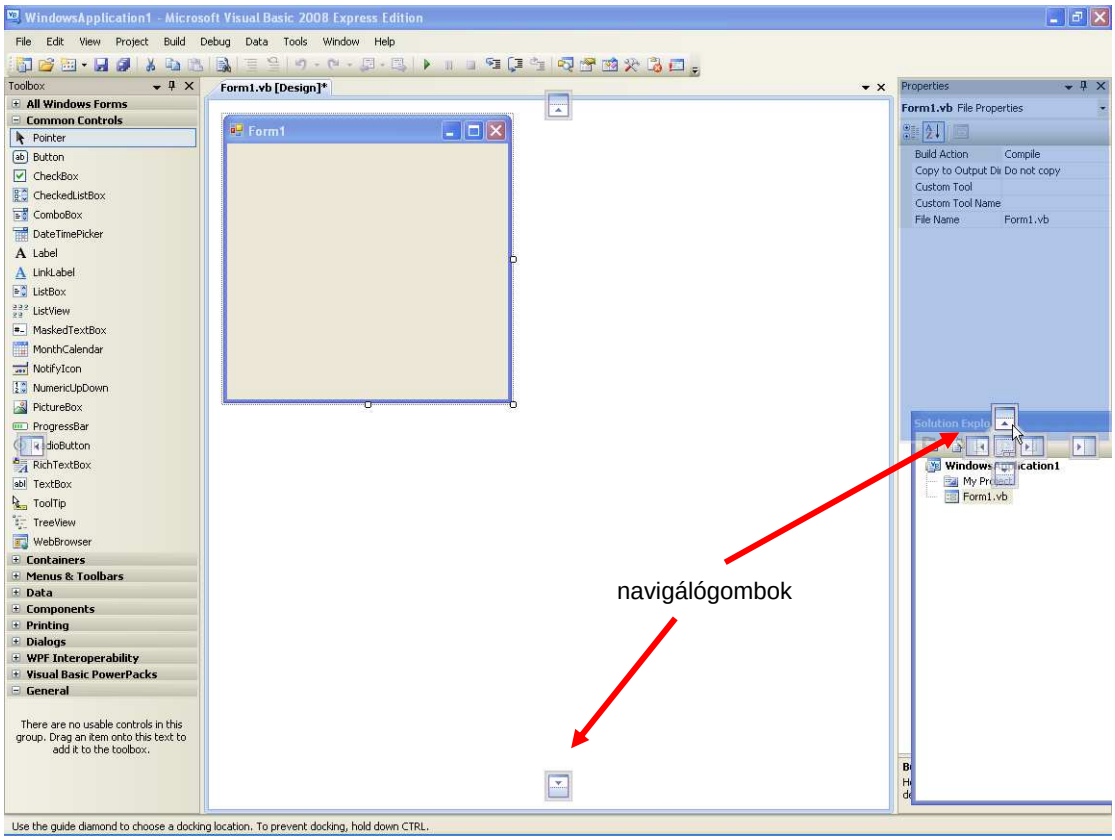
Megjegyezzük, hogy a *Solution Explorer* munkaablak *Show All Files* ikonjával a rejtett fájlokat, illetve a projekt teljes mappaszerkezetét megjeleníthetjük. Érdemes megtekinteni egy Windows-alkalmazás *Form1.Designer.vb* fájlját. A *Partial* kulcsszó az osztálydefiníció előtt arra utal, hogy itt az osztály forráskódjának csak egy része szerepel (a többi a *Form1.vb* fájlban). Az *Inherits* utasítás pedig a *Form1* osztály szülőosztályát jelöli ki (öröklődés). A forráskódban megtaláljuk az űrlapra helyezett vezérlőelemek létrehozását és tulajdonságainak beállítását végző utasításokat.

A lebegő ablakok elrendezése

A munkaablakok *Auto Hide* tulajdonságának alkalmazását a *Programozás Visual Basicben* fejezet elején már bemutattuk.

A munkaablakok alapértelmezés szerint a programablak széléhez vannak rögzítve. A rögzítés feloldásához kattintsunk jobb egérgombbal a munkaablak címsorára, majd válasszuk a *Float(ing)* menüpontot. Így a munkaablakot szabadon elmozdíthatjuk a képernyőn.

Az ismételt rögzítéshez a *Dock(able)* menüponttal térhetünk vissza. Majd fogjuk meg a munkaablakot, és mozgassuk a képernyőn. Az így megjelenő navigálógombok segítségével a kiválasztott helyre illeszthetjük be.



A Solution Explorer visszatétele a szokásos helyére

A súgó használata – Visual Basic 2008

A súgó telepítése

A Visual Studio 2008 súgóját elérhetjük a Microsoft webelyén (MSDN Online), illetve telepíthetjük a számítógépre is[21]. A telepített változatot az *MSDN Express Library* alkotja. A letöltést a következő címről végezhetjük:

http://www.microsoft.com/express/Downloads/

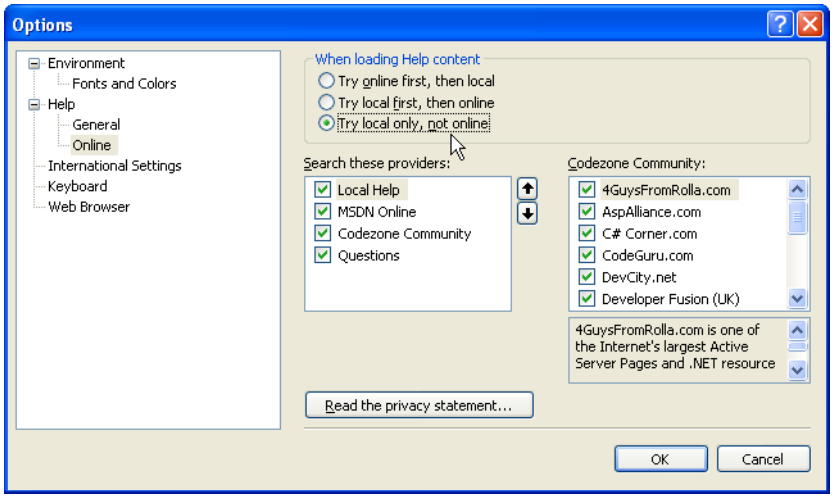
Kattintsunk a *Visual Studio 2008 Express* feliratra, majd a megjelenő listában az *MSDN Express Library* hivatkozásra. Válasszuk ki a nyelvet, és kattintsunk a *Free Download* feliratra. A telepítő fájl mérete 360 megabájt.

A súgóban többek között olvashatunk egy angol nyelvű bevezetést a Visual Basic használatába (Visual Basic Guided Tour), egy programozói kézikönyvet (Visual Basic Programming Guide), illetve szerepel benne a nyelv teljes leírása (Reference). Az online súgóban frissítéseket, aktuális információkat és további kiegészítéseket találunk a Visual Basichez.

A súgót a szokásos módon, az *F1* funkcióbillentyűvel vagy a *Help*/*Contents* parancs segítségével érhetjük el.

Az online súgó letiltása

A programozás során célszerű letiltani az online súgót, mert a gyakorlati érettségin nem használhatjuk az Internetet! A súgó első megnyitásánál a fejlesztői rendszer rákérdez az online *Help* engedélyezésére. Utólag a súgóablak *Tools*/*Options* parancsával módosíthatjuk a beállítást. Válasszuk a *Help*/*Online* listaelemet, majd jelöljük be a *Try local only, not online* választógombot!



Az online súgó letiltása

Ne felejtsük el a fejlesztői környezet Internet-elérését is letiltani. Ezt otthon a számítógépre telepített tűzfal segítségével tehetjük meg. Megfelelő beállítás esetén a tűzfal rákérdez az Internet-elérés engedélyezésére. Válasszuk a tiltás funkciót!

A Visual Basic referencia

A súgó egyik leggyakrabban használt része a nyelv leírása, melynek megértése csak elemi angoltudást igényel. A referenciát a súgó tartalomjegyzékében (*Contents*) a *Microsoft MSDN Express Library 2008*/*Visual Basic Express*/*Reference (Visual Basic)*/*Visual Basic Reference* címszó alatt találjuk. Legfontosabb elemei:

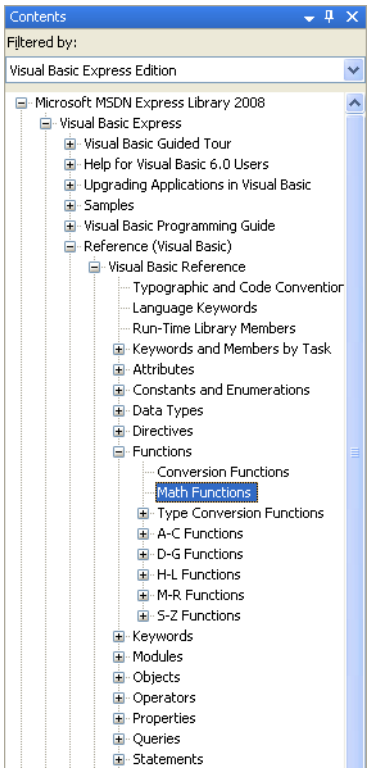
Constants and Enumerations	előre definiált konstansok és felsorolások
Data Types	adattípusok
Functions	függvények (A matematikai függvények ismertetését ne az ábécérend szerint, hanem a *Math Functions* csoportban keressük!)
Operators	operátorok, műveleti jelek
Statements	utasítások

[21] A 2008-as verzió eredeti változata még beépítve tartalmazta az MSDN Express Library-t, de az újabb változatból már kimaradt.

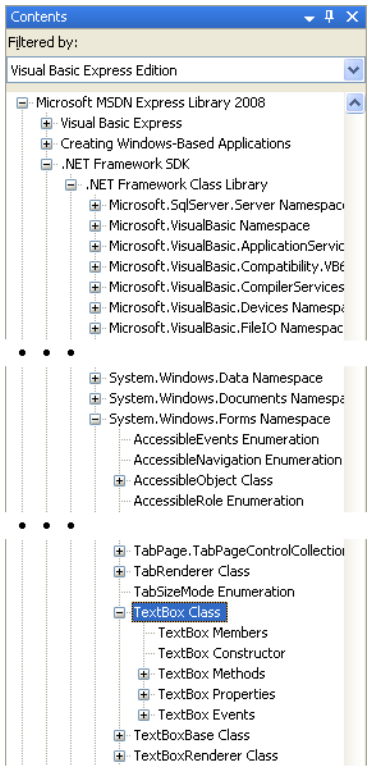
A leírások végén egy vagy több forráskódpéldát is láthatunk, amely bemutatja a megfelelő nyelvi elem használatát.

A grafikus felhasználói felület objektumai a súgóban

A grafikus felület objektumait (illetve a nekik megfelelő osztályokat) a .NET Framework definiálja, így a súgó tartalomjegyzékének *.NET Framework SDK*/*.NET Framework Class Library* címszavát kell kibontatnunk. Az elemeket a névterek szerint rendezték. Egy-egy osztály kiválasztásakor a rövid leírás mellett megtaláljuk a tulajdonságok és metódusok (members) ismertetését. A szövegdoboz (textbox) osztály leírásának helyét például az alábbi ábrán mutatjuk be.



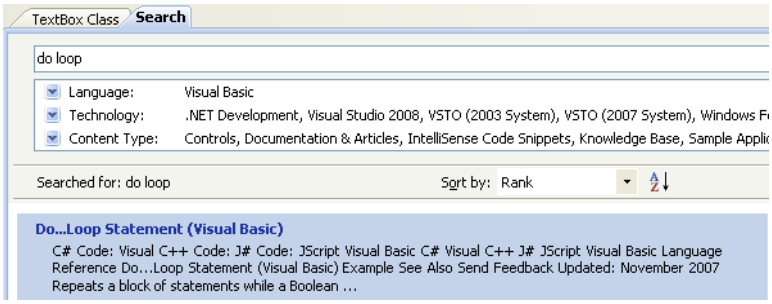
A súgó tartalomjegyzéke



A szövegdoboz a súgó tartalomjegyzékében

Keresés a súgóban

Szükség esetén meg is kereshetjük a kérdéses fogalmat, kulcsszót a súgóban. Ehhez válasszuk a *Help*/*Search* parancsot vagy a súgóablakban a *Search* fület! Írjuk be a keresett szót a szövegdobozba, majd nyomjuk le az Entert (vagy kattintsunk a *Search* gombra). A megjelenő listában a súgó kék háttérrel jelöli a kereséshez legjobban illeszkedő találatot. A sötétkék címszóra kattintva megjelenik a keresett elem leírása.



A Do…Loop utasítás keresése és a találati lista első eleme

A Visual Basic 2010 súgója

A súgó telepítése

A 2010-es verziónál az úgynevezett *Help Library Manager* kezeli a súgót. Válasszuk a *Help* menü *Manage Help Settings* parancsát. A megjelenő ablak legfontosabb elemei:

Choose online or local help	online vagy helyi menü használatának kiválasztása
Install content from online	helyi menü telepítése

A helyi menü telepítésekor először rákérdez az igényelt komponensekre. Válasszuk ki a *.Net Framework* (*.NET Development* csoport) és a *Visual Basic* (*Visual Studio 2010* csoport) súgójának a hozzáadását. Ezek után egy elég hosszadalmas folyamat során, de automatikusan elvégzi a telepítést.

A Visual Basic referencia a súgóban

A súgó használata

A súgó a *Help*/*View Help* parancsra egy külön böngészőablakban jelenik meg (a címsávban ellenőrizhetjük, hogy a helyi súgót látjuk-e).

A Visual Basic referenciát a következő választással érjük el:

Visual Studio/
Visual Studio Languages/
Visual Basic and Visual C#/
Visual Basic/
Reference (*Visual Basic*)/
Visual Basic Language Reference

A .NET Framework osztálykönyvtárainak helye a súgóban:

.NET Framework4/*.NET Framework Class Library*

A megnyíló weblapon ábécérendben találjuk az egyes névtereket, majd a kiválasztott névteret ismertető weblapon az osztályok listáját.

A súgó használata a továbbiakban lényegében megegyezik a 2008-as változatéval. Megjegyezzük, hogy az egyes weblapokat felvehetjük a böngésző kedvencei (könyvjelzői) közé.

Tartalom