

# Bevezetés

Ez a könyv nagyjából 2001 nyara óta létezik, ha nem is ebben a formában: a C# és a .NET platform első kiadása ugyanis akkor jelent meg, a .NET 1.0 Beta 2 verziójával karöltve. Nagy örömmel és hálával tölt el, hogy munkám kedvező fogadtatásra lelt a sajtóban, és ami még fontosabb, az olvasók körében is. Az évek során jelölték a Jolt Award döntőjére (vesztettem... kellemetlen) és a 2003-as Referenceware Excellence Awardon a programozási könyvek kategóriájában (nyertem?... de jó!)

Azóta azon dolgoztam, hogy a könyvet frissítsem és aktualizáljam a .NET platform újabb verzióinak megfelelően. Eközben limitált darabszámmal megjelentettem egy speciális kiadványt, amely bemutatta a .NET 3.0 verzióját (Windows Presentation Foundation, Windows Communication Foundation, és a Windows Workflow Foundation), valamint áttekintést nyújtott olyan különböző technológiákról, amelyek még megjelenésre vártak, és amelyeket most LINQ néven ismerünk.

Jelenleg a könyv negyedik kiadását tartja a kezében az Olvasó. Ez a kiadás egyrészt tömören megismétli a korábbi ismereteket, azaz a .NET 3.5-ös verziójában található legfontosabb változások magyarázatát, másrészt számos teljesen új fejezetet is tartalmaz, de ezen túlmenően több eddigi rész is jelentősen kibővült.

Ahogy a korábbiakban megszokhattuk, ez a kiadás is egyszerű és érthető nyelven ismerteti a C# nyelvet és a .NET-alapoztálykönyvtárakat. Soha nem értettem azokat a szakírókat, akik úgy adják elő mondandójukat, hogy az jobban hasonlít egy GRE-szöszedetről írt tanulmányra, mint egy szakkönyvre. Munkám továbbra is arra fekteti a hangsúlyt, hogy a benne található információk segítségével jól használható szoftverrendszereket lehessen készíteni, így nem töltök túl sok időt az ezoterikus részletek tanulmányozásával, hiszen ezekkel csak nagyon kevés olvasóm fog valaha is foglalkozni.

## Mi egy csapat vagyunk, az Olvasó és én

A szakírók az emberek igényes csoportjának írnak (nekem már csak tudnom kell – egy vagyok közülük). Tisztában vagyunk azzal, hogy egy szoftver elkészítése – bármelyik platformmal történjen is (.NET, J2EE, COM stb.) – rendkívül aprólékos munka, ugyanakkor nagyon specifikus a részleg, a cég, az ügyfél vagy a tárgykör szerint. Hiszen lehet, hogy épp az elektronikus kiadói iparágban dolgozunk, vagy éppenséggel rendszert fejlesztünk az államnak, esetleg a helyi önkormányzatnak, a NASA-nak vagy egy katonai részlegnek. Én a magam részéről gyermekeket oktató szoftvert fejlesztettem, számos n-rétegű rendszert, valamint projekteket a gyógyászati és a pénzügyi ágazatokban. Majdnem nulla az esélye, hogy annak a kódnak, amelyet egyikünk ír a munkájához, köze lesz ahhoz a kódhoz, amelyet a másikunk ír a saját munkájához (kivéve, ha véletlen épp együtt dolgoztunk korábban!).

Ezért szándékosan kerülöm az olyan mintapéldákat, amelyeket konkrét iparághoz vagy programozási feladathoz köthetnénk: a C#, az objektumorientált programozás és a .NET 3.5 alapsztály könyvtárait ipari alkalmazásoktól független példákon keresztül mutatom be. Ahelyett, hogy minden példám adatok tömkelegét szolgáltatná, bérlistát számolna, vagy ehhez hasonlókkal szolgálna, az én példámhoz mindenkinek van valami köze: autóról szólnak (ráadásként néhány geometriai struktúrával és alkalmazottakkal). És ez az a pont, ahol összetalálkozunk.

*Az én feladatom*, hogy legjobb tudásom szerint elmagyarázzam a C# programozási nyelvet és a .NET platform alapvető jellegzetességeit. Ugyanígy mindent megteszek azért, hogy ellássam a kedves Olvasót különféle eszközökkel és stratégiákkal, amelyek lehetővé teszik, hogy a könyv alapján folytatni lehessen a téma tanulmányozását.

*Az olvasó feladata* pedig, hogy a megszerezett tudást képes legyen alkalmazni saját programozási feladatában. Persze egyértelmű, hogy a projektek nem állatnevekkel ellátott autók körül forognak, de hát pont erről szólna az alkalmazott ismeret. Biztos lehet benne az Olvasó, hogy ha egyszer megértette a könyvben bemutatott elveket, ismereteinek birtokában olyan .NET-alkalmazásokat írhat, amelyek használhatók saját programozási környezetükben.

# A könyv áttekintése

A *Pro C# 2008 and the .NET 3.5 Platform* (A C# és a Microsoft .NET 3.5.) negyedik kiadása logikailag nyolc különálló részből áll, s ezek mindegyike további fejezeteket tartalmaz. Jó néhány témakör, például az alapvető C#-szerkezetek, az objektumorientált programozás és a platformfüggetlen .NET fejlesztése kibővült a korábbiakhoz képest, saját fejezetet kapott. Az új kiadás a .NET 3.0–3.5 programozás tulajdonságait (LINQ , WCF , WPF , WF stb.) összefoglaló fejezeteket is tartalmaz.

---

**Megjegyzés** A magyar változat kiadásának előkészítése során a SZAK Kiadó úgy döntött, hogy az eredeti könyv tartalmát két kötetben jeleníti meg. Ezt a viszonylag nagy terjedelem indokolja. Az alábbiakban mind a két kötet tartalmát ismertetjük, így biztosítva, hogy az is képet kapjon a mű egészéről, akinek csak az egyik kötet kerül a kezébe. (Ezt a Bevezetést változatlan formában mind a két kötetben szerepeltetjük.)

---

## Első kötet

### 1. rész: Bevezetés a C#-ba és a .NET platformba

Az első rész bemutatja a .NET platform alapvető természetét és számos olyan fejlesztőeszközt (köztük több nyílt forráskodút), amelyeket .NET-alkalmazások fejlesztéséhez használhatunk. Mindeközben néhány alap C# programozási nyelvi részlettel és a .NET-típusrendszerrel is megismerkedünk.

#### 1. fejezet: A .NET filozófiája

Az első fejezet voltaképpen a könyv további részeinek a gerincét képezi. A hagyományos Windows-fejlesztések világának vizsgálatával kezdjük, és feltárjuk a korábbi hátrányokat is. A fejezet elsődleges célja azonban az, hogy megismerjünk jó néhány .NET-központú építőkövet, például az egységes nyelvi futtatót (Common Language Runtime – CLR), a közös típusrendszert (Common Type System – CTS), az egységes nyelvi specifikációt (Common Language Specification – CLS) és az alaposztálykönyvtárakat. Egyszerű betekintést nyerhetünk a C# programozási nyelvbe és a .NET-szerelvényformátumába, valamint áttekintést kapunk a .NET platform platformfüggetlen természetéről (a B függelék részletesen tárgyalja ezt a témakört).

## 2. fejezet: C#-alkalmazások készítése

A fejezet célja, hogy különféle eszközök és technikák ismertetésével bevezessen minket a C#-forráskódfájl fordításának folyamatába. Először a parancssoros fordító (csc.exe) és a C#-eredményfájlok használatát ismerhetjük meg. A fejezet további részében pedig számos forráskódszerkesztőt és integrált fejlesztési környezetet (integrated development environments – IDEs) tanulmányozhatunk, többek között a következőket: TextPad, Notepad++, SharpDevelop, Visual C# 2008 Express és a Visual Studio 2008. Ezenkívül számos olyan további programozási eszközt ismertetünk, amelyet bármely .NET-fejlesztőnek azonnal elő kell tudnia húzni a farzsebéből.

## 2. rész: A C# alapvető építőelemei

A 2. részben tárgyalt témakörök kiemelten fontosak, hiszen az itt megjelenő tudnivalókat folyamatosan használjuk függetlenül attól, hogy milyen jellegű .NET-szoftvert szándékozunk fejleszteni (webes alkalmazás, asztali grafikus alkalmazás, forráskódkönyvtárak, Windows-szolgáltatások stb.). Itt derül ki, hogy hogyan működnek a C# nyelv alapvető építőkövei, beleértve az objektumorientált programozás (OOP) részleteit is. Ez a rész ismerteti továbbá, hogy hogyan kell kezelni a futásidejű kivételeket, és bevezet minket a .NET személggyűjtő szolgáltatásának részleteibe is.

## 3. fejezet: A C# alapvető építőelemei, I. rész

Ebben a fejezetben kezdjük el részletesen tanulmányozni a C# programozási nyelvet. Megismerkedünk a `main()` metódus szerepével és számos részlettel a .NET platform belső adattípusaival kapcsolatban, beleértve a szövegesadatkezelést a `System.String` és a `System.Text.StringBuilder` névterek alkalmazásával. Megvizsgáljuk továbbá az iterációs és a döntési szerkezeteket, a szűkítő és a bővítő műveleteket, valamint az `unchecked` kulcsszó használatát.

## 4. fejezet: A C# alapvető építőelemei, II. rész

A 4. fejezet befejezi az alapvető C#-szemléletek meghatározását, a típusülterhelt metódusok létrehozásától az `out`, `ref` és `params` kulcsszavakon keresztüli paraméterek definiálásáig. Vizsgáljuk, hogyan kell adathalmazt létrehozni és kezelni, hogyan kell olyan adattípust készíteni, amelynek `null` is lehet az értéke (a `?` és a `??` operátorokkal), és megmutatjuk az *értéktípus* és a *referenciátípus* közötti különbséget is.

## 5. fejezet: Egységbe zárt osztálytípusok definiálása

Ebben a fejezetben az objektumorientált programozással (OOP) ismerkedhetünk meg a C# programozási nyelv használatán keresztül. Először meghatározzuk az OOP pilléreit (beágyazás, származtatás és polimorfizmus), majd a fejezet további részében azt nézzük meg, hogyan kell konstruktorok, tulajdonságok, statikus tagváltozók, konstansok és csak olvasható fájlok használatával robusztusosztály-típusokat létrehozni. Végezetül ismertetjük a részleges típus definícióját és a C# XML-kódjának dokumentációs szintaktikáját.

## 6. fejezet: A származtatás és a polimorfizmus

Ebben a részben az OOP maradék pilléreit (származtatás és polimorfizmus) vizsgáljuk meg; ezek a segítségével összetartozó osztálytípusok családját hozhatjuk létre. Megnézzük a virtuális metódusok és absztrakt metódusok (és az absztrakt alaposztályok) szerepét, valamint a *polimorf interfész* természetét. Végül, de nem utolsósorban, ez a fejezet ismerteti a .NET platform legfontosabb alaposztályának, a `System.Object` osztálynak a szerepét is.

## 7. fejezet: Strukturált hibakezelés

Ez a fejezet elsősorban azt mutatja be, mit kell tenni a strukturális kivételkezelés segítségével a futási időben keletkezett anomáliákkal. Nemcsak az ilyen jellegű problémák megoldására szolgáló C#-kulcsszavakat ismerjük itt meg (`try`, `catch`, `throw` és `finally`), hanem az alkalmazásszintű és a rendszerszintű kivételek közötti különbséget is. Megismerhetünk továbbá többféle, a Visual Studio 2008-ban található eszközt, amelyek segítségével hibakeresést végezhetünk a figyelmen kívül hagyott kivételek között.

## 8. fejezet: Az objektumok életciklusa

Ennek a résznek az utolsó fejezete azt tárgyalja, hogy hogyan kezeli a CLR a memóriát a .NET szemétyűjtőjének a segítségével. Megérthetjük az alkalmazásgyökereket, az objektumnemzedékeket és a `System.GCType` szerepét. Az alapok megismerése után, a fejezet további része az *eldobható objektumokkal* (az `IDisposable` interfészen keresztül) és a véglegesítő folyamattal (a `System.Object.Finalize()` metóduson keresztül) foglalkozik.

### 3. rész: Haladó programozási szerkezetek a C#-ban

A könyv 3. része ismerteti a C# nyelv haladóbb jellegű (de ugyancsak nagyon fontos) elveit. Az interfészek és metódusreferenciák vizsgálatával befejezzük a .NET típusrendszerével való ismerkedést. A továbbiakban betekintést nyerhetünk a generikus típusok és a számos új C# 2008 nyelvi elem szerepébe, valamint a LINQ-ba is.

#### 9. fejezet: Interfészek használata

Ennek a fejezetnek az anyagát az interfészalapú programozást is magába foglaló objektumalapú fejlesztés alkotja. Megismerhetjük belőle, hogy hogyan kell olyan típusokat definiálni, amelyek többszörös viselkedést támogatnak, hogyan lehet ezeket a viselkedéseket futásidőben feltérképezni, és hogyan lehet szelektíven elrejteni bizonyos viselkedéseket az *explicit interfészmegvalósítás* segítségével. Továbbá a számos előre definiált .NET-interfésztípus megismeréséhez el kell sajátítanunk azoknak az egyedi interfészeknek a használatát, amelyek ad hoc jellegű eseményarchitektúra készítéséhez szükségesek.

#### 10. fejezet: Gyűjtemények és generikus típusok

Ez a fejezet a `system.collections` névtérgyűjtemény típusainak tanulmányozásával kezdődik; ezek már a .NET platform kezdeti kiadásának is részei voltak. Noha a .NET 2.0 megjelenése óta a C# programozási nyelv lehetővé teszi *generikus* típusok használatát, ahogy a későbbiekben kiderül, a generikus programozás lényegesen megnöveli az alkalmazás teljesítményét és a típusbiztonságot. Nemcsak azt vizsgáljuk meg, hogy milyen különböző generikus típusok léteznek a `system.collections.Generic` névtérben, de azt is, hogyan készíthetjük el saját generikus metódusainkat és típusainkat (megszorításokkal vagy anélkül).

#### 11. fejezet Metódusreferenciák, események és lambdák

A 11. fejezetnek az a célja, hogy érthetővé tegye a *metódusreferencia*-típusokat. Leegyszerűsítve: a .NET-metódusreferencia olyan objektum, amely az alkalmazás más metódusaira mutat. Ez a minta olyan rendszerek készítését teszi lehetővé, amelyek több objektum összekapcsolódását engedik meg kétirányú együttműködés segítségével. A .NET-metódusreferenciák vizsgálata után megismerkedünk a .NET event kulcsszavával, amelyet arra használunk, hogy egyszerűsítsük a nyers metódusreferencia programozásának kezelését. Végezetül megvizsgáljuk a C# 2008 lambda-operátorát (`=>`), és feltárjuk a kapcsolatot a metódusreferenciák, a névtelen metódusok és a lambda-kifejezések között.

## 12. fejezet: Indexerek, operátorok és mutatók

Ez a fejezet a C# programozási ismereteinket mélyíti el számos haladó programozási technika segítségével. Megismerhetjük, hogyan kell operátorokat túlterhelni, és hogyan kell a típusaink számára egyedi konverziós rutinokat írni (akár implicit, akár explicit formában). Megtanulhatjuk azt is, hogy hogyan kell létrehozni és használni a *típusindexelőket*, és hogyan kell kezelni a C-stílusú mutatókat a „nem biztonságos” kódkörnyezetben.

## 13. fejezet: C# 2008 nyelvi újdonságai

A .NET 3.5 kiadásában a C# nyelv számos új programozási eszközzel bővült, amelyek közül jó néhány arra szolgál, hogy lehetővé tegye a LINQ API használatát (erről a 14. fejezetben többet is megtudhatunk). Megismerjük a lokális változók implicit típusalattörténi ellátásának a szerepét, a részleges metódusok, az automatikus tulajdonságok, a bővítő metódusok, a névtelen típusok és az objektuminicializáló szintaxis használatát.

## 14. fejezet: Bevezetés a nyelvbe ágyazott lekérdezésekbe (LINQ)

Ez a fejezet ismerteti a LINQ-et (Language Integrated Query – nyelvi integrált lekérdezés), amelyet nyugodtan nevezhetünk a .NET 3.5 legérdekesebb részének. Ahogy a fejezet későbbi részéből kiderül, a LINQ segítségével lehetőségünk nyílik erősen típusos *lekérdezéseket* írni, amelyeket számos LINQ-célpontra alkalmazhatunk, hogy a szó legtágabb értelmében „adatot” kezelhessünk. Megtanuljuk a LINQ objektumokra történő alkalmazását, ennek segítségével a LINQ-kifejezéseket adattárolókra (tömbök, kollekciók, egyedi típusok) fogjuk tudni alkalmazni. Ez a tudás nélkülözhetetlen lesz akkor, amikor azt vizsgáljuk meg, hogyan kell LINQ-kifejezéseket relációs adatbázisokon (a LINQ az ADO-n keresztül) és XML-dokumentumokon (a LINQ az XML-en keresztül) használni.

## 4. rész: Programozás .NET-szerelvényekkel

A negyedik rész a .NET-szerelvényformátum részleteit taglalja. Nemcsak azt tanulhatjuk meg, hogyan kell .NET-kódkönyvtárakat telepíteni és konfigurálni, de azt is, hogy mi a .NET bináris kép belső összeállítása. Ebben a részben lesz szó a .NET-attribútumok szerepéről és a többszálú alkalmazások létrehozásának módjáról is. A későbbi fejezetek néhány haladó témát érintenek, például az objektumkörnyezetet, a köztes nyelvi kódot és a dinamikus szerelvényeket.

## 15. fejezet: A .NET-szerelvények

Első megközelítésben a *szerelvény* nem más, mint egy felügyelt \*.dll vagy \*.exe bináris fájl leírására szolgáló kifejezés. Ám a .NET-szerelvények története sokkal több ennél. Ebben a fejezetben megtudjuk, mi az egyfájlos és a többfájlos szerelvények közötti különbség, és, hogy ezeket hogyan kell készíteni és telepíteni. Megvizsgáljuk, hogyan kell privát és megosztott szerelvényeket konfigurálni XML-alapú \*.config fájlokkal és kibocsátó házirenddel rendelkező szerelvényekkel. Mindeközben megtudhatjuk, hogy milyen a GAC (Global Assembly Cache – globális szerelvénytár) belső struktúrája, és mi a .NET keretrendszer konfigurációs segédprogramjainak a szerepe.

## 16. fejezet: Típusreflexió, késői kötés és attribútumalapú programozás

A 16. fejezetben folytatjuk az ismerkedést a .NET-szerelvényekkel. A `System.Reflection` névtér használatával kapcsolatban azt vizsgáljuk, a futási időben hogyan valósul meg a típusmeghatározás folyamata. Az ilyen jellegű típusok használatával lehetőségünk van olyan alkalmazások létrehozására, amelyek futás közben képesek a szerelvények metaadatait olvasni. Kiderül, hogyan lehet a *késői kötés* segítségével futási időben dinamikusan betölteni és létrehozni típusokat. A fejezet utolsó témaköre – mind a szabványos, mind az egyedi – .NET-attribútumok szerepének ismertetése. Az egyes témakörök hasznosságának szemléltetésére a fejezet konklúziójaként bemutatunk egy kiterjeszthető Windows Forms-alkalmazást.

## 17. fejezet: Folyamatok, alkalmazástartományok és objektumkörnyezetek

A szerelvényekről szóló tudnivalók után ez a fejezet a betöltött .NET végrehajtható fájlok felépítését ismerteti. Az elsődleges cél a folyamatok, az alkalmazástartományok és a kontextusbeli határok közötti kapcsolat illusztrálása. Ezek alapozzák meg a következő fejezet témáját: a többszálú alkalmazások készítésének módját.

## 18. fejezet: Többszálú alkalmazások készítése

Ez a fejezet azt mutatja be, hogy hogyan kell többszálú alkalmazásokat készíteni, és számos olyan technikát ismertet, amelyeket a szálbiztos kód készítéséhez alkalmazhatunk. A fejezet elején átismételjük a .NET metódusreferencia-típusait, hogy megértsük, valójában hogyan támogatják a metódusreferenciák

az aszinkronfüggvény-hívásokat. Majd megvizsgáljuk a `System.Thread` névtérben található típusokat. Ezek között számos olyan van (`Thread`, `ThreadStart`), amelynek segítségével könnyen lehet újabb végrehajtási szálakat készíteni. Végül megismerkedünk a `BackgroundWorker` típussal, amely nagyban megkönnyíti a felhasználói felületről történő szálkészítést.

## 19. fejezet: A köztes nyelv (CIL) és a dinamikus szerelvények

A 4. rész utolsó fejezetének szerepe kettős. Először a korábbiaknál részletesebben megvizsgáljuk CIL szintaktikáját és szemantikáját, majd a `System.Reflection.Emit` névtér szerepét mutatjuk be. Ezeknek a típusoknak a használatával lehetőségünk nyílik olyan szoftver készítésére, amely képes .NET-szerelvényeket futásidőben generálni a memóriába. Az olyan szerelvényeket, amelyeket a memóriában definiálunk és futtatunk, *dinamikus szerelvényeknek* nevezzük.

## Második kötet

### 5. rész: Bevezetés a .NET alaposztálykönyvtáraiba

Ha elérkeztünk ehhez a részhez, akkor már biztos tudásunk van a C# nyelvről és a .NET- szerelvényformátumok részleteiről. Az ötödik rész továbbfejleszti ezeket az ismereteket azáltal, hogy számos általánosan használt alaposztálybeli szolgáltatást mutat be, többek között a fájl I/O-t és az adatbáziselérést az ADO.NET segítségével. Ez a rész tárgyalja továbbá az elosztott alkalmazások készítésének témakörét a Windows Communication Foundation (WCF) segítségével, valamint azokat a munkafolyamat-engedélyezett alkalmazások készítését, amelyek a Windows Workflow Foundation (WF) API-t használják.

### 20. fejezet: Fájlműveletek és elkülönített tárolás

A `System.IO` névtér segítségével lehetőségünk van a számítógép fájl- és könyvtárszerkezetének elérésére. Ebben a fejezetben megismerjük, hogyan kell programozottan könyvtárrendszert létrehozni (és törölni), valamint adatokat különböző folyamatba (fájlalapú, sztringalapú, memóriaalapú stb.) be-, illetve kimotozni. A fejezet utolsó része megvizsgálja az *elkülönített tárolást*, amelynek segítségével lehetőségünk van biztonságos helyen külön tárolni az adatokat függetlenül attól, hogy a célgép biztonsági beállításaitól. Hogy megértsük a `System.IO.Isolated` API bizonyos elveit, áttekintést kell kapnunk a kódhozzáférés-szabályozásról (Code Access Security – CAS).

## 21. fejezet: Bevezetés az objektumsorosítás világába

Ez a fejezet a .NET-platform objektumsorosító szolgáltatásába nyújt betekintést. Egyszerűen szólva a *sorosítás* segítségével lehetőségünk nyílik arra, hogy egy objektum (vagy a kapcsolódó objektumok egy halmazának) állapotát egy folyamatban tároljuk a későbbi használat céljából. A *deszerializálás* (a sorosítás ellentétes művelete) az a folyamat, amelynek segítségével kivethetjük az objektumot a folyamból, és a memóriában az alkalmazás számára használható formába hozhatjuk. Az alapok elsajátítása után megtanulhatjuk, hogyan kell egyedivé tenni a sorosítási folyamatot az `ISerializable` interface és a .NET-attribútumok halmazának segítségével.

## 22. fejezet: ADO.NET 1. rész: Az élő kapcsolat

Az adatbázisokról szóló két fejezet közül az elsőben az ADO.NET programozási API-ról kaptunk ismereteket. Ez a rész bevezeti a .NET adatszolgáltató szerepét, valamint azt, hogy hogyan kell a relációs adatbázissal kommunikálni az ADO.NET *élő kapcsolatán* keresztül. Ez a kapcsolati objektumokat, a parancsobjektumokat és az adatolvasó objektumokat jelenti. Ezenkívül ez a fejezet mutatja be az egyedi adatbázis létrehozásának módját, és ismerteti azokat az adatelérési könyvtárakat, amelyeket majd a könyv további részében használunk.

## 23. fejezet: ADO.NET 2. rész: A bontott kapcsolat

Ez a fejezet az adatbázis-kezelés különböző módokon történő manipulálási lehetőségeit tárgyalja tovább, jelen esetben az ADO.NET *bontott kapcsolaton* keresztül. Megismerkedhetünk a `DataSet` típus és az adatfeldolgozó objektumok szerepével, valamint számos Visual Studio 2008 eszközzel, amelyek nagyban megkönnyítik az adatvezérelt alkalmazások létrehozását. Mindeközben rávilágítunk arra, hogyan kell a `DataTable`-objektumokat felhasználói felületi elemekhez kapcsolni, olyanokhoz, mint például a Windows Forms `API Gridview` típusa.

## 24. fejezet: A LINQ API programozása

A 24. fejezet a LINQ programozási modellt mutatja be, pontosabban a LINQ „az objektum irányába” részét. Megvizsgáljuk, hogyan kell LINQ-lekérdezéseket alkalmazni relációs adatbázisokon, `DataSet`-objektumokon és XML-dokumentumokon. Mindeközben megtanuljuk az `adatkontextus`-objektum, valamint az `sqlmetal.exe` segédprogram szerepét, és megvizsgáljuk a Visual Studio 2008 különböző LINQ-támogató funkcióit.

## 25. fejezet: A WCF

A .NET 3.0 bevezetett egy teljesen új API-t, a WCF-et, amely lehetőséget ad arra, hogy szimmetrikus módon, függetlenül az alatta található rétegek felépítésétől elosztott alkalmazást fejlesszünk. Ez a fejezet feltárja a WCF-szolgáltatások, -gazdagépek és -kliensek készítésének lehetőségeit. A WCF-szolgáltatások rendkívül flexibilisek, a gazdagépek és a kliensek XML-alapú konfigurációs fájlokat használhatnak, hogy deklaratív módon adjanak meg címeket, kötéseket és szerződéseket.

## 26. fejezet: A Windows Workflow Foundation – Bevezetés

A .NET 3.0 a WCF mellett bevezette a WF API-t is, amely lehetőséget ad arra, hogy *munkafolyamatokat* definiáljunk, futtassunk és monitorozzunk. Ennek segítségével komplex üzleti folyamatok modellezhetők. Megtanuljuk a WF általános célját, ahogy azt is, hogy mi a szerepe az aktivitásoknak, a munkafolyamat tervezőnek, a munkafolyamat futtatási motornak, és a munkafolyamat-engedélyezett forráskód könyvtáraknak.

# 6. rész: Felhasználói felületek

Általánosan hibás feltételezés a .NET platformmal kapcsolatban az, hogy ez a keretrendszer kizárólag a webalapú felhasználói felületekkel foglalkozik (ez feltehetően a „.NET” kifejezés miatt van, hiszen felidézi az „Internet” szót, azaz a „webprogramokat”). A .NET kiemelkedő módon támogatja a webalkalmazások készítését, a könyv ezen része azonban a hagyományos felhasználói felületekre koncentrál két grafikus felület keretrendszerén, a Windows Forms és a WPF használatán keresztül.

## 27. fejezet: Windows Forms-programozás

Az eredeti asztali grafikus felhasználói felület, amely a .NET platformmal együtt kapható, a *Windows Forms*. Ez a fejezet ismerteti a felhasználói felület keretrendszerének szerepét, és bemutatja, hogyan kell főablakokat, párbeszédablakokat és menürendszereket készíteni. Feltárja továbbá az úrlapszármaztatás szerepét, és azt, hogy hogyan kell kétdimenziós adatot rajzolni a `System.Drawing` névtér segítségével. A témakör illusztrálására a fejezet végén készítünk egy (korlátozott funkcionalitással bíró) rajzolóalkalmazást.

## **28. fejezet: A WPF és az XAML**

A .NET 3.0 teljesen új grafikus felhasználói felületet vezetett be, amelyet *WPF*-nek nevezett el. Röviden, a WPF kimagaslóan interaktív és médiában gazdag front endek készítését teszi lehetővé asztali alkalmazásokhoz (és indirekt módon webalkalmazásokhoz). A Windows Formsszal ellentétben, ez a túltelített felhasználófelület-keretrendszer számos kulcsfontosságú szolgáltatást (2D-s és 3D-s grafika, animációk, gazdag dokumentumok stb.) integrál egyetlen egységesített objektummodellbe. Ebben a fejezetben megkezdjük az ismerkedést a WPF-fel és a kiterjeszhető alkalmazásjelölő nyelvvvel (Extendable Application Markup Language – XAML). Megtanuljuk, hogyan kell XAML-mentes, csak XAML-t használó és a kettő kombinációjából felépülő WPF-programokat létrehozni. A fejezet végén készítünk egy egyedi XAML-nézetgetőt, amelyet a további WPF-fel foglalkozó fejezetekben is használni fogunk.

## **29. fejezet: Programozás WPF-vezérlőelemekkel**

Ebben a fejezetben megtudjuk, hogy hogyan kell dolgozni a WPF-vezérlőelem tartalommodelljével, és érintünk számos vezérlőelemekkel kapcsolatos témát is, például a függőségi tulajdonságokat és irányított eseményeket. Jó néhány WPF-vezérlőelem használatát bemutatjuk, s szintén ebben a fejezetben magyarázzuk el az elrendezéskezelő, -vezérlő utasítások és a WPF-adatkötés modelljének a használatát.

## **30. fejezet: WPF 2D grafikus renderelés, erőforrások és témák**

Az utolsó fejezet a WPF tárgyalását három látszólag független téma vizsgálatával zárja. A WPF grafikus renderelő szolgáltatásának általában szüksége van arra, hogy egyedi erőforrásokat definiáljunk hozzá. Ezeknek az erőforrásoknak a használatával lehetőségünk van egyedi WPF-animációk készítésére, és grafikát, erőforrásokat és animációkat alkalmazva egyedi témák készítését fogjuk tudni megoldani WPF-alkalmazások számára. A különböző témák összegzéseképpen a fejezet utolsó részében azt illusztráljuk, hogyan kell futásidőben egyedi grafikus témákat alkalmazni.

## 7. rész: Webes alkalmazások fejlesztése ASP.NET segítségével

A 7. rész az ASP.NET programozási API használatával készített webes alkalmazásokat vizsgálja. Az ASP.NET szándékosan úgy lett kialakítva, hogy az asztali felhasználói felület létrehozását modellezze, úgy, hogy szabványos HTTP kérésekre/válaszokra helyez egy eseményvezérelt, objektumorientált keretrendszert.

### 31. fejezet: ASP.NET weboldalak készítése

Ez a fejezet vezet be bennünket az ASP.NET használatával történő webes alkalmazásfejlesztésbe. A kiszolgálóoldali szkripteket felváltják az igazi objektumorientált nyelvek (mint a C#, VB.NET és hasonlók). Bemutatjuk egy ASP.NET weboldal készítését, valamint az alatta található programozási modellt és az ASP.NET egyéb kulcsfontosságú elveit, például azt, hogy hogyan kell webkiszolgálót választani, valamint a `web.config` fájlok használatát.

### 32. fejezet: ASP.NET-vezérlőelemek, -témák és -mesteroldalak

Ez a fejezet azokkal a vezérlőelemekkel foglalkozik, amelyek a belső vezérlőfát töltik fel adattal. Megismerjük az alapvető ASP.NET webes vezérlőelemeket, többek között a ellenőrző vezérlőelemet, a belső oldali navigációs vezérlőelemeket és a különböző adatkötő műveleteket. Ugyancsak ennek a fejezetnek a feladata, hogy bemutassa a *mesteroldalak* és az ASP.NET témamotorjának a szerepét, amely a hagyományos stíluslapok szerveroldali megfelelője.

### 33. fejezet: ASP.NET állapotkezelési technikák

Ez a fejezet kiterjeszti az eddigi ASP.NET-ismereteinket, ugyanis megvizsgálja, milyen különböző módokon lehet állapotot kezelni a .NET alatt. Ahogy a klasszikus ASP, az ASP.NET is egyszerűen teszi lehetővé sütik létrehozását, ahogy alkalmazásszintű és munkafolyamat-szintű változók használatát is. Az ASP.NET azonban bevezet egy új állapotkezelő technikát: az alkalmazás-gyorsítótárat. Ha már tudjuk, milyen módokon lehet állapotot kezelni az ASP.NET alatt, megismerkedünk a `System.HttpApplication` alapsztály szerepével (a `Global.asax` fájlban belül), valamint azzal, hogy hogyan kell dinamikusan megváltoztatni a webalkalmazásunk futásidejű viselkedését a `web.config` fájl segítségével.

## 8. rész: Függelékek

A könyv utolsó része két fontos témát vizsgál meg, amelyek valójában nem illeszkedtek szervesen a könyv felépítésébe, ezért a függelékbe kerültek. A függelék kiteljesíti a C#-pal és .NET platformmal kapcsolatos ismereteinket, megvizsgáljuk ugyanis, hogyan kell régebbi kódokat integrálni a .NET-alkalmazásainkba, valamint hogyan kell a .NET-fejlesztést a Windows operációs rendszer családján kívül használni.

### A függelék: A COM és a .NET együttműködése

Azok, akik már programoztak Windows operációs rendszer alatt a .NET platform használata előtt, nagy valószínűséggel ismerik a Component Object Modelt (COM). Noha a COM-nak és a .NET-nek semmi köze egymáshoz (azon túl, hogy mindkettő a Microsoft szerzeménye), a .NET platform teljes névteret biztosít ahhoz (`System.Runtime.InteropServices`), hogy .NET-szoftverek COM-komponenseket használjanak, és fordítva. Ez a függelék az együttműködési réteget mutatja be egészen részletesen; ez a téma azért nagyon fontos, hogy a korábban megírt programjainkat át tudjuk menteni az újonnan készített .NET-alkalmazásainkba.

### B függelék: Platformfüggetlen .NET-fejlesztés a Monóval

Végül, de nem utolsó sorban a B függelék a .NET nyílt forráskódú implementációjával, a *Monóval* foglalkozik. A Mono segítségével lehetőségünk van gazdag tulajdonságokkal rendelkező .NET-alkalmazás létrehozására, telepítésére és futtatására különböző operációs rendszereken, mint például a MacOS X, Solaris, AIX és számos Linux-disztribúció. Mivel a Mono nagyjából kompatibilis a Microsoft .NET platformjával, így már tisztában vagyunk azazal, hogy mit tud nyújtani számunkra. Ezért a függelék a Mono installálását tárgyalja, valamint azt, hogy, milyen fejlesztői eszközök állnak a rendelkezésünkre, és hogyan működik a Mono futtatómotorja.

## Öt szabadon letölthető fejezet – még több információ

Azoknak, akiknek a 33 fejezet és a két függelék nem lenne elég, szabadon letölthetnek további öt fejezetet. A könyv korábbi verziói három olyan fejezetet tartalmaztak, amelyek a Windows Forms fejlesztésének szenteltünk (egyedi vezérlők vizsgálatával egyetemben), egy másik fejezetet a .NET remoting rétegének (`System.Runtime.Remoting` és társai), és a végül egy fejezet a hagyományos XML-webszolgáltatások készítésével foglalkozott az ASP.NET Web Service projekt sablonjainak a segítségével.

A könyv jelen kiadása nem tartalmazza ezt az öt fejezetet. Ennek az a legfőbb oka, hogy a .NET 3.0-ban a WCF és a WFP API-k rendre a .NET-remoting/XML-webszolgáltatások és a Windows Forms API-k örökösei. Ha mélyebben szeretnénk elmerülni a Windows Forms világában (azon túl, amit a 27. fejezet nyújtani tud), vagy meg akarjuk nézni, hogyan kell használni a (örökségül hagyott) .NET-remotingot és az XML-webszolgáltatás-API-t, egyszerűen csak meg kell nézni az Apress honlapján a megfelelő részt:

<http://apress.com/book/view/1590598849>.

Az itt található linke kattintva digitális formában letölthetjük a könyv további fejezeteit, ha sikerült válaszolni a könyv szövegéből véletlenszerűen feltett kérdésekre. (A letölthető fejezetek csak angol nyelven érhetők el – a SZAK Kiadó megjegyzése).

## A könyv forráskódjának igénylése

A könyvben szereplő összes forráskód-mintapélda (az öt szabadon letölthető további fejezetben levők is) szabadon és azonnal elérhető az Apress honlapjának Source Code/Download részlegéről. Egyszerűen gépeljük be a <http://www.apress.com> címet, válasszuk ki a Source Code/Download linket, és keressük meg a megfelelő címet, és, töltsük le az önmagát kicsomagoló \*.zip fájlt. A forráskódokat fejezetenként osztottuk fel.

Figyelem: a fejezetekben a Forráskód megjelölés, mint amilyen az alábbi is, arra utal, hogy a szóban forgó példákat be lehet tölteni a Visual Studio 2008-ba további vizsgáldások és módosítások céljából:

---

**Forráskód** Ez a forráskód-megjelölés egy kiválasztott könyvtárra hivatkozik.

---

Egyszerűen nyissuk meg a \*.sln fájlt a megfelelő alkönyvtárból. Ha nem használjuk a Visual Studio 2008-at (lásd a 2. fejezetet további integrált fejlesztői környezetekért), akkor manuálisan töltjük be a felkínált forráskódfájlokat a választott fejlesztői eszközbe.

---

**Megjegyzés** A magyar nyelvű változatban a forráskódok megjegyzéseit lefordítottuk. A letölthető forráskódok fordítása nem állt módunkban, így azok teljesen angol nyelvűek. Ilyen módon inkább megfelelnek a letölthető kódok használati céljának – a SZAK Kiadó megjegyzése.

---

## A lehetséges javítások

Elképzelhető, hogy a könyv végigolvasása során nyelvtani vagy forráskódbeli hibákat vél felfedezni a kedves Olvasó, bár remélem, hogy erre nem kerül sor. Ha mégis, elnézést kérek. Az aktuális hibalistát az Apress weboldalról lehet igényelni (amely szintén a könyv honlapján található meg), és ugyan-csak itt található az elérhetőségem, amelyen értesíteni tudnak.

## Elérhetőségem

Ha bármilyen kérdés merül fel a könyvhöz tartozó forráskódokkal kapcsolatban, vagy szükség van valamelyik példa tisztázására, vagy egyszerűen csak véleményt szeretne nyilvánítani a .NET platformról, nyugodtan küldjön levelet a következő e-mail címre (hogy biztosak legyünk benne, hogy a levele nem kerül a kéretlen levelek közé, kérem, tegye a tárgymezőbe a „C#FE” szöveget): [atroelsen@Intertech.com](mailto:atroelsen@Intertech.com).

Annak ellenére, hogy megpróbálok minden levélre lehetőség szerint válaszolni, sajnos, ahogy mindnyájan, időnként nagyon elfoglalt vagyok. Ha nem válaszolok egy-két héten belül, akkor ez nem jelenti azt, hogy nem akarok foglalkozni a problémával, csak épp nagyon nem érek rá (vagy ha szerencsém van, valahol épp nyaralok).

Nos, akkor, köszönöm, hogy megvásárolta a könyvemet (vagy legalábbis, hogy belenéz a könyvesboltban, miközben azon gondolkodik, hogy megvegye-e). Remélem, élvezni fogja a kedves Olvasó a tanulmányozását, és hasznosítani tudja az újonnan szerzett ismereteket.

Minden jót!  
Andrew Troelsen