

2009



Andrew Troelsen

Második kötet (20-33. fejezet)

A C# 2008
és a .NET 3.5



Főszervező: Kis Balázs MSc., MCT, e-mail: balazs.kis@sarak.hu
info@sarak.hu ■ Kiadóvezető: Kis Ádám, e-mail: adam.kis@sarak.hu ■
36-22-350-209 ■ Fax: 36-22-565-311 ■ www.sarak.hu ■ e-mail:
Könyvterjesztő Gyenes László a tagja ■ 2060 Bicske, Díjfa u. 3. ■ Tel.:
SZAK Kiadó Kft. ■ Az 1795-ben alapított Magyar Könyvkiadók es

Minden jog fenntartva. Jelen könyvet, illetve annak részét a kiadó engedélye
nélküli másolás, rögzítés, adatátvitel rendszerben táralmi, bármilyen formában
vagy eszközökkel elektronikus úton vagy más módon kozolni.

A könyvvá Fordítása a Kilgray Kft. MemooQ (<http://www.memooq.com>) progra-
májával készült, a szöveg helyessegétszerelése az elválasztásokat a Morphologic He-
lyesek névű programjával ellenoriztuk.

ISBN 978-963-9863-10-1

Magyár nyelvi kiadás: Laczko Krisztina és Trepák Monika
Lektor: David Zoltán, MVP és Gincsai Gábor, MVP
Varga Péter
Eros Szilvia, Ivancsy Renáta, Szalay Márton, Tibor Melinda, Varga Ágnes,
Fordította a SZAK Kiadó fordítócsapat: Domoszai László, Endredi Gabriele,
Magyar fordítás (Hungarian translation) © SZAK Kiadó 2009.

Original English language edition published by Apress Co. Sound View Court 3B,
Greenwich CT 06830 USA. Copyright © 2008 by Apress Co. Hungarian-language
edition copyright © 2009 by SZAK Kiadó. All rights reserved.
Copyright 2008 by Apress.

A negyedik kiadás magyarul, ketet kötetben
A C# 2008 és a .NET 3.5 - Magasolt körzet (20-33. fejezet és két függelék)
Pro C# 2008 and the .NET 3.5 Platform, Fourth Edition
Andrew Troelsen

Ezt a kiadást Miklónnak, a csodamacska nákkal, a 412-es szám alatt eltoltott
éveknek és csodálatos felleségemnek, Amandának ajánlom, aki tisztelmesen
várta, hogy elkezdi írni újabb könyvet megírásával.

Mi egy csapatt vagyunk, az Olivásó és en.....	xxxii
A könnyv áttekintése.....	xxxiii
Első kötet.....	xxxiii
1. rész: Bevezetés C#-ba és a .NET platformba.....	xxxiii
2. fejezet: C#-alkalmazások készítése	xxxiv
3. rész: A C# alapvető építőelemi	xxxiv
4. fejezet: A C# alapvető építőelemi, II. rész	xxxv
5. fejezet: Egysegébke zárt osztálytipusok definíciója	xxxv
6. fejezet: A származtatás es a polymorfizmus	xxxv
7. fejezet: Struktúrált hibakezelés	xxxv
8. fejezet: Az objektumok elektikusa	xxxv
3. rész: Haladó programozási szerekkel a C#-ban	xxxvi
9. fejezet: Interfeszkek használata	xxxvi
10. fejezet: Gyűjtmemenyek és generikus típusok	xxxvi
11. fejezet Metódusreferenciák, események és Lambdák	xxxvi
12. fejezet: Indexterlek, operatork és mutatók	xxxvii
13. fejezet: C# 2008 nyelvi újdonságai	xxxvii
14. fejezet: Bevezetés a nyelvbe ágyazott	xxxviii
lekérdezésekhez (LINQ)	xxxviii
4. rész: Programozás.NET-szerelvények	xxxviii
15. fejezet: A .NET-szerelvények	xxxviii
16. fejezet: Tipusreflexió, késői kötés és	xxxviii
attribútumalapú programozás	xxxviii
17. fejezet: Folyamatok, alkalmazásstartományok és	xxxix
a dinamikus szerelvények	xxxix

Tartalomjegyzék

Köszönnetnyilvánítás	xxix
Bevezetés	xxxi
Mi egy csapatt vagyunk, az Olivásó és en.....	xxxii
A könnyv áttekintése	xxxiii
Első kötet	xxxiii
1. rész: Bevezetés C#-ba és a .NET platformba	xxxiii
2. fejezet: C#-alkalmazások készítése	xxxiv
3. rész: A C# alapvető építőelemi	xxxiv
4. fejezet: A C# alapvető építőelemi, II. rész	xxxv
5. fejezet: Egysegébke zárt osztálytipusok definíciója	xxxv
6. fejezet: A származtatás es a polymorfizmus	xxxv
7. fejezet: Struktúrált hibakezelés	xxxv
8. fejezet: Az objektumok elektikusa	xxxv
3. rész: Haladó programozási szerekkel a C#-ban	xxxvi
9. fejezet: Interfeszkek használata	xxxvi
10. fejezet: Gyűjtmemenyek és generikus típusok	xxxvi
11. fejezet Metódusreferenciák, események és Lambdák	xxxvi
12. fejezet: Indexterlek, operatork és mutatók	xxxvii
13. fejezet: C# 2008 nyelvi újdonságai	xxxvii
14. fejezet: Bevezetés a nyelvbe ágyazott	xxxviii
lekérdezésekhez (LINQ)	xxxviii
4. rész: Programozás.NET-szerelvények	xxxviii
15. fejezet: A .NET-szerelvények	xxxviii
16. fejezet: Tipusreflexió, késői kötés és	xxxviii
attribútumalapú programozás	xxxviii
17. fejezet: Folyamatok, alkalmazásstartományok és	xxxix
a dinamikus szerelvények	xxxix

A System.IO nevűter	3
20. Fájlműveletek és elszigetelt tárókás	3
Az absztrakt FileSystemlinie osztály	5
A Directory (info) és File (info) típusok	6
A DirectoryInfo típus használata	7
Fájlök lista ásása a DirectoryInfo típus segítségével	9
Alkonyvtárak letrehozása a DirectoryInfo típus segítségével	10

5. rész: Bevezetés a .NET alaposztálykonnyvtáribába

Második kötet	xxxix
5. rész: Bevezetés a .NET alaposztálykonnyvtáribába	xxxix
20. rész: Fájlműveletek és elszigetelt tárókás	xxxix
21. fejezet: Bevezetés az objektumsortostas világábba	xli
22. fejezet: ADO.NET 1. rész: Az élő kapcsolat	xli
23. fejezet: ADO.NET 2. rész: A bonott kapcsolat	xli
24. fejezet: A LINQ API programozása	xli
25. fejezet: A WCF	xli
26. fejezet: A WF	xli
27. fejezet: Windows Forms-programozás	xli
28. fejezet: A WPF és az XAML	xlii
29. fejezet: Programozás WPF-vézeteljellemekkel	xlii
30. fejezet: WPF 2D grafikus rendszerek	xlii
7. rész: Webes alkalmazások fejlesztése ASP.NET segítségével	xliii
erőforrások és temák	xliii
temák és mesterdalak	xliii
33. fejezet: ASP.NET állapotkezelési technikák	xliii
A függelék: A COM és a .NET egységtimunkodése	xliii
B függelék: Platformfüggelek.NET-felisztés a Monorail	xliii
C szabadon letölthető fejezet – még több információ	xliii
A könnyű forráskódjainak igénylése	xlv
A lehetőséges javítások	xlv
Elehetőségek	xlv
20. Fájlműveletek és elszigetelt tárókás	3

A Directroy típus használata	11
A Drivelfifo osztály típus használata	12
A Fileinfo osztály használata	14
A Fileinfo Open() metodus	15
A Fileinfo Create() metodus	16
A Fileinfo OpenWrite() és a Fileinfo OpenRead() metodusok	17
A Fileinfo OpenText() és a Fileinfo AppendText() metodusok	18
A Fileinfo OpenFileDialog() metodus	18
A Fileinfo CreateText() és a Fileinfo AppendText() metodusok	18
A absztrakt Stream osztály	22
További fajlözöponttú tagok	20
A StreamWriter és StreamReader típusok használata	23
A StreamWriter es StreamReader típusok használata	29
A StringWriter és StringReader típusok közvetlen leterhezása	29
Olvásás szövegfüjlöl	27
Szövegfájl írása	26
A StreamWriter es StreamReader típusok használata	25
A absztrakt Stream osztály	22
A File típus használata	19
A Fileinfo CreateText() metodus	20
A absztrakt Stream osztály	22
A FileStream típusok használata	23
A StreamWriter es StreamReader típusok használata	29
A BinaryWriter és BinaryReader osztályok használata	31
Fájlok programozott „folyelés”	34
Asztiniron fájllolvásás és -írás	36
Az elszigetelt fájlok szerepe	38
Bizalom kerülete	38
Az elszigetelt fájlok szerepe	38
Bizalom kerülete	38
Akciós működés	50
A CAS működése	50
Full Trust jogosultság visszatartása	52
A My Computer Zone kódcsoporthoz	52
Az elszigetelt fájlok hélye	55
Az elszigetelt fájlok hatókörre	55
Az elszigetelt fájlok segítőjével	57
A System.IO.IsolatedStorage típusa	57
Tároló leterhezása az IsolatedStorage objektummal	58
Adat irása a tarolóbaba	60

Az olvasásra a tárloból	61
Felhasználói adat törlése a tárloból	62
Egyedi konnyvtársstruktúra létrehozása	62
Az elszigetelt tárlohoz közelés CíkCOne-telpeiles	64
Az IsolatedStorageFfilePermission attributum	65
Az biztonságí zóna korlátozása	65
Az alkalmazás kódzettetelé webkiszolgálón	67
Az eredmény megtékinthese	67
Osszefoglalás	69
Az objektumosről színesítés	71
21. Bevezetés az objektumosről színesítés világába	71
Nyilvános mezők, privat mezők es nyilvános tulajdonságok	76
A sorosító formázó kiválasztása	77
Az Iformatter és az IremotingFormatter interfejszek	78
A formázók közötti típuskontosság	80
Objektumok visszaállítása a BinaryFormatterrel	81
Objektumok sorosítása a BinaryFormatterrel	83
Objektumok sorosítása a SoapFormatterrel	83
Objektumok sorosítása az XmlSerializerrel	85
Generalt XML-adatok szabalyozása	86
Objektumok sorosítása az XmlDocumentrel	88
A sorosítási folyamat tesztelése	90
Az objektumosről tesztelés hatere	91
Sorosítások teszteléséhez használtak	92
Sorosítások teszteléséhez a Serializable használataval	92
Sorosítások teszteléséhez a Streamrel	96
Osszefoglalás	97
22. ADO.NET, 1. rész: Az elő kapcsolat	99
Az ADO.NET magas szintű meghatározása	99
Az ADO.NET kez arca	101
Az ADO.NET-adatszolgáltatók működése	102
A Microsoft által származó ADO.NET-adatszolgáltató	104
Harmandik feltétel származó ADO.NET-adatszolgáltató	106

További ADO.NET-nevek	107
A System.Data névter típusai	108
Az IDbConnection interfész szerepe	109
Az IDbTransaction interfész szerepe	109
Az IDbCommand interfész szerepe	110
Az IDbDataAdapter és az IDbAdapter interfész szerepe	111
Az IDbDataAdapter interfész szerepe	112
Az AdatReader és az AdatRecord interfész szerepe	113
Rügalmasság növelése az alkalmazásoknál figuraelsőjének	115
Fajlokkal	115
Az AutoLot adatbázis leterhözés	117
Az Inventory tabla leterhözés	118
Az GetPartName() tárolt eljárás leterhözés	120
Az Customers és az Orders tablák leterhözés	121
Tablakapcsolatok vizuális bemutatása	123
Az ADO.NET data provider factory modell	124
Regisztrált data provider factory	125
Egy teljes data provider factory példa	126
A data provider factory modell lehetséges hárány	130
A ConnectionString elém	130
Az ADO.NET kapcsolatlapú modelle	131
A kapcsolatobjektumok használata	133
A ConnectionStringBulider objektumok	136
Az adatolvások	139
Tobb eredményhalma közrese adatolvasval	141
Ürítelhasználható adatleírések készítése	142
A kapsolatiobjektumok leterhözés	144
A beszürést végező logika leterhözés	145
A torleszt végrehajtó logika leterhözés	145
A módosítást végező logika leterhözés	146
A lekérdezést végrehajtó logika leterhözés	147
A paramétereit parameterekkel	148
A paramétereit parameterekkel	148
Tárolt eljárások végrehajtása	150
Parameterek megadása a DbParameter típus segítségével	150
Parameterek front end leterhözés	152

A Main() metódus implementálása	153
A ShowInstruments() metódus implementálása	155
A ListInventory() metódus implementálása	156
A DeleteCar() metódus implementálása	156
A InsertNewCar() metódus implementálása	157
A UpdateCarWithName() metódus implementálása	157
A tarothi eljárásunk megérválasa	158
Aszinkron adatleírás az SqlCeCommand használatával	159
Az adatbázis-tranakciók	161
Az ADO.NET-tranakcióobjektum külcsfotosságát tagjaí	162
Tranzakciómódus hozzáadása az InventoryDAL	162
Az adatbázis-tranzakciók tesztelése	166
Osszefoglalás	167
23. ADO.NET, 2. rész: A bontott kapcsolat	169
Az ADO.NET kapcsolat nekűli modellje	170
A DataSet szerepe	171
A DataSet alapvető tulajdonságai	172
A DataSet külcsfotosságú módusai	173
DataColumn típusok használata	174
DataSet letrehozása	174
DataColumn típusok használata	174
A DataColumn letelezoása	176
A mezők automatikus novellesenek engedélyezése	177
DataColumn objektum hozzáadása egy	177
Datatable típusok használata	178
Datatable típushoz	178
A RowState típusok használata	178
A RowState tulajdonság	179
DataRow típusok használata	180
A DataRowVersion tulajdonság	182
DataTable típusok használata	183
DataTable típusok beszűrása DataSet objektumokba	185
A DataTable adatnak feldolgozása DataTableReader	186
A DataSet objektumok sorosítása XML-Kennet	188
objektumokkal	188
A DataTable/Dataset objektumok sorosítása bináris	189
formátumban	189
Datatable felületei egy generikus List<T> használatával	190
Datatable objektumok körte séle felhasználói felületekhez	192

Sorok programozott törlese	194
Sorok kiállásztása szüresi feltételek alapján	196
Sorok modosítása	199
A DataView típus használata	200
Egy utolsó felhasználói felületbővítmény: sorok számának megjelenítése	202
DatASET/DATABLE objektumok felületeihez adattílesztőkkel	203
Egy egszerű adattílesztő	205
Adatbázisnevek leképezésére barátágos nevekre	206
Az AutoLotDAL.dll ismételt vizsgálata	207
A kiinduló osztálytípus definíciója	207
Az adattílesztő konfigurálása az SqlCommandBuilder	208
használataival	210
Windows Forms front end letrehozása	211
Navigálás a többtáblázatos DataSet objektumokban	212
A táblázatok közötti kapcsolatok kiépítése	215
Az adattílesztők előkezelése	216
Navigálás a kapcsolódó táblázatok között	216
A Visual Studio 2008 adattílerési eszközei	220
A DataGridview vizuális megtervezése	220
Az App.config fájl és a Settings.Settings fájl	224
A generált DataSet vizsgálata	226
A generált DataTable és DataRow típusok vizsgálata	228
A generált adattílesztő	230
Generált típusok használata a kódban	231
Az automatikusan generált kod elválasztása	233
Egy felhasználói felület front end: újra a felhasználói felület-renderol	236
Osszefoglalás	237
24. A LINQ API programozása	239
A LINQ to ADO.NET szerepe	239
Programozás LINQ to DataSettel	240
A DataSET bővítmenyek szerepe	242

Datatable LINQ-kompatibilitás használata	243
A DataRowExtensions.Field<T>() bővíti metódus szerepe	245
Üj Datatable objektumok felületese	246
Az entitássztruktúrok szerepe	248
Egy egyszéria LINQ to SQL Példa	249
Fősen tipusos DataContext leterhözés	251
A [Table] és [Column] attribútumok: további részletek	253
Entitássztruktúrok generálása az SqlMetal.exe használatával	254
Kapcsolatok definíciósa entitássztruktúrok használatával	258
Az erősített tipusos DataContract	259
A generált tipusok használata	260
Erittességekkel entitássztruktúrok LINQ to SQL Studio 2008 használatával	262
Üj elemek beszürése	264
Letező elemek törlése	265
XML-dokumentumok kezelése a LINQ to XML használatával	267
LINQ to XML: egy jobb DOM	267
A System.Xml.Linq névter	268
XML-dokumentumok leterhözés programozottan	269
Dokumentumok leterhözés LINQ-lekérdezésekkel	271
XML-tartalom betöltese és elemzése	272
Navigálás egy memoriabán lévő dokumentumban	272
Adatok módosítása egy XML-dokumentumban	275
Összefoglalás	276
25. A WCF	277
Nehány előzettszolgáltató API	277
A DCOM szerepe	278
A COM+/Enterprise Services szerepe	279
A MSMQ szerepe	280
A .NET-rendezés szerepe	281
Az XML-webszolgáltatás szerepe	282

Pelida .NET-wébszolgálatasra	282
Wébszolgálatasi szabványok	285
Named pipe-ok, soketek és P2P	286
A WCF szerepe	286
A WCF-funkciók áttekintése	287
A szolgáltatásorientált architektúra áttekintése	288
1. alapelve: A határok explicitek	289
2. alapelve: A szolgáltatások autonómok	289
3. alapelve: A szolgáltatások szerződésen keresztül es nem implementáció keresztül kompatibilis	289
4. alapelve: A szolgáltatás kompatibilitása	289
házirendben alapul	289
WCF: A lényeg	290
Az alapvető WCF-szerelvények	290
A Visual Studio WCF projektsablonok	292
A WCF Service Website projektsablon	293
A WCF-falkalmazás alaposszéallítása	294
A WCF ABC-jé	296
A WCF-szerződésök	297
A WCF-Kötések	298
HTTP-alapú Kötések	299
TCPIP-alapú Kötések	300
MSMQ-alapú Kötések	301
A WCF-címek	302
WCF-szolgáltatás készítése	304
A [ServiceContract] attribútum	306
Az [OperationContract] attribútum	307
Szolgáltatástípusok mint működési szerződésök	308
A WCF-szolgáltatás hoztola	308
ABC-k leírások az App.config fájban	309
A ServiceHost típus	313
Hosszfejlesztési lehetőségek	311
A ServiceHost típus használata	310
A ServiceHost típusok szerződési szabványai	313
A <system.serviceModel> elem jellemezői	315
Metadatcsere (Metadata Exchange) engedélyezése	317
WCF-ügyfélalkalmazás készítése	320
Proxykód generálása az svcsutil.exe segítségével	320
Proxykód generálása Visual Studio 2008-ban	321
TCP-alapú kötes konfigurálása	323

A WCF Service Library projektsablon használata	325
Egyeszerű Matér-szolgáltatás készítése	325
A WCF-szolgáltatás tesztelése a WcfTestClient.exe-vel	326
A konfigurációs fájl modosítása az SvcConfigEditor.exe	
Programmal	327
WCF-szolgáltatás használása Windows-szolgáltatásként	329
Az ABC-k megadása a forrásokban	330
A Windows-szolgáltatás telepítése	332
Windows-szolgáltatás tesztelése a Windows-szolgáltatásra	332
A MEX engedélyezése	332
Szolgáltatás azinikron hívása	334
WCF-adatszerek tervezése	337
Webközpontú WCF Service projektsablon használata	338
Szolgáltatásokat implementálva	340
A *.svc fájl szerkeze	341
A Web.config fájl módosítása	342
A szolgáltatás tesztelése	342
Osszefoglalás	343
26. A Windows Workflow Foundation – Bevezetés 345	
Egy üzleti folymat definíciója	345
A WF szerepe	347
A WF építőköckái	347
A WF futtatáskörnyezete	348
A WF alapvető szolgáltatási	349
A WF-tevékenységek elég megtölthetők	350
Szekvenciális és állapotcsoport-munkafolyamatok	352
WF-szerelvénylek, -növeterék és -projektek	355
A .NET 3.5 WF-támogatása	356
Visual Studio munkafolyama-alkalmazás Lethozás	358
A kezdeti munkafolyamathoz tartozó kod vizsgálata	358
A Code tevékenysége hozzáadása	359
While tevékenysége hozzáadása	361
A WF-motor használási Kodja	364
Egyedi indítási paraméterek hozzáadása	365
Webszolgáltatások hívása a munkafolyamatunkban	368

A MathWeb szolgáltatás leterhezása	368
A WF-wébszolgáltatás-fogyaztó leterhezása	370
Az IFEleszterkezés ségek konfigurálása	372
Az InvokeWebszolgáltatás-ségek konfigurálása	374
Kommunikáció WC-szolgáltatással a SendActivity	
Segítségevel	377
Ugrafelehasználható WF-kódoknýtár leterhezása	382
Hitelellenőrzés végrehajtása	384
Windows Forms-kleinákkalmazás leterhezása	387
Az eggyedi tevékenységek	390
Osszefoglalás	391
27. Windows Forms-programozás..... 395	
Egyesrejtett Windows Forms-kalkmazás (IDE-mennet)	
A Windows Forms-neveték	396
A system.EventArgs és A System.EventHandler szerepe	402
A Visual Studio Windows Forms-projektsablonja	404
A vizuális tervezőfelület	404
A kezdeti trilap	406
A program osztály	408
Mennitrendszerk vizuális építése	409
A Control osztály funkcionálitása	411
A Form osztály funkcionálitása	417
A Form típusa elérési	419
Reagálás az égér eseményeire	422
Az egér gombokkal műveletei	424
Reagálás a billentyűzet eseményeire	425
Parbeszédbalkok tervezése	427
A DiálogResult tulajdonság	429
A tabulátoros rend-konfigurálása	430
Az ürlap alapértelmezett beviteli gömbjának a beállítása	431
Parbeszédbalkok megjelenítése	431

6. rész: Fehasználói felületek

A WF-wébszolgáltatás-fogyaztó leterhezása	370
Az IFEleszterkezés ségek konfigurálása	372
Az InvokeWebszolgáltatás-ségek konfigurálása	374
Kommunikáció WC-szolgáltatással a SendActivity	
Segítségevel	377
Ugrafelehasználható WF-kódoknýtár leterhezása	382
Hitelellenőrzés végrehajtása	384
Windows Forms-kleinákkalmazás leterhezása	387
Az eggyedi tevékenységek	390
Osszefoglalás	391

28. A WPF és az XAML	453
A WPF mozgatásúrólja	453
A különböző API-k egyégesítése	454
Kapcsolatok elküldésére a XAML segítségével	455
Optimalizált rendszerek modell biztosítása	456
További WPF-központhasznos tulajdonoságok	457
A WPF-alkalmazások különböző típusai	458
Hagyományos asztali alkalmazások	458
NAVIGÁCIÓALAPÚ WPF-alkalmazások	459
XBAP-alkalmazások	460
Silverlight-alkalmazások	461
A WPF-szerelvények vizsgálata	461
Az Application osztály szerépe	463
A Window osztály szerépe	465
A System.Windows.Controls.ContentControl	465
A System.Windows.Controls.Control alaposztály szerépe	468
A System.Windows.Controls.Window alaposztály szerépe	469
A System.Windows.FameworkElement alaposztály szerépe	470
A System.Windows.UITelement alaposztály szerépe	471

28. A WPF es az XAML 453

Az utánpók származtatása	433
GDI+ -alapú gráfikus adatok renderelése	436
A System.Drawing névter	437
A Graphics típus szerpe	438
Graphics objektumok megszerzése a Paint	439
Teljes Windows Form-s alkalmazás létrehozása	442
A formirendszer készítése	443
A ShapeData típus meghatározása	444
A ShapePictureDialog típus meghatározása	445
Interaktív struktúra hozzáadása a MainWindow típushoz	446
A Tools menü funkcióitának implementálása	447
A grafikus kiemelő rögzítése és renderelése	449
A sorosítási logika implementálása	450
Oszzefoglalás	452

A System, Windows, Media, Visual szerpe	470
A System, Windows, DependencyObject osztály szerpe	470
A System, Windows, Threadинг, DispatcherObject szerpe	471
(XAML-ménetes) WPF-alakmazás készítése	471
A Window osztálytipus kibővítése	474
Egyszerű felhasználói felület letrehozása	475
Az Application típus további jelemezői	477
Az Application típus további eseményei	479
feloldogozása	479
Az Application típus Windows gyűjtőeleménnye	480
A Window objektum Ellettertama	480
A Window objektum Closung eseményének kezelése	482
Ablakszintű egérresemények kezelése	484
(XAML-központhoz) WPF-alakmazás készítése	486
A MainWindow definíciója XAML-ben	487
Alkalmazásobjektum definíciója XAML-ben	488
XAML-fájlok feloldogozása az msbuild.exe segítségével	489
Markup átalakítása .NET-szerelvénye	492
A BAML szerpe	494
XAML-leképezése C#-kódra	492
A XAML-szerkezet	499
A kapcsolatok elküldönösére mógsöttes Kodfájlökkel	497
XAML-ból szerelvénnyel: a Foleyamat összefoglalása	496
A XAML-szintaxis	499
XAML-kiszereletek a XamlPad segítségével	501
XAML-élémek és -külcsszavak	505
A XAML tulajdonságok szintaxisa	506
A XAML markupbólmenyei	512
WPF-alakmazások készítése a Visual Studio 2008 segítségével	517
WPF-projektsablonok	518
A kezdeti ablak névenek módosítása	519
A WPF-tervzó	520

XAML feldolgozása futásidőben: A SimpleXamlPad.exe	523
A Loaded esemény megvalósítása	525
A Button kattintási eseménynek megvalósítása	526
A Closed esemény megvalósítása	527
A alkalmazás tesztelése	528
A Microsoft Expression Blend szerpe	529
Az Expression Blend előnyei	530
Osszefoglalás	531
29. Programozás WPF-vezérlőelemekkel	531
A WPF vezérlőelem-könnyvtár vizsgálata	531
A reszletek a dokumentacióban találhatók	534
Vezérlőelemek deklarációja a XAML-ben	535
Együttműködés a vezérlőelemekkel a forrásokkaljukban	536
A vezérlőelemek definíciója a XAML-ben	537
A függőségi tulajdonságok szerpe	539
A függőségi tulajdonságok szerpe	540
A tövábbi tulajdonságok szerpe	541
További események	544
A tövábbi események szerpe	546
A buörök események folytatása és leállítása	547
A tövábbi események szerpe	548
A jelölőnégyzetek és radiogombok használata	549
Logikai csoporthasználat	550
A kapcsolódó elemek bonyolultabb csoporthasználat	551
A button típusok használata	551
A buttonbase típus	551
A button típus	552
A button típus	553
A repeatbutton típus	554
A checkbox típusok használata	556
A jelölőnégyzetek és radiogombok használata	558
Logikai csoporthasználat	559
A kapcsolódó elemek összeállítása GroupBox típusokba	560
A listbox és a combobox típusok használata	562
Listá-vezérlőelemek felületes programozt módon	563
Textszöveges tartalom használása	564
Akciális kválasztás meghatározása a beágyazott	566
Tartalom esetében	566

A többszöros szövegbeviteli mezők használata	568
A TextBox típus használata	569
A PasswordBox típus használata	570
A taralomelrendezés kezelése panelek használatával	572
A WPF alapvető paneleitípusai	573
Tartalom elhelyezése a Canvas panelekben belül	574
Tartalom elhelyezése a WrapPanel panelekben belül	577
Tartalom elhelyezése a StackPanel panelekben belül	579
Tartalom elhelyezése a Grid panelekben belül	580
Racsok GridSplitter típusokkal	582
Tartalom elhelyezése a DockPanel panelekben belül	583
A lapozás engedélyezése a Paneltípusoknál	584
Ablak keretének kezelése beágyazott panelek használatával	585
A menürendszer kezelése	587
A ToolBar típus kezelése	588
A StatusBar típus kezelése	589
A felhasználói felület véglegesítése	590
A WPF vezetőkötött tisztasági	592
A belső vezetőelem parancsobjektumok	593
Ütastátsok kapcsolása a felhasználói felület tetszőleges	595
Ütastátsok kapcsolása a Command tulajdonoságba	596
elemeihez	596
A WPF adatkötési modell	598
Ismerkedés az adatkötessel	599
A DataContract tulajdonság	600
Adattalálás az ValueConverter segítségével	602
Konvertációk különöző adattípusok között	605
Kötés egyedi objektumokhoz	606
Az ObservableCollection<T> típus használata	608
Egyedi adatsablon kezelése	610
A felhasználói felület elérésének kötése	611
XML-dokumentumokhoz	612
Egyedi parbeszédbalk kezelése	614
A DiálogResult eretk hozzárendelése	615
Az aktuális kiállásztás megszerezése	616
Egyedi parbeszédbalk megjelenítése	617

30. WPF 2D grafikus renderelés, erőforrások	619
és témák	
A WPF grafikus renderelési szolgáltatásának filozófiaja	620
A WPF grafikus renderelei lehetségei	621
A Shape leszámított típusainak használata	622
A Drawing leszámított típusainak használata	623
A Visual leszámított típusainak használata	624
Egyedi vizuális renderelei program készítése	627
A megfelelő megoldás kiválasztása	629
A Shape leszámított típusainak felülvizsgálat	630
A Shape leszámított típusainak funkcionálisai	631
A Rectangule, az Ellipse és a Line típusok használata	631
A Path típusok használata	637
A ImageBrush típus	636
A Atmenetes esetek használata	635
Egyéb típusok használata	633
A WPF-ecsettípusok használata	634
Egyéb típusok használata	635
A Geometri típusok szerpe	639
Egyéb típusok a DrawingImage típusban	641
Drawing típusok a DrawingBrush típusban	642
Osszetett rajzoló geometria	643
A fehásználófehér-tranzformációk szerpe	645
A Transform leszámított típusok	646
A WPF animációs szolgáltatásai	647
Az Animation utolaggal rendelkező típusok szerpe	648
A Timeline összetűlő szerpe	649
Animáció készítése C#-forrásokból	650
Az animáció ütemezésének szabályozása	652
Animáció lépteszásához használata	653
Animáció készítése XAML-leírásval	654
A Storyboard típus szerpe	654
Az <EventTrigger> használata	655
A külcsépkocka-animáció szerpe	655
Animáció diszkrét külcsépkockakkal	656

A HTTP szerepe	685
A HTTP Kérés/Válasz-ciklus	686
A HTTP szerepe	686
Webs alkalmazások és webkiszolgálók	687
Az IIS virtuális környvű tárának szerepe	688
Az ASP.NET féllezesztési Galéria	689
A HTML szerpre	691
HTML-dokumentumstruktúrák	692
HTML-típusok féllezesztése	693
HTML-alapú felhasználói felület készítése	695

31. ASP.NET-weboldalak készítése

segítségevel

7. rész: Webs alkalmazások féllezesztése ASP.NET

Animáció linéaris kulcsépekkel	658
A WPF erőforrásrendszer	660
A bináris erőforrások használata	660
A Resource Build Action	661
A Content Build Action	662
Az objektum (vagy másnéven logikai) erőforrások szerepe	663
A WPF készítés es alkalmazása WPF-vezérlőelemeken	663
A megnevezett stílusok használata	665
Stílusbeállítások felülbírálása	667
Létező stílusok leszármaztatása	667
A stílusok kiterjesztése	669
A stílusok korlátosása	669
Stílusok hozzárendelése implicit módon	672
Stílusok definíciója többekkel	670
Egységi sablon készítése	676
Típustípusokhoz hozzáadása a sablonokhoz	677
Sablonok használata a stílusokban	679
Összefoglalás	681

Az ügyféloldali szkriptek szerepe	698
Példa az ügyféloldali szkriptekre	699
A default.htm tulajdonságok ellenorizése	700
Az ügyféloldali szkriptekkel tövábbítás (a GET és a POST)	701
Klasszikus ASP-oldal kezelése	702
A klasszikus ASP problémái	705
Az ASP.NET 1.x röbbelnyei	705
Az ASP.NET 3.5 legrégebbi webes üjdonságai	707
Az ASP.NET-neveték	707
Az ASP.NET weboldal-kódolási modelje	709
Adatkezponthoz kötött példák manuális hivatalozása	710
Az AutoloTDDL.dll fájl manuális hivatalozása	710
A felhasználói felület tervezése	711
Adatelerei logikai hozzáadása	712
Az ASP.NET-direktívák szerepe	715
A szkriptblokk elmezese	716
Az ASP.NET vezetőelem-alkatrészök attétele	717
A mögötteskod-modell használata	718
Hivatalozás az AutoloTDDL.dll szerevénnyére	720
A kodfájl modosítása	720
Az ASP.NET-oldal fordítási cíklusa	726
Tobbfájlos oldalak fordítási cíklusa	727
A Page típus származtatás ízlése	729
Együttműködés a bejövő HTTP-kereséssel	730
Bonugesztszabályok	732
Hozzáérés a bemeneti tulajdonságokhoz	733
Az ISPostBack tulajdonság	734
Együttműködés a kimenő HTTP-válaszokkal	735
HTML-tartalom kiírásátás	736
Felhasználók átirányítása	737
Az ASP.NET-weboldalak elérhetősége	738
Az AutoEventWireup attribútum szerepe	740

32. ASP.NET-vezérlőelemek, -téma&szkék és -mesteroldalak ..	747
A webes vezérlőelemek visszakedésének megértese ..	747
Kiszolgálóoldali események kezelése ..	748
Az AutoPostBack tulajdonság ..	749
A System.Web.UI.HtmlControls tipus ..	751
Vezérlőelemek fejlesztés ..	752
Vezérlőelemek dinamikus hozzáadása (es torlesse) ..	754
A System.Web.UI.WebControls.WebControls WebControl típus ..	755
Nehány szöveg a System.Web.UI.HtmlControls típusokról ..	756
Sokoldali ASP.NET-webelem készítése ..	759
Mesteroldalak használata ..	760
Az Inventory tartalomlap tervezése ..	770
A Build-a-Car tartalomlap tervezése ..	774
Rendelés es lapozás engedélyezése ..	774
Az elérhetőségi szintek szerepe ..	778
A RequiredFieldValidator vezérlőelem ..	780
A RegularExpressionValidator vezérlőelem ..	781
A RangeValidator vezérlőelem ..	782
A CompareValidator vezérlőelem ..	782
Ellenorzes összegzesenek leterhözesa ..	783
Ellenorzes összegzesenek vezérlőelem ..	785
Témák használata ..	787
A *Skin Fajlok ..	788
Témák alkalmazása a teljes webhelyre ..	790
Témák alkalmazása oldalanként ..	791
A SkimID tulajdonság ..	791
Tartalomjegyzék	
Az Error esemény ..	741
A Web.config fájl szerkeze ..	742
Az ASP.NET Website Administration segedprogramja ..	745
Összefoglalás ..	746
32. ASP.NET-vezérlőelemek, -téma&szkék és -mesteroldalak ..	747
A webes vezérlőelemek visszakedésének megértese ..	747
Kiszolgálóoldali események kezelése ..	748
Az AutoPostBack tulajdonság ..	749
A System.Web.UI.HtmlControls típus ..	751
Vezérlőelemek fejlesztés ..	752
Vezérlőelemek dinamikus hozzáadása (es torlesse) ..	754
A System.Web.UI.WebControls.WebControls WebControl típus ..	755
Nehány szöveg a System.Web.UI.HtmlControls típusokról ..	756
Sokoldali ASP.NET-webelem készítése ..	759
Mesteroldalak használata ..	760
Az Inventory tartalomlap tervezése ..	770
A Build-a-Car tartalomlap tervezése ..	774
Rendelés es lapozás engedélyezése ..	774
Az elérhetőségi szintek szerepe ..	778
A RequiredFieldValidator vezérlőelem ..	780
A RegularExpressionValidator vezérlőelem ..	781
A RangeValidator vezérlőelem ..	782
A CompareValidator vezérlőelem ..	782
Ellenorzes összegzesenek leterhözesa ..	783
Ellenorzes összegzesenek vezérlőelem ..	785
Témák használata ..	787
A *Skin Fajlok ..	788
Témák alkalmazása a teljes webhelyre ..	790
Témák alkalmazása oldalanként ..	791
A SkimID tulajdonság ..	791

33. ASP.NET állapotkezelési technikák	799
Témák beállítása kódoló	793
Vezérzőelemek elhelyezése HTML-táblákkal	795
Összefoglalás	797
33. ASP.NET állapotkezelési technikák	799
Az állapot	799
Állapotkezelési módszerek az ASP.NET-ben	802
Az ASP.NET-nezetláppot szerepe	803
A nézetláppot bemutatása	804
Egyedi nézetláppot-adat hozzáadása	806
A Global.asax fájl szerepe	808
A visszglobális kiüvelekkel	810
A HttpApplication-sosztály	811
Az alkalmazások módosítása	812
Alkalmazások módosítása	813
Az alkalmazások kezelése	816
A webalkalmazások kezelése	818
Az alkalmazás-gyorsítótár használata	818
Az alkalmazások gyorsítótárazása	819
Az * .aspx fájl módosítása	822
Munkamenedzserek kezelése	824
A HttpSessionState további tagjai	828
A cookie	829
Sútitkrol	830
Bemeneti sútitk adattímak olvasása	831
A <SessionState> elem szerepe	832
Munkamenedzserek ASP.NET munkamennete	835
Állapotkezelőgalin	836
Munkamenedzserek tárолоса ASP.NET munkamennet	836
Az ASP.NET profil-API-ja	836
Munkamenedzserek tárолоса dedikált adatbázisban	835
Állapotkezelőgalin	836
Az ASP.NET profil-adatbázis	836
Felhasználói profil meghatározása a Web.config fájban	838
Profileddatok hozzáterése programoztatni	839
Profileddatok csoporthoztasa és egyedi objektumok tárолоса	843
Összefoglalás	845

A	A COM és a .NET együttműködése	849
A .NET	együttműködési képessége hatókörre	849
A .NET	és a COM együttműködésének egy szerű példája	851
A C#-ügyfélalkalmazás	együttműködési ellékesztése	852
Egy .NET	együttműködési szereleme	855
A futási időben	hívható burkoló	857
RCW:	a COM-trípusok mint .NET-trípusok	858
RCW:	klasszok referenciázásmájának kezelése	860
RCW:	alacsony szintű COM-interfeszek elréjítése	860
A COM	IDL szerepe	861
A VB	COM-kiszolgálóinkhoz generált IDL	863
Az IDL	attribútumok	864
Az IDL-könnyvtár-utastárs	865
A [default]	interfész szerepe	865
Az IDL	paraméterattribútumok	868
Az IDL	elérhetősége	869
Az IDL	szerepe	870
Bonyolultabb COM-Kiszolgálók	megfelelő COM-interfesz támogatása	871
Belső objektumok	együttműködési szereleme	872
Az együttműködési szereleme	szájat C#-Klassenakkalmazás készítése	874
COM-menedzserek	ellogására	877
A COM	es a .NET együttműködési képessége	879
A System	Runtíme InteropServices attribútumai	880
A CCW	szerepe	881
A.NET-osztály	interfész szerepe	882
Saját .NET-trípusok	készítése	884
Elosz nevű definíálása	888
A típuskönnyvtár letelezhéséa	és a .NET-trípusok regisztrálása	886
Az exportálási típus adatáinak a vizsgálata	887	
Visual Basic 6.0	tesztelés készítése	888
Osszejöglalás	908

8. rész: Függelékek

B Platfromfüggételek .NET-felisztes a Monoval	891
A .NET platformfüggételek termesztete.....	891
A CLI szerepe	892
A nepszerű CIL-dísztribúciók	894
A Mono hatóköré	895
A Mono konyvtárszerkezetének vizsgálata	898
A Mono beszerzése és telepítése	896
A Mono-felisztesek	668
A Microsoft-kompatibilis Mono-felisztesek	900
Mono-szigetílus felisztesek	901
A monoop(2) használata	902
.NET-alakmazások felisztese Monoval	903
Mono-kódkönyvvár felisztese	903
Egy név hozzárendelése a CoreLibDumper.dll	904
használatai	905
Szerelvények telepítése a Mono GAC-ba	906
Konzolalkalmazások felisztese Monoiban	907
Ügyfélalkalmazásunk betöltsése a Mono-OSzefoglalás	913
Windows Forms alkalmazásunk futtatása Linux alatt	911
Windows Forms ügyfélprogram készítése	909
futatások nyíezetbe	908
Tárgymutató	915
A szakmai lektorol	927
A szerzőről	928

Jölléhet egyedül az en névem jelenik meg a könnyű portfólión, de a munka soha nem készülhetet volna el jó néhány tethetőséges ember segítsége nélkül. A következőben azokhoz szólók, akik lehetségesek, hogy ezt a könnyvet megíjják.

Mindenekellett koszönheti azt a szolgáltatást, amely a kezirat működését elősegít. Mint minden más, az esetleges gápelesei hibákért vagy technikai hibákért felelős.

Végül, de nem utolsósorban, koszönni a munkáját, barátainak és tanácsadóinak, hogy ügy érzi, ez a kiadás sokkal színvonalasabb lett, mint az előzőek. Mindegy, hogy milyen technikai szolgáltatásokat használ a szerkesztő, aki annyi boldogságot ér el, hogy időre.

Külön koszönni a szerkesztőt, aki mindenekellett koszönheti azt a szolgáltatást, amely a kezirat működését elősegít. Mindegy, hogy milyen technikai szolgáltatásokat használ a szerkesztő, aki annyi boldogságot ér el, hogy időre.

Koszönetnyilvánítás

Bevezetés

Ahogy a korábbiakban megszokhattuk, ez a kódas is egy szertés erthetére nyelven ismerte a C# nyelvet és a .NET-alapozott alkonytárakat. Soha nem kerettem azokat a szaktíkokat, akik úgy azták elő mondanodjukat, hogy az jobban hasonlít egy GR-E-szövedetrol it tanulmányra, mint egy szakkonyvre. Munkám tövábbra is arra fekete a hangsúlyt, hogy a benne található írásbeli szövegekkel jól használható szoftverrendszereket lehessen készíteni, így nem töltök túl sok idejt az előteríkusz részletek tanulmányozásával, hiszen ezekkel csak nagyjón kevés olvasom fog valaha is foglalkozni.

Azota azon dologzatam, hogy a könnyvet frissítsem és aktualizáljam a .NET- platformról verzióinak megfelelően. Ekozben limitált darabszámmal megjelenítem egy speciális kiadávánnyal, amely bemutatta a .NET 3.0 verzióját (Windows Presentation Foundation, Windows Communication Foundation, Windows Workflow Foundation), valamint a többi kiadásra vonatkozó technológiáról, amelyek még meglehetősen elérhetők. A kiadásokban található legfontosabb változások magyarázatait, másrészt számos teljesen új fejezetet is tartalmaz, de ezek tulmenően több eddig is jelentőségen nem szerepeltek a könyvben. A könyvben minden részben a .NET 3.5-s verzióra épít, így a kiadásnak a következő részei nem érhetők el.

Mi egy csapatt vagyunk, az Olvasó és én

A szakirkok az emberérek igényeinek csoporthanak írnak (nekem min által csak tudom mire kell - hogy vagyok kozúlik). Tisztában vagyunk azzal, hogy egy szoftverrel rendkívül aprólekos munka, úgyanakkor nagyon specifikus a részleg, a cége, az ügyfél vagy a tárlykör szerint. Hiszen lehet, hogy őpp az elektronikus ki- és számos n-retegű rendszert, valamint projekteket a gyögyászati és a penzügyi ágasztokban. Majdnem nulla az esetére, hogy annak a kodnak, amelyet egy-ként mutatnak mindenhol. A magam részéről gyermekként oktatás szoftverrel fellesztem, legnék. En a tárlyk plattormával történjen is (NET, ZEE, COM stb). - Készítse - barmeiylik plattormával (NET, ZEE, COM stb). - rendkívül aprólekos munka, úgyanakkor nagyon specifikus a részleg, a cége, az ügyfél vagy a tárlykör szerint. Hiszen lehet, hogy őpp az elektronikus ki- és számos n-retegű rendszert, valamint projekteket a gyögyászati és a penzügyi ágasztokban. Majdnem nulla az esetére, hogy annak a kodnak, amelyet egy-ként mutatnak mindenhol. A magam részéről gyermekként oktatás szoftverrel fellesztem, legnék. Ezért szándékosan kerülöm az olyan mintapeldákot, amelyeket konkrét iparágba vagy programozási felelősséghez köthetünk: a C#, az objektumorientált programozás es a .NET 3.5 alaposztály konyvtárait ípari alkalmazások. Az en felelősséghez köthetünk, hogy melyikből tudásom szérint elmagyarázzam a C# prog- mindent megtessék azért, hogy elláttam a kedvezes Olivásoit különféle eszközökkel. Azt olvaso szolgáltatában. Persze egyszerűen tudásat kérők számára az alkalmazásokat ismeret. Bíztos lehet benne az Oliváso, hogy ha egyszer megérte a könnyebben bemutatott elveket, ismerteinek birtokában olyan szövetszabályt, hogy a könyv használhatók saját programozási körben.

1. fejezet: A NET filozófiaja

Az első rész bemutatja a .NET platform alapvető termések között a számos olyan fejlesztőszokozt (konzolik, több nyílt forrásokból), amelyeket .NET-alkalmazások fejlesztéséhez használhatunk. Mindeközben néhány alap C# programozási nyelvi részleteket és a .NET-típusrendszert is megismertedünk.

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

Elsö koret

Megjegyzés A magyar vallottaknak kereszténysége a szabadságban elköszöntő során a SZAK kiadás úgy döntött, hogy az eredeti konnyv tarthatmára két kötetben jeleníteti meg. Ez a viszonylag nagy terjedelűm indokolja. Az alábbiakban minden a két kötet tartalmára ismertetjük, így biztosítva, hogy az is kepett kapcsolatban mindenek csak az egyik kötetet kerüli a kezébe. (Ezt a Bevezetésben szerepeltejük.)

kialdas a logikaiakban nyolc különálló részben áll, s ezek minden egyike tövábbi fejlesztéket támogat. Jo néhány témaikor, például az alapvető C#-szerekzetek, az objektumorientált programozás és a platformfüggetlen .NET felhasználókhoz kötött a korábbiakhoz képest, saját fejlesztet kapott. Az új kiadás a .NET 3.0-3.5-vitillet a korábbiakhoz képest, saját fejlesztetet kapott. Az új kiadás a .NET 3.0-3.5-programozás tulajdonaságait (Linq, WCF, WPF, WF stb.) összefoglaló fejezetet tartalmaz.

A konyv áttekintése

A kohyv' aktiekjhutese

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

Előbbének a teljesítésben kezdetjük el részletekben tanulmányozni a C# programozási nyelvet. Megismerekedünk a Main() metódus szerkezetével és számos részlettel a.NET platform belső adattípusai val körül. Később a kapcsolatban, beleértve a szövegesadat-kezelést a system.String osztálytól. Továbbá az interakciós és a döntései szereketeket, a szöveg-szavakat. Megvizsgáljuk tövábbá az text. Stringből induló nemrég elhalászatát.

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

A 2. részben taragyalt temakörök kiemeltekn öntössük, hiszen az itt megjelenő tudnivalókat folyamatosan használjuk függelékeni attól, hogy milyen jellegrű alkalmazás, forrásokkal könnyítenek, Windows-szolgáltatások stb.). Itt derül ki, hogy hogyan működnek C# nyelvű alapvető eszközökkel, beleértve az objektumorientált programozás (OOP) részleteit is. Ez a rész ismerteti továbbá, hogy hogyan kell kezelni a futásidőt kivételeket, és bevezet miatt a .NET személygyűjtő szolgáltatásnak részleteibe is.

2. rész: A C# alapvető epítőelemei

A legezetű céjük, hogy különöte ezeket a szabványokat és technikákat ismertetésvel bővezzék sen minket a C#-forrásokkal foglalkozókat. Először a paraméterekről beszélünk, majd részben pedig számos forrásrészletet ismertetünk. Ezután többek között a környezetekről beszélünk, amelyet baromecyt (NET-fel-) oljának nevezünk azonban elő kell tudnia húzni a fárasztékból.

2. fejezet: C#-alkalmazások készítése

Ezután a résznekek az utolsó feljegyzésekkel szembenek a CLR a memória által. A .NET személyiségeknek a segítségevel. Megérthetők az alkalmazásokre, az objektummezőknek és a szabványoknak. Az alapok megismerése után, a feljegyzet többébbi része az elbőlhető objektumokkal (az időspórolható interfészben keresztül) és a véglegesítő folyamatokkal (a szövegben említett objektumokkal) foglalkozik.

8. fejezet: Az objektumok elektiklusa

Ez a lejelzett elbesorbaan azt mutatja be, mit kell tenni a stukkutrális kivételekhez zelés segrítésével a futási időben keletkezett anomáliaikkal. Nemcsak az ilyen jellelégű problémák megoldására szolgáló C#-kulcsszavakat ismerjük itt meg (try, catch, throw és finality), hanem az alkalmazászintű és rendszerszintű kivételek kozzat különbségeit is. Megismernéhetünk többébb a többfélé, a Visual Studio 2008-ban található eszközöt, amelyek segítségevel hibakeresést végezhetünk a fogyelmen kívül hagyott kivételek kozzat.

7. fejezet: Strukturált hibakezelés

Ebben a részben az OOP maradek pilleret (származtatás és polimorfizmus) vizsgáljuk meg; ezek a segítségekkel összehozhatók osztálytípusok családját hozzájuk letre. Megnezzük a virtuális metódusok és absztrakt metódusok szerepét. Végül, de nem utolsó sorban, ez a fejezet ismerteti a .NET platform legfontosabb alaposszabalyának, a system-object osztálynak a szerepét is.

6. fejezet: A szarmaztatás és a Polimorfizmus

Bbben a legezetben az objektumorientált programozás (OOP) ismerekedhetünk meg a C# programozási nyelv használatán keresztül. Először megírhatunk az OOP pillerét (bólgyázás, származtatás és polimorfizmus), majd a fejezet tövábbi részeiben azt nézzük meg, hogyan kell konstruktorok, tulajdonságok, stakkus tagvaltozók, konstansok és csak olvasható fajlok használatával robustosítjuk a típusosokat létrehozni. Végezetül ismertetjük a részle-geket típus definiciját és a C# XML-kódjának dokumentációs szintaktikáját.

5. fejezet: Egyésgébe Zárt osztálytípusok definiálása

metodusreferenciák, a névtelen metodusok és a lambda-kifejezések között. megvizsgáljuk a C# 2008 lambda-operatort (\Rightarrow), és feltártuk a kapcsolatot a szerűsítők a nyers metodusreferenciával programozásnak kezelését. Végezetül ismerkedünk a .NET event kulcsszavával, amelyet arra használunk, hogy egy-együttműködés segítségével. A .NET-metodusreferenciák vizsgálata utan még hétfővén, amelyek több objektum osszékapcsolását engedik meg kérirányú mazás más metodusára mutat. Ez a mintha olyan rendszerek készítésére tetszi le-ledgezésről: a .NET-metodusreferenciá olyan objektum, amely az alkalmazásban más metodusának része.

A 11. fejezetek az a célja, hogy erthetővé tegye a metodusreferencia-típusokat.

11. fejezet Metodusreferenciák, események és lambda

Ez a fejezet a system.Collections.Generic névterügyütemény típusainak tanulmányozása. Noha a .NET 2.0 meglehetősen ota a C# programozási nyelv lehetővé tetszi generikus típusok használatát, ahogy a kezdetekben kidért, a generikus programozás lényegesen megújvári az alkalmazás teljesítményét és a típusbiztonságát. Nem-csak azt vizsgáljuk meg, hogy milyen különbszöző generikus típusok leteznek a számára, de azt is, hogyan készíthetjük el saját generikus metodusainkat és típusainkat (megszorításokkal vagy átneleltük).

10. fejezet: Gyűjtemények és generikus típusok

Ez a fejezet a system.Collections névterülyütemény típusainak tanulmányozása. Ez a fejezet a szabályozók, ezek már a .NET platform kezdeti kiadásának részei voltak. Igenek a fejezetek az anyagát az interfészalapú programozást is magába fog-
laló objektumalapú részesítés alkotta. Megismertetőként belölle, hogy hogyan kell olyan típusokat definiálni, amelyek többesztős viselkedést támogathatnak, hogyan lehet ezeket a viselkedéseket futáridőben feltérképezni, és hogyan lehet szellemítően elérni bizonyos viselkedéseket az explicit interfészmezőkkel. Továbbá a számos előre definált .NET-interfész típus megtalálható, amelyek ad hoc jellegrő eseményarchitektúra készítéséhez szükségesek.

9. fejezet: Interfészek használata

A könnyv 3. része ismerte a C# nyelv haladóbb jellegrő (de ugyancsak na-
gyon fontos) elvét. Az interfészek es metodusreferenciák vizsgálataval befe-
jezzük a .NET típusrendszerével való ismerkedést. A továbbiakban betekint-
teszt nyerhetünk a generikus típusok es a számos új C# 2008 nyelvi elem szerető-
pébe, valamint a LINQ-ba is.

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

A negyedik rész a NET-szerelvényformárium részleteit taglalja. Nemcsak azt tanulhatjuk meg, hogyan kell.NET-kódokon kívül raktatni a telepíténi és konfigurálni, de azt is, hogy mi a.NET bináris kód belső összeállítása. Ebben a részben lesz szó a.NET-attribútumok szerkezetéről és a többszállitási alkalmazások letrehozásárólnak modjáról is. A kesőbbi fejezetek néhány haladó témastrétnéknél, például az objektumkörnyezetet, a köztes nyelvi kódot és a dinamikus szerelemeinek.

4. rész: Programozás .NET-szerelvénnyekkel

14. fejezet: Bevezetés a nyelvbe ágyazott lekérdezésekbe (Linq)

A .NET 3.5 kiadásában a C# nyelv számos új programozási eszközöt bonyolt, amelyek közül jó néhány arra szolgál, hogy lehetővé tegye a LINQ API használatát (erről a 14. fejezetben többet is megtudhatunk). Megismerjük a lokális szállítmányozási szintaxis használatát.

13. fejézet: C# 2008 nyelvi újdonságai

12. fejezet: Indexterek, operátorok és mutatók

M8. fejezet: Többszálu alkalmazások készítése

A szerevbenyekről szórol tudnivalok után ez a fejezet a betöltötők NET végrehajtásáról felírja a felépítését ismerteti. Az elrendelések cél a folyamatok, az alkalmazások működésének megtervezése. A következő fejezet témáját: a többszállú alkalmazások kezeltetésének modelljeit.

17. fejezet: Folyamatoik, alkalmazásokat tömörítik és objektumkörnyezetek

16. fejezet: Típusreftleksió, keszítő körtes es attributummalapú programozás

15. fejezet: A .NET-szerelvénnyek

A system, ío nevter segeslegével lehetőségeink van a számtalogszerű környezetekkel programoztan könnyvtárszerkezetek elérésre. Ebben a felületben minden megismertük, hogyan tudunk a kodhozazzáírás-szabályozásrol (Code Access Security - CAS).

20. fejezet: Fajlműveletek és elküldönötött tárrolás

Hála elérkezettünk ehhez a részhez, akkor már biztos tudásunk van a C# nyelvrol és a.NET-szerelvényformátumok részleteiről. Az ottoldik rész tövábbfejleszti ezeket az ismereteket azaztálal, hogy számos általánosan használt alapossztályból szolgáltatást mutat be, többek között a `File` I/O-t és az adatbázisréseket azADO.NET segítségével. Ez a rész tárnyalja továbbba az előzőt alkalmazások szégeivel, valamint azokat a mutatófolyamat-engedélyezett alkalmazások készítésével, amelyek a Windows Workflow Foundation (WF) API-t használják.

5. rész: Bevezetés a .NET alaposztálykoniytáraiba

Masodik kötet

A. rész utolsó részében szerepeltek a korábban részletezettben megvizsgáltuk CIL szintaktikáját és szemantikáját, majd a system. Reflection. A rész utolsó részében szerepeltek kétféle. Elsőször a korábban részletezettben nememoriában definiáltunk és futtatunk, dinamikus szerelvénnyeket nevezünk.

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

Az asztalról ülőknek hívásokat. Majd megvizsgáljuk a számrendszerben található típusokat. Ezek között számos olyan van (Thread, Threadstart), amelynek segítségevel könnyen lehet újabb végrehajtásit szállítani. Végül megszabadítunk a BackGroundWorker tippsszel, amely nagyban megkönnyíti a felhasználói felülettel történő szálkezelést.

21. fejezet: Bevezetés az objektumsortisztas világába

Bevezetés

Ez a fejezet a .NET-platform objektumsortiszt szolgáltatásába nyújt betekintést. Egyesületen szolva a sortiszták segítségevel lehetőségeink nyílik arra, hogy egy objektum (vagy a kapcsolódó objektumok egy halmazának) állapotát egy test. Egyesületen szolva a sortiszták segítségevel lehetőségeink nyílik arra, hogy objektumot a jövőműl, és a memoriában az alkalmazás számára használható szerepet, valamint azt, hogy hogyan kell a relációs adatbázissal komunikálni. Az adatbázisokról szóló két fejezet közül az elsőben az ADO.NET programozási API-ról kapott ismereteket. Ez a rész bevezeti a .NET adatszolgáltatásokat az objektumsortiszt körül ismeretlenek. Az adatbázisokról szóló két fejezet közül az másodikban történő manipulálási fejezetet: ADO.NET 1. rész: Az elő kapcsolat.

22. fejezet: ADO.NET 1. rész: Az elő kapcsolat

Az adatbázisokról szóló két fejezet közül az elsőben az ADO.NET programozási API-ról kapott ismereteket. Ez a rész bevezeti a .NET adatszolgáltatásokat az objektumsortiszt körül ismeretlenek. Az adatbázisokról szóló két fejezet közül az másodikban történő manipulálási fejezetet: ADO.NET 2. rész: A bontott kapcsolat.

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

Ez a fejezet az adatbázis-közlelés különöző módokon történő manipulálási lehetőségeit tárgyalja tovább, jelen esetben az ADO.NET bontott kapcsolat API-vel. Ez a rész tipus az adattípusokkal kapcsolatos részben részletezi a Windows Forms fejilletet elemekhez kapcsolni, olyanokhoz, mint például a Visual Studio 2008 környezet funkcióit.

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-lekérdezésekkel alkalmazni relációs adatbázisokon, dataset-objektumokon és XML-dokumentumokon. Mindeközben megtanuljuk az adattípusokat és megvizsgáljuk a Visual Studio 2008 környezet funkcióit.

27. fejezet: Windows Forms-programozás

A látálosan hibás lehettelezés a .NET platform körülönbsége az, hogy ez a kezretendész kiürölve a webalapú felhasználói felületekkel foglalkozik (ez fehetően a „NET” kifejezés miatt van, hiszen felidézi az „Intermet” szót, melyet a „weboprogramokat”). A .NET kezretendész modon támogatja a webalapú kalmazások készítését, a könnyű ezek része azonban a hagyományos felhasználói felületekre koncentrál, két grafikus felület kezretendészéről, a Windows alkalmazások készítését, a könnyű ezek része azonban a hagyományos felhasználásra, a „weboprogramokat”. A .NET kezretendész modon támogatja a webalapú kalmazások készítését, a könnyű ezek része azonban a hagyományos felhasználásra, a „weboprogramokat”.

6. rész: Felhasználói felületek

A.NET 3.0 a WCF mellett bevezetve a WF API-t is, amely lehetőséget ad arra, hogy munkafolyamatokat definíáljunk, futtassunk és monitorozzunk. Ennek ellenére a munkafolyamatokat tervezőnek, a munkafolyamat futtatási motornak, és a munkafolyamat tervezőnek, ahogy azt is, hogy mi a szerepe az aktivitásoknak, a munkafolyamok celjai, a munkafolyamat futtatási motorok, és a munkafolyamok modelljei között. Megtakarítva a WF segítségével komplex üzleti folyamatok modellezhetők. Még tanuljuk a WF használatát, aholgy munkafolyamatokat definíáljunk, futtassunk és monitorozzunk.

26. fejzett: A Windows Workflow Foundation – Bevezetés

arrá, hogy szimmetrikus módon, függetlenül az alatta található relégek fel-
építésétől elosztott alkalmazásról leírunk. Ez a részlet feltárja a WC-F-szo-
lgáltatások, -gázágépek és -klinensek kezelésének lehetségeit. A WC-F-szo-
lgáltatások rendkívül flexibilisek, a gazdagépek es a klinensek XML-alapú kon-
figurációk felhasználhatók, hogy deklaráti módon adjanak meg ci-
meket, kötéseket és szereződéseket.

25. fejezet: A WCF

30. fejezet: WPF 2D grafikus renderelés, erőforrások és temák

Ebben a felzérben megtudjuk, hogy hogyan kell dolgozni a WP-felzérőt. Elém taralommodelljevel, és eritink számos vezérlőlemekekkel kapcsolatos témával is, például a függőségi tulajdonságokat és irányított eseményeket. Jo nehaian WP-felzéről elém használata bemutatjuk, s szintén ebben a fejezetben magyarázzuk el az előrendezéskezelő, -vezérlő utasítások és a WP-feladatokat.

29. fejezet: Programozás WPF-vezérlőelemekkel

A NFT 3.0 teljesen új grafikus felhasználói felületeit vezetőt bő, amelyet WPF-nek nevezett el. Röviden, a WPF kimagaslóan interaktív es mediasban gazdag front endnek kezeltetését teszi lehetővé azáltal alkalmazásokhoz (és indirekt módon webalkalmazásokhoz). A Windows Formsszal ellentében, ez a tulajdonság felhasználófellelhetőséget kínálja számos külcsfonosságú szolgáltatást (2D-s és 3D-s grafika, animációk, gazdag dokumentumok stb.). Intégrál egyszerűen WPF-fel és kiterjeszthető alkalmazásjelölő nyelvvel (Extendable Application Markup Language - XAML). Megtakarítja, hogyán kell XAML-mintes, csak XAML-t használó es a kettő kombinációjából felépülő WPF-programkort lehet horizontálisan. A fejezet végén készítünk egy egységesi XAML-nézetegrot, amelyet a további WPF-fel foglalkozó fejezetekben is használhati fogunk.

28. fejzett: A WPF és az XAML

[Web.config Settings](#)

33. fejezet: ASP.NET állapotkezelési technikák

Ez a feljezet azokkal a vezérloelemekkel foglalkozik, amelyek a belső vezérlésről, fájlat töltésről adatokkal. Megismertük az alapvető ASP.NET webs vezérlőelemeket, többek között aellenörzö vezérlőelemet, a belső oldalai navigációs vezérlőelemet, és a különösen adatkötő vezérlőelemet. Ügyancsak ennek a részletben leírunk a feladatait, hogyan mutassa meg a masteroldalak es az ASP.NET temamotorja.

32. fejjezet: ASP.NET-vezérlőelemek, -témaik és -mesteroldalak

Ez a részlet vezet be benneinket az ASP.NET használatával történő webes alkalmazásfejlesztésbe. A kiszolgálóoldali szkripteket felváltók az igazi objektumorientált nyelvök (mint a C#, VB.NET és hasonlók). Bemutatjuk egy ASP.NET weboldal készítését, valamint az alatta található programozási modellet és az ASP.NET egypélyű kulcsfontosságú elvét, például azt, hogy hogyan kell wékízsolgálni a web. Ez a részlet folytatását a következő fejezetben írunk.

31. fejezet: ASP.NET weboldalak készítése

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

7. rész: Webes alkalmazások felépítése ASP.NET

B függelék: Platformfüggelén .NET-fejlesztes a Monoval

Azok, akik már programoztak Windows operációs rendszereken kívül ismerik a COM-t (Component Object Model) és a .NET-t (Platform). Noha a COM-nak és a .NET-nek semmi közös egymáshoz nem kötődik, mégis a .NET-alkalmazásai kompatibilisek a COM-al. Ez azt jelenti, hogy a .NET-alkalmazásoknak a COM-komponensekkel szemben használhatatlanul kell működniük. Ez a lehetőség lehetővé teszi, hogy a .NET-alkalmazásoknak a Windows operációs rendszereken kívül is használhatók.

A függelék: A COM és a .NET együttműködése

8. rész: Függelékek

Forráskód Ez a forráskód-megjelölés egy kiválasztott környzettára hivatkozik.

Fényelem: a felzértekbén a Forrásokkal meglélesek, mint amilyen az alábbi 2008-ba további vizsgaloldások es módszertárosk célfájához:

A könnyben szereplő összes források-miniatűpedala (az ot szabadon letölthető törvábbi fejlesztében levők is) szabadon és azonnal elérhető az Áprész honlapjának Source Code/Download részlegérol. Egyszerűen گépeljük be a http://www.apress.com címét, válasszuk ki A Source Code/Download linket, és keressük meg a megfelelő címet, és, töltük le az önmagát kicsomagolt zip fájlt. A forrásokodat fejlesztenkemt osztottuk fel.

A környéki forrásokról jól van igénylésre

AZ itt található linkek kattintva digitális formában letelethetük a konvéniai további fejlesztéti, ha sikeresítő válaszolni a környezetek csak angol nyelven érhetők el - a valamennyi szöveghez a teljesen meglévő magyar fordítás is elérhető.

A könnyű jelen kiadása nem tarthatmazza ezt az öt lejáratet. Elírni az a leg-több oka, hogy a .NET 3.0-ban a WCF es a WPF API-k rendte a .NET-remo-tin/g/XML-webszolgáltatások és a Windows Forms API-k tökötéi. Ha me-lyebben szeretnék elmeríteni a Windows Forms világában (azon tul), amit a 27. fejezet nyújtani tud), vagy meg akarjuk nezni, hogyan kell használni a (öröksegtől hagyott). NET-remotingsot es az XML-webszolgáltatás-API-t, egy-szervezőn csak meg kell nezni az Press honlapján a megfelelő részről.

Azoknak, akiknek a 33 tétezet es a két tűzegelék nem lenne elég, szabadon le-tölthetek további öt fejezetet. A könnyű karábbi verziót harom olyan fejezetet tartalmaztak, amelyeket a Windows Forms felületeknek szenteltünk (egyedi vezérlők vizsgálatával egyetemben), egy másik fejezetet a .NET remoting rendszereknél (systrém, RunTime, Remoting és trásai), és a végül egy fejezet a hagyományos XML-wébszolgáltatások kezeltével foglalkozott az ASP.NET Web Service projekt sablonjainak segítségével.

Ot szabadon letoölhető fejjezet – még több információ

Andrew Troelsen
Minden jöt!

Nos, akkor, koszönöm, hogy megvásárolta a környezetet (vagy legalábbis, gy-e-e). Remélem, elvezeti fogja a kedves Olvasó a tanulmányozását, és hasz- hagy belenek a környezetsabotban, miközben azon gondolkodik, hogy megeve- nosztani tudja az újonnan szerezett ismereteket.

Ha gyűjteményt szeretne kiírni, akkor ez nem jelenti azt, hogy nem akarok valasztozok egy-két heten belül, akkor ez nem jelenti azt, hogy nem szem van, valahol épí nyaralok).

Ammak elleneré, hogy megpróbálunk minden levélre lehetsége szerint vala- szovágejt: atrolesten@intertech.com.

Ha barmilyen kérdés merül fel a környezőtől tartozó forrásokkal kapcsol- latban, vagy szíksege van valamelyik példa részletei, vagy egyszerűen lehetségeit szeretné nyilvántartani a .NET platformról, nyugodtan küldjön casak vissza!

Ha bármihez közelítünk, vagy a környezetet értékelünk, amelyen értesítést tudunk,

Elerhetőségem

Ekképzelhető, hogy a környezetet végigolvassa során nyelvtani vagy forrásokba- hibákat vél feldezzeti a kedves Olvasó, bár reméltem, hogy erre nem kerül sor. Ha mégis, elmezeit kérlek. Az aktuális hibálistát az Appress weboldalról lehet igényelni (amely szintén a környezetet támogató megegyezése).

A lehetséges javítások

Megjegyzés A magyar nyelvű változatban a forrásoknak megjegyzéseit lefordítottuk. A lefordít- to forrásoknak fordítása nem állt módunkban, így azok teljesen angol nyelvűek. Ilyen módon in- kább megfelelnék a lefordított kódok használatát C#ban – a SZAK kiadó megegyezése.

Egy személyes nyissuk meg a *.sln fájlt a megfelelő alkonyvatartábol. Ha nem használjuk a Visual Studio 2008-at (lásd a 2. fejezetet többi integrált fejlesz- tői környezetekről), akkor manuálisan töltük be a felkínált forrásokfájlokat a valasztot fejlesztfi eszközbe.

könyvtáraiba

alaposztály-

a.NET

Bevezetés

5. rész

kat további rendszerhez azonban használhatók.

lelmézes szereint minden rendszervére elérhetők, így az itt említett típusok a számítógép memória (RAM) és a processzor (CPU) teljesítménye, a rendszerek stabilitása, a rendszerek által kínált funkciók és a rendszerek által kínált felhasználói felületek minősége.

A rendszerek stabilitását többféle módon javíthatók:

- 1. A rendszerek frissítése: minden rendszernél fontos, hogy a rendszerek legújabb frissítésekkel legyenek ellátva.
- 2. A rendszerek optimalizálása: minden rendszernél fontos, hogy a rendszerek teljesítményét optimalizálva legyenek.
- 3. A rendszerek telepítése: minden rendszernél fontos, hogy a rendszerek helyesleg telepítve legyenek.
- 4. A rendszerek konfigurációja: minden rendszernél fontos, hogy a rendszerek konfigurációja megfeleljen a rendszerek használatahoz.

A System.IO névter

A CAS-t gyakran az elszigetelt tárrolóval együtt használjuk.

(Code Access Security, Kódrendszer-alapú biztonság) kell megvizsgálnunk.

mutatásaihoz először a .NET platform egyik biztonsági megaladását, a CAS-t sajnosan, korlátolt száműzetéssel hozhatják le. Ezáltal az API-nak a be-nyezetben friss alkalmazások a felhasználói es az alkalmazásadatokat bizton-tedstorage névteren keresztül. Ennek segítségével a szigorúbb biztonsági kor-sok segítségével, bennük az elszigetelt tárroló használata (a rendszer I/O tárrolásán megismertük a fájlak és könyvtárak kezelését az alapvető I/O tárrolásban).

Miután megismertük a fájlak és könyvtárak kezelését az alapvető I/O tárrolásban,

szintén a rendszerek adattárolók olvasásával és írásával foglalkozunk.

számítógép mappa - a fájlstruktúráját, majd a különöző karakter-, bináris,

állományok, és megvásárlási, hogy hogyan módosították programoztatni a rendszerek. Elsőként megismerteink a rendszerek definíciót alapvető makróit. Elsőként megismerteink a rendszerek mutatóit, az OS-kapcsolatot te-

.NET rendszerek szemzéséből mutat be számos, I/O-val kapcsolatos tárrolásban.

Idegen elérési lehetőségek a rendszerek munkamenedzser funkcióit. Ez a rendszer a rendszerek erettségi szintjére alakíthatását készítse ki, hogy a program kérés

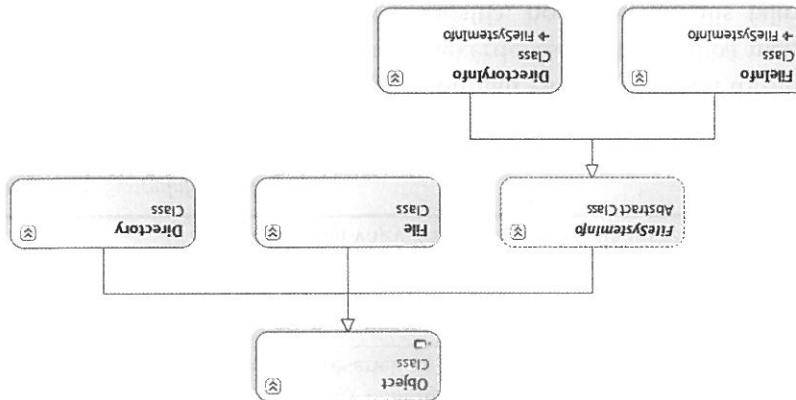
Fájlműveletek és elszigetelt tárrolás

H U S Z A D I K F E J E Z E T

Nem abstrakt /O osztálytipusok	Jelenlés	BinaryReader, BinarWritter	Primítiív adattípusok (egész számok, logikai értékek, sztringek stb.), bináris értékek történetű tárolását és visszaolvasását teszik lehe írve.
Buffereďstream		DIrectory, DIrectoryInfo	Ezekkel a számítógép meghatározott szolgáltat részleteit információkban.
DriveInfo		FÍle, FÍleInfo	A számítógép meghajtójáról szolgáltat részleteit információkban.
MemoryStream		FÍlestream	Velelén fajlhozszállítmányt (pl. keresési képesességek) tesz lehe írve, és az adatokat byte-folyamként jeleníti meg.
FÍlesystemwatcher		FÍlesystemwatcher	Egy megaladott környvétában található kijelölt mappákban folyamhoz (helyi példig egy fizikai fájlhoz).
Path		MemoryStream	Velelén hozzáérhetőbbíti a memoriában tárolt adat-
Stream		Olyan System, string típusú objektumokon hajt végre környvtárelérést utakról tartalmaznak információkat.	
StreamReader		Ezeken folyamhoz (helyi példig egy fizikai fájlhoz).	
StreamWriter		MemoryStream	Ölyan System, string típusú objektumokon hajt végre műveleteket, amelyek plátformájúggelten fájl-, illetve folyamhoz (helyi példig egy fizikai fájlhoz).
TextReader		Path	Ezeken folyamhoz (helyi példig egy fizikai fájlhoz).
TextWriter		StreamReader	Velelén hozzáérhetőbbíti a memoriában tárolt adat-
XmlTextReader		TextReader	Ölyan System, string típusú objektumokon hajt végre környvtárelérést (illetve folyamként jeleníti meg).
XmlTextWriter		TextWriter	Egy megaladott környvétában található kijelölt mappákban folyamhoz (helyi példig egy fizikai fájlhoz).

A system, ío neveter több típusa a hizkai mappek es fajosik programozott kezelésre összponosztí. Vanmak azonban további típusok, amelyek a sztring- és mérőszáma manipulációkat támogatják. A 20.1. táblázat bemutatja a sztring- és mérték (hem absztrakt) osztályait.

20.1. ábra: A File- és Directory-képzőponti típusok



Iyok az absztrakt FileInfo típusból származnak.

A System.Object osztályt bővíti ki, míg a DirectoryInfo és FileInfo osztályt foglalja. A 20.1. ábrán látható, hogy a DirectoryInfo és File típusok közvetlenül eredetivé példányozásnál metódusokkal (így ezeket a new kulcsszóval le kell kapcsolni) a FileInfo osztályhoz tartozó statikus tagokon keresztül közvetlenül bázisoltanak. Az ezekhez szorosan tömök letelezhését, törlést, másolását és mozgatását. Az elso két típus, a DirectoryInfo és a File, különöző statikus tagokon keresztül bázisoltanak. A konnyvártartukról ismerjük a felületet. Az elso két típus, a DirectoryInfo osztályt különöző statikus tagjaihoz közelítünk a kezdőkörből. A System.IO névterrel negy típus segítségével tesszi lehetővé a számlálókép-

A DirectoryInfo és FileInfo típusok

Az imént felsorolt konkrét osztálytípusok mellett a system. IO több féloldalt származott számára. A kezdeti körben számos ilyen típusot bemutatunk. Az StreamReader osztályt Stream, StreamReader, TextWriter stb.) is definiált, amelyek egy megsosztott polimorf interfeszst definálnak az osszes leírást. Az ilyan absztrakt osztályt (Stream, StreamReader, TextWriter stb.) is definiált, amelyek egy megsosztott polimorf interfeszst definálnak az osszes leírást. Ez az absztrakt osztály minden típusnak a kezdeti fájlhoz, hanem

20.1. táblázat: A System.IO névter külcsfonsosságú tagjai

Nem absztrakt	I/O osztálytípusok	Jelentés
StreamReader, StreamWriter,	A StreamReader és StreamWriter ter típusokhoz hasonló -	sztringpufferek.
StringReader,	nék, hárterratorolik azonban nem fizikai fájlok, hanem	an ezek az osztályok is szöveges információkat kezel-

A FileSysteinfo osztály definíciója a Delete() metódust is. Ez a származtatott típusok valósítják meg azért, hogy egy adott fájl vagy konnyvtárat törleni lehessen a merevlemezről. Az attribútumok lekérdezése elöl tövábbá még- (vagy konnyvtárra) vonatkozó statisztikák ne legyenek elavultak.

20.2. táblázat: A FileSysteinfo tulajdonságai

Tulajdonság	Jelelhetős
Name	Az aktuális fájl vagy konnyvtár neveit adja vissza.
LastWriteTime	Visszaadja vagy beállítja az aktuális fájl vagy konnyvtár utolsó módosításának az idejét.
LastAccessTime	Visszaadja vagy beállítja az aktuális fájl vagy konnyvtár legutolsó előresemeket az időpontról.
FullName	Visszaadja a fájl vagy konnyvtár teljes előrei utvonalaát.
Extension	Visszaadja a fájl kiterjesztését.
Exists	Segítségevel megláthatók, hogy egy adott fájl vagy konnyvtár létezik-e.
CreationTime	Visszaadja vagy beállítja az aktuális fájl vagy konnyvtár létrehozásának az idejét.
Attributes	Visszaadja vagy beállítja az aktuális fájlnak azokat az attribútumait, amelyeket a Filetributus felisrolás ír le.
Jelentés	Tablázat felisrolja az erdekesebb alap tulajdonságokat.

A direktoriinfo és a Fileinfo típusok az absztrakt FileSysteinfo osztálytól öröklök számos képeségüket. Legnagyobbreszt arra használjuk FileSysteinfo osztály tagjait, hogy lekérdezzük egy fájl vagy konnyvtár attribútumait, amelyeket a Filetributus felisrolás ír le. A direktoriinfo és a Fileinfo osztály tagjai, hogy lekérdezzük egy fájl vagy konnyvtár attribútumokat stb.). A 20.2. táblázat felisrolja az erdekesebb alap tulajdonságokat.

Az absztrakt FileSysteinfo osztály

Altalanosságban elmondhatunk, hogy a Fileinfo és a direktoriinfo osztály inkább egy szerű sztrukturáltan kezeli terneke vissza.

ben a direktori és a File osztályok tagjai erősen töröknek helyett ugyanis ezek tagjai erősen típusos objektumokat adnak vissza. Ezzel szem-máccsal (pl. letrehozás ideje, működési/olvashatóság stb.) van szükségtük, lyokat célszerűbb használni, ha egy fájrról vagy konnyvtárról részletes információkat szeretnénk, ha például a direktoriinfo osztálytól

A második példában azt feltételezzük, hogy a konstruktorokat átadott átveatal (C:\Windows) már letérzik a számítógépen. Ha nem letérzik konvertátoroknak (meg, eloszor még kell hívunk a Create() metódust):

```
// A C:\Windows konvertárhoz kötés,
// // Az aktuális konvertárhoz kötés.
// DirectorInfo dir1 = new DirectorInfo("C:\Windows");
// szözerinti szerkezetet használataval.
```

A DirectorInfo típus használatahoz az előreki ütvonalat kell megadnunk konstruktorparaméterként. Ha az aktuális konvertárhoz (pl. a futó alkalmazásban bemutatunk néhány példát):

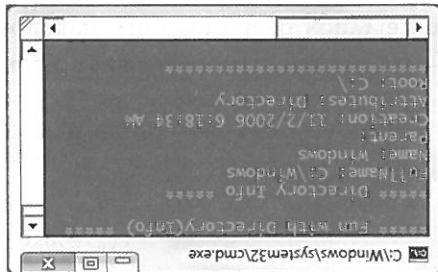
20.3. tablázat: A DirectorInfo típus kulcsfunkciói tagjai

Tag	Jelentés
Root	Megkaphja az előreki ütvonal gyökerét.
Parent	Az adott előreki ütvonal gyökerének szülőkonvertárat adja vissza.
MoveTo()	A konvertárat és a taratlanat új előreki ütvonalat álla mozgatja.
GetFiles()	Eltérő típusok tömbjevel teríssza, amely az adott konvertárat felfogható tartalmazzák.
GetDirectories()	Visszaad egyszerűen az aktuális konvertártosszák alkalmazára nevezett tartalmazó sztringtömböt.
Delete()	Egy adott konvertártól lesz teljes tartalmának a törlése.
CreateSubdirectory()	Egy konvertár (vagy több alkönvertár) létrehozása a megalapozott ütvonalon.
Create()	Az előző, I/O műveletekkel kapcsolatos típus, amelyet közélebbrol megvizsgálunk, a DirectorInfo osztály. Tagjainak a segítségevel létrehozhatunk, át helyezhetünk, torolhatunk és kilitszthetünk konvertáratokat és alkönvertákat. Az összefüggésben, a DirectorInfo osztály tagjai közül a Create() funkcióval mindenkorán felülbírálható (FileSystemInfo) objektum funkcióitársainon til a DirectorInfo osztályától (FileSystemInfo) eredően a DirectorInfo osztály által rendelkezett funkciókat.

Az első, I/O műveletekkel kapcsolatos típus, amelyet közélebbrol megvizsgáltuk, a DirectorInfo osztály. Tagjainak a segítségevel létrehozhatunk, át helyezhetünk, torolhatunk és kilitszthetünk konvertáratokat és alkönvertákat. Ez a 20.3. tablázatban feltüntetett kulcsfunkciók tagokkal rendelkezik.

A DirectorInfo típus használata

20.2. ábra: Információk megléntese a Windows-környezetben



```

    static void Main(string[] args)
    {
        Class Program
        {
            static void Main(string[] args)
            {
                Console.WriteLine("***** Fun with Directory(Info) *****\n");
                ShowWindowsDirectoryInfo();
                Console.WriteLine("***** Run with Directory(Info) *****\n");
                Console.ReadLine();
            }
        }
    }
}

```

Mután letéhetők a Directorийnfo objektumot, megvizsgálhatók a konviktusokat, tarthatmálat bárminely, a FII rendszertől függetlenül, tipikusan a korábbi tulajdonos által. Például hozunk letegy új, directoryapp nevű konzolalkalmazást. A program osztályt egészítünk ki egypty új statikus metodusossal, amely leterhezozza a szövegeket modositásuk az utvonalában), amely számos erdekes statisztikát olajan új direktoriyhho objektumot a C:\Windows Konviktura leképzésre (szükséges esetén módszert az utvonalat), amely számos erdekes statisztikát jelenít meg (a kiemelte a 20.2. ábrán látható):

```
    ////////////////////////////////////////////////////////////////////  
    // Kossukk hozza egy nem Letezzo konvitarhoz, majd hozzuk letre.  
    ////////////////////////////////////////////////////////////////////  
    new DirectoryInfo(dr3).Create();  
    dr3 = new DirectoryInfo(@"C:\MyCode\Testing");
```

Ha futtak az alkalmazás, akkor a 20.3. ábrához hasonló lista tár fel a termékek.

```

static void displayImageFiles()
{
    DirectoryInfo dir = new DirectoryInfo("C:\Windows\Web\WallPaper");
    foreach (FileInfo imageFile in dir.GetFiles("*.jpg"))
    {
        Console.WriteLine("Found {0} *.jpg files", imageFiles.Length);
        // Menyita talaatunk
        Console.WriteLine("File name: {0}", f.Name);
        Console.WriteLine("File size: {0}", f.Length);
        Console.WriteLine("File type: {0}", f.ContentType);
        Console.WriteLine("Attributes: {0}", f.Attributes);
        Console.WriteLine("Creation: {0}", f.CreationTime);
        Console.WriteLine("Last Access: {0}", f.LastAccess);
        Console.WriteLine("Last Write: {0}", f.LastWrite);
    }
}

```

Megjegyzés Ha a számítógépen nem található C:\Windows\WebViewer\konvijat\alakzatú bitmap fájlokat olvasza be.

Fajlok lista zására a DirectoryInfo típus segítségevel

A directory into which haszhalata

A *Createsubdi rectory()* metodus visszatérési eredményt kell tudni, hogy sikeres végrehajtás esetén a metodus visszad ad egy *Dictionary* objektumot, amely az újonnévű leterhozott környvtárat jellepzi. Nezzük meg azt a példát!

Ha meghívjuk ezt a metoduszt a Main() függvényből, és Windows Intézvöl meghenyezzük a Windows-könyvtárat, írható, hogy megjelennek az új alkotónyv-

```

    static void ModifyAppDirectory()
    {
        DirectoryInfo dir = new DirectoryInfo("..\\");

        // A \MyFolder\Data Térhezásá az alkalmazás konviktárában.
        // A \MyFolder\Subdir\rectory "MyFolder" kijelölése.
        dir.CreateSubdirectory("MyFolder");

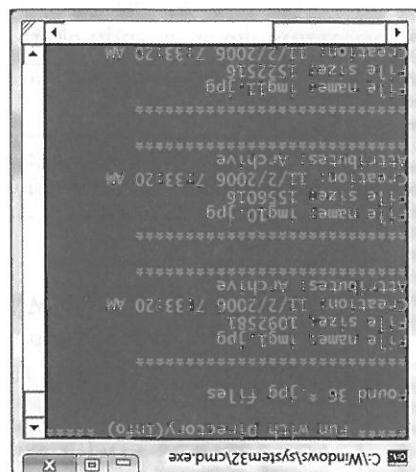
        // A \MyFolder\Subdir\rectory "MyOlder" kijelölése.
        // A \MyFolder\Subdir\rectory "MyOlder\Data" kijelölése.
        dir.CreateSubdirectory("MyOlder");
    }
}

```

Egy környvétér szerkezete a direktoriánhoz. Címetes szabadi réctörök (métodusai programozottan bonyolítják). Ezzel a metodussal akár egyszerű, akár több egyszerű műveletet alkalmazás telepítési utvonalainak megágyazott alkonyváttarat is letervezhetünk egyszerszerűen gyorsabban.

A alkonyvatarak letrehozása a DirectoryInfo segítségével

20.3. abra: Információk kezeléséről



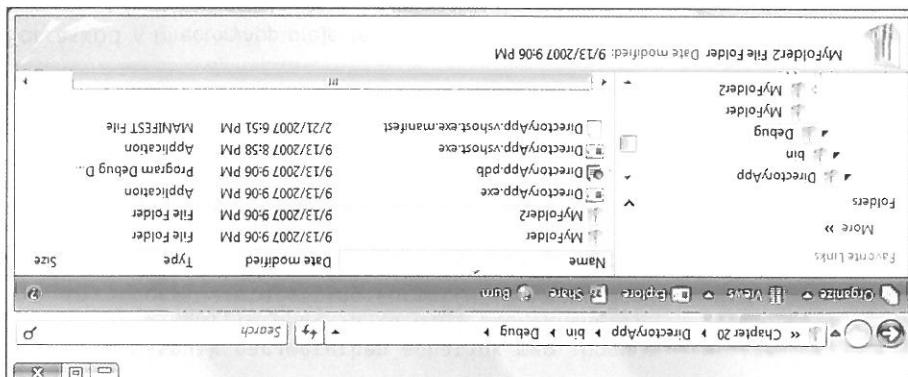
20. fejezet: Fajlmuveletek és elszigetelt tarolás

A következőkben ismerekedjünk meg a Directory tipussal. A Legtöbb esetben a Directory osztály statikus tagjainak működése megtételel annak funkcionálisának, amelyet a DirectoryInfo példányszintű tagjai nyújtanak. Ne feledjük azonban, hogy a Directory tagjai általában szinteghipusokat adnak vissza, nem pedig őrökön típusos FileInfo/DirectoryInfo típusokat.

A Directory típus működésének szemléletezésre ez a végső segédíűggvény megjelenít a számítógépen található összes meghajtó nevét (a Directory.GetFiles() metódussal), és a Directory.Delete() statikus metódus a korábban leírttal a korábban leírtható MyFolder és MyFolder2/Data alá.

A Directory típus használata

20.4. abra: Alkonyetárrak letrehozása



```

    static void Main(string[] args)
    {
        // Create a new directory object
        DirectoryInfo dir = new DirectoryInfo(@"C:\Temp\MyFolder");

        // Create a subdirectory named "Data"
        dir.CreateSubdirectory("Data");

        // Create a file named "Info.txt" in the Data folder
        FileInfo infoFile = new FileInfo(Path.Combine(dir.FullName, "Info.txt"));

        // Write some text to the file
        infoFile.WriteAllText("Hello, World!");

        // Read the contents of the file
        string content = File.ReadAllText(infoFile.FullName);

        // Print the contents to the console
        Console.WriteLine(content);
    }
}

```

```
A system. IO nevterben található a DriveInfo nevű osztály. A Directory, Get-  
LogicalDrives() metódushoz hasonlóan a statikus DriveInfo.GetDrives()  
metódus is a számítógép meghajtóműk a nevet adja vissza, de a Directory.  
GetLogicalDrives() metódus() metódusban hasonlóan a statikus DriveInfo.GetDrives()  
metódus is a számítógép meghajtóműk a nevet adja vissza, de a Directory.  
GetLogicalDrives() metódusban elérhetően a DriveInfo szamos egyséb részle-  
tetetőkkel (pl. a meghajtó tipusa, a szabad területek, a kö-  
tetetők megadása). Nezzük a következő, DriveInfoApp nevű új konzolalkalma-  
zásban definiált Program osztályt:  
class Program  
{  
    static void Main(string[] args)  
    {  
        Console.WriteLine("***** Fun with DriveInfo *****\n");  
    }  
}
```

A DriveInfo osztálytípus használata

Förträskod A DirectoYapp projektet a förträskodkonyvtárban a 20. fejezet alkonyvtára tár-találmazza. A forrásokkal nyilvántartott lásd a Bevezetés részében a xlvi. oldalat.

```

static void Main(FunWithDirectories directoryType)
{
    try
    {
        // A számítógép meghajtójainak kijelölése.
        string[] drives = directoryType.Directories();
        Console.WriteLine("Here are your drives:");  

        foreach (string s in drives)
            Console.WriteLine(s);
        // Letöröljük, amit létrehoztunk.
        // Delete the directories we created to delete them.
        Console.WriteLine("Press enter to delete directories");
        Console.ReadLine();
        Directory.Delete(string.Format("{0}\\MyFolder",  

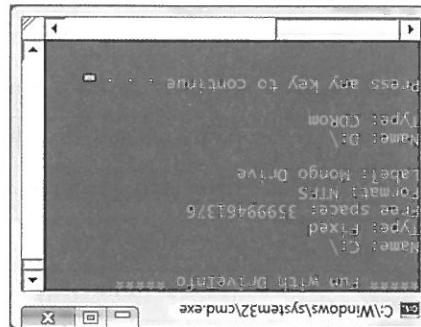
            Environment.CurrentDirectory));
    }
}

```

Torrások A Driveníappp profilprojekt a forrásokkal összefüggően a 20. fejezet alkonyvátra tar-talmazza. A forrásokkal összefüggően a 20. fejezet alkonyvátra tar-talmazza. A forrásokkal összefüggően a 20. fejezet alkonyvátra tar-talmazza.

A Director, a Directoryinfo és a Driveinfo összefoglalók alapvető működésének áttekintése után a következőkben megvizsgáljuk, hogyan hozhatunk létre, nyithatunk meg, zárhatsunk be és törlhetünk adott könnyvtárban lévő fájlokat.

20.5. ábra: Meghajték adatának lekérdezése a Drivelinfo használatával



A 20.5. ábra egy lehetséges kimenetet mutat.

A Driveinfo osztálytípus használata

20.4. tablázat: Fileinfo alapvető tagjai

Tag	Jelentés
Appendedtext()	Egy streamwriter tipusú hozzáférés a fájhoz. Eltérítésekkel szövegeket frízhetünk a fájlhoz.
CopyTo()	A megfelelő fájlt átmásolja egy új fájlba.
Create()	Új fájlt hoz létre, és egy streamwriter tipusú (lásd <code>Kesobb</code>) tervez.
CreateText()	Egy streamwriter tipusú hozzáférés a fájlhoz, amelynek minden karakterét az információhoz közelítően kell.
Delete()	Törli a fájlfájlt példányát a lakkúrtól.
DirectoryName	Visszaadja a szülgókönyvtár teljes elérési útját.
Length	Visszaadja az aktuális fájl vagy környvű rész metereit.
Moveto()	A fájlt áthelyezí, és lehetőséget ad arra, hogy új nevet adjunk neknek.
Name	Visszaadja a fájl nevét.
Open()	Megnyitja a fájlt különöző olvasási/trási és megosztási jogaival.
OpenRead()	Csak olvasáshoz függetlenül hozzáférhető.
OpenText()	Egy streamreader tipusú hozzáférés (lásd <code>Kesobb</code>), amelyet egy fájlt olvasunk.
OpenWrite()	Csak írható függetlenül hozzáférhető.

tagját a 20.4. táblázatban foglaltuk össze.

Ahoogy a direktoriapp-peldabán is látunk, a Fileinfo osztály segítségével a számítógépen található fájlokrol kerdezhetünk le adatokat (leterhezszám, méret, attribútumok stb.), tövábbá az osztály segítségével leterhezszámolhatunk, aholgyezhetünk es törlhetünk fájlokat. A Filesystemről tisztesebb részletekkel foglalkozunk, amelyekben részletesen bemutatjuk a Fileinfo osztály funkcióit.

A Fileinfo osztály használata

```

    {
        // Használjuk a Filestream objektumot...
    }
}

using (Filestream fs = f.Create())
{
    Fileinfo f = new Fileinfo(@"C:\Test\dat");
    // Idéális a fájl I/O trübsöközö.
    // Egy 'using' hatókor definíálása
    static void Main(string[] args)
}

```

C# using blokkjának segítségével a forrátra biztatók a „takarító”-logika erőforrásait. Minthogy a Filestream megvalósítja az idioszabálásokat, az azonosítót, hiszen így a rendszer felszabadítja az adattöolyam nem felügyelt azonosítót, minden fileszámlát számára teljes olvasási/írásra hozzáérésre bíztat. Minthány befejeztük a Filestream objektummal a munkát, zárjuk le a fájlt.

A Fileinfo.Create() metódus Filestream típusú ter vissza, amely a fájlon minden szimikron, minden aszimikron írás/olvasási műveleteket tesz lehetővé (a részleteket lásd később). A Fileinfo.Create() által visszadott Filestream objektum minden részletekkel rendelkezik, amelyeket minden részletekkel használhatunk. Minthogy a részletekkel rendelkező Filestream minden részletekkel rendelkezik, minden részletekkel rendelkező Filestream minden részletekkel rendelkezik.

```

    {
        fs.Close();
    }
}

// Zárjuk le a fájlt önmagot.

// Használjuk a Filestream objektumot...
static void Main(string[] args)
{
    Filestream fs = f.Create();
    Fileinfo f = new Fileinfo(@"C:\Test\dat");
    // Létrehoznunk egy új fájlt a C meghajtón,
    static void Main(string[] args)
}

```

A fájlazonosító létrehozásának első módja a Fileinfo.Create() metódus használata:

A Fileinfo.Create() metódus

A Fileinfo osztály többsege tehát egy specifikus I/O-központú objektumot (Filestream, Streamwriter stb.) ad vissza. Ezek segítségevel a fájl nyozásá elöl azonban vizsgáljuk meg azokat a különbségeket, amelyekkel a Fileinfo osztályal megszerezhetjük a fájlazonosítóját.

Tag	Jelentés
CreateNew	Az operációs rendszert új fájl leterhezására utasítja. Ha a fájl már létezik, akkor felülírja azt.
Create	Az operációs rendszert új fájl leterhezására utasítja. Ha a fájl megnyitva van, mindenki a fájl kiolvasására jogosult.
Open	Megnyít egy másik fájlt. Ha az már létezik, ellenkező esetben új fájlt hoz létre.
OpenOrCreate	Megnyítja a fájlt, ha az már létezik, ellenkező esetben új fájlt hoz létre.

```
{
    Append,
    Truncate,
    OpenOrCreate,
    Open,
    Create,
    CreateNew,
    CreateNew,
    {
        public enum FileMode
    }
}
```

Felsoolt tipussal határozhatunk meg (lásd 20.5. táblázat):
szége. Az első paraméter az I/O-kérés tipusát határozza meg (pl. új fájl leterhezására, fájl megnyitása, hozzáírásra, hozzáírásokhoz stb.), amelyet a FileMode típusú paramétere van szűk-

```
{
    {
        // Használjuk a FileStream objektumot...
    }
}

FileStream Readwrite, FileMode, Open(Create, FileMode, OpenOrCreate,
using(FileStream fs2 = new FileInfo("C:\Test2.dat"));
FileInfo fs2 = new FileInfo("C:\Test2.dat");
// Leterhezunk egy új fájlt a FileInfo.Open() metódussal.
{
    static void Main(string[] args)
}
```

zük meg a következő logikát:
Az Open() meghibásza után egy FileStream objektumot kapunk vissza. Néz-
az Open() jellemezően több paramétert vesz fel a fájl kezelésének a leírására.
néhányamis preízébeben használhányuk a fájlok megnyitására és leterhezására, úgyani-
A FileInfo.Open() metódust a FileInfo.Create() által nyújtott lehetőségek-

A FileInfo.Open() metódus

2009



Andrew Troelsen

Második kötet (20-33. fejezet)

es a .NET 3.5
A C# 2008



Fösszerkesztő: Kis Balázs MScE, MCT, e-mail: balazs.kis@szak.hu
Info@szaak.hu ■ Kiadóvezető: Kis Ádám, e-mail: adam.kis@szaak.hu ■
36-22-350-209 ■ Fax: 36-22-565-311 ■ www.szaak.hu ■ e-mail:
Könnyvterjesztők Egyesületeinek a tagja ■ 2060 Bicske, Díjfa u. 3. ■ Tel.:
SZAK Kiadó Kft. ■ Az 1795-ben alapított Magyar Könnyvkiadók és

vagy eszközökkel elektronikus úton vagy más módon közölni.
Néhány filos reprodükálni, adatátvitő rendszereken tárolni, bármilyen formában
Mindenn jog fenntartva. Jelen könnyvét, illetve annak részét a kiadó engedélye

lyesek nevű programjával elérhetőtök.
A könnyv fordítása a Kilgray Kft. Memoo (<http://www.memooqm.com>) progra-
májával készült, a szöveg helyesegét az eválásztásokat a Morphologic He-

ISBN 978-963-9863-10-1

Magyár nyelvi gondozás: Laczko Krisztina és Trepák Monika
Lektor: David Zoltán, MVP és Gimcsai Gábor, MVP
Varga Péter,
Eros Szilvia, Ivancsy Renáta, Szalay Márton, Tibor Melinda, Varga Ágnes,
Fordította a SZAK Kiadó fordítószöpörje: Domoszlai László, Endredi Gabriella,
Magyar fordítás (Hungarian translation) © SZAK Kiadó 2009.

Copyright 2008 by Apress.
Original English language edition published by Apress Co. Sound View Court 3B,
Greenwich CT 06830 USA. Copyright © 2008 by Apress Co. Hungarian-language
edition copyright © 2009 by SZAK Kiadó. All rights reserved.

A C# 2008 es a .NET 3.5 - Maissoldik kötet (20-33. fejezet es ket függelék)
A négyedik kiadás magyarul, ket kötetben
Pro C# 2008 and the .NET 3.5 Platform, Fourth Edition
Andrew Troelsen

Ezt a kiadást Miklónnak, a csodamacskanak, a 412-es szám alatt eltoltott éveknek és csodálatos felleségemnek, Amandának ajánlom, aki tisztelmesen várta, hogy elkeszítjék újabb könnyvem megírásával.

Mi egy csapata vagyunk, az Olivaso és en.....	xxxii
A könnyv áttekintése	xxxiii
Első kötet.....	xxxiii
1. rész: Bevezetés a C#-ba és a .NET platformba.....	xxxiii
2. fejezet: C#-alkalmazások készítése	xxxiv
3. rész: A C# alapvető építőelemi, I. rész	xxxv
4. fejezet: A C# alapvető építőelemi, II. rész	xxxvi
5. fejezet: Egységebbe zárt osztálytípusok definíciója	xxxv
6. fejezet: A származtatás es a polymorfizmus.....	xxxv
7. fejezet: Struktúratípusek.....	xxxv
8. fejezet: Az objektumok elektikusa.....	xxxv
3. rész: Haladó programozási szerekkel a C#-ban	xxxvi
9. fejezet: Interfészek használata	xxxvi
10. fejezet: Gyűjtőmenyek és generikus típusok	xxxvi
11. fejezet Metódusreferenciák, események és Lambdák	xxxvi
12. fejezet: Indextíperek, operatork és mutatók	xxxvii
13. fejezet: C# 2008 nyelvi újdonságai	xxxvii
14. fejezet: Bevezetés a nyelvbe ágyazott	xxxvii
lekérdezésekhe (Linq)	xxxvii
4. rész: Programozás .NET-szerelvények	xxxviii
15. fejezet: A .NET-szerelvények	xxxviii
16. fejezet: Tipusrelaxio, Késői kötés es	xxxviii
attribútumalapú programozás	xxxviii
17. fejezet: Folyamatok, alkalmazásoktól mások es	xxxix
a dinamikus szerelvények	xxxix

Tartalomjegyzék

Kioszönenetnyilvánítás xxix

Bevezetés xxxi

A System.IO nevűr	3
20. Fájlműveletek és elszigetelt tárókás	2
Az absztrakt FileSystemlinüösöztetés	6
A DirectoryInfo és FileInfo típusok	7
A DirectoryInfo típus használata	9
Fájlak listaízása a DirectoryInfo típus segítségével	10

5. rész: Bevezetés a .NET alaposztálykonvertációba

Második kötet	xxxix
5. rész: Bevezetés a .NET alaposztálykonvertációba	xxxix
20. fajlmezők és elszigetelt tárókás	xxxix
21. fajlmezők Bevezetés az objektumsortas világábba	xli
22. fajlmezőt ADO.NET 1. rész: Az élő kapcsolat	xli
23. fajlmezőt ADO.NET 2. rész: A bonott kapcsolat	xli
24. fajlmezőt A LINQ API programozása	xli
25. fajlmezőt A WCF	xli
26. fajlmezőt A WF	xli
6. rész: Fehaszszáli fajlmelek	xli
27. fajlmezőt Windows Forms-programozás	xlii
28. fajlmezőt A WPF és az XAML	xlii
29. fajlmezőt Programozás WF-vézetelőlemekekkel	xlii
30. fajlmezőt WPF 2D grafikus renderelek	xlii
7. rész: Webes alkalmazások felhasználók számára ASP.NET segítségével	xliii
erőforrások és temák	xliii
temák és mesteralak	xliii
33. fajlmezőt ASP.NET állapotkezelési technikák	xliii
A függelék: A COM és a .NET egységtípusok kezelése	xliii
B függelék: Platfromfüggelek.NET-felisztés a Monoval	xliii
C függelék: Utazáson letooltott fejezet – még több információ	xliii
A könnyű forrásködjanak igénylés	xlv
A lehetőséges javítások	xlv
Elerhetőségek	xlv
20. Fájlműveletek és elszigetelt tárókás	2

A Directive típus használata	11
A DriveInfo osztály típus használata	12
A Fileinfo osztály használata	14
A FileInfo.Open() metódus	15
A FileInfo.Create() metódus	16
A FileInfo.OpenRead() és a FileInfo.OpenWrite() metódusok	17
A FileInfo.OpenText() és a FileInfo.AppendText() metódusok	18
A FileInfo.OpenText() metódus	18
A Fileinfo.CreateText() és a Fileinfo.AppendText() metódusok	18
A StreamWriter és StreamReader típusok használata	29
A StringWriter és StringReader típusok közvetlen használása	29
A StreamWriter/SreamReader típusok közvetlen használása	29
A StreamWriter és StreamReader típusok közvetlen használata	31
A BinaryWriter és BinaryReader osztályok használata	31
Fajlok programozott „folyélesse”	34
Asztinkron fájolvásás és -írás	36
Az elszigetelt fájlok szerepe	38
Bizalom kerdesei	38
Az elszigetelt fájlok szerepe	38
Bizalom kerdesei	38
Az elszigetelt tarolo típusot API egysége	39
fehásználási módjai	40
Bevezetés a kódredefekt-látható biztonságba	40
A bizonyítékok szerepe	41
A kodcsoporthoz köthető	45
Az engedélykészletek szerepe	49
A CAS működése	50
Full Trust jogosultság viszazzállítása	52
a My_Computer_Zone kódcsoporthoz	52
Az elszigetelt tarolo	52
Az elszigetelt tarolo hélye	53
Az elszigetelt tarolo hatókörre	55
Az elszigetelt tarolo objektummal	57
Az elszigetelt tarolo szerepe	57
Az elszigetelt tarolo	58
Tarolo leterhezsa az IsolatedStorageFile objektummal	58
Adat rösa a taroloiba	60

Az olvasásra a tárloból.....	61
Felhasználói adat törlése a tárloból	62
Egyedi könnyvtárt készítő leírások	62
Az elszigetelt tárlohoz közelben: ClickOnce-teljesítmények	64
Az IsolatedStorageFelfelmelegítési szabályok	65
Az biztonságírók zóna korlátozása	65
Az alkalmazás kiszolgáltatása webkiszolgálón	67
Az eredmény megtékinthése	67
Osszefoglalás.....	68
21. BEVEZETÉS AZ OBJEKTUMSOROSÍTÁS VILÁGÁBA.....	71
Az objektumsortás	71
Objektumok konfigurálása sorosításban	73
Nyilvános mezők, privát mezők és nyilvános tulajdonságok	76
A sorosító formázó kiállítása	77
Az Iforamtér és az IRelemezFormattér interfések	78
A formázók közötti típusösszesség	80
Objektumok sorosítása a BinaryFormatterrel	81
Objektumok viszazzállítása a SoapFormatterrel	83
Objektumok sorosítása a GeneralXML-adatok szabalyozása	86
Objektumok sorosítása az XmlSerializerrel	88
A sorosítási folyamat tesztelése	90
Az objektumsortás hatérfelületei	91
Sorosítások teszteléséhez használatai	92
Sorosítások teszteléséhez használatai	96
Osszefoglalás.....	97
22. ADO.NET, 1. rész: Az előkapcsolat	99
Az ADO.NET magas szintű meghatározása	99
Az ADO.NET körére	101
Az ADO.NET-adatszolgáltatók működése	102
A Microsoft által származó ADO.NET-adatszolgáltatók	104
Harmonikus felhalmozás ADO.NET-adatszolgáltatók	106

További ADO.NET-nevek	107
A System.Data névter típusai	108
A IDbConnection interfész szerepe	109
Az IDbCommand interfész szerepe	109
Az IDbTransaction interfész szerepe	109
Az IDbDataAdapter és az IDatarRecord interfészek szerepe	111
Az IDbDataAdapter interfész szerepe	112
Adatszolgáltatók absztrahálása interfészszekel	113
Rúgalmasság novellese az alkalmazásoknál gyűrűkön	115
Az Autolot adatbázis leterhözés	117
Az Inventory tabla leterhözés	118
A GetPetName() tarolt eljárás leterhözés	120
Az Customers és az Orders tablák leterhözés	121
Táblakapcsolatok vizuális bemenetek	123
Az ADO.NET adat provider factory modell	124
Regisztrált adat provider factoryk	125
Egy teljes adat provider factory példa	126
A data provider factory modell lehetséges hártya	130
A <connectionString> elem	130
Az ADO.NET kapcsolatlapú modelle	131
A kapcsolatobjektumok használata	133
A ConnectionStringBuilder objektumok	136
Az adatolvások	139
Több eredményhalmaz kinyerése adatolvásval	141
Ürítelhasználható adatleírései könnyebb készítésre	142
A kapcsolatlogika leterhözés	144
A beszürás törlesztve logikai leterhözés	145
A módosítás törlesztve logikai leterhözés	146
A lekérdezést végző logikai leterhözés	147
A paramétereit paraméterekkel	148
Tárolt eljárások végrehajtása	150
Paraméterek meghadása a IDbParameter típus súrgótsegével	152

A Main() metodus implementálása	153
A ListInventory() metodus implementálása	155
A DeleteCar() metodus implementálása	156
A InsertNewCar() metodus implementálása	157
Az UpdateCarName() metodus implementálása	157
A CarrollEljárásunk megírása	158
Az Aszinkron adatleírás az SqlCommand használatával	159
Az Ado.NET-tranzakciókhoz	161
Az adatbázis-tranzakciók	162
Az Ado.NET-tranzakcióobjektum külcsfotosságú tagjai	162
Tranzakcióméthodus hozzáadása az InventoryDAL	
Osszefoglalás	167
23. ADO.NET, 2. rész: A bontott kapcsolat	169
osztályhoz	163
Az adatbázis-tranzakciók tesztelése	166
Az ADO.NET kapcsolat nelekű modellje	170
A DataRow típusok használata	178
A RowState tulajdonság	180
A DataRowVersion tulajdonság	182
A DataTable típusok beszűrása DataTableReader	185
A DataTable adattámak feloldogozása DataTableReader	186
A DataSet objektumok sorosítása XML-Ként	188
Objektumokkal	186
Formatumban	189
DataSet objektumok sorosítása bináris	
DataTable fejtöltese egy generikus List<T> használatával	192
DataTable objektumok körte sérelhetőkéhez	190
Datatable fejtöltese egy generikus List<T> használatával	192

Sorok programozott törlése	194
Sorok kiállásztasa szüresi feltelek alapján	196
Sorok modosítása	199
A DataView típus használata	200
Egy utolsó felhasználói felületbővítmény: sorok számlával	202
Dataset/Datatable objektumok felületek adattípusokkal	203
Egy egyszerű adattípus	205
Adatbázisnevek leképezésére barátágos nevekre	206
Az AutoLotDAL.dll ismertetett vizsgálata	207
A kinduló osztálytípus definíciója	207
Az adattípusok konfigurálása az SqlCommandBuilder	208
használatával	
A GetAlmInventory() implementációja	210
Windows Forms front end letélezása	211
Navigálás a többtáblázatos DataSet objektumokban	212
Az adattípusok elérési szintezise	213
A táblázatok közötti kapcsolatok kiépítése	215
Az adatbázistablak modosítása	215
Navigálás a kapcsolódó táblázatok között	216
A Visual Studio 2008 adattípusi eszközei	220
A DataGridView vizuális megtervezése	220
Az App.config fájl és a Settings.Settings fájl	224
A generált DataSet vizsgálata	226
A generált DataTable és DataRow típusok vizsgálata	228
A generált adattípuselt	230
Generált tippusk használata a kódban	231
Az automatikusan generált kod elválasztása	233
Egy felhasználói felület front end: újra a felhasználói felület-rendszer	236
Összefoglalás	237
24. A LINQ API programozása	239
A LINQ to ADO.NET szerepe	239
Programozás a LINQ to DataSettel	240
A DataSet bővítmények szerepe	242

Datatable LINQ-kompatibilitás használata	243
A DataRowExtensions.Field<T>() bővíti metódus szerepe	245
Üj Datatable objektumok felülete	
Az entitássztruktúrok szerepe	248
A DataContext típus szerepe	248
Egy egyezségi LINQ to SQL Példa	249
Erosen típusos DataContext letrehozása	251
A [Table] és [Column] attribútumok: további részletek	253
Enittássztruktúrok generálása az SqlCommand.exe használatával	254
Kapcsolatok definíciókra enittássztruktúrok használatával	258
Az erősített típusos DataContract	259
A generált típusok használata	260
Erittássztruktúrok generálása a Visual Studio 2008	
Üj elemek beszűrása	264
Letező elemek módosítása	265
Letező XML: egy jobb DOM	267
használatával	267
XML-dokumentumok kezelése a LINQ to XML	
használatával	262
Üj elemek beszűrása	264
Letező elemek módosítása	265
LINQ to XML: egy jobb DOM	267
használatával	267
A XML-dokumentumok letrehozása LINQ-lekérdezésekkel	271
XML-tárolom betöltsése és elmezés	272
Navigálás egy meghatározott dokumentumban	272
Adatok módosítása egy XML-dokumentumban	275
Összefoglalás	276
25. A WCF.....	277
Nehány előzettsel API	277
A DCOM szerepe	278
A COM+/Enterprise Services szerepe	279
A MSMQ szerepe	280
A .NET-rendezés szerepe	281
Az XML-webszolgáltatás szerepe	282

Példa .NET-wébszolgáltatásra.....	282
Wébszolgáltatási szabványok.....	285
Named pipe-ok, socketek és P2P.....	286
A WCF szerepe	286
A szolgáltatásorientált architektúra áttekintése	288
I. alaplevi: A határök explicitek	289
2. alaplevi: A szolgáltatások autonómok	289
3. alaplevi: A szolgáltatások szerződésen kerestül es nem implementációban kerestül kompatibilitásra	289
4. alaplevi: A szolgáltatás kompatibilitása	289
házi rendben alapul.....	289
WCF: A lényeg	290
Az alapvető WCF-szerelvények	290
A Visual Studio WCF projektsablonok	292
A WCF Service Website projektsablon	293
A WCF-felalkmazás alaposszeállítása	294
A WCF-Kötések	298
A WCF-szerződések	297
A WCF ABC-jé	296
A WCF-Kötések	299
HTTP-alapú Kötések	300
MSMQ-alapú Kötések	301
A WCF-címek	302
WCFSzolgáltatás Kötések	304
A [ServiceContract] attribútum	306
Az [OperationContract] attribútum	307
Szolgáltatástípusok mint működési szerződések	308
A WCF-szolgáltatás hoztölása	309
ABC-k letrehozása az App.config fájban	310
A ServiceHost típus	313
Hozzáférési jogosultságok	311
A ServiceHost típus használata	310
A ServiceHost típusok lehetségek	313
A <system.serviceModel> elem jellemezői	315
Metadatcsere (Metadata Exchange) engedélyezése	317
WCF-ügyfelalkalmazás készítése	320
Proxykód generálása az svcsutil.exe segítségével	320
Proxykód generálása Visual Studio 2008-ban	321
TCP-alapú kötes konfigurálása	323

A WCF Service Library projektsablon használata	325
Egyeszerű Matérk-szolgáltatás készítése	325
A WCF-szolgáltatás tesztelése a WcfTestClient.exe-vel	326
A konfigurációs fájl modosítása az SvcConfigEditor.exe	326
Programmal	327
WCF-szolgáltatás használása Windows-szolgáltatáskenet	329
Az ABC-k megaladása a forráskódban	330
Windows-szolgáltatás telepítő letrehozása	332
A MEX engedélyezése	332
Szolgáltatás aszinkron hívása	334
WCF-adatszerek tervezése	337
Webközpontú WCF Service projektsablon használata	338
Szolgáltatás szerverzések implementálása	340
A *.svc fájl szerepe	341
A WF építőköckái	347
A WF futtatásának megoldása	348
A WF alapvető szolgáltatási	349
A WF-tevékenységek elülső megközelítésben	350
Szekvenciális és állapotföldelt-munkafolyamatok	352
WF-szerelvénnyek, -névtérök és -projektek	355
A .NET 3.5 WF-támogatása	356
Visual Studio munkafolyamat-projektsablonok	356
A munkafolyamat menete	357
Egyeszerű munkafolyamat-alakmazás létrehozása	358
A kezdeti munkafolyamathoz tartozó kod vizsgálata	358
A Code tévékenysége hozzáadása	359
White tévékenysége hozzáadása	361
A WF-motor hozstolási kódja	364
Egyedi indítási paraméterek hozzáadása	365
Webszolgáltatások hívása a munikafolyamatasunkban	368

A MathWeb szolgáltatás leterhezásá 368	A WF-webszolgáltatás-fogyaztó leterhezásá 370	A WF-webszolgáltatás-fogyaztó leterhezásá 370
segítségeivel 377	Ujratelthetők WF-kódoknáyer leterhezásá 382	Hittelellenőrzés végrehajtása 384
Kommunikáció WC-szolgáltatással a SendActivity 374	Az invokéWebservice tevékenységek konfigurálása 372	Windows Forms-kleinálkalmazás leterhezásá 387
szisztemával 391	Az invokéWebservice tevékenységek konfigurálása 372	Windows Forms-kleinálkalmazás leterhezásá 387
6. rész: Felhasználati felületek 395	Ujratelthetők WF-kódoknáyer leterhezásá 382	Hittelellenőrzés végrehajtása 384
27. Windows Forms-programozás 395	A vezetőlemekek gyűjtémenyének felülete 399	A Visual Studio Windows Forms-projektsabloná 402
A Windows Forms-neveterek 396	A System.EventArgs és A System.EventHandler szerepe 402	A Visual Studio Windows Forms-Projektsabloná 404
Egyesített Windows Forms-kleinálkalmazás (IDE-mennet) 397	A vizuális tervezőfelület 404	A kezdett útjáról 406
397	Mennitrendszerlek vizuális építése 409	A Program osztály 408
396	Az urlapok anatómiája 412	A Control osztály funkcionálitása 414
396	Az urlapok kattintás meghatározása 419	A Form osztály funkcionálitása 417
396	Reagálás az egér eseményére 422	A Form típusa elérhetősége 419
396	Az egergomb kattintás meghatározása 424	Reagálás a billentyűzet eseményére 425
396	Reagálás a billentyűzet eseményére 427	Parbeszedők tervezése 427
396	A DiálogResult tulajdonság 429	A tabulátorosirány-választ 430
396	A tabulátorosirány-konfigurálása 430	Az úrlap alapértelmezett beviteli gömbjának a beállítása 431
396	Parbeszedők megléltetése 431	Parbeszedők megléltetése 431

28. A WPF és az XAML 453

Az utánpók származtatása	433
GDI+-alapú grafikus adatok renderelése	436
A System.Drawing névter.	437
A Graphics objektumok megszerzése a Paint	438
eseményen keresztül	439
Az utánpók felületeinek eredménytelenítése.....	442
Teljes Windows Forms-alakmazás létrehozása.....	443
A fómenürendszer készítése	443
A ShapeData típus meghatározása	444
A ShapePictureDialog típus meghatározása	445
Ilfástsukláról hozzáadása A MatinWindow típushoz	446
A Tools menü funkcióitásának implementálása	447
A grafikus kiemeneti rögzítése és renderelese	449
A sorosítási logika implementálása	450
Összefoglalás.....	452

A System, Windows, Media, Visual szerépe	470
A System, Windows, DeependencyObject osztály szerépe	470
A Syatem, Windows, Threadинг, DispatcherObject szerépe	471
(XAML-mennet) WPF-alakmazás készítése	471
A Window osztálytipus kidövítése	474
Egyszerű felhasználói felület letrehozása	475
Az Application típus további jelelmzői	477
Az Application típus Windows gyűjteményének feloldogozása	477
Az alkalmazás adatai és a paraméteri argumentumok feloldogozása	477
Az alkalmazás készítése	480
A Window objektum Ellettertama	480
A Window objektum Closung eseményének kezelése	482
Ablakszintű egéreresemények kezelése	484
Ablakszintű billentyűzetesemények kezelése	485
(XAML-központról) WPF-alakmazás készítése	486
A MainWindow definíciója XAML-ben	487
Alkalmazásobjektum definíálása XAML-ben	488
XAML-fájlok feloldogozása az msbuild.exe segítségével	489
Markup átalakítása.NET-szerelvénye	492
XAML-lekepzeze C#-kódra	492
A BAML szerépe	494
XAML-ból szerelvénnyel: a Foley mat összefoglalása	496
A kapcsolatok elküldönösére mógsöttes Kodfajlakkal	497
A XAML-szintaxis	499
A XML-kiértelek a XamlPad segítségevel	499
XAML-nevérek és -külcsszavak	501
XAML-élémek és -attribútumok	505
A XAML tulajdonságok szintaxisa	506
XAML-típusokatalkotk	510
A XAML csatolt tulajdonságai	509
A XAML markupból menyei	512
WF-alakmazások készítése Visual Studio 2008 segítségevel	517
WPF-projektsablonok	518
A kezdeti alaknak nevenek módosítása	519
A WPF-tírvező	520

XAML feldolgozása futásidőben: A SimpleXamlPad.exe	523
A Loaded esemény megvalósítása	525
A Button kattintási eseménynek megvalósítása	526
A Closed esemény megvalósítása	527
Az alkalmazás tesztelése	527
A Microsoft Expression Blend szerepe	528
Az Expression Blend elnyei	529
Osszefoglalás	530
A WPF vezérlőelemek és a Visual Studio 2008	533
A részletek a dokumentációnban találhatók	534
Vezérlőelemek deklarálása a XAML-ben	535
Egy utiműködés a vezérlőelemekkel a forráskódjálokban	536
A tövábbított buforékesemények szerepe	546
Továbbított események	544
mezők	543
Wrappper tulajdonság definíciója a DependencyProperty	548
A tövábbított lefutó események szerepe	547
A buforékesemények fölytatása és leállítása	547
A Button típusok használata	551
A ButtonBase típus	551
A ToggleButton típus	552
A Button típus	553
A RepeatButton típus	554
A Logikai csatornák leírása	558
A kapcsolódó elemek közötti összefüggés GroupBox típusokba	559
A kapcsolódó elemek közötti összefüggés GroupBox típusokba	560
Listbox és a ComboBox típusok használata	562
Lista-vázlethelek felületei programozt módon	563
Tesztoleges tartalom hozzáadása	564
Az aktuális kiállások meghatározása a beágyazott	565
Tartalom esetében	566

A többszöros szövegbeviteli mezők használata	568
A TextBox típus használata	569
A PasswordBox típus használata	570
A WrapPanel kezelés Canvas paneleken belül	574
Tartalom elhelyezése a WrapPanel paneleken belül	577
Tartalom elhelyezése a StackPanel paneleken belül	579
Tartalom elhelyezése a Grid paneleken belül	580
Rácsok GridSplitter típusokkal	582
Tartalom elhelyezése a Grid paneleken belül	583
Tartalom elhelyezése a DockPanel paneleken belül	585
Ablak keretének kezelése beágyazott panelek használataval	587
A mentirendszer kezelése	588
A Toolbar típus kezelése	589
A Statusbar típus kezelése	590
A megvalósítás végelesiése	591
A WPF vezérlőutastási	592
A belső vezérlőelem parameterek objektumok	593
Utastáskapcsolás a Command tulajdonságok	595
Utasítások kapcsolása a felhasználói felület tetszőleges	596
Elemeihöz	596
A WPF adatkötési modell	598
Ismerkedés az adatkötéssel	599
A DataContract tulajdonság	600
A Mode tulajdonság	601
Adattáblák az ValueConverter segítségével	602
Konveritals kiülönöködő adattípusok körökt	605
Kötés egyedi objektumokhoz	606
AZ ObservableCollection<T> típus használata	608
Egyedi adatsablon kezelése	610
XML-dokumentumokhoz	611
A felhasználói felület elemeket köröz	612
Egyedi parbeszedelak kezelése	614
A DiálogResult eretk hozzárendelése	615
AZ aktuális kiválasztás megszervezése	616
Egyedi parbeszedelak megjelenítése	617

30. WPF 2D grafikus renderelés, erőforrások	619
és témák	
A WPF grafikus renderelési szolgáltatásának felozófiaja	620
A WPF grafikus renderelési lehetőségei	621
A Shape leszámított típusainak használata	622
A Drawing leszámított típusainak használata	623
A Visual leszámított típusainak használata	624
Egyedi vizuális renderelési program készítése	627
A megfelelő megoldás kiválasztása	629
A Shape leszámított típusainak felidézése	630
A Shape leszámított típusainak funkcionálisai	631
A Rectangle, az Ellipse és a Line típusok használata	631
A Polyline, a Polygon és a Path típusok használata	632
A WPF-ecsettípusok használata	633
Egyéni ecsettípusok készítése	634
Atmenetes ecsetek használata	635
Az ImageBrush típusok	636
A WPF-tollak használata	637
A Drawing leszámított típusainak vizsgálata	638
A Geometry típusok szerepe	639
Egyesített rajzoló geometria	641
Drawing típusok a DrawingImage típusban	641
Drawing típusok a DrawingBrush típusban	642
Összetett rajzoló geometria	643
A Transform leszámított típusok	645
A Transformation leszámított típusok	646
A WPF animációs szolgáltatásai	647
Az Animation utoltaggal rendelkező típusok szerepe	648
A Timeline összefüggő szerepe	649
Az animáció készítése C#-forráskódbal	650
Az animáció ütemezésének szabályozása	652
Az animáció lejátszása viszszafelé és folyamatos ismétlése	653
Animáció készítése XAML-leírásval	654
A Storyboard típus szerepe	654
Az <EventTrigger> használata	655
A kulcsépkocka-animáció szerepe	655
Animáció diszkrét kulcsépkockákkal	656

31. ASP.NET - Weboldalak készítése	685
A HTTP szerelpe	685
A HTTP keresés/válasz-ciklus	686
A HTTP állapotmentes protokoll	686
Webs alkalmazások és webkiszolgálók	687
Az IIS virtuális környvtáriainak szerelpe	688
Az ASP.NET féllesztőkiszolgálója	689
A HTML szerelpe	691
HTML-dokumentumstruktuárak	692
HTML-fájlak féllesztése	693
HTML-alapú felhasználói felület készítése	695

segitségevel

7. rész: Webs alkalmazások fejlesztése ASP.NET

A WPF erőforrásrendszere	690
A bináris erőforrások használata	690
A Resouce Build Action	691
A Content Build Action	692
Az objektum (vagy másnéven Logikai) erőforrások szerepe	693
Az inline stílusok használata	693
A megnevezett stílusok használata	695
Stílusok lezármaztatása	697
A stílusok kiterjesztése	699
A stílusok korlátzása	699
Letézo stílusok lezármaztatása	699
A stílusok kiterjesztése	700
Stílusok hozzárendelése implicit módon	700
Stílusok definíciója tripperekkel	702
Stílusok hozzárendelése programozt módon	702
Vezetőelem felhasználói felületeinek módosítása sablonok	705
Egyedi sablon készítése	706
Triplek hozzáadása a sablonokhoz	707
Sablonok használata a stilusokban	709
Osszefoglalás	711

Az ügyféloldali szkriptek szerepe.....	696
Pelda az ügyféloldali szkriptekről.....	699
A default.htm ürlapadatnak ellenörzése.....	700
Az ürlapadatok tövábbítása (a GET és a POST)	701
Klasszikus ASP-oldal készítése	702
A klasszikus ASP problémái	705
Az ASP.NET 1.x fájlok elérési	705
Az ASP.NET 3.5 legfőbb webes újdonságai	706
Az ASP.NET legfőbb újdonságai	707
Az ASP.NET-neveték	707
Az ASP.NET weboldal-kódolási modelje	709
Adatkapozásban ügyféloldali készítés	710
Az AutoloTDAI.dll fájl manuális hivatközösség	710
A felhasználói felület tervezése	711
Adateleírásban ügyféloldali hivatközösség	712
A szkriptblokk elemzése	716
Az ASP.NET vezetőlelem-műkärmük áttekintése	717
A mögötteskod-modell használata	718
Hivatközösség az AutoloTDAI.dll szervezésére	720
A kodfájl modosítása	720
AZ ASP.NET-oldalak hibakeresése és nyomkövetés	721
ASP.NET-webhely könnyvtárszerkezetének részletei	723
Hivatközösszeg szereleme	724
Az App_Code mappa szerepe	725
Az ASP.NET-oldal forrásait clickusa	726
Folytatás oldalak forrásait clickusa	726
Többfájlos oldalak forrásait clickusa	727
A Page típus származtatási lincsa	729
Együttműködés a bejövő HTTP-kereséssel	730
Boncgeszszövetszövek	732
Hozzáérés a bememelt ürlapadatokhoz	733
Az ISPostBack tulajdonság	734
Együttműködés a kimenő HTTP-válaszokkal	735
HTML-tartalom kihozásától	736
Felhasználók attribútumai	737
Az ASP.NET-wебoldalak elérési készlete	738
Az AutoEventWireup attribútum szerepe	740

Az Error esemény	741
A Web.config fájl szerépe	742
Az ASP.NET Website Administration segedprogramja	745
Összefoglalás	746
32. ASP.NET-vezérlőelemek, -téma&k és -mesteroldalak .. 747	
A webes vezérlőelemek viselkedéseinek megerősítése	747
Kiszolgálóoldali események kezelése	748
Az AutoPostBack tulajdonság	749
A System.Web.UI.Control típus	751
Vezérlőelemek felisrölés	752
A ASP.NET webes vezérlőelemek dinamikus hozzáadása (és törlése)	754
Nehány szó a System.Web.UI.HtmlControls típusoskról	758
Sokoldalú ASP.NET-vezérlőelem	759
Mesteroldalak használata	760
A Menü vezérlőelem és a *.sitemap fájlok használata	763
Kenyérmorzsza-vezérlőelem letrehozása a SiteMapPath	765
Az Inventorystartalomlap tervezése	770
A Default.aspx startalomlap definíciója	776
Típus-szigetezével	776
Az AdRotator használata	777
Az Inventory startalomlap tervezése	778
Az elenörzés szereleme	779
A RegularExpressionValidator vezérlőelem	781
A RangeValidator vezérlőelem	782
A RequiredFieldValidator vezérlőelem	780
A RequiredFieldValidator vezérlőelem	782
A CompareValidator vezérlőelem	782
A RangeValidator vezérlőelem	783
Ellenőrzés összegzésének létrehozása	785
Ellenőrzési csoporthoz definíálás	786
Témák használata	787
A *skin fájlok	788
Témák alkalmazása oldalanknál	790
Témák alkalmazása a teljes webhelyre	790
A SkinID tulajdonság	791

33. ASP.NET állapotkezelési technikák	799
Témák beadállásai kódolásai	793
Vezetőeljelmezékek elhelyezése HTML-táblákkal	795
Összefoglalás	797
Az ASP.NET-nezetállapot szerepe	803
A nézetállapot bemutatása	804
Egyedi nézetállapot-adat hozzáadása	806
A Global.asax fájl szerepe	808
A vevő, globális kivételekkelő	810
A HttpApplication-sosztaly	811
Az alkalmazásadatok modosítása	816
A webalkalmazás Leállításának kezelése	818
Az alkalmazás-gyorsítótár használata	818
Adatak gyorsítótárazása	819
Az *.aspx fájl modosítása	822
Munkamenetadatok kezelése	824
A HttpSessionState további tagjai	828
A sessionid	830
Bemeneti szűrők olvasása	831
A <SessionState> elem szerepe	832
Munkamenetadatok tárolása ASP.NET munkamenet-	
Állapotkiszolgálón	833
Munkamenetadatok tárolása ASP.NET munkamenet-	
Állapotkiszolgálás	835
Munkamenetadatok dedikált adatbázisban	836
Az ASP.NET profili-API-ja	836
Állapotkiszolgálás	837
Munkamenetadatok dedikált adatbázisban	838
Felhasználói profil meghatározása a Web.config fájban	839
Profileddatok hozzáfelese programoztatni	843
Profileddatok csoporthoztasa és egyedi objektumok tárolása	845
Összefoglalás	

A	A COM és a .NET együttműködése	849
8.	rész: Függelékek	
A .NET	es a COM együttműködésének hatolkozóre	849
A C#-ügyfélkiszolgálók	ellenesére	852
Egy .NET	együttműködési szereleme	855
A futási időben	hívható burkoló	857
RCW:	COM-típusok mint .NET-típusok	858
RCW:	COM-típusok generált IDL	863
A COM	IDL szerepe	861
A [default]	interfész szerepe	865
Az IDL-konviktár-utastáts	865
Az [deleau]ttribute	szerepe	866
Az IDL-paraméterattribútumok	866
IDL-paraméterattribútumok	998
Típuskönyvtár használata	együttműködési szereleme	997
Készítéshez	987
Készítéshez a CoCalc	coccashoz	988
Bonyolultabb COM-kiszolgáló készítése	989
Meg egy COM-interfész támogatására	990
Beleső objektumok feltrácsa	992
Az együttműködési szereleme	993	
Saját C#-kliensalkalmazás készítése	994
Együttműködés a CoCar típusossal	995	
COM-szemények előágása	997	
A COM és a .NET együttműködési képessége	999	
A System.Runtime.InteropServices attribútumi	999
A CCW szerepe	999	
A .NET-osztályinterfész szerepe	999	
Osztályinterfész definíálása	999	
Saját .NET-típusok készítése	999	
Erios név definíálása	999	
A típuskönyvtár leterhözésével	999	
Az exportált típus adatának a vizsgálata	999	
Visual Basic 6.0 tesztelés készítése	999	
Osszefoglalás	999	

B Platformfűggetlen .NET-felisztés a Monoval	891
A .NET platformfűggetlen termések	891
A CLI szerepe	892
A nepszervi CLI-distribúciók	894
A Mono hatókör	895
A Mono beszerzés és telepítése	896
A Mono környvtárszerkezetnek vizsgálata	898
A Mono-felisztőszközök	668
A C#-fordítók használata	600
Microsoft-kompatibilis Mono-felisztőszközök	900
Mono-szigetkiszolgálók	901
A mono(2) használata	902
.NET-alakalmazások kezeltése Monoval	903
Mono-kódkönyvvár kezeltése	903
Eros név hozzárendelése a CoreLibDumper.dll	904
szerelevenyhez	905
A modosított manifesztrum megtekintése a monodis	906
Szerelvények telepítése a Mono GAC-ba	907
Konzolalkalmazás kezeltése Monoval	907
Ügyfélalkalmazásunk betöltsése a Mono-	908
futtatókörnyezetbe	909
Windows Forms ügyfélprogram kezeltése	911
Windows Forms alkalmazásunk futtatása Linux alatt	912
Java-szablonok további tanulásba	913
Összejöglalás	915
Tárgymutató	915
A szakmai lektorrol	927
A szerzőről	928

Jölléhet egyedül az en névem jelenik meg a könnyű bortróján, de a munka soha nem készülhetet volna el jó néhány tehetséges ember segítsége nélkül. A Kovett közökben azokhoz szólók, akik lehetővé tettek, hogy ezt a konyvet megírjam.

Mindenekellett készített illlet az Ápriss összes munkatársát, akikkel már több éve van szerencsém együtt dolgozni. Béndekvilli emberék, akit kípesek voltak az en eredeti Word-dokumentumáimat tökéletesen prorázva varázsolni. Hálás készített érte. Már most nagyon váróm, hogy mikor dolgozhatunk is – met együtt egy kovettkező könyvön (de azért elölte szabadságolom magam tanáccsal segítek, hogy úgy érzem, ez a kiadás sokkal szívonalaabb lesz, mint az előzőek. Mint mindenig, az esetleges gápeleseti hibákért vagy technikai tervésekért egyedül en vagyok a felelős.

Végül, de nem utolsósorban, koszónom családomnak, barátainak és munkatársaimnak, hogy elviseltek mögörva viselkedésemet, amely a kezirat elkeszítésének utolsó szakaszában jó néhány szor jellemezett engem.

Koszónetnyilvánítás

Ez a könnyű nagyjából 2001 nyara óta létezik, ha nem is ebben a formában: a C# Azota azon dolgozatam, hogy a könnyvet frissítsem és aktualizáljam a .NET rövidabban (nyerem? ... de jó!) 2003-as Reference Excellence Awardon a programozási környék kategóriában évek során jutottak a Jolt Award döntőjére (veszettetem... kellemetlen) és a fogadtatásra lett a sajtóban, és ami még fontosabb, az olvasók körében is. Az verziójával karolható. Nagy örömmel es halával tölt el, hogy munkám kedvező es a .NET platform első kiadása úgyanis akkor jelenik meg, a .NET 1.0 Beta 2 verziójával kialakult. Ez a könnyű megfelelően. Eközben limitált darabszámmal megplatorma újabb verzióinak megfelelően. Eközben limitált darabszámmal meg- (Windows Presentation Foundation, Windows Communication Foundation, Windows Workflow Foundation), valamint attíkintést nyújtott olyan különöző technológiákról, amelyek még megfelelne a varrak, és amelyeket sen új fejezetet is tarthatnak, de ezek tulmenően több eddigí rész is jeleniksen jában található legfontosabb változások magyarázzák, másrészt számos teljes- egyreészettől tömörén megismeli a korábbi ismereteket, azaz a .NET 3.5-ös verzio- jelleg a könnyű negevédik kiadását tárja a kezében az Olvasó. Ez a kiadás most LINQ néven ismerünk.

Ahogy a korábbiakban megszokhattuk, ez a kiadás is egyszerte és erthe telekkel csak nagyon kevés olvasóm fog valaha is foglalkozni. termi, így nem töltök töl sok időt az ezoterikus részletek tanulmányozásával, formációk segítségével jól használható softverrendszereket lehessen készí- re. Munakam továbbra is arra fektet a hangsúlyt, hogy a benne található in- jobban hasonlít egy GRC-szisztemához a tanulmányra, mintegy szakkönyv- erettem azokat a szakirokat, aikik úgy adják el a mondanivalókat, hogy az nyelven ismereti a C# nyelvet és a .NET-alapoztatlykönyvtárat. Soha nem

BEVEZETÉS

Az olvasók feladata példáig, hogy a megszerezett tüdőst kepes legyen alkalmazni saját programozási feladatában. Persze egyéretlennü, hogy a projektek meghatározott ismeret. Bízatos lehet benne az Olvasó, hogy ha egyszer szólma az alkalmazottal, minden erről szólva az alkalmazott ismeret. Megérte a könyvbén bemutatott elvüköt, ismerteinek birrekban olyan megérte a könyvbén bemutatott elvüköt, ismerteinek birrekban olyan NET-alkalmazásokat írhat, amelyek használhatók saját programozási körben.

Az *en feladatait*, hogy legjobb tudásom szerint elmagyarázzam a C# programozási nyelvet és a .NET platform alapvető jellegezéseit. Ügyanolyan mindenent megbeszék azért, hogy ellássam a kedvező Olvaszt különféle eszközökkel és stratégiákkal, amelyek lehetővé teszik, hogy a könnyű laptopban Jolly-lehessen a tema tanulmányozása.

Ezért szándékosan kerüljük az olyan mintapéldákat, amelyeket konkréter írásban vagy programozási feladathoz köthetünk: a C#, az objektumorientált programozás és a.NET 3.5 alaposztály konyvtárai ípár alkalmazások-tól független példakön kereszttüli mutatók be. Ahelyett, hogy minden példam datok tömökélegét szolgáltatja, berüsti a számolára, vagy ehhez hasonlók algoritmusokat írunk, az elérhetőbb lesz a felhasználónak. Ez az egyik legfontosabb előny a géometria struktúrával és alkalmazottakkal. Ez az a rész, ahol összetállkozunk.

A szakirkok az emberérek legényes csoportháznak itinák (nekem már csak tudom műszaki tanulmányt készítésé - bárminelyik platformmal történjen is (NET, ZEE, COM stb.) - kelet - egy vagyok koztulik). Tisztában vagyunkazzal, hogy egy szoftverrel rendkívül aprólekos munika, úgyanakkor nagyon specifikus a részleg, a cégek, az ügyfél vagy a tárlykör szerint. Hiszen lehet, hogy epp az elektronikus kiadói paragrafan dolgozunk, vagy éppen esetleg a hagyományosnak, a NASA-nak vagy a legnemesebbek. En a magam részéről gyermekkötőkkel szembeni szoftverrel fejlesztem, számos n-retegű rendszert, valamint projektet a gyögyászati és a penzügyi agazatokban. Majdnem nulla az esetle, hogy annak a Kodnak, amelyet egy-kink ir a munakájhoz, közé lesz aholoz a Kodhoz, amelyet a másikunk ír a sa-

Mi egy csapat vagyunk, az Olvasó és en

Az első részletekben a könyv többi részének a gyakorlati alkalmazására szolgál. A használati részletekkel szemben a gyakorlati alkalmazásban előforduló problémák megoldásai, a kódok részletei, valamint a részletekkel kapcsolatos tanácsok a könyv második részében találhatók.

1. fejezet: A .NET filozófiaja

Az első rész bemutatja a .NET platform alapvető terméseket és számos olyan fejlesztőszokat (konzik, több nyílt forrásokból), amelyeket .NET-alkalmazások fejlesztéséhez használhatunk. Mindeközben néhány laptop C# programozási nyelvi részleteit is megismertedünk.

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

Elsö kötet

A konyv áttekintése

A 4. fejezet befejezéi az alapvető C#-szemléletek meghatározását, a típusú terhelet metodusok leterhözésétől az out-, ref és params kulcsszavakon kereszttüli paraméterek definíálásáig. Vizsgáljuk, hogyan kell adathalmazt letrehozni és kezelni, hogyan kell olyan adattípuszt készíteni, amelynek nincs lehet az értelmezése (a ? és a ?? operátorokkal), és megmutatjuk az eredményes es a referenciátipus kiszövött különbségeit.

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

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

A 2. részben taragyalt témakörök kiemelten fontosak, hiszen az itt megjelenő utánivalókat folyamatosan használjuk függetlenül attól, hogy milyen jellegű NET-szoftveret szándékozunk felhasználni (wébes alkalmazás, asztali grafikus alkalmazás, forrásokkal összhangban, Windows-szolgáltatások stb.). Itt derül ki, hogy hogyan működnek a C# nyelv alapvető objektumorientált programozás (OOP) részleteit is. Ez a rész ismerteti tövábbá, hogy hogyan kell kezelni a futásidejű kivételeket, és bevezet miniket a .NET személygyűjtő szolgáltatásnak részleteitől is.

2. rész: A C# alapvető építőelemei

A tölzeti célfelé, hogy különféle eszközök és technikák ismeretével bővezzük minke a C#-források felületét, amelyet barátulunk. Ezekből számos lehetőség azonban elöl kerül tudnia hűtői a faraszbebol.

2. fejezet: C#-alkalmazások készítése

tem, object, Finalize() metóduson keresztül) foglalkozik. Ez időszakban interfészben kereszttüll) es a vélegesítő folyamatot (a sys-
alapok megismerése után, a fejzett tövábbi része az eldobjható objektumokkal
mazásig járhat, az objektummezőkkel és a system. GetType() típusra szerepet. Az
memoriáit a .NET személyi típusainak a segítségevel. Megérhetők az alkali-
Elnék a résznek az utolsó fejzete azt tárnyálja, hogy hogyan kezeli a CLR a

8. fejezet: Az objektumok elektromos

hezük a figyelmen kívül hagyott kivételek között.
Studio 2008-ban található ezeket, amelyek segítségevel hibakeresést végez-
kivételek közötti különbségeket is. Megismertetünk többéle, a Visual
(try, catch, throw es finaliy), hanem az alkalmazászintű es rendszerszintű
jellegrő problémák megoldására szolgáló C#-klasszakat ismerjük itt meg
ezelés segítségevel a futási időben keletkezett anomáliaikkal. Nemcsak az ilyen
EZ a fejzett elősorban azt mutatja be, mit kell tenni a strukturalis kivételek-

7. fejezet: Strukturalit hibakerzelés

fontosabb alaposszabalyának, a system. object osztálynak a szerepeit is.
szetet. Végül, de nem utolsó sorban, ez a fejzett ismerte a .NET platform leg-
(es az absztrakt alaposszabalyok) szerepét, valamint a polymorf interfesz terme-
hözhatók letre. Megnezzük a virtuális metodusok es absztrakt metodusok
vizsgájuk megl. ezek a segítségevel összehozza osztalystpusok családját
Ebben a részben az OOP maradék pillerét (származtatás es polimorfizmus)

6. fejezet: A származtatás es a polimorfizmus

ges típus definicijait es a C# XML-kódjának dokumentációs szintaktikáját.
látható robustusosszabaly-típusokat létrehozni. Végezetül ismertetjük a részle-
donságok, statikus tagvaltozók, konstansok es csak olvasható fajlok haszná-
fejzett tövábbi részben azt nézzük meg, hogyan kell konstruktorok, tulaj-
rozzuk az OOP pillerét (beágyazás, származtatás es polimorfizmus), majd a
Ebben a fejzettelben az objektumorientált programozással (OOP) ismerkedhе-
titük meg a C# programozási nyelv használatán kereszttüll. Először megírhat-

5. fejezet: Egységbé zárt osztalystpusok definíciója

A II. telezethék az a célja, hogy érthetővé tegye a metodusréferenciát-típusokat. Légegyszerűsítve: a .NET-metodusréferenciák olyan objektum, amely az alkálmazás más metodusaira mutat. Ez a mintha olyan rendszerek készítését teszi lehetővé, amelyek több objektum összekapcsolását engedik meg kétiranuyt eggyütteseknél. A .NET-metodusréferenciák vizsgálata után megismertekdink a .NET event külcsszavaval, amelyet arra használunk, hogy szükséges esetben meghívjuk a C# 2008 lambda-operátorát (\Rightarrow), és feltájíuk a kapcsolatot a megvizsgáltuk a nyers metodusréferenciák programozásának kezelését. Végezetül szükséges a nyers metodusréferenciák programozásának C# 2008 lambda-operátorát (\Rightarrow), és feltájíuk a kapcsolatot a megvizsgáltuk, a nettelén metodusok és a Lambda-kifejezések között.

1. fejezet Metodusréferenciák, események és lámdbák

10. fejezet: Gyűjtemények és generikus típusok

Eltérők a telezethetők az anyagokat az interfejszalapú programozást is magába foglaló objektumalapú fejlesztés alkotta. Megismertetők belül, hogy hogyan kezeli olyan tipusokat definícióit, amelyek többszörös viselkedést támogatnak, hogy minden objektummalapú fejlesztés alkotta. Megismertetők belül, hogy hogyan kezeli olyan tipusokat definícióit, amelyek többszörös viselkedést támogatnak, hogy minden telezethető az interfejszalapú készítéshez szükségesek.

9. fejezet: Interfeszékek használata

A környéki része ismerte a C# nyelv haladóbb jellegrégi (de gyáncsnak négyon fontos) elvét. Az interfejszek és metodusriferenciák vizsgálataval belfelezuzzuk a .NET típusrendszerevel való ismerkedést. A tövábbiakban betekintést nyerhetünk a generikus típusokról és a számos új C# 2008 nyelvi elem szerepében, valamint a LINQ-ba is.

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

A negyedik rész a .NET-szerelvénnyel formázott részleteit taglaja. Nemcsak azt objektumkörnyezetet, a köztesz nyelvi kódot és a dinamikus szerelevenyeket.nak modjával is. A kezdői felületek néhány haladó tematérimentek, például az szé a .NET-attribútumok szerkezeti es többszáz alkalmazások letrehozásá- de azt is, hogy mi a .NET bináris kép belső összeállítása. Ebben a részben lesz tanulhatjuk még, hogyan kell .NET-kódok nyilvántartákat telepíteni es konfigurálni, nemcsak azt

4. rész: Programozás .NET-szerelvénnyel

XML-en kereszttől) használhati. sokon (a LINQ az ADO-n kereszttől) es XML-dokumentumokon (a LINQ az kor azt vizsgáljuk meg, hogyán kell LINQ-kifejezésekkel relációs adatbázis-pusok) fogjuk tudni alkalmazni. Ez a tudás nekülözhetetlen lesz akkor, ami- gőtsegevel a LINQ-kifejezésekkel adattárolunk (tömbök, kolllekciók, egységi tresszünk. Megtanuljuk a LINQ objektumokra történő alkalmazását, ennek se- cílpontra alkalmazhatunk, hogy a szövegűkkel számlos LINQ-toségeink nyilván tippeseket lekérdezésére írni, amelyeket számos LINQ- résznekk. Ahogy a felület kezdői részben kidérit, a LINQ segítségével lehe- rált lekérdezés), amelyet nyúgoda tan nevezhetünk a .NET 3.5 legérdekesebb Ez a felület ismerteti a LINQ-et (Language Integrated Query - nyelvi integratív-

14. fejezet: Bevezetés a nyelvre ágyazott lekérdezésekbe (LINQ)

objektumorientált szintaxis használatát. az automatikus tulajdonoságok, a bővíti módszerek, a névtelen típusok es az valtozók implicit tippessel történő ellátásának a szerkezet, a részleges módszusi- nálatait (erről a 14. fejezetben többet is megtudhatunk). Megismertük a lokális amelyek közül jó néhány arra szolgál, hogy lehetővé tegye a LINQ API hasz- A .NET 3.5 kiadásában a C# nyelv számos új programozási eszközöt bonyol- valók. Ez a fejezet a C# programozási ismereteket mélyít el számos haladó progr- amozási technika segítségével. Megismertetjük, hogyan kell operátorokat tüntethetni, es hogyan kell a típusaink számára egyédi konverziós rutinokat írni (akkor implicit, akár explicit formában). Megtanulhatunk azt is, hogy ho-

13. fejezet: C# 2008 nyelvi újdonságai

C-stílusú mutatókat "nem biztosítás" kódörnyezetben. gyán kell letrehozni es használni a típusindextípust, es hogyan kell kezelni a típust (akkor implicit, akár explicit formában). Megtanulhatunk azt is, hogy ho- amelyek közül jó néhány arra szolgál, hogy lehetővé tegye a LINQ API hasz- A .NET 3.5 kiadásában a C# nyelv számos új programozási eszközöt bonyol- valók. Ez a fejezet a C# programozási ismereteket mélyít el számos haladó progr- amozási teknika segítségével. Megismertetjük, hogyan kell operátorokat tüntethetni, es hogyan kell a típusaink számára egyédi konverziós rutinokat írni (akkor implicit, akár explicit formában). Megtanulhatunk azt is, hogy ho-

12. fejezet: Indextípuk, operátorok es mutatók

Ez a pélezeit azt mutatja be, hogy hogyan kell többszáz alkalommal szoktatni a felhasználókat a megértesítésekhez alkalmazhatunk. A feljelzés elégén általában hogyan tanulmánytak a módszertanikai technikákat, amelyeket a szablonos kód kezelésére használnak. A pélezeit eljelzésekben általában a .NET módszertanikai pélezeitet használjuk.

18. fejezet: Többszálú alkalmazások készítése

Készítésének módjai.

Ezek alapozzák meg a következő feljezet temóját: a többszázta alkalmazások mazásástarományok és a kontextusbeli határok közötti kapcsolat illusztrálása. A hasitható fajlok felépítését ismerte el. Az elsődleges cél a felgyamaik, az alkali-

17. fejezet: Folyamatok, alkalmazásokatrományok és objektumkörnyezetek

A 16. feljelzésben foglaltak az ismerekedést a .NET-szerelvénnyekkel. A syss-tem. Reflektion névter használatával kapcsolatban azt vizsgáljuk, a futási időben hogyan valósul meg a típusmeghatározás folyamata. Az ílyen jellegrű típusok használatával lehetőségeink van olyan alkalmazások letrehozására, amelyek futás közben kepesek a szerelvénnyek metadatát olvasni. Kiderül, hogyán lehet a kész kötés segítségével futási időben dinamikusan beoltalni esetleges változtatásokat. A feljelzett konstrukciókban azonban nem mutatunk egyszerűbb alternatívát a szerepeknél ismertetésre. Az egyes temakörök elegendő - .NET-attribútumok szerépének ismertetése. Az egyes temakörök hasznosságának szemléleteit ismerhetjük meg a fejezet könyvtájaihoz hasznosítva.

16. fejezet: Tipusreflexio, keszi Kotcs Es attributumalapú programozás

Bár so megközelítésben a szerelteny nem más, mint egy felügyelet * . Így az exé bináris fajl leírásra szolgáló kifejezés. Am a .NET-szerelvenyek töre- műve sokkal több ennel. Ebben a fejezetben megtudjuk, mi az egységekkel többfajlos szerelvenyek közötti különbség, és, hogy ezeket hogyan kell kezeli a telepíténi. Megvizsgáljuk, hogyan kell privat es megosztott szerelvén- nyeket konfigurálni XML-alapú *. config fájlokkel és kibocsátó hozzárendelésekkel. Mindeneközben meg tudhatunk, hogy minden a CAC (Global Assembly Cache - globális szerelvenytár) belső struktúrája, és mi a .NET keretrendszer konfigurációs sereďprogramjainak a szerkeze.

15. fejezet: A .NET-szerelvénnyek

A system. I0 névter segeségevel lehetőségeink van a számítógép fájl- és környvtarszerekzeteinek eleresése. Ebben a feljelzésben megismertük, hogy an- tikell programoztan könnyvtárrendszer letrehozni (es torolni), valamint, ada-tolakat különöbözö folyamokba (fájlalapú, sztringalapú, memoriatalapú stb.) be-, illetve kímrozgatni. A feljelzett utolsó része megvizsgálja az alkülöntetit trólokat, amelynek segítségével lehetőségeink van biztonságos helyen külön tárolni az adatokat függeltenül attól, hogy a célgép biztonságát bennlőttetési. Hogyan működhet a rendszer? I0. Isolated API bizonyos elővet, attékintést kell kap-nunk a Kodhozzaférés-szabályozásról (Code Access Security - CAS).

20. fejezet: Fájlműveletek és elküldönötött tárolas

5. rész: Bevezetés a .NET alaposztálykörnyzetáraiba

Masdik kötet

A 4. rész utolsó fejezetének szerepe kétféle. Először a korábban részletezésben megvizsgáljuk CIL szintaktikáját és szemantikáját, majd a system. Reflection. Emiatt néhány szerelőt mutatunk be. Ezeknek a típusoknak a használataval lehetőségeink nyílik olyan szoftver készítésre, amely képes.NET-szerelvényeket futásidőben generálni a memoriából. Az olyan szerelvényeket, amelyeket a memoriában definiáltunk es futtatunk, dinamikus szerelvénynévenk nevezzük.

19. fejezet: A köztes nyelv (CL) és a dinamikus szerelemeinek

Az aszinkrontúggyűny-hívásokat. Majd megvizsgáljuk a számokat. Ezek között számos olyan van (Thread, Threadstart), ben található típusokat. Ezek közül történő szakképzést. Nagyban megkönnyíti a felhasználói felülettel a szakképzést.

A 24. Jelölje a LINQ programozási modellt mutatja be, pontosabban a LINQ "az objektum irányába" részét. Megvizsgáljuk, hogyan kell LINQ-lekérdezé- kumentumokon. Mindeközben megtanuljuk az adatkörnyezets-objektum, va- laminth az sQLmetá. A LINQ-tamogató funkciót.

24. fej zet: A LINQ API programoz sa

Ez a feljelzés az adatbázis-kezelés különöző módokon történő manipulálási lehetőségeit tükrözi tovább, így minden adatbázisban használható. Például a Windows Forms API fejlesztői elemeikhez kapcsolni, olyanokhoz, mint például a Windows Forms

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

Az adatbazisokról szólok! Két félézet kozúj az elsohán az ADO.NET programozási API-rol kaputunk ismertetik. Ez a rész bevezeti a .NET adatszolgáltatás szerepét, valamint azt, hogy hogyan kell a relációs adatbázissal komunikálni. Kéthi az ADO.NET török kapisztáin keresztül. Ez a kapcsolati objektumokat, a parancsobjektumokat és az adatolvasó objektumokat jeleníti. Ez a rész azonban csak az adatleirei könnyvtárakat, amelyeket majd a könnyű tövábbi része azokat az adatleirei könnyvtárakat, amelyeket majd a könnyű tövábbi része fejezet mutatja be az egyedi adatbázis letrehozásának módját, és ismerteti a fejezetet.

22. fejezet: ADO.NET 1. rész: Az előkapcsolat

71. fejezet: Bevezetés az objektumsortisták világába

Az eredetit asztali grafikus felhasználói felület, amely a .NET platformra együtt kapható, a Windows Formok. Ez a felület ismerteti a felhasználói felület szereit és menürendszereket késztíeni. Felületek tövábba az interakcióval számaztatás szerépet, és azt, hogy hogyan kell kezdeményezni adatot rajzolni a rendszerben. Drawing nevető segítségével. A témakör illusztrációkra a felület végezhetők.

27. fejezet: Windows Forms-programozás

6. rész: Fehasznalói felületek

A .NET 3.0 a WCF mellett bevezetve a WF API-t is, amely lehetőséget ad arra, hogy inkasoljuk a műveleteket definícióink, futtatásunk es monitorozzunk. Ennek segítségével komplex üzleti folyamatoik modellezhetők. Még tanuljuk a WF alatlansos célt, ahogy azt is, hogy mi a szerepe az aktivitásknak, a munkafolyamat tervzónék, a munkafolyamat futtatási motorok, és a munkafolyamat-engedélyezett forrásokkal könnyítünk.

26. fejzett: A Windows Workflow Foundation – Bevezetés

arra, hogy szimmetrikus módon, függelékenyül az alatta található részegék fel-
építésétől elosztott fejlesztések. Ez a részlet felátfogja a WC-szol-
ágatások, -gazdagépek és -klinensek készítésének lehetségeit. A WC-szol-
ágatások rendkívül flexibilisek, a gazdagépek és a klinensek XML-alapú kon-
figurációk feljölkötéséhez használhatók, hogy deklaráció módon adjának meg ci-
meket, kötéséket és szerződéseseket.

25. fejezet: A WCF

Az utolsó feljelzés a WPF targyálását hárrom latszolag tüggetlen téma vizsgála-tával zárja. A WPF gráfikus rendszerrel szolgáltatásnak általában szüksege van arra, hogy egységi erőforrásokat definíáljunk hozzá. Ezekenkéz az erőforrásoknak a használatával lehetőségeink van egységi WPF-animációk készítése, és grafikák, erőforrásokat és animációkat alkalmazva egységi temák kezére, és színesekben a fejezet utolsó részében azt illusztráljuk, hogyan temák összegezésével WF-kalkulációk számára. A különöző színtestet fogunk tudni megoldani WF-funkciók alkalmazása egyedi temák kezére, és színesekben a fejezet utolsó részében azt illusztráljuk, hogyan temák összegezésével WF-kalkulációk számára. A különöző színtestet fogunk tudni megoldani WF-kalkulációk számára. A különöző színtestet fogunk tudni megoldani WF-funkciók alkalmazása egyedi temák kezére, és színesekben a fejezet utolsó részében azt illusztráljuk, hogyan temák összegezésével WF-kalkulációk számára.

30. fejjezet: WPF 2D grafikus renderelés, erőforrások és temák

Ebben a fejezetben megtudjuk, hogy hogyan kell dolgozni a WP-vezetőknek tartalommodelljevel, és érmitunk számos vezetőlelemekkel kapcsolatos témáit is, például a függősségi tulajdonsgokat és irányított eseményeket. Jó nehány WP-vezetőlelem használatát bemutatjuk, s szintén ebben a fejezetben magyarázzuk el az elrendezésekhez, -vezető utasítások és a WPF-adat-

29. fejezet: Programozás WPF-vezérlőelemekkel

A .NET 3.0 teljesen új gránthús felhasználói felületet vezetett be, amelyet WPF-nek nevezett el. Röviden, a WPF ki magaslatban interaktív és medíaban gazdag front endnek kezeltetését teszi lehetővé aztól alkalmazásokhoz (es indirekt módon webalkalmazásokhoz). A Windows Formssal ellentében, ez a telepített felhasználófelület-kerevenrendszer számos külcsontosságú szolgáltatást (2D-s 3D-s grafika, animációk, gazdag dokumentumok stb.) integrál egyszerűen gyakorlati objektummodellbe. Ebben a részben a WPF-objektumokkal szembenekedést a WPF-fel és a kiterjesztett alkalmazásokkal nyelvvel (Extendable Application Markup Language - XAML). Megtanuljuk, hogyan kell XAML-mintes, csak XAML-t-használó es a kettő kombinációjából felépülő WPF-programokat létrehozni. A részben végén kezeltünk egy egyszerű XAML-nézetegyetl, amelyet a to-

28. fejzett: A WPF és az XAML

`web.config fail seguelevel.`

Ez a féljelző kiterjeszeti az eddigi ASP.NET-ismertetőnek, úgyanis megvizsgálja, milyen különbsöző módon lehet állapotot kezelni a .NET alatt. Ahogy a klasszikus ASP, az ASP.NET is egyszerűen teszi lehetővé olyan leírásokat, amelyekkel a webalkalmazásunk futásidéjű viselkedését a rendszerbeli (a Global.asax fájlban belüli), valamint azzal, hogy hogyan kell dinamikusan megváltoztatni a webalkalmazásunk futásidéjű viselkedését a ASP.NET alatt, mégismérkőzünk a szabám. Ha a többi alkalmazás-szintű változók használata is. Az ASP.NET azonban bevezetett egy új alkalmazás-szintű változót, amelyen modokon lehet állapotot kezelni az ügyorstotrárt. Ha már tudjuk, milyen módon lehet állapotot kezelni az ASP.NET alkalmazás-szintű változókat, miyennek módosítani lehet a szabályozásokat.

33. fejezet: ASP.NET állapotkezelési technikák

Ez a féllezet azokkal a vezérlövelmekkel foglakozik, amelyek a belső vezérlo-
fárat töltik fel adatthal. Megismertük az alapvető ASP.NET webelek vezérlöveleme-
ket, többek között aellenörző vezérlövelmeket, a belső oldalai navigációs vezér-
lövelmeket, többlek között a hagyományos vezérlövelmeket, a belső oldalai navigációs vezér-
lövelmeket, többlek között a hagyományos vezérlövelmeket. Ügyancsak ennek a féllezet-
telélemeket és a különöző adatkörök mivételeket. Ugyancsak ennek a féllezet-
telélemeket és a feladatait, hogy bemutassa a mesteralak es az ASP.NET temamotorja-
nak a szerepet, amely a hagyományos stiluslapok szerveroldali megfelelője.

32. fejjezet: ASP.NET-vezérlőelemek, -témaik és -mesteroldalak

Ez a fejezet vezet be benyútnak az ASP.NET használatával történő webes alkalmazásfejlesztésbe. A kiszolgálóoldali szkripteket felváltók az igazi objektumorientált nyelvek (mint a C#, VB.NET és hasonlók). Bemutatjuk egy ASP.NET weboldal készítését, valamint az alatta található programozási móddelit és az ASP.NET egyéb kulcsfontosságú elveit, például azt, hogy hogyan kell webkiszolgálati válaszszám, valamint a web-confinig fajlok használatát.

3.1. fejezet: ASP.NET weboldalak készítése

A 7. rész az ASP.NET programozási API használatával készített webes alkalmazásokat vizsgálja. Az ASP.NET szándekeisan úgy lett kiállítva, hogy az asztali felhasználói felület leterhözését modellezze, úgy, hogy szabványos HTTP kérésekre/válaszokra helyez egy eseményvezérelt, objektumorientált kere�trendszerrel.

7. rész: Webes alkalmazások felületei ASP.NET

Végeül, de nem utolsó sorban a B függelék a NET nyitott forrásokhoz ímplementációival, a Monoval foglalkozik. A Mono segítségevel lehetőségeink van gázdaság tulajdonsgókkal rendelkezni. NET-alkalmazás letrehozására, telepítésére és futtatásra különöző operációs rendszerekben, mint például a Max OS X, Solaris, AIX es számos Linux-distroval, mint például a Max kompatibilis Microsoft .NET platformaval, így már tisztaban vagyunk az üzleti alkalmazások fejlesztésében.

B függelék: Platformfüggelén .NET-felhasználók a Monovall

AZOK, akik már programoztak Windows operációs rendszerei alatt a .NET platform használata előtt, nagy valósztályosból ismertek a Component Object Model (COM). Noha a COM-nak és a .NET-nek semmi közé egymásba azon túl, hogy mindketten a Microsoft szervezénnyel, a .NET Platform teljes mevetereit biztosítják (rendszer, RunTime, InteropServices), hogy .NET-softverek COM-komponenseket használjanak, és fordítva. Ez a tüggelek az együtt-működési reteget mutatja be egeszben részleteiben; ez a témá azért nagyon fontos, hogy a korábban meglévő programjainkat át tudjuk menteni az újonnan készített .NET-alkalmazásainkba.

A frissek: A COM és a .NET együttműködése

A konv^y utolsó része két fontos téma volt vizsgálat meglétén, amelyek valójában nem illusztráltak a lezárásokat szerevben a könnyű felépítésről, ezért a függelékhez kerültek. A függelék kifejezési C#-pal és.NET platformmal kapcsolatos ismertetéinek, a megvizsgáljuk ügyanyi, hogyán kell a.NET-felületet a Windows operációs mazásainkba, valamint hogyan kell a.NET-regebbi kodolat integrálni a.NET-alkalmasrendszer családján kívül használni.

8. rész: Függelékek

Forráskód Ez a forráskód-megjelenés egy kiválásztott könnytárra hivatkozik.

2008-ba további vizsgálatok es modosítások céljából: is, arra utal, hogy a szoban forgó pelekát be lehet tölteni a Visual Studio Fügyletm: a fejlesztéken a Forráskód megjelenés, mint amilyen az alábbi *.zip fájlit. A forráskódokat fejleszteneket osztottak fel. ket, és keresztsük meg a megfelelő címet, és, töltünk le az önmagát kicsomagoló http://www.apress.com címét, várasszuk ki a Source Code/Download linket. Így szerűen géljük be a jának Source Code/Download részlegéről. Egyeseknél előfordulhat, hogy a fejlesztőben levők is) szabadon és azonnal elérhető az Apress honlap-további fejlesztés szerepében szabadon (az ot szabadon letölthető A könnyben szereplő összes forráskód-mintapélda (az ot szabadon letölthető

A könny forráskódjának igénylése

SZAK Kiadó megjegyzése). Az itt található linkek kattintva digitális formában letölthetik a könnyt to-vabbí fejlesztőit, ha sikertörténetet követően szövegesből végletiszterűen felettes kérésekre. (A letölthető fejlesztek csak angol nyelven érhetők el - a

<http://apress.com/book/9781590598849>.

A könny jelen kiadása nem tartalmazza ezt az ot fejlesztőt. Ennek az a leg-több oka, hogy a .NET 3.0-ban a WCF es a WPF API-k rendre a .NET-remo-típus XML-webszolgáltatások es a Windows Forms API-k bőrkösei. Ha me-lyebben szeretnének elmerílni a Windows Forms világában (azon tul, amit a 27. fejleszt nyújtani tud), vagy még akárjuk nézni, hogyan kell használni a szerűen csak meg kell nézni az Apress honlapján a megfelelő részt: (öröksegi hagyott) .NET-remotinigot es az XML-webszolgáltatás-API-t, egy-

Service projekt sablonjaihoz a segítségevel. manyos XML-webszolgáltatások készítésével foglalkozott az ASP.NET Web tegének (rendszer, RunTime, Remoting es más) es a végül egy fejleszt a .NET remotinig re-vézelők vizsgálatával (egytagú), es a másik fejlesztések szenteltünk (egyedi tarthatmazat), amelyeket a Windows Forms fejlesztéseknek (egyedi tollthelek további ot fejlesztet. A könny korábbi verziói hárrom olyan fejlesztet-azoknak, akiknek a 33 fejleszt es a két függelék nem lenne elég, szabadon le-

Öt szabadon letölthető fejleszt - meg több információ

Andrew Troelsen

Minden jöt!

Nos, akkor, koszönöm, hogy megvásárolta a könnyűmete (vagy legalábbis, gyere). Reméllem, elvezetni fogja a kedves Olvasó a tanulmányozását, és hasz- hagy belenéz a könnyvesboltba, miközben azon gondolkodik, hogy mege- Nos, akkor, koszönöm, hogy megvásárolta a könnyűmete (vagy legalábbis, gyere). Reméllem, elvezetni fogja a kedves Olvasó a tanulmányozását, és hasz- hagy belenéz a könnyvesboltba, miközben azon gondolkodik, hogy mege- valasztozok egy-két heten belül, akkor ez nem jelenít azt, hogy nem akarok szólni, sajnos, ahogy mindenki, időnkent nagyon előglat vágynak. Ha nem csém van, valahol ép párval.

Azaz, mégproblémánk minden levéltári lehetsége szerint vala- szovaggal: atrolesten@mitretech.com.

Ha bármiilyen kérdez merrit fel a könnyűhöz tarozó forrásokkal kapcsol- nem kerül a kerelelén levelek közé, kérlem, tegyé a tárnymezőbe a „C#FE” levelet a következő e-mail címre (ahogy biztosak legyünk benne), hogy a levele csak véleményt szereesse nyilvánítani a .NET platformról, nyugodtan küldjön lapban, vagy szíksege van valamelyik példa tiszazzára, vagy egyszerűen szövegeket): atrolesten@mitretech.com.

Elerhetőségem

Elképzelhető, hogy a könnyű végigolvassa során nyelvtani vagy forrásokba- hibákat vél feldezi a kedves Olvasó, bár reméltem, hogy erre nem kerül sor. Ha mégis, elmezeit kerek. Az aktuális hibálistát az Appress weboldalról csak itt található az elerhetőségem, amelyen értesíteti tudnák.

A lehetséges javítások

Megjegyzés A magyar nyelvű változatban a forrásoknak megjegyzéseit lefordítottuk. A lefordított forrásoknak nem állt modunkban, így azok teljesen angol nyelvűek. Ilyen módon in- kább megfelelnék a lefordított kodok használatát Céjlának – a SZAK kiadó meggijegyzése.

Egy szírűen nyissuk meg a *.sln fájlt a megfelelő alkonyvatrabol. Ha nem használjuk a Visual Studio 2008-at (lásd a 2. fejezetet töltse be a fellkínált forrásokkal fejlesz- tői környezeteket), akkor manuálisan töltse be a fellkínált forrásokkal fejlesz- a valasztot fejlesztő eszközbe.

könyvtáraiba

alaposztály-

a.NET

Bevezetés

5. rész

Fájlműveletek és elszigetelt tarolás

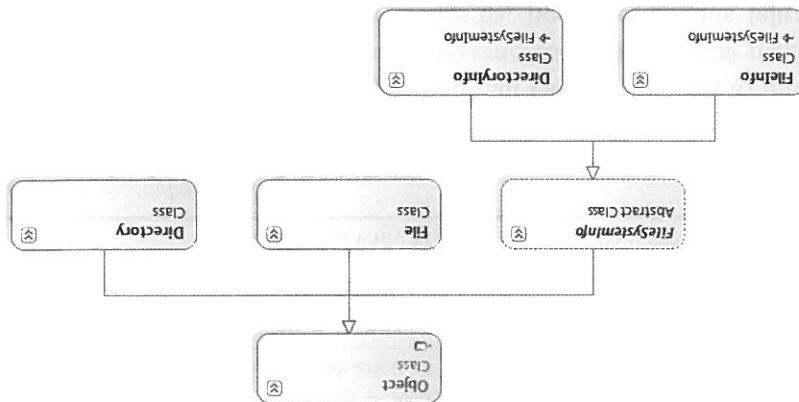
H U S Z A D I K F E J E Z E T

A NET keretrendszerben a fajlalapú (és a memoriálapú) b-e- és kvítelei (I/O szolgáltatásokat biztosító alapvető osztálykoni várak helye a system, jó név-ter. Minth minden névter, a system, jó is szamos osztályt, interfészét, felosrotlt tipusát, struktúrát és metodusreferenciát hataroz meg, amelyek tulajomó többsegé az mscorelib, d11 fajlban található. A system, jó névter tövábbi tagjait a system, d11 szerelvénnyre hívhatkozunk, így az itt említett típuso- telemzes szereint mindkét szerelvénnyre hívhatkozunk, így az itt említett típuso- kat tövábbi tennedök nekül, azonmáll használhatatlályuk).

A System.IO newer

Nem absztrakt	I/O osztálytípusok	Jelentés
A számítógépnek minden részlete teljesen elérhető.	Büfferesztream	Bytet-folyamok ideiglenes tárolójaként szolgál, ezeket a részleteket szabványosan más tárolókra lehetővé.
Ezekkel a számítógépen található fajlok módosíthatók.	Fájloinfo	A Fájl típusa a funkcionális statikus tagok segítségével leírja, hogy melyik részre módosítható.
Ezekkel a számítógépen található fajlok módosíthatók.	File, Fájloinfo	A Fájl típusa a funkcionális statikus tagok segítségével leírja, hogy melyik részre módosítható.
Véletlen fájlhozzáférés (pl. keresési kezelésére).	Fájstream	Véletlen hozzáférés biztosít a memóriaiban tárolt adat-
Egy megalapított könnyvtárban található különböző meg-	Fájlystemwachter	Egy megalapított könnyvtárban található fájlok mege-
lehetővé, és az adatokat byte-folyamként jeleníti meg.	Fájlystem	Könnyvtárlétreseit az adatokat byte-folyamként teszi le-
Véletlen fájlhozzáférés (pl. keresési kezelésére).	MemoryStream	Véletlen hozzáférés biztosít a memóriaiban tárolt adat-
Egy megalapított könnyvtárban található különböző	Path	Olyan számítási rendszertípusú objektumokon hat végre műveleteket, amelyek platformfüggetlen fájl-, illetve könyvtárlétreseit utárolják tartalmaznak informaciót.
Ezekkel fájlokban tárolhatunk (illetve azokból visszaol-	StreamWriter, StreamReader	Ezekkel fájlokban tárolhatunk (illetve azokból visszaol-
vasztunk) szöveges információkat. A végén fájhoz-		zászerest nem támogathák.

20.1. ábra: A File- és Directory-közötti típusok



A system. IO névter negy típusa segítségével teszi lehetővé a számlítógep környvtársi funkcióinak és különálló fajtájainak kezelését. Az elso két típus, a directory és a file, különöző statikus tagokon keresztül biztosítja az objektumok leírását, törlését, másolását és mozgatását. Az ezekhez szorosan kapcsolódó FileInfo és DirectoryInfo típusok hasonló működést tesznek el-erhetővé példányaizintől meuduoskál (így ezeket a new kulcsszavval le kell foglalni). A 20.1. ábrán látható, hogy a directory és file típusok közvetlenül a system.object osztályt bővíti ki, míg a DirectoryInfo és FileInfo osztályt a logikai). A 20.1. ábrán látható, hogy a directory és file típusok közvetlenül a system.object osztályt bővíti ki, míg a DirectoryInfo és FileInfo osztályt a logikai).

A Directory(info) és File(info) típusok

Az implement felsorolt konkrét osztálytípusok mellett a system. IO több felsorolt származott számlára. A kezeltíkeben számos ilyen típusot bemutatunk. Az típusokat olyan absztrakt osztályt (stream, TextReader,TextWriter stb.) is definiál, amelyek egy megosztott polimorf interfeszit definálnak az osztályoknak. Ez a típus a Stream osztályt azonban nem fizikai fajlhoz, hanem a Streamwríter, Streamreader és Textwríter ter típusokhoz hasonlóan ezek az osztályok is szöveges információkat kezeli, ha törölök azonban nem fizikai fajlhoz, hanem sztringpufferre.

20.1. táblázat: A System.IO névter külcsfunkciósai tagjai

Nem absztrakt	I/O osztálytípusok	Jelentés
StringWriter, Streamwriter, StreamReader, StreamWriter	A Streamreader és Streamwriter ter típusokhoz hasonlóan ezek az osztályok is szöveges információkat kezelnek, ha törölök azonban nem fizikai fajlhoz, hanem sztringpufferre.	

A FileSyste mindinfo osztály definíciója a Delete() metódust is. Ez a származtatott típusok valósítják meg azért, hogy egy adott fájl vagy konviktatot törlni lehessen a merevlemezről. Az attribútumok lekérdezése elöl tövábbá meg- hívhatók a Refresh() metódust, amely biztosítja, hogy az aktuális fájlra töltött részről minden információ megtalálható.

20.2. táblázat: A FileSyste mindinfo tulajdonságai

Tulajdonság	Jeleintés
Exists	Visszaadja a fájl létezőképességét, hogy egy adott fájl vagy konviktat.
Attributes	Visszaadja a fájl tulajdonságait, amelyeket a FileSyste mindinfo tulajdonságai tartalmaznak.
CreationTime	Visszaadja vagy beállítja az aktuális fájl vagy konviktat létrehozásának idejét.
Extension	Visszaadja a fájl kiterjesztését.
FileName	Visszaadja a fájl vagy konviktat teljes elérési utvonalaát.
LastAccessTime	Visszaadja vagy beállítja az aktuális fájl vagy konviktat legutolsó elérésének idejét.
LastWriteTime	Visszaadja vagy beállítja az aktuális fájl vagy konviktat utolsó módosításának idejét.
Name	Az aktuális fájl vagy konviktat neveit adja vissza.

A direktoriyinfo és a FileSyste mindinfo absztrakt FileSyste mindinfo osztálytól öröklök számos képességeket. Legnagyobb részt arra használjuk FileSyste mindinfo osztály tagjait, hogy lekérdezzük egy fájl vagy konviktat általános tulajdonságait (pl. a létrehozás idejét, s különöző attribútumokat stb.). A 20.2. táblázat felosorolja az eredményesből alap tulajdonságokat.

Az absztrakt FileSyste mindinfo osztály

Alábbiakban elmondható, hogy a FileSyste mindinfo és a Direktoriyinfo osztályokat célszerűbb használni, ha egy fájtorl vagy konviktatról részletek információkat szeretnék megtudni, mivel a FileSyste mindinfo osztályban elérhető tulajdonságokban több használható, mint a Direktoriyinfo osztályban.

20. fejezet: Fájlműveletek és elszigetelt tárrolás

A második példában azt feltételezzük, hogy a konstruktornak átadott utvonalon találunk műveleteket végrehajtaní, a rendszer egy származéka. Az adott konstruktor megszabott módon kívánta, hogy a konstruktor a Create() metódust:

```
DirectoryInfo dir2 = new DirectoryInfo("C:\Windows");
// Az aktuális konnyvtárhoz kötés.
// A C:\Windows konnyvtárhoz kötés.
// Az aktuális konnyvtárhoz kötés.
```

A DirectoryInfo típus használatahoz az elérési útvonalat kell megadunk konstruktorparaméterként. Ha az aktuális konnyvtárhoz (pl. a fürt alkalmazás konstruktorához) szeretnénk hozzáérni, használjuk a „.” jelölést. Az alábbiakban bemutatunk néhány példát:

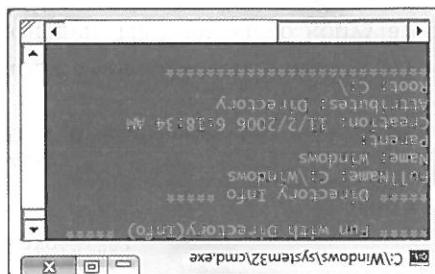
20.3. táblázat: A DirectoryInfo típus kulcsfontosságú tagjai

Tag	Jelentés
Root	Megkapja az elérési útvonal gyökerétől.
Parent	Az adott elérési útvonal gyökerétől adja vissza.
MoveTo()	A konnyvtárat és a tartalmat új elérési útvonalra helyezte át.
GetFiles()	Fájlokat felvesz a megadott útvonalról.
GetDirectories()	Visszad egyszerűen az aktuális konnyvtár összes alkonyvtárat.
Delete()	Egy adott konnyvtárnak és teljes tartalmának a törlése.
CreateSubdirectory()	megadott útvonalon.
Create()	Egy konnyvtár (vagy több alkonyvtár) létrehozása a Create(),

Az első, I/O műveletekkel kapcsolatos típus, amelyet közlebbről megvizsgáltunk, a DirectoryInfo osztály. Tagjainak a segítségevel létrehozhatunk, átnevezhetünk, törlhetünk és kiiktathatunk konnyvtáratokat az alkonyvtárat. Az osztálytól (FileSystemInfo) örökölt funkcióit használva tölthetjük le a direktoriába. Ez a 20.3. táblázatban feltüntetett kulcsfontosságú tagokkal rendelkezik.

A DirectoryInfo típus használata

20.2. ábra: Információk megléte a Windows-környezetben



```

    static void Main(string[] args)
    {
        Class1.Program
        {
            static void ShowWindowsDirectoryInfo()
            {
                Console.WriteLine("***** Fun with Directory(Info) *****\n");
                ShowWindowsDirectoryInfo();
            }
            static void Main(string[] args)
            {
                string s = "A konnyvtárrinformációk kifirásá.";
                DirectoryInfo dir = new DirectoryInfo("C:\Windows");
                Console.WriteLine("***** DirectorY Info(0:C:\Windows) : ");
                Console.WriteLine("***** Full Name: {0}", dir.FullName);
                Console.WriteLine("***** Name: {0}", dir.Name);
                Console.WriteLine("***** Parent: {0}", dir.Parent);
                Console.WriteLine("***** CreationTime: {0}", dir.CreationTime);
                Console.WriteLine("***** Attributes: {0}", dir.Attributes);
                Console.WriteLine("***** Root: {0}", dir.Root);
                Console.WriteLine("***** Writeline(Root) : ");
                Console.WriteLine("***** Writeline(Root) : ");
                Console.WriteLine("***** Writeline(Root) : ");
            }
        }
    }
}

```

```
// kosszuk hozza egy nem Letezo konvijatrhoz, majd hozzuk letre.  
DirectoryInfo dir3 = new DirectoryInfo(@"C:\MyCode\Testing");  
dir3.Create();
```

Hátha füttetők az alkalmazásról, akkor a 20.3. ábrához hasonló lista teljesítésére vonatkozik.

```

static void displayImageFiles()
{
    directoryInfo dir = new DirectoryInfo("C:\Windows\Web\Wallpaper");
    foreach (FileInfo file in dir.GetFiles("*.jpg"))
    {
        Console.WriteLine("Found {0} *.jpg files", file.Length);
        Console.WriteLine("File name: {0}", file.Name);
        Console.WriteLine("File size: {0}", file.Length);
        Console.WriteLine("File type: {0}", file.Extension);
        Console.WriteLine("Creation time: {0}", file.CreationTime);
        Console.WriteLine("Attributes: {0}", file.Attributes);
        Console.WriteLine("File path: {0}\n", file.FullName);
    }
}

```

Ez a metodus olyan **F1** leírásra tömörbeli tervezés, amelyek mindenegyike egy adott fajtól taralmaz információkat (a **f1** leírás tipus használatával részletezett keszöbbszöbökkel). Vagyík a program osztály kovetkező, a Main() függvényből meghívott, statikus metódust:

Megjegyzés Ha a számítógépen nem található C:\Windows\Web\Wallpaper, akár útközött megléhetően a kodot, például úgy, hogy a C:\Windows\Könyvtár „. bmp fajljaiat olvassa be.

A fent példálat kiengesztíthetők azzal, hogy az alapvető információkat lekerdezzük le adatokat a C:\Windows\Web\Wallpaper könnyvtárban található szén túl a direktoriyutó típus néhány metodustól is használjuk. Először keresszük le adatainkat a GetTiles() metódussal.

Fajlok lista zására a DirectoryInfo típus segítségevel

A Directoryinfo típus használata

A `CreateSubDirectory()` metódus visszatereti értékeről annyit kell tudni, hogy sikeres végrehajtás esetén a metódus visszad egy `DirectoryInfo` objektumot, amely az újirányú leterhezott könyvtárat jelképezi. Nezzük meg azt a példát:

Ha megírjuk ezt a metódust a `Main()` függvényben, és Windows Intézővel megnézzük a Windows-könyvtárat, látható, hogy megjelennek az új alkonyvtárak (lásd a 20.4. ábrát).

```

static void ModifyDirectory()
{
    // A \MyFolder\Data leterhezása az alkalmazás könyvtárában.
    // A \MyFolder leterhezása az alkalmazás könyvtárában.

    DirectoryInfo dir = new DirectoryInfo("..\\");

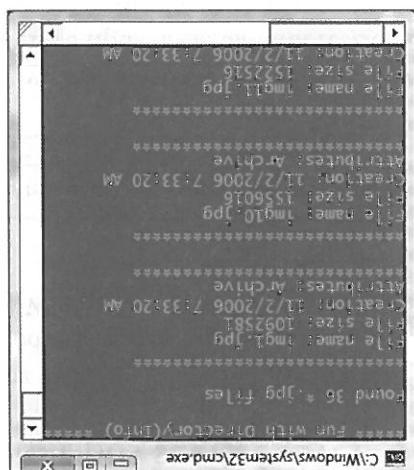
    dir.CreateDirectory("MyFolder");
    dir.CreateSubdirectory("MyFolder");
}

```

Egy könyvtár szerkezetet néha ny alkonyvtárral: A következő metódus például kibövíti az alkalmazás telepítési utvonalának másba ágyazott alkonyvtárat is leterhezatunk egyszeres függvénytással. Programozottan bővíthetik. Ezzel a metodusnal akár egy, akár több egy-alkonyvtárat szerkeszthetünk. A `CreateSubdirectory()` metodusnal

Alkonyvtárak leterhezása a DirectoryInfo segítségével

20.3. ábra: Információk kezeltőablak



20. fejezet: Fájlműveletek és elszigetelt tárolás

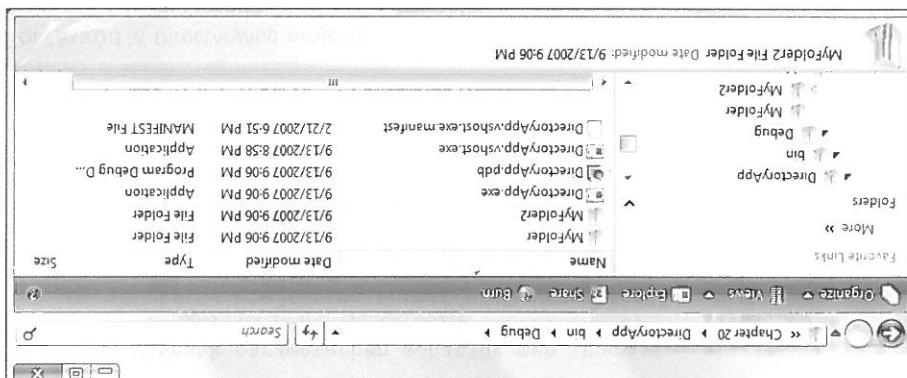
kōnyvtárakat:

A Directory típus működésének szemléleteire ez a végzős szegediügyi-vény megjelenítése a számítógépen található összes meghajtó nevét (a Directory metódusával), és a directory.Delete() statikus metódusával törlendő. A MyFolder2\MyFolder1\letrabban leírtakban minden lehetséges mappát törlhetünk.

A következőkben ismerekedjünk meg a Directori tipusokat. A legtöbb esetben a Directori osztály statikus tagjainak működése megelel annak funkcionálisban, hogy a Directori tagjai általban szintegtipusokat adnak vissza, amelyet a Directori példányszintű tagjai nyújtanak. Ne feledjük azonban, hogy a Directori tagjai általban szintegtipusokat adnak vissza, nem pedig erősen tipusos Előnefo/Dirекториfinfo tipusokat.

A Directory tipus használata

20.4. abra: Alkonytárak leterhözés



```
    DirectorYInfo dir = new DirectorYInfo(".");

    static void Main(string[] args)
    {
        DirectorYInfo dir = new DirectorYInfo(".");

        // A \MyFolder\Tereholás a kezdetkörnyékről.
        // A visszaadott DirectorYInfo objektum tárolása.
        // DirectorYInfo myDataFolder =
        //     CreateSubdirRecursively("MyFolder2\Data");
        // A .\MyFolder2\Data létrehozni utjának kiirása.
        // A ..\MyFolderder2\Data\leírás a myDataFolder;
        // ConsolE.WriteLine("New Folder is: {0}", myDataFolder);
    }
}
```

```

class Program
{
    static void Main(string[] args)
    {
        Console.WriteLine("**** Fun with DriveInfo ****\n");
    }
}

A szabán definiált program osztályt:
GetLogicalDrives() metódushoz hasonlóan a statikus DriveInfo.GetDrives()
logicalDrives() metódusban található a DriveInfo nevű osztály. A Directory.Get-
terekmétet stb.). Nezzük a következő, DriveInfoApp névű új konzolalkalmá-
tet is megad a meghasítkrol (pl. a meghasított tulaj, a szabad területek, a kö-
tetlalogicalDrives() metódussal elérhetően a DriveInfo számos egyéb részlege-
metodus is a számtögep meghasítóink a nevet adja vissza, de a directory.
A system.IO névterben található a DriveInfo nevű osztály. A Directory.Get-

```

A DriveInfo osztálytípus használata

Forráskód A DirectoryApp projektet a forráskönyvtárban a 20. fejezet alkönyvtára tar-
talmazza. A forrásokonnyvtárról lásd a Bevezetés xl. oldalat.

```

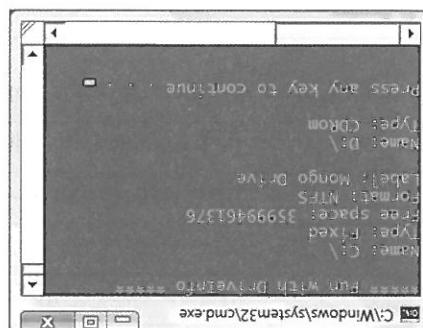
try
{
    Console.WriteLine("Press Enter to delete directories");
    Console.ReadLine();
    Directory.Delete(string.Format("{0}\MyFolder2",
        // Az alkönyvtárakat is törlni szeretnénk-e.
        // A második paraméterben adhatjuk meg, hogy
        // Lethatójuk, amit törölhetünk.
        // A számítógép meghasítóink ki tiltására.
        // string[] drives = Directory.GetLogicalDrives();
        // Console.WriteLine("Here are Your drives:");
        // foreach (string s in drives)
        //     Console.WriteLine("--> {0} ", s);
        // Envíroment.CurrentDirectory, true);
    catch (IOException e)
    {
        Console.WriteLine(e.Message);
    }
}
}


```

Förträskod A Drivemföapp projektet a förträskodkonyvtarroll lásd a Bevezetés xli. oldalat.

A Director, a Directroyinfo es a Drivetimeinfo osztalyok alapvető működésének alatt elvileg zárt hálózatnak tekinthető, hogyan hozhatunk letre, nyithatunk meg, szárhatalmuk be és förlöhetünk adott környezetben levő feljelöltek.

20.5. ábra: Meghajtják adatáinak lekérdezésére a DriveInfo használataval



A 20.5. ábra egy lehetséges körmenetet mutat.

Tag	Jelentés
AppendText()	Egy StreamWriter típusú hozzáférési módszer, amelynek segítségével szövegeket rögzíthetünk a fájlhoz.
CopyTo()	A meglévő fájlt átmásolja egy új fájlba.
Create()	Új fájlt hoz létre, és egy FileStream típusú (lásd később) tervezett adatakat tartalmazó.
CreateText()	Egy StreamWriter típusú hozzáférési módszer, amelynek segítségével szövegeket rögzíthetünk a fájlhoz.
Delete()	Törli a FileFrom Példányát a filtralementesített állományt.
Directory	A szülekekönyvtár egy Példányát adja vissza.
DirectoryName	Visszaadja a szülekekönyvtár teljes elérési útját.
Length	Visszaadja az aktuális fájl vagy könyvvárt méretét.
MoveTo()	A fájlt áthelyezi, és lehetőséget ad arra, hogy új nevet adjunk neki.
Name	Visszaadja a fájl nevét.
Open()	Megnyíja a fájlt különöző olvasási/trási és megosztási jogaival.
OpenRead()	Csak olvasáshoz FileStream típusú hozzáférési módszer.
OpenText()	Egy StreamReader típusú hozzáférési módszer (lásd később), amelyel egy lemez szövegfájljait olvashatunk.
OpenWrite()	Csak írásáshoz FileStream típusú hozzáférési módszer.

20.4. táblázat: FileInfo alkalmi tagjai

Ahogy a DirectoryApp példában is láttuk, a FileInfo osztály segítségével a számtípusokon található fájlokrol kerdezhetünk le adatokat (lehetőzás ideje, méret, attribútumok stb.), továbbá az osztály segítségével letehetünk, másolhatunk, áthelyezhetünk és törlhetünk fájlokat. A Filesystemről pusboldorokkal funkcionálásom tul a FileInfo osztály néhány egyszerű alapvető funkcióval. A Filestream osztály segítségével a fájlokat, melyeket adatainkból írunk ki, a fájlokba írunk adatainkat. A fájlokat a fájlba írásra, adatainkat pedig a fájlba olvasásra használjuk.

A FileInfo osztály használata

20. fejezet: Fájlműveletek és elszigetelt tárolás

```

static void Main(string[] args)
{
    // Ez a fájl eléréséhez használjuk a FileStream objektumot...
    using (FileStream fs = f.Create())
    {
        // Írásra használtuk a filerelőnévöt.
        fs.WriteLine("C:\Test.dat");
        // Újra írásra használhatjuk a filerelőnévöt.
        fs.WriteLine("A másik sor");
    }
}

```

A `FileNotFoundException` metódus `FileInputStream` típusú ter vissza, amely a fájl minden szinikron, minden aszinkron irási/olvásási műveletekkel tesz lehetővé (a részleteket lásd később). A `FileNotFoundException` Create() által visszadott `FileNotFoundException` miatt minden részleteket lásd később. A `FileInputStream` részleteit olvasási/ríiasí hozzáérését biztosít.

Miután befejezettük a `FileInputStream` objektummal a munkát, zárjuk le a fájl-azonosítót, hiszen így a rendszer felszabadítja az adatolyam nem felügyelt erőforrásait. Minthogy a `FileInputStream` megvalósítja az időspasabé interfejszt, a C#-ban blokkjának a segítségevel a rendszer a rendszerválasztók a "takarító"-logika leterhelzását (a részleteket lásd az első kötet 8. fejezetében):

```

    static void Main(string[] args)
    {
        // Letrehozunk egy új fájlt a C meghajtón
        // Használjuk a FileStream objektumot...
        // Zárjuk le a fájlt Yamot.
        fs.Close();
    }
}

```

A felhalmozási leírásbanak elso modja a feltérkép. CreateU metódus hasz-

A `Fileinfo.Create()` methods

A **F1** értefo osztály megtérülésére többesegére tehet az **specifikus I/O**-konzonciót. A objektumot (**Filestream**, **Streamwriter** stb.), ad vissza. Ezek segítségevel a fajl- és többfelékeppen olvashatunk, illetve írhatunk adatokat. A példák tanulmányozásához előt azonban vizsgáljuk meg azokat a különöző módszereket, amelyekkel a **F1** értefo osztályal megszerelhetik a fajlazonosítóját.

```
    public enum FileMode {  
        Create, CreateNew, Open, OpenOrCreate, Truncate,  
        Append, CreateTruncate, OpenOrCreateTruncate  
    }  
}
```

```

static void Main(string[] args)
{
    // Létrehozunk egy új FileStream objektumot.
    FileStream f2 = new FileStream(@".\Temp\test2.dat", FileMode.Open);
    // Fájlinfo f2 = new FileInfo.Open();
    // Létrehozunk egy új FileShare objektumot.
    FileShare fs2 = f2.CreateFileMode.OpenOrCreate,
    using(Filereadwrite f2 = f2.OpenFileMode.OpenOrCreate,
        // Használjuk a FileStream objektumot...
}
}

```

A `Fileinfo.open()` metódust a `FileInfo.create()` által hívott lehetőségekkel precízebben használhatjuk a fájlok megnyitására és lezárására, ugyanis az `open()` jellemezően több paramétert vesz fel a fájl kezelésének a létrekészítésekor. Az `open()` meghívása után egy `Filestream` objektumot kapunk vissza. Nez-

A Fileinfo.Open() methods

meghajtjan letezik a Test3.dat és Test4.dat nevű fájlt:

```
strean objektummal ter vissza (a következő kódban feltételezzük, hogy a C
info.open() metodusokhoz hasonlóan az openread() és az openwrite() is File-
rolásértékkel kellene ehhez megadunk. A FileInfo.create() és a File-
sak írható FileInfo típusokat annak vissza átadáskul, hogy különféle feso-
metodusokat is tartalmaz. Ezek megfelelően konfigurált irányedett, illetve
fájlazonosítók, a FileInfo osztály további, openread() és openwrite() névű
Habár a FileInfo.open() metodusossal rendkívül rugalmassan kezelhetők a
```

A FileInfo.OpenRead() és a FileInfo.OpenWrite()

```
public enum FileMode
{
    None,
    Read,
    Write,
    ReadWrite
}
```

Végül a harmadik paraméter (Fileshare) azt határozza meg, hogy milyen módon osztjuk meg a fájlt más fájlkezelőkkel. Az alapnévvel a következők:

```
public enum FileAccess
{
    Read,
    Write,
    ReadWrite
}
```

A második paraméter a FileMode részről típusa egy olyan érték, amellyel a folyam olvasási/trási viselkedését határozza meg:

20.5. Táblázat: A FileMode felsorolt típusai tagja

Tag	Leírás
Truncate	Megnyitja és 0 byte méretűre vágja le a fájlt.
Append	Megnyitja a fájlt, a végeire pozicionál, és trási műveleteket kezd
	meget kapcsoló kizárolag csak írható folyamatokkal hossz- nálható). Ha a fájl nem létezik, új fájlt hoz létre.

A két utolsó eredékes metódus a `CreateText()` és az `AppendText()`, amelyek streamwriter-referenciát adnak vissza:

A `Fileinfo.CreateText()` és a `Fileinfo.AppendText()`

A strelamreader tipus segítségevel karakteradatokat olvashatunk a móglötter fejiből.

```
// Visszakapunk egy Streamreader objektumot.
// Filtelenfo f5 = new Filtelenfo("C:\boot.ini");
using(Streamreader srreader = f5.openText());
{
    // Hasznaljuk a Streamreader objektumot.
}
```

Az openText() a FileInfo típus egy másik, a fájlmegnyitásról kapcsolatos tagja. A Create(), az open(), az openRead(), és az openWrite() metódusokkal szemben A Create(), az open(), az openRead() és az openWrite() metódusokkal szemben az openText() metódus a StreamReader típus helyett. Ezellel a C# megoldásban található egyszerűbb. Íme néhány példája:

A `Fileinfo.OpenText()` methods

```

static void Main(string[] args)
{
    // Trasvezet Filestream objektum megszerzesre.
    // Fileinfo f3 = new FileInfo(@"C:\Test3.dat");
    // Using(Filestream readonlystream = f3.OpenRead());
    // Hasznaljuk a FileStream objektumot...
}

// Most peddig a csak irhato FileStream objektumot kapjuk vissza.
// Fileinfo f4 = new FileInfo(@"C:\Test4.dat");
// Using(Filestream writeonlystream = f4.OpenWrite());
// Hasznaljuk a FileStream objektumot...
}

```

```

        }
    }

using(FileStream readonlyStream = File.OpenRead("Test3.dat"))
// Irányelvűtől FileStream objektum megszerzése.

{
}

FileStream mode, openOrCreate,
using(FileStream fs2 = File.Open("C:\Test2.dat",
// FileStream objektum megszerzése a File.Open() metódus révén.

{
}

using(FileStream fs = File.Create("C:\Test.dat"))
// FileStream objektum megszerzése a File.Create() metódus révén.

{
}

static void Main(string[] args)
{
}

```

bemutatni tipusok alkalmazásával például egyszerűbbé tehetjük:
 Fileinfo tipusok egymással felcserélhetők. Az előző, a FileStream használata
 Read(), az Openwrite() és az openText() metódusok. Sok esetben a File és a
 megfelelők az AppendText(), Create(), a CreateText(), az openText(), az open-
 tagokon keresztül. Ugyanúgy, mint a Fileinfo típusban, a File típusban is
 A File típus a Fileinfo osztályhoz hasonló funkcionálisit biztosít a statikus

A File típus használata

A streamwritter típusnal karakteradatokat irhatunk a móglottak fájlba.

```

        }
    }

using(StreamWriter swriteAppend = f7.AppendText())
Fileinfo f7 = new Fileinfo("C:\FinalTest.txt");
{
}

using(StreamWriter swriter = f6.CreateText());
Fileinfo f6 = new Fileinfo("C:\Test5.txt");
{
}

// Használjuk a Streamwritter objektumot...
{
}

// Használjuk a Streamwritter swriteAppend objektumot...
{
}

static void Main(string[] args)
{
}
```

20.6. tablázat: A File tipus metodusa

Metódus	Jelentés
ReadAllBytes()	Megnyíti a megadott fájlt, visszadás a bináris adatokat byte-tombként, majd lezárja a fájlt.
ReadAllLines()	Megnyíti a megadott fájlt, visszadás a karakteradatokat sztringtömbként, majd lezárja a fájlt.
ReadAllText()	Megnyíti a megadott fájlt, visszadás a karakteradatokat sztringben. Stríng típusként, majd lezárja a fájlt.
WriteAllBytes()	Megnyíti a megadott fájlt, kírja a byte-tömböt, majd lezárja a fájlt.
WriteAllLines()	Megnyíti a megadott fájlt, kírja a sztringtömböt, majd lezárja a fájlt.
WriteAllText()	Megnyíti a megadott fájlt, kírja a szöveget, majd lezárja a fájlt.

A félre írt használhatók néhány egységi taggal is (lásd 20.6. táblázat), amelyek jelentősen megkönnyítik a szöveges adatok olvasását és irását.

További fájlok között a tagok

```

using(StreamWriter swriter = File.AppendText("C:\FinalTest.txt"))
{
    using(StreamReader sreader = File.OpenText("C:\boot.ini"))
    {
        string strakapunk_egy Streamreader objektumot.
        string sziszterezese.
        string streamwriter megiszerezese.
        string file.create("C:\Test3.txt");
        file.appendtext("C:\FinalTest.txt");
    }
}

```

Források A SimpleFileIO projekt megtalálható a 20. fejezet alkonyvtárában. A forrásokat kinyitva rövid lásd a Bevezetés xlv. oldalát.

Ha gyorsan szeretnénk fájlazonosítóhoz jutni, a File típus egyszerűen megekönnyít a dolgunkat. Ha pedig elobb leterhözünk egy File-től objektumot, ennek megvan az az előnye, hogy megrázásállhatunk a fájlt az absztrakt File-szinten belül a legtöbb fájlazonosítóhoz jutni, a File típus egyszerűen

```

    }

}

Console.WriteLine("TODO: {0}", task);

}

foreach (string task in File.ReadAllLines(@"C:\tasks.txt"))
{
    Console.WriteLine(task);
}

// Visszaolvassuk az egészet, és kiirjuk a kepernyőre.

// Az összes adatot kiirja egy fájlba a C meghajtón.

File.WriteAllLines(@"C:\tasks.txt", myTasks);

// "Call Mom and Dad", "Play Xbox 360";
// "Fix bathroom sink", "Call Dave",
// string[] myTasks = {
//     "Call Mom and Dad", "Play Xbox 360",
//     "Fix bathroom sink", "Call Dave"
// };

Console.WriteLine("***** Simple IO with the File Type *****\n");
static void Main(string[] args)
{
    Class Program
    {
        static void Main()
        {
            using System;
            using System.IO;
        }
    }
}

```

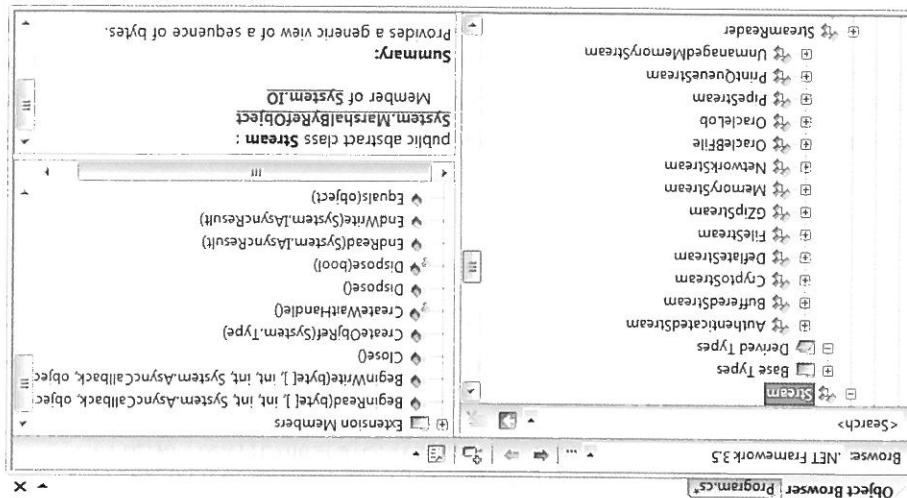
A File típus új metodusaival, néhány sornyi kódval olvashatók vagy írhatunk le is zárják a mögöttes fájlazonosítót. A következő konzolalkalmazás (SimpleFileIO) például a lehető legkevesebb munkával menti a sztringadatot (kullan le is) a myTasks stringben. Ezek a tagok a mielőtt végén automatikusan konszolra írják a módszert. A myTasks stringben a konzolállomásnak (SimpleFileIO) megfelelő lehetségesből minden a sztringadatot (String) megadható a C meghajton (májd ottan beolvassa memoriába):

A File típus használata

A stream osztály Leszámolásottai tethat az adatokat nyers byte-folyamkent jelezik. Ezeket a stream-sorozatokat támogatók a keresés, és ez azt jelenti, hogy mindenik meg, így gyakran elég nehézkes a nyers adatfolyamokkal dolgozni. Bárazonysos stream-leszámolásokat a belülről használhatunk az adatfolyamokban lévő pozitív.

Megjegyzés Az adaptólamok segítségével nem csak fajtakat kezelhetünk. A NET-konváta rakkal adaptólam-alapú hozzáfrejt biztosítanak a halozatokhoz, a memoriáterületekhez és az egypéldányban szereplő adaptólam-alapú absztraktokhoz.

20.6. abra: A Streamable *szarmazó tipusok*



Bárdig a **F1** leseztream, a **Streampreader** és a **streampwriter** objektumokat még nem használunk **fájlok** adatához irásra vagy olvasásra. Hogy megértsük ennek menetét, meg kell ismernünk az adatfolyam konceptjét. Az I/O kezelésben az **adatfolyam** olyan adatrendszerként jellemez, amely a fórrás és a cél között továbbít. Az adatfolyamokkal egyformán kezelhetünk egy **byte-sorozatot** attól függően, hogy milyen eszköz (fájl, halozat kapcsolat, nyomtatás stb.) tárja vagy jeleníti meg azt.

Az absztrakt Stream osztály

A **FileStream** osztály úgy valósítja meg az absztrakt stream típus tagjait, hogy az a **FileStream** folyamok számára a legmegfelelőbb. Ez egy eleg egszerű taggal. Ehezett inkább **kiolvasás** és **írás** műveket. Ennek ellenére példaként nézzük meg a **FileStream** típus szinkron munkát. Ígyek megkönnyítik a szöveges adatokkal vagy a **NET**-tipusokkal végzett szaggan közvetlenül nem tiliságosan gyakran használjuk a **FileStream** típus rú adattölyam; csak egy byte-ot vagy byte-ok tömbjeit tudja olvasni. A való-ahogyan az a **FileStream** típusnak számára a legmegfelelőbb. Ez egy eleg egszerű taggal.

A FileStream típusok használata

20.7. táblázat: A Stream absztrakt tagjai

Tag	Jelentés
CanRead, CanSeek	Megadja, hogy az aktuális adattölyam támogatja-e az olva-sást, a keresést és/vagy az írást.
Close()	Lézárja az aktuális adattölyamot, és felszabadít minden, az adattölyammal kapcsolatos erőforrásat (pl. csatlakozópontról) kat es fajlazonosítókatt). Belülleg ez tulajdonképpen a funkcionálisan megégyezik az "adattölyam eldobásával".
Dispose()	Frissít a mögöttes adattörrést vagy adattárat a puffer jelen-tégi állapotával, majd torli a pufferit. Ha egy adattölyamhoz több puffer van, akkor csak a második, harmadik, stb. lesz eldobásával.
Length	Meghatozza a pozíciót az aktuális adattölyamban.
Position	Frissít a mögöttes adattörrést vagy adattárat a puffer jelen-tégi állapotával, majd torli a pufferit. Ha egy adattölyamhoz több puffer van, akkor csak a második, harmadik, stb. lesz eldobásával.
Read()	Kiolvás egy byte-sorozatot (vagy egységen byte-ot) az adatto-lóból, és az olvasott byte-ök számával elérhető lepette az aktuális pozíciót az adattölyamban.
Seek()	Pozícióval az aktuális adattölyamban.
SetLength()	Beállítja az aktuális adattölyam hosszát.
Write()	Kír egy byte-sorozatot (vagy egységen byte-ot) az adatto-lóból, és a kiírt byte-ök számával elérhető lepette az aktuális pozíciót az adattölyamban.
WriteByte()	Iyamba, és a kiírt byte-ot számával elérhető lepette az aktuális pozíciót az adattölyamban.

A stream osztály által nyújtott funkcionálitas megerősítéshez nézzük meg annak alapvető tagjait a 20.7. táblázatban.

Tetlezük fel, hogy van egy `FileStream`, `FileStream`app névű konzolalkalmazásunk. A célunk az, hogy egy gyorsen szöveges üzenetet írunk a myMessage, dat névű `FileStream`ba. Mivel a `FileStream` csak nyers byte-kkal tud dolgozni, a system.String fejlesztője minden karaktert byte-kké alakítja át. Ezért a `FileStream` appban a `Encoding.UTF8.GetBytes()` metódust használjuk. Ez a metódus a stringet byte-kká alakítja át. Ezután a `FileStream` appban a `Write()` metódust használjuk, hogy a byte-kat írja a `FileStream`ba.

A kódokat végzével a byte-tombot a `FileStream`.`Write()` metódussal írjuk ki a `FileStream` objektum megszerzésére. // Ne fejleszünk el importálni a system.Text-öt. // A FileStream objektum megszerzése. using(FileStream fs = new FileStream("C:\myMessage.dat", FileMode.Create)) { static void Main(string[] args) { Console.WriteLine("**** Fun with FileStreams ****"); // a System.IO nevtereket. // A FileStream objektum megszerzése. // A sztring byte-tombbe történő kódolása. // A bytet[] fájlba frásza. FileStream fs = Encoding.UTF8.GetBytes(msgASByteArray); fs.Position = 0; msgASByteArray.Length); // Az adattöolyam belső pozíciójának alaphelyzetbe állítása. // A tifpusok kitolvásása a fájlbeli es megjelenítésük a konzolon. // A tifpusok kitolvásása a fájlbeli es megjelenítésük a konzolon. // A sztring byte-tombbe történő kódolása. // A sztring msg = "Hello!"; string msg = Encoding.UTF8.GetString(fs.ToArray()); // A bytet[] fájlba frásza. fs.Position = 0; msgASByteArray = Encoding.Default.GetBytes(msg); // A sztring byte-tombbe történő kódolása. // A sztring msg = "Hello!"; string msg = Encoding.UTF8.GetString(fs.ToArray(), 0, msgASByteArray.Length); // Az adattöolyam belső pozíciójának alaphelyzetbe állítása. // A tifpusok kitolvásása a fájlbeli es megjelenítésük a konzolon. // Dekodolt üzenetek megjelenítése. // Dekodolt üzenetek megjelenítése. Console.WriteLine("Default Encoding.Default Message: " + Console.WriteLine(Encoding.UTF8.GetString(fs.ToArray(), 0, msgASByteArray.Length))); // Konsole.Writeln("Encoded Message: " + Console.WriteLine(Encoding.UTF8.GetString(fs.ToArray()))); }

A Streamwriter es a stringwriter osztalyok jobb megteresenek erdekeben 20.8. tablazat osszefoglalja a Textwriter osztaly legfontosabb tagjait.

A Streamwriter es a stringwriter osztalyok jobb megteresenek erdekeben 20.8. tablazat osszefoglalja a Textwriter osztaly legfontosabb tagjait.

A Streamwriter típus (a stringwriter terehez hasonlóan, ezt lásd kessőbb) az absztrakt Textwriter típusok származtatott típusok szövegek adatot áthatnak egy meglévő típusról, amelyekkel a származtatott típusok szövegek általában tagolhatók.

A Streamwriter típus (a stringwriter típusok származtatott típusok szövegek általában tagolhatók) az absztrakt Textwriter típus (a stringwriter típusok származtatott típusok szövegek általában tagolhatók) általában tagolhatók.

A Streamreader típus (erről a kezeltípusról még lesz szó). A Textreader típuson keresztül lehetővé a leszárás.

A kapcsolódó Streamreader típus (erről a kezeltípusról még lesz szó). A Textreader típuson keresztül lehetővé a leszárás.

A Streamreader az absztrakt Textreader típusból származik úgyanúgy, mint a Streamwriter típuson keresztül.

Únicode-kódolás megfelelő.

A StreamWriter és StreamReader

Förerraszkod A Filestremapp projekt megtalálható a 20. fejzett alkonyvtárban. A forrásokat környezetbarát lásd a Bevezetés xlv. oldalán.

Bár a tenni Példa valóban feltölti a fájlt adatokkal, kidérül a Filez stream típus kiszávezélen használataink legmagobb hatánya is: nyers byt-e-okkal kell dolgozniuk. Más stream-lezárásra írt típusok is hasonlóképpen működnek. Ha például egy byt-e-sorozatot szeretnénk elérni, addott memoriáit felülné kiírni, lefordítani, majd a Netwerkstream típus lesz a halhatunk egy Memorystream objektumot. Hasonlóképp, ha egy byte-tombot egy halálzati kapcsolaton keresztül kell továbbítanunk, a Netwerkstream típus lesz a megfelelő típus.

A StreamWriter és StreamReader tipusok használata

ki az íj fajtái:
 A Streamwriter teremtőként minden objektummal szöveges adatot rögzíthet a szövegmezőben. A Streamwriter teremtőként minden objektummal szöveges adatot rögzíthet a szövegmezőben. A Streamwriter teremtőként minden objektummal szöveges adatot rögzíthet a szövegmezőben. A Streamwriter teremtőként minden objektummal szöveges adatot rögzíthet a szövegmezőben.

Szövegfájl irása

A Streamwriter teremtőként minden objektummal szöveges adatot rögzíthet a szövegmezőben. A Streamwriter teremtőként minden objektummal szöveges adatot rögzíthet a szövegmezőben. A Streamwriter teremtőként minden objektummal szöveges adatot rögzíthet a szövegmezőben. A Streamwriter teremtőként minden objektummal szöveges adatot rögzíthet a szövegmezőben. A Streamwriter teremtőként minden objektummal szöveges adatot rögzíthet a szövegmezőben. A Streamwriter teremtőként minden objektummal szöveges adatot rögzíthet a szövegmezőben.

Megjegyzés: A TextWriter teremtőként minden objektummal szöveges adatot rögzíthet a szövegmezőben. A TextWriter teremtőként minden objektummal szöveges adatot rögzíthet a szövegmezőben. A TextWriter teremtőként minden objektummal szöveges adatot rögzíthet a szövegmezőben. A TextWriter teremtőként minden objektummal szöveges adatot rögzíthet a szövegmezőben.

20.8. táblázat: A TextWriter teremtőként rögzített szövegek

	Tag	Jellemzők
CLOSE()	A metodus lezárja az írott, és feleszabadt minden kapcsolódó erőforrásat. A folyamat során a Puffer automatikusan kiürül (ez a tag funkcionálisan megengyezik a DISPOSE() metodus megelőzését).	
FLUSH()	Ez a metodus kiüríti a pufferrel adott minden szövegmezőt sorvegmezőt. A Windows operációs rendszerek alapértelmezett sorvegmezőt. Ez a tag funkcionálisan megengyezik a DISPOSE() metodus megelőzését.	
NEWLINE	Ez a tulajdonoság jelelői a származtatott írott osztály üjisor-konszol-konstansai.	
WRITELINE()	Ez a tulajdonoság jelelői a származtatott írott osztály üjisor-konszol-konstansai.	
WRITETE()	Ez a tulajdonoság jelelői a származtatott írott osztály üjisor-konszol-konstansai.	
WRITETEXT()	Ez a tulajdonoság jelelői a származtatott írott osztály üjisor-konszol-konstansai.	
WRITESCREEN()	Ez a tulajdonoság jelelői a származtatott írott osztály üjisor-konszol-konstansai.	

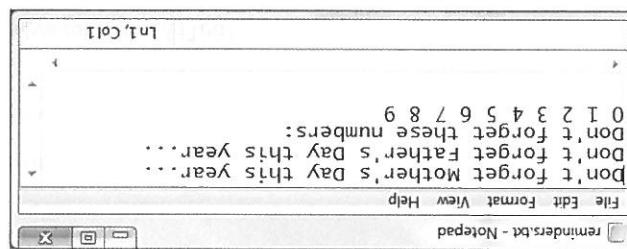
20. fejezet: Fájlműveletek és elszigetelt tárrolás

A következőkben megnezzük, hogyan olvashatunk programozottan adatot textreáder osztályból származik, amely a 20.9. táblázatban összegzett funkcióinakat nyújtja.

Felülböl a megfelelő streamreáder típus segítségével. Ez az osztály az absztrakt `TextReader` osztálytól派生 (derives) a `StreamReader` osztálytól派生 (derives).

Olvásás szövegfájlból

20.7. ábra: A *.txt fájl tartalma



Ha lefuttatjuk a programot, megvizsgálhatjuk az új fájl tartalmát (lásd 20.7.). A fájl az aktuális alkalmazás binárisdebug mapjában található, feltéve, hogy nem abszolút elérési útvonalat adtunk meg a `CreateText()` függvénynek.

```
{
    Console.WriteLine("Created file and wrote some thoughts...");

    writer.WriteLine();
    // Új sort szürűnk be.

    writer.WriteLine("Don't forget Father's Day this year...");

    for(int i = 0; i < 10; i++)
        writer.WriteLine("Don't forget these numbers:" + i);

    Console.WriteLine("File.CreateText(\"reminders.txt\")");
}
// Megkapjuk a StreamWriter-t, majd kiirjuk a sztringadatot.

using System.IO;
// Megkapjuk a StreamWriter-t, majd kiirjuk a sztringadatot.

static void Main(string[] args)
{
    Console.WriteLine("**** Fun with StreamWriter / StreamReader ****");
    string s = "Don't forget Mother's Day this year...";

    StreamWriter writer = File.CreateText("reminders.txt");
    writer.WriteLine(s);
    writer.WriteLine("Don't forget Father's Day this year...");
```

Miután lefuttatjuk a programot, a reminders.txt fájl tartalmát láthatunk a konzolon.

```

    {
        Console.WriteLine("Here are your thoughts:\n");
        string input = null;
        while ((input = sr.ReadLine()) != null)
        {
            string[] lines = sr.ReadToEnd().Split(new char[] { '\n' });
            lines[0] = lines[0].Trim();
            if (lines[0] == "exit")
                break;
            else
                Console.WriteLine(lines[0]);
        }
        Console.WriteLine("Goodbye!");
    }
}

```

Ha úgy megoldottuk a MyStreamWriterReader osztályt, hogy az StreamReader-tól szöveges tartalmat olvashassuk be a reminders.txt fájl szöveges tartalmát, akkor az alábbiak szerint olvashatjuk be a reminders.txt fájl szöveges tartalmát:

20.9. táblázat: A TextReader-alapjaitól tagjai

Típus	Jelentés
ReadLock()	Legelejebb annyi karaktert olvas be az aktuális adattolymaból, amennyit a Count paraméterben megadunk, az adatokat pedig kiirja a pufferbe a megadott indextől kezdve.
Read()	Adatai olvas a bemenneti adattolymaból.
Peek()	Lehet, hogy az adattolymam végre erőltünk, visszaadja a következő elérhető karaktert amelkül, hogy az olvashatóját módosítaná. Ha -1 értekel a tel.ter.vissza, az azt jelez, hogy az adattolymam végre erőltink.
ReadLine()	Beolvas a pozíciótól kezdve, és egyszerűen sztringként adja vissza a fájl végén vagyunk).
ReadToEnd()	Beolvasza az adattolymaban található valamennyi karaktert az aktuális pozíciótól kezdve, és egyszerűen sztringként adja vissza a fájl végén vagyunk).

20. fejezet: Fájlműveletek és elszigetelt tárók

A stringettában találhatók a legtöbb személyes információ aki részt vett az emlékezetben. Ez akkor lehet hasznos, ha egy mögöttes pufferhez karakterlátható információkkel hozzájárulunk.

A StringWriter és StringReader tipusok használata

Förfraskold A stremmávitereppäprojekti tartoontaa tarjamalla. A förfraskodkonyvatarröll läsää Breveteet xvi, oltalat. -jezet a 20. fejezet al-

Bár zavaró lehet, hogy a fájl I/O-műveleteit ennyire leképíteni lehetőleg nem minden esetben megoldható, am ezek minden nagyobb rugalmasságot biztosítanak. A következőkben a sztringwritter és a stringreader osztályok szerepére vizsgáljuk meg.

A system. I0 névtert tpusaszt hsználva zavaró lehet, hogy úgyanazt az ered-
ményt gyakran többelé megközelítéssel is elérhetjük. Megszerezhetünk például egy streamwritter objektumot a File vagy a FileInputStream osztályból.
Van azonban még egy módszer, amelyet a Streamwriter osztályból használhatunk: ha közvetlenül hozzuk lete-
tes a Streamreader típusoskall hsználhatunk: ha közvetlenül a kezdőképpen is modosithatjuk:

A StreamWriter/StreamReader típusok közvetlen letrehozása

A StringWriter es StringReader tipusok használata

Hátha gyógy szövegek adattolagyamboöl szeretmenk olvasni, használjuk a megfelelő stříngreader tipust, amely (ahogy az várható) a kapesoldo streamreader osztályhoz hasonlóan működik. A stirringreader osztály tulajdonképpen nem tesz mászt, mint fejűldéfinálja az orokot tagokat, hogy azok ne járjol, hanem szövegek adattbol olvassanak:

Mivel a stringwritter es a streamwritter ugyanabban az osztályból (`Text`-
writer) származik, az irás logikája többé-kevésbe megegyezik. A string-
writer osztályból azonban - természetesen fogva - kinyerhető egy sys tem.

```

static void Main(string[] args)
{
    Console.WriteLine("***** Fun with StringWriter /");
    string str = "Létrehozzuk a StringWritert, és memóriaiba írjuk";
    str += " a karakteradót.";
    using (StringWriter strwriter = new StringWriter())
    {
        strwriter.WriteLine("don't forget Mother's Day this year...!");
        // Megkaphjuk a tartalom másolatát (sztringben tárolva),
        // írás környékük a konzolra.
        Console.WriteLine("Contents of StringWriter:\n{0}", strwriter);
    }
}

```

újunk. Végyük példaként azt, amikor a stringreaderwrappert alkalmazás a helyi mérvelmézen levő fail helyett egy stringwriter objektumba ír sztringadatblokkot.

Tags	Jelentés
baseStream	Ez az irásvezető tulajdonsgához kötött bázisfolyamhoz.
Close()	Ez a metódus lezárja a bináris adatfolyamatot.
Flush()	Ez a metódus kiüríti a bináris adatfolyam bufferét.

A BinaryWriter és BinaryReader osztályok használata

Forráskód A StringReaderWriterAPP projektet a forráskódokonvátrabán a 20. fejezet alkójnyátra tartalmazza. A forráskódokonvátról lásd a Bevezetés Xliv. oldalat.

```
using (StringReader strReader = new StringReader("Contents of StringWriter:\n[0]", strWriter))
{
    string input = null;
    while ((input = strReader.ReadLine()) != null)
    {
        Console.WriteLine(input);
    }
}

using (StringWriter strWriter = new StringWriter())
{
    string input = null;
    while ((input = Console.ReadLine()) != null)
    {
        strWriter.WriteLine("Don't forget Mother's Day this year...!");
    }
}
```

A BinaryWriter és BinaryReader osztályok használata

```

20.11. feladat: A BinaryReader alapötölgéje az
    adattípuszt ír ki egy új *-dat fájlba:
    A körvonalzó példa (BinaryWriterReader névű konzolalkalmazás) többfelé
    statikus void Main(string[] args)
    {
        Console.WriteLine("***** Fun with Binary Writers /");
        Readers *****\n");
    }

    // Megnyitunk egy bináris írót egy fájl számára.
    // Filtérífo f = new FileInfo("BinaryFile.dat");
    using(BinaryWriter bw = new FileInfo("BinaryFile.dat"));
    {
        // Kifirjük a BaseStream tulajdonság típusát.
        // (Jelen esetben System.IO.FileStream).
        // Consolé.Writeline("Base stream is: {0}", bw.BaseStream);
        // Létrehozunk néhány adatot, amelyet elmenthetünk a fájlba.
        double adouble = 1234.67;
        int anint = 34567;
        string astring = "A, B, C";
    }
}

```

A binary reader osztály a bináriyri ter által nyújtott működést a 20.11. tábla szerinti ki.

Tag	Jelentés	seek()	write()	read()	BinaryWriter legfontosabb tagjai
ez a metodus beállítja a pozíciót az aktuális adattölyamban.	Ez a metodus kír egy értéket az aktuális adattölyamba.	Ez a metodus kír minden karaktert a megadott számig.	Ír minden karaktert a megadott számig.	Olvas minden karaktert a megadott számig.	20.10. tablázat: A BinaryWriter legfontosabb tagjai
szektor	szektor	szektor	szektor	szektor	szektor

Források A BinaryWriter-reader projektekkel láss a Bevezetés xlv. oldalat.

```

static void Main(string[] args)
{
    string f = new FileInfo("binfile.dat");
    ... // Kiolvassuk a bináris adatokat az adaptorfolyamból.
    using(BinraryReader br = new BinraryReader(f.OpenRead()))
    {
        ... Konsole.WriteLine(br.ReadDouble());
        ... Konsole.WriteLine(br.ReadInt32());
        ... Konsole.WriteLine(br.ReadString());
        ... Konsole.WriteLine(br.ReadLine());
    }
}

```

A BinaryWriter és BinaryReader osztályok használata

A fajlok megfigyelésének folyamata kivonatkezõ példára szemlélteti, ehhez azt fejtetézzük, hogy a C meghajtott lethehozunk egy MyFolder nevû új konviktárat, amelyben különbozó, text fajlok találhatók (tételezés szerint elhelyezve). Az alábbi (MDI rectorywachter névű) konzolalkalmazás a MyFolder konnyvtában van. A *.txt fájlokat fogja felügyeleti, és tizennégyet ír ki a konzola, ha új fájlt hozunk letre, illetőleg megújvározza fájlt torlunk, módosítunk vagy átméretezzük:

/// A Renamedeeventhandedler metodusreferenciának egy,
/// a következő szignatúranak megfelelő metodust kell meghívni.a
void MyNotifiicatorHandle(MessageBoxSource source, Renamedeeventargs e)

A Renaudé eseményt is kezeltetjük a Renaudé Event Handler er metodusaihoz kapcsolódva.

/// A fő részletekben mindenhangulatban megfelelő módszerrel leírható, hogy melyik részben milyen módszert alkalmaztak a környezetvédelemhez. A részletekben mindenhangulatban megfelelő módszerrel leírható, hogy melyik részben milyen módszert alkalmaztak a környezetvédelemhez. A részletekben mindenhangulatban megfelelő módszerrel leírható, hogy melyik részben milyen módszert alkalmaztak a környezetvédelemhez.

A **F1** lexisystemswocher hasenhalatnamak else lipeskegent be kell allitamunuk a Path tulajdonssagot a megfigyeleni fajlokat taratalmazo mappa eleresi utvonalara, valamint a filter tulajdonssagban meg kell adunk a felügyelende fajlok ki-

```
{  
    public enum NotifyFilters  
    {  
        Attributes, CreationTime,  
        Directories, FileName,  
        LastAccess, LastWrite,  
        Security, Size,  
    }  
}
```

A kovetkezokben nezzük meg a Filesystemwatcher osztály szerepet. Ez a típus megeléhetően hatékonyan segít a rendszerekben található fillok programozott felügyeleteben (vagy megfigyelésében). Ez azt jelenti, hogy a Filesystem-watcher típusú utastírával a felügyeletező a számítógépen (bar a tagok többsegége egyérelelmű, célszerű) teljesről műveltek tekintetben (ez a tagok többsegége egyérelelmű, célszerű).

Fajlok programozott „folyamatos”


```
public abstract class Stream : IDisposable
{
    public void EndWrite(IAsyncResult result);
    public virtual void EndRead(IAsyncResult result);
    public virtual int Read(byte[] buffer, int offset, int count);
    public virtual IAsyncResult BeginWrite(byte[] buffer, int offset, int count, AsyncCallback callback, object state);
    public virtual IAsyncResult BeginRead(AsyncCallback callback, object state);
    ...
}
```

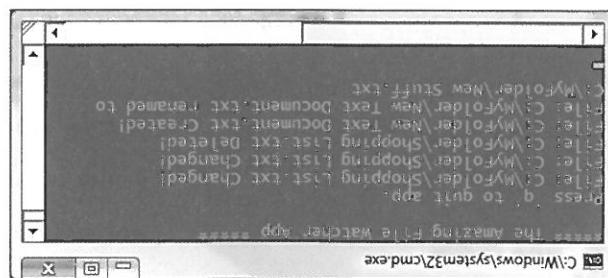
A szystem, ío névtert áttekinetesének belfejlezéséket nezzük meg, miként használja:

túnuk vége részére asztinkor műveleteket a Filestream tipussal. Az elso kötetben, a többszájú fejedelmi logozás elemzésekor már látottuk, hogyan támogatja a .NET kerekrendszert az asztinkor műveleteket (lásd a 18. fejezetet). Minden nagy részjelölök részén olvasásra idlegényes művelet lehet, a számítás, a szolgáltatás, a szolgáltató rendelkezni. Ezek a módszerek az Iasyincresult tipusát használják:

Azszinkron fázisjavasás esetén

Förträskod A Mydirektoriwachter alkalmazását förträskodkonyvtárban a 20. fejezet alkonyvátra terültetettet. A forrásokkal összhangban minden másik részleteket kivéve a 20. fejezet alkonyvátra terültetettet.

20.8. abra: Szövegfüjlok megfigyelése



A program kiprobabilasához inditsuk el az alkalmazást, és nyissuk meg a Windows hitezet. Probáljuk meg átmerezni a fájlokat, hozunk leter e s torolunk egy *.txt fájlt stb. A MyOlder konverterban lévő szövegfájlok állapota val kapcsolatos különbségekkel. Láthatóva valamik (lásd 20.8. ábrát).

20. fejezet: Fájlműveletek és elszigetelt tárolas

A **fehér** peldával érdekelhető az, hogy a **F11** streamek típusa aszinkron viselkedésenek engedélyezéséhez csak a **Peldában szereltő** konstruktor használható. A legtöbbször, szabályon kívül van a típusú paraméter (ha igaz erőkötés után). A **lehetőleg**, minden típusú paraméter (ha igaz erőkötés után).

Kivállásokat adhat a végrehajtó szerevnynek; többek között az alábbiakat:
.NET-központú biztonság mechanizmus. A CAS segránségevel a CLR biztonság
Bárcsak a .NET platform valasztja a kódrendszer-alapú biztonság (CAS) elnevezésű,
nem jelent egyséb potenciális biztonságkockázat?

(rosszindultat) hivataloskatt az operációs rendszerek mögöttes API-jaihoz, vagy
probabil erzékeny adatokat kiolvassan a rendszerszintű adatbázisból, nem intet
tölgépről tolltűnk le végrehajtható fájlt, mi garantálja, hogy a szerevny nem
hogyan nem akar erzékeny adatokat olvasni? Használkozzunk meg arról,
tölgépről fájlokat probabil olvasni, vagyon hogyan bizonyosodhatunk meg arról,
amely tavoli helyről letölthetőkodott futtat. Ha ez a kódönnyvátra helyi számít-
számossá válik. Tegyük fel például, hogy van egy olyan alkalmazásunk,
zen kivül számos különöző helyről tolltethetünk be, a bizalom kérdezse ige-
megyelembe véve, hogy .NET-szerevnyeket a számítógép helyi merevleme-

Bizalom kérdezse

mozottan tolltethetünk be különböző URL-rol szerevnyt.
Míg az Assembly.LoadFrom() metódussal peddig (lásd a 16. fejezetet) progra-
mási utat határozzunk meg, ahonnan a CLR különböző szerevnyeket tollíthat be,
nak <codebase> elemre lehetséges teszí, hogy deklarátvá modon tesszük el.
Az első kötet 15. fejezetben szó volt arról, hogy az úgyfelek * .config fájlia-
előző kötet 19. fejezetet).

Mindehelyen minden letrahozott új szerevnyel is megtethetünk ezt (lásd az
szám, a memoriaban letrahozott új szerevnyel is megtethetünk ezt az
de akár a System.Reflection.Emitt neverter típusainak segítségevel dinamikku-
szerevnyeket tolltűnk le, többek között különböző webhelyekről, az internetről,
eredetetlől függően. A .NET platform támogatja, hogy különöző helyekről
vagy a lemezre írt potenciális biztonságát fenyegetést jelenthet az alkalmazás
trust) biztonságát prioritási kap. Egy számítógép merevlemezéről olvasni
sakor azt feltételezik, hogy a program a CLR-tól minden megnegedő (full
minden előző, fájl I/O-val kapcsolatos példánkban az alkalmazás végrehajtás-

Az elszigetelt tároló szerepe

Források A Asynchronikus szerep a forrásokonvátról lásd a Bevezetés XL. oldalát.
taralmazza. A forrásokonvátról lásd a Bevezetés XL. oldalát.

Az elszigetett tártozókhoz tartható API használatának másik előnye az, hogy a kodnak nincs szüksége bemeneteket elérési utakra vagy konnyvtárnevekre a programban. Ehelyett az elszigetelt tártozó használatakor az alkalmazás indítható minden módon, mert a kód azonosítójaval (pl. URL, erős név vagy X509 digitális aláírás) valamelyen modern összefüggésekhez hozható egyédi adattároló területre az adatokat (a Kodazonosítók ismerteset lásd kezdetben).

Az elszigetett törölőhoz tartozó API-t nem kizárolag a tavoly számítógépekkel, de a felhasználók számára is elérhető. A töröléshez először a törölendő adatokat kell meghosszabbítani, majd a törölésre alkalmas API-t használva hívhatunk ki a törölést. Az API visszaadja a törlés eredményét, amely az adott mutatók alapján megadja, hogy mennyi idővel lehetséges a törlés. Ez a módszer lehetővé teszi a felhasználók számára a törölést, és segíti a felhasználókat a törölés hatékonyságának javításában.

Az elszigetelt tárrolóhoz tartozó API egyéb felhasználási modjai

A system, ío, isoláltedstorrage névvel tűpuszt hozzávalhatók olyan alkalmazások, amely a .NET-ét futtató számítógép szpecialis, elszigetelt tárrolónak letrehozására, amely a .NET-t futtató számítógép szpecialis, elszigetelt tárrolónak nevezett területen tárolja az alkalmazás adatát. Ez valójában egy biztonságos futtatásokról szóló „sandbox”, amelyben a CLR lehetsége teszi az irás/olvasási műveleteket még akkor is, ha az alkalmazás URL-rel folytatottak le.

- a szamitógép környékről - es tájrendszerekkel modellözés,
 - halozati, internet - és adatbáziskapcsolatok kezelése,
 - új alkalmazásfejlesztések és dinamikus szerelvénnyek letrehozása,
 - a NET reflexiós szolgáltatás használata,
 - a nem felügyelt kod meghívása a Plmvoke segítségével.

Mintútan a CLR kiteretekkelte a szerelemei bázisozonytékait, és eldönötötte, hogy melyik kodcsoporthiba kerüljön, amikor erdekeben, hogy meghatározhasa, a szerelemei mit tehet, és fogkent mi nem tölhet, lekerdezi a Kodcsoporthoz tartozott engedélyezőszablonot (ez egy szérszínű a Különálló engedélyek nevezésére készítve).

Megjegyzés Szigorúan véve kettétele bizonyítékot beszélethetünk: ezek a szerelvénylelapból és hosszalapú bizonyítékok. Mi a szerelvénnyalapú bizonyítékot lefordítjuk a szerelvénylebbe, a hosszalapú bizonyítékokat. Azonban az adott szabálytól eltérően a szerelvénnyalapú bizonyíték esetében nem minden részletekkel kell megfelelnie.

ugyanazon tettelekmek telemeket meg (pl. ugyanaz az eredetük). Azt a félteletet, amely alapján a CLR eldöntheti egy szerelvénnyel, hogy az melegílik kedcsportba tartozik, bizonyítéknak nevezzük. A szerelvénnyeket erre- diktikon kívül más bizonyítékoknak is besorolhatjuk őket. A szerelvénnyeket erre- llyen lehet a szerelevny eros neve, egy bázagyazot X509 digitális tanúsítvány vagy bármely, általunk programozottan elérkezett egyszerű féltelet.

Bevézetés a kódereket-alapú
diztonságba

Megjegyzés A LAS tejes demutatás leágiládból egy (vagy kec) teljes fejezetet ígelyenhez, ezért a LAS működését kizárolódik olyan szinten tekintjük át, hogy megérthessük az elszigetelt területekkel szembeni működési lehetőségeket. A CAS működésének további részleteit a .NET Framework 3.5 tartalmazza.

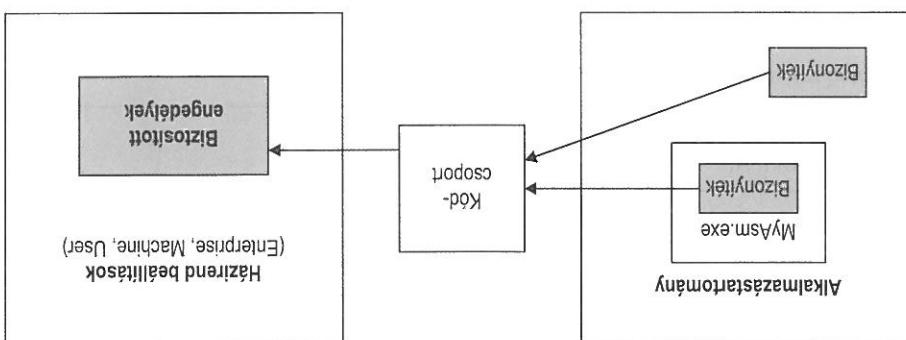
Szerencsere az elszigetelt táróló kezelésre ígern egyszerű azoknak, akik erthetően nyozásra elott nezzük át a.NET Kodhozzáírás-szabalyozási modellet.

Amma k erdekeben, hogy a CLR meghatározza, hogy egy szerevny melyik táblázat íjá le azokat a több bizonyítéktípusokat, amelyet a szerevny betástartományból) szereznik meg, amikor az betölödik a memóriaba. A 20.12. tek tehát olyan adat, amelyet a szerevnybel (vagy az azt hozzóló alkalmakodcsoportba kerüljön, elosztó körülissa a megadott bizonyítéket. A bizonyítékek szerepét a CLR meghatározza, hogy egy szerevny melyik

A bizonyíték szerepe

készve a szerevnybizonyíték fogalmával. Erdemes azonban kiiscit részletesebben szemügyre venni a CAS építőelemet. CAS közötti interakció a határoban történik, felhasználói beavatkozás nélkül. többször az alapértelmezett CAS biztonságát beállítás, valamint a CLR es a .NET-alkalmazás futtatasakor a határoban, szerevhelyetlenül zálik le. Leggyorsultságú készletek hozzárendelésének folyamata minden alkalmal, egy gosztalmasa szerevny készletekkel hozzárendeléséhez használható. A szerevny készletek kiterjesztésének, a szerevny kodcsoportba sorolásának es a jo-

20.9. ábra: A CAS építőelemei



Miután minden harom (vállalati, számítógép- és felhasználói) biztonságát hozzárend alkalmazsa megtörtént, a szerevny lefut a .NET futásidejű környezetében. Ha a szerevny a jogosultságú készletebe nem tartozó kódot mutatja be, hogy a CAS ezek biztonságú kivételet fog dobni. A 20.9. ábra példájához, a CLR futásidejű biztonságú kivételet fog dobni. A 20.9. ábra példájához, a CLR futásidejű biztonságú kivételet fog dobni. A 20.9. ábra példájához, a CLR futásidejű biztonságú kivételet fog dobni. A 20.9. ábra példájához, a CLR futásidejű biztonságú kivételet fog dobni.

```

Ezután egy olyan egyszírű alkalmazást írunk, amely megkereti a felhaszná-
lót, hogy adja meg a betölteni szerelelveny nevét. Szamba vesszők valamely-
nyi szerelelvenyből kiválasztva. Ezután megadja a kiteretkelezendő szerelelveny teli-
szerűítését, majd az eredményt kijelíti a konzolablakba. Elő-
ször is, a program osztály Main() metódusa lehetővé teszi, hogy a felhasználó
megadja a kiteretkelezendő szerelelveny teljes elérési útját. Ha az L-lehetőségeket
valasztja, megállapítja, hogy a felhasználó megadott szerelelvenyt a memoriából. Ha sikeresít, átadja az Assembly-referen-
ciáját egy másik, DíszPályázsmegoldásra. Ez a segédmetódusnak. A megvalósít-
tatás a következő:
}

class Program
{
    static void Main(string[] args)
    {
        bool isSuccess = false;
        string userOption = "...";
        Assembly assembly = null;
    }
}

```

20.12. táblázat: Kulonbøző típusú szerelemeinek összefoglaló összehasonlítás

Hosszúítási nyiterek-típus	Jelentés	Alkalmazásokonnyv-tár	A szereleveny telepítési környezetére.	Szerelevenyhashskód	A szereleveny tartalmának hashszereletek.	Kibocsátó tanúsítványai	A szerelevenyhez rendelt Authentication X509-es digitális tanúsítvány (ha van).	Hejly	A forráswebhely, amelyről a szerelevenyt betölthetek (nem érvényes a helyi gépről betöltött szerelevenyek esetében).	A szereleveny erős neve	A szereleveny erős neve (ha van ilyen).	URL	Az az URL, ahonnan a szereleveny betölthetőt (HTTP, FTP, rögzített stb.).	Zóna	Amik a zónának a neve, aholnan a szereleveny betölthető.
Hostszűrőkönnyítők-típus	Jelentés	Alkalmazásokonnyv-tár	A szereleveny telepítési környezetére.	Szerelevenyhashskód	A szereleveny tartalmának hashszereletek.	Kibocsátó tanúsítványai	A szerelevenyhez rendelt Authentication X509-es digitális tanúsítvány (ha van).	Hejly	A forráswebhely, amelyről a szerelevenyt betölthetek (nem érvényes a helyi gépről betöltött szerelevenyek esetében).	A szereleveny erős neve	A szereleveny erős neve (ha van ilyen).	URL	Az az URL, ahonnan a szereleveny betölthetőt (HTTP, FTP, rögzített stb.).	Zóna	Amik a zónának a neve, aholnan a szereleveny betölthető.
Hostszűrőkönnyítők-típus	Jelentés	Alkalmazásokonnyv-tár	A szereleveny telepítési környezetére.	Szerelevenyhashskód	A szereleveny tartalmának hashszereletek.	Kibocsátó tanúsítványai	A szerelevenyhez rendelt Authentication X509-es digitális tanúsítvány (ha van).	Hejly	A forráswebhely, amelyről a szerelevenyt betölthetek (nem érvényes a helyi gépről betöltött szerelevenyek esetében).	A szereleveny erős neve	A szereleveny erős neve (ha van ilyen).	URL	Az az URL, ahonnan a szereleveny betölthetőt (HTTP, FTP, rögzített stb.).	Zóna	Amik a zónának a neve, aholnan a szereleveny betölthető.
Hostszűrőkönnyítők-típus	Jelentés	Alkalmazásokonnyv-tár	A szereleveny telepítési környezetére.	Szerelevenyhashskód	A szereleveny tartalmának hashszereletek.	Kibocsátó tanúsítványai	A szerelevenyhez rendelt Authentication X509-es digitális tanúsítvány (ha van).	Hejly	A forráswebhely, amelyről a szerelevenyt betölthetek (nem érvényes a helyi gépről betöltött szerelevenyek esetében).	A szereleveny erős neve	A szereleveny erős neve (ha van ilyen).	URL	Az az URL, ahonnan a szereleveny betölthetőt (HTTP, FTP, rögzített stb.).	Zóna	Amik a zónának a neve, aholnan a szereleveny betölthető.

Végeül pedig a display assembly evidence C) metodus olvasással ki a betoltoott szerelemeiből a bizonyítékot az Assembly típus Evideance tulajdonosának rövén, vagyiból a bizonyítékot az Assembly típus Gethostevidence() me-

```
private static Assembly LoadAssembly()
{
    try
    {
        Console.WriteLine("Enter path to assembly: ");
        string assemblyPath = Console.ReadLine();
        Assembly assembly = Assembly.LoadFrom(assemblyPath);
        return assembly;
    }
    catch (Exception ex)
    {
        Console.WriteLine("Load error...");
```

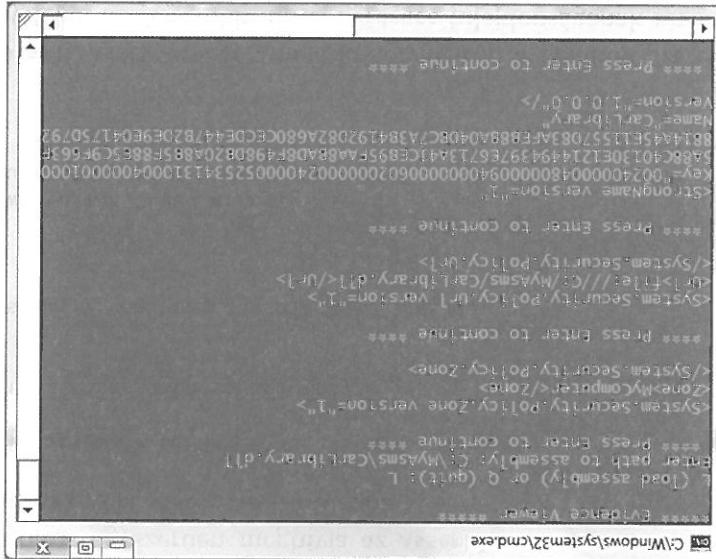
A LoadForm() metódus egyszerűen megálvija az Assembly.LoadForm() módot, hogy bállitsa a privat Assembly tagvaltozót:

```

do
    Console.WriteLine("***** Evidence Viewer *****\n");
    Console.WriteLine("Load assembly) or Q (quit): ");
    useroption = Console.ReadLine();
    switch (useroption)
    {
        case "T":
            asm = Loadasm();
            if (asm != null)
                displayAssembly(asm);
            break;
        case "q":
            if (userdone = true)
                break;
            else
                Console.WriteLine("I have no idea what you want!\"");
        default:
            break;
    }
}
} while (!isuserdone);
}
}

```

20. 10. ábra: A Carl library-*dll* bizonyítékaink megtékinthető



Az alkalmazás teszteléséhez érdemes a C megírásához gyökeresen leírható interfész típus megléteitől és az alkalmazás konstrukcióját szemben minden részletet tükrözni. Az alkalmazás körülállása az összes bizonyítékokat, ahogy az 20.10. részletben írt előírásoknak megfelelően el kell végezni. A felületekkel kapcsolatos tesztelések során minden felületet ki kell vizsgálni, hogy minden felületen minden funkció működik helyesen. A felületekkel kapcsolatos tesztelések során minden felületet ki kell vizsgálni, hogy minden felületen minden funkció működik helyesen.

```

private static void DisplayAssemblyEvidence(Assembly asm)
{
    // A bijoznyittekjutemeny es a mogottes felisoroló lekerese.
    // A bijoznyittekjutemeny = e.Evidence;
    // Ienumerator itfenum = e.GetHostEnumerator();
    Evidence e = asm.Evidence;
    IEnumerator itfenum = itfenum.MoveNext();
    while (itfenum != null)
    {
        // A bemeneteket kiirasa.
        Console.WriteLine("**** Press Enter to continue ****");
        Console.ReadLine();
        itfenum.MoveNext();
    }
}

```

Megjegyzés A .NET Framework 3.5 SDK egy csapatl. exe nevű parancssor programot is tartalmaz, amellyel úgyanazeket a beállításokat érhetjük el.

A .NET keretrendszer 3.5 SDK egy olyan grafikus felügyeleti eszközt biztosít, amellyel a rendszergazdák meghozhatók és módosíthatók a megfelelő kodcsatornák, vagy szüksége szerint újat hozhatnak létre. Az eszköz segítségével beállíthatók a CLR minden részleteit, például, hogy a CLR minden, adott címre (pl. http://www.interfreech.com) lektorolt különböző szerelemeit egy testre szabott biztonsági környezetben futtasson.

20.13. táblázat: Néhány gyakorlati kódcsaport

Alapértelmezett	Hozzárendelt	Jelentés	kódcsaport	My-Computer-Zone	Full Trust	Közvetlenül a helyi merevlemezről bér-	LocallIntranet-Zone	LocallIntranet	A helyi internet egy megszisztaisi pontjá-	Internet-Zone	Internet	A webről lektorolt szerelemeit jellez.

A bizonyítékok használataval a CLR a szerelemeinek kódcsaportba sorolhatja. Mindehnen kódcsaportot egy biztonságú zónához rendeltek, amelyek gyárilag meghatározott biztonságú beállításokkal rendelkeznek. A CLR ezek segítségével rendezi el a szerelemeit (ezekről a következő részben lesz szó).

A kódcsaport szerepe

Forráskód A MyEvidenceViewer projekt a forráskódok nyitáraiban a 20. fejezet alkójának tárta tartalmazzá. A forráskódok nyitárról lásd a Bevezetés XI. oldalát.

Látható, hogy a CarLibrary.dll a MyComputer-zónába került a C:\MyAsms\ CarLibrary.dll URL-ről, és rendelkezik erős névvel. Ha a szerelemeit más hélyről toljuk be (mondiuk tavolí webhelyről), akkor teljesen más kimeneteet fogunk kapni. A lenyeg az, hogy a szerelemy memóriaiba való betöltesekor a CLR megvizsgálja a szerelemy által szolgáltatott bizonyítékokat.

Bevezetés a kódrendszer-alapú biztonságba

A Membership Condition fut le kattintva látható, hogy a CLR mi alapján dönthet el, hogy egy adott .NET-szerelvénynek ebbe a zónába kell-e teroznia. A 20.13. ábra mutatja, hogy a tagság felettel a nem túl beszédes Zone-értek, amely azt a helyet jelöli, ahonnan a szerelevényt betölthetik.

20.11. ábra: A CAS-központi számítógéphálózirányítás

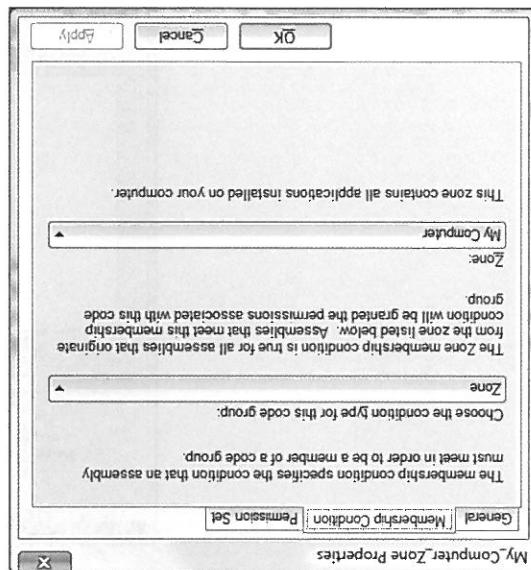


Ezt az eszközt a Control Panel Administrative Tools mappájában találjuk Microsoft.NET Framework Configuration néven. Az eszköz elindítása után a CAS-beállításokat hárrom szinten módosíthatjuk: vállalati szinten (pl. minden hálózatba kapcsolt gép), a helyi gép szintén, illetve felhasználónak (A CAS-t alkalmazásstartományi szinten is beállíthatjuk, de ez csak programoztatni lehet meg). A 20.11. ábra a CAS-alapértelmezett „számtológépszintű hálózirend” beállításainak leírását bemutatja a kiallásban.

Az All_Code kódcsoporthoz (amely az összes .NET-szerelvénnyt jelenti) több Puter_Zone közösport (amely a közvetlenül a helyi merevlemezről betölthető laptop, amely további részletekkel szolgál). A 20.12. ábra például a My_Computer_Zone tartalmát mutatja. A számtológépszintű hálózirendben minden szerelvénnyeket tárta ki a számítógép, a számtológépszintű hálózirendben pedig csak a számítógép hálózatához köthető szerelvénnyeket.

Ha a My_Computer_Zone kodcsoporthoz a My_Computer_Zone Properties ablakban a Membership Condition tabban lévő Zone membership condition részletet kattintunk, azt lehetővé teszi biztonságba engedélyezését kapja meg (lásd a 20.14. ábrát).

20.13. ábra: A My_Computer_Zone kodcsoporthoz tartozó fejlettei

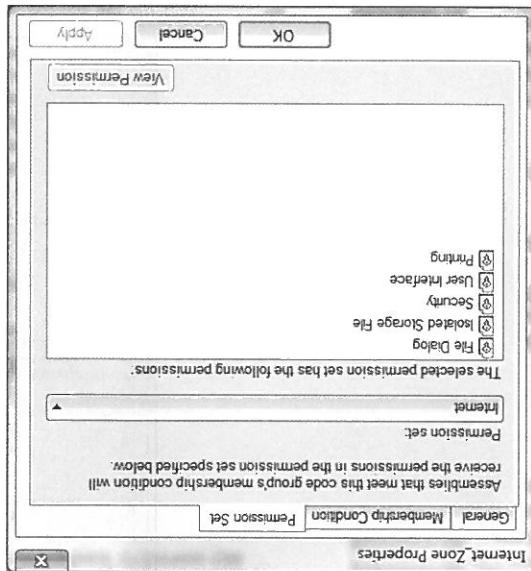


20.12. ábra: A My_Computer_Zone kodcsoporthoz tartozó részletei



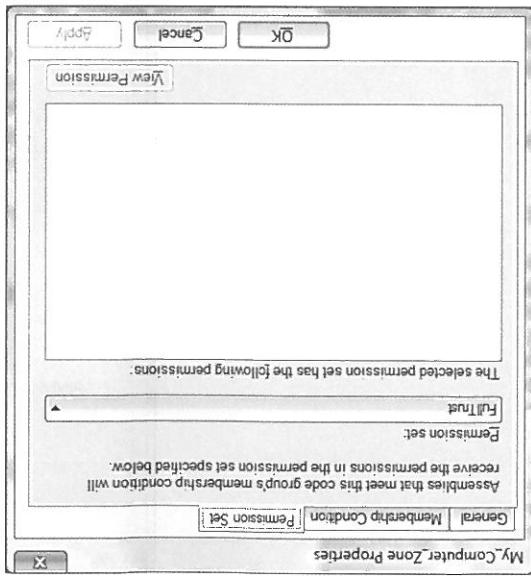
20. fejezet: Fájlműveletek és elszigetelt tárols

20.14. ábra: Az adapterrelmezés szervint a helyi merevolémerevől betölthet szervevények Full Trust engedélyt kapnak



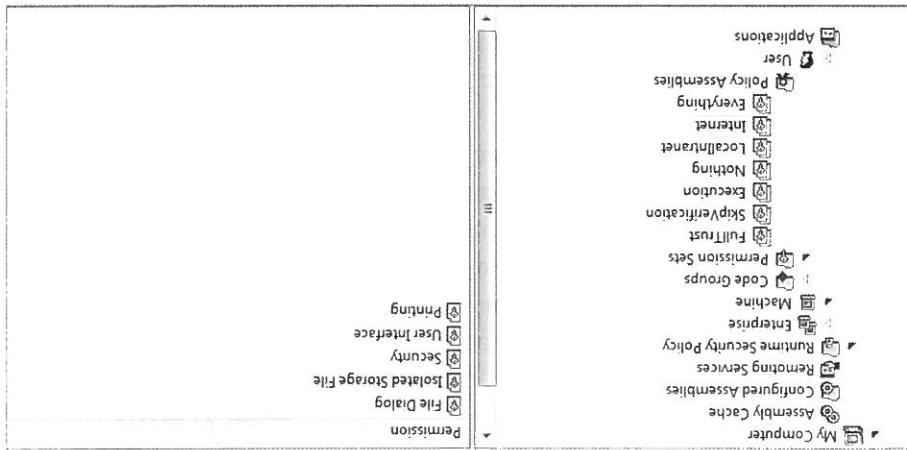
Full Trust engedélyt kapnak

20.15. ábra: A különböző URL-rel betöltött szervevények az adapterrelmezés szervint nem kapják meg a Full Trust engedélyezését



AZ itt látható ikonok (FileDialog, Isolated Storage File, Security stub), minden a kezelt egység specifikus engedélyt készítik, amelyeket még részletesebbben bemutatunk, ha duplan kattintunk rájuk. Ha például ketszer kattintunk a Security engedélyre (amely az általános biztonsági beállítások gyűjtőnévre), azt láthatunk, hogy az Intermet-Zóna által futó szerelvény (ameleyet

20. 16. ábra: Internet engedélyezészet

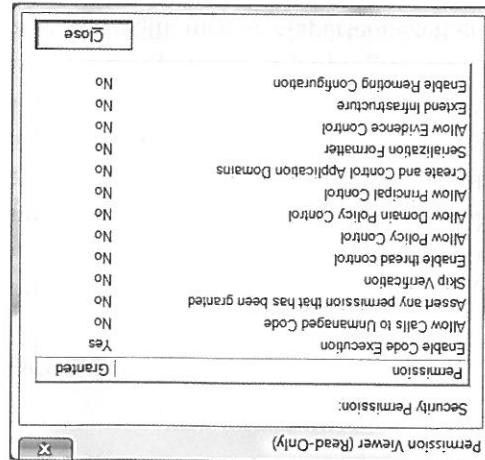


Az engedélykészletek szerepe

Ez az eseményt a Kultuszminisztérium és a Magyar Nemzetkörök Szövetsége szervez. A résztvevőknek mindenki köszönheti, hogy részt vették a programban. A résztvevőknek mindenki köszönheti, hogy részt vették a programban.

A CAS működése

20.17. ábra: Az engedélykészlet egyes engedélyeinek megtérítése



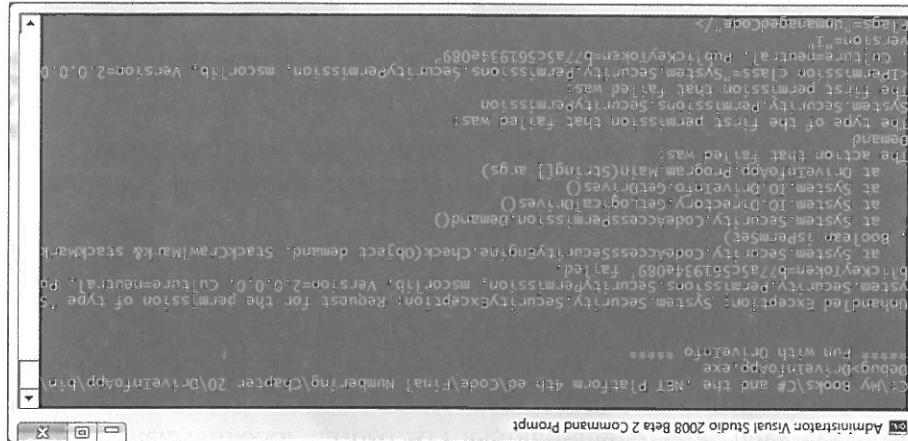
Szemerédi et al. (László Szemerédi, János Komlós, Endre Szemerédi) (1975).

szemért az általános körfelvétel körülbelül 1000 km-es területen elterjedt. A korlátos területekben a leggyakoribb a szemérvízszintű földszintű talajvíz, amelyet a talajvízszintű földszintű talajvíznek nevezünk. A szemérvízszintű földszintű talajvíz a talajvízszintű földszintű talajvíznek a leggyakoribb formája, amelyet a talajvízszintű földszintű talajvíznek a leggyakoribb formája.

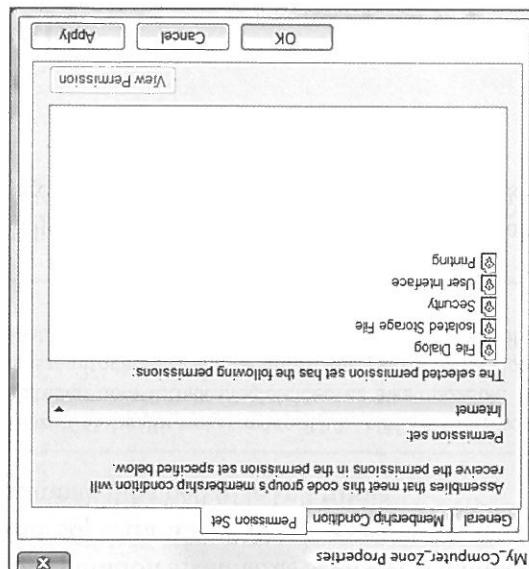
A 20.19. ábrán látszik, hogy az alkalmazás nem tudott hozzáérni a helyi merevlemezhöz, és a CLR biztonsági kivételet döböt.

Bevézetés a kódredefekt-alapú biztonságba

20.18. ábra: A My_Computer_Zone kodcsoporthoz engedélyezett minden modosításra



20.19. ábra: Biztonsági ellenőrzésről. A My_Computer_Zone kodcsoporthoz nem engedélyezték



- adatokat.
- XAML böngészők alkalmazását (XAP, XAML Browser Application – hasd a 28. fejezetet) telepítünk, amely a felhasználó számítógépen táról átnevezett tartományba integrálódik, és az ügyfélgépen kell beállítani.
 - Olyan Windows Forms-vezérlőelemet telepítünk le az internetre, amelyről egy webalkalmazásba integrálódik, hogy biztosításos futtatáskor nyezetben működik, és nincs korlátlan hozzáérés a helyi fizikai szerehez.
 - Egy ClickOnce alkalmazását telepítünk, amely biztosításos futtatáskor használható kapcsolódó adatokat (Datasetek, XML-fájlok stb.).
 - El kell menteni az alkalmazás felhasználói beállításait vagy egyéb felhasználóhoz közzött is célszerű lehet.

Az elszigetelt tarolóhoz tártozó API legfontosabb szerepe, hogy egy olyan modosítani kellene. Eznekvül ennek a technológiának a használata az alábbi portban találhatók, amely azzal olvasztanak adatokat, hogy melegítik Kodcsobi függeléknél rövidítve a virtuális környezetet hoz létre, amelybe az alkalmazások attól bizonyságos virtuális környezetet hoz létre, amelybe az alkalmazások attól körülmények között is célszerű lehetséges.

Az elszigetelt taroló

A fenti rövid leírás nem adhat teljes áttekintést erről a rendkívül sokoldalú API-ról, de ez alapján jobban megérthető az elszigetelt taroló szerepének a funkcióiról.

Megjegyzés Ha a számítógépre vonatkozó CAS-beállításokat módosítjuk, előfordulhat, hogy vissza kell álltaní az alapértelmezett számítógéphezéről-beállításokat, ha a rendszerekkel való kompatibilitás miatt. Ezáltal a Runtime Security Policy mappában található Machine Icons, majd a helyi menüből gombbal a Runtime Security Policy mappában található Machine Icons, majd a helyi menüből vissza kell álltaní az alapértelmezett számítógéphezéről-beállításokat, kattintva a jobb vissza a My Computer-Zone kodcsoporthoz. Ezután a My Computer-Zone kodcsoporthoz a Full Trust engedélyezését.

A Microsoft .NET Framework Konfiguráció alkalmazás segítségével állítsuk ki a Full Trust jogosultság visszalíttatása a My Computer-Zone kodcsoporthoz. Ezután a drivernelekkel, például hibák kezelésével, a My Computer-Zone kodcsoporthoz a Full Trust engedélyezését.

Full Trust jogosultság visszalíttatása a My Computer-Zone kodcsoporthoz

A felhasználószintű elszigetelésen kívül az elszigetelt táról szereleveny és/vagy alkalmazásstartomány-azonossági szintjén is képes az adatokat izolálni. Ha az elszigetelést felhasználónkent és szerelevenyeket is engedélyezünk, az alkalmazás ügyanazt a tárolt fogja használni függeltenül attól, hogy a programot melyik alkalmazásstartomány hasznotja.

Teremezettetel fogva az elszigetelt taroló (legálabb) felhasználónakent elszigetelt egy másik adatból az adatokat. Ezért, ha a programunk az elszigetelt tarolóból teljesen elszigetelt taroló (legálabb) felhasználónakent elszigetelt egy másik adatból az adatokat. Ezért, ha a programunk az elszigetelt tarolóból ment adatokat, azokat a .NET az aktuálisan bejelentkezett felhasználó szinten ment adatokat, ahol a munkaállomásra a működésben részt vevő felhasználó szinten lehetséges, és ugyanabboldal az alkalmazásból adatokat ment el, az eltarolt adatokat egypti, az adott felhasználó számára egyéb helyen fogzta.

Az elszigetelt táróló hatókörre

Csakúgy, mint a fajtrendszer más részein tarolt adatokat, a végfelehasználó az elszigetelt tárrolóba helyezett filiokat is atmásolhatja, általában a legtöbb felehasználó soha sem találkozik közvetlenül a számítógép elszigetelt tárrolójával.

Röhetően, hogy a rendszergráfot nagy-ságától, Ha az alkalmazásunk mellétt más alkalmazások is meghennek a felületet, mégis körülönként adatot tárrolnak, lehetséges, hogy a tárroló mérte eléri a megelegensegítőt, hogy a struktúrálta ki a felületekkel szembeni személyes élményt.

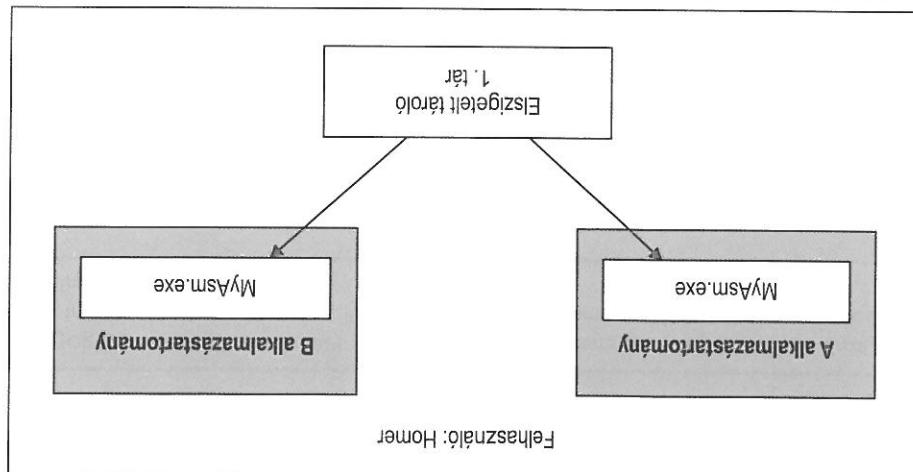
MessageZes A .NET teknikai API-jának használata nem remége ennek a konyvnek. Erről rész-letes információval szolgál a .NET Framework 3.5 SDK dokumentációjának Cryptographic Service Címlü része.

Az elszigetelt táróló használata nem jelenthet azt, hogy ne tárolhatunk adatot a konfígi fájlban (például Kapcsolatstringett), vagy helyezhetünk el külön-bőzö alkalmazásbaellátásokat a rendszerekben adatbázisban. Minthi minden teknológiának, az elszigetelt tárólónak is vanakk hártyányai. Első legfontosabb az, hogy az elszigetelt tárólóba helyezett adat nincs automatikusan titkosítva. Ezért csaknugy, mint a hagyományos IO-műveletek esetén, az erzékelőnyű adatokat (pl. hitelekártya-információkat) manuálisan kell titkosítanunk (és visszafejtendük).

vonatkozó részét.

Az elszügegettől használataival vándorló (roaming) profilok is kialakítják. A vándorló határok lehetségeset teszi, hogy bár a felhasználó kilotonnára számítogépekre jelekkel kezeli be, mégis ügyanazt az alkalmazásadatot kapja meg. Llyen esetben az adattarolás halozzáti helyen törtenik, és igény szerint letöltsődik, ha a felhasználó bejelentkezik egy adott munikaálomástra. Ennek ellenére ezzel a felhasználó belépési idejétől függetlenül minden szolgáltatótól megérheti a hozzáférést.

20.20. ábra: Felhasználó- és szerelemeyszintű izoláció



Elszügítetéles felhasználó és szerelvénny alapján

használjak (lásd a 20.20. ábrát).

Fbbenn az esetben az elérhető legelérgezettebb bizonyíték alapján (pl. az erős név) jön letre a tarolo neve. Igyl elerhetségb, hogy ugyanazon a gépen egy időben a program több példányát használ, amelyek minden ügynámazt a tarolo

A könnyvátrák neveivel vagy azzal, hogy ezeket programozzattan megad-
juk a tarolók leírására során, nem kell foglalkozunk. Ezek a névek auto-
matikusan jönnek letre a felhasználó azonosságához, és a szükséges szerelemy
es/vagy alkalmazásstartomány-bizonyítékok alapján.

Ez alatt található számítalan (teleisen körülöndhatlan névű) alkonyvatár, ame-
lyeket az elszigetelt tarolóhoz API-hoz leter érhet tart karban. A 20.22.
ábra a taroló helyét mutatja egy Vista futtatás számítógépen.

C:\Documents and Settings\<felhasználó>\Local Settings\Application
Data\IsolatedStorage

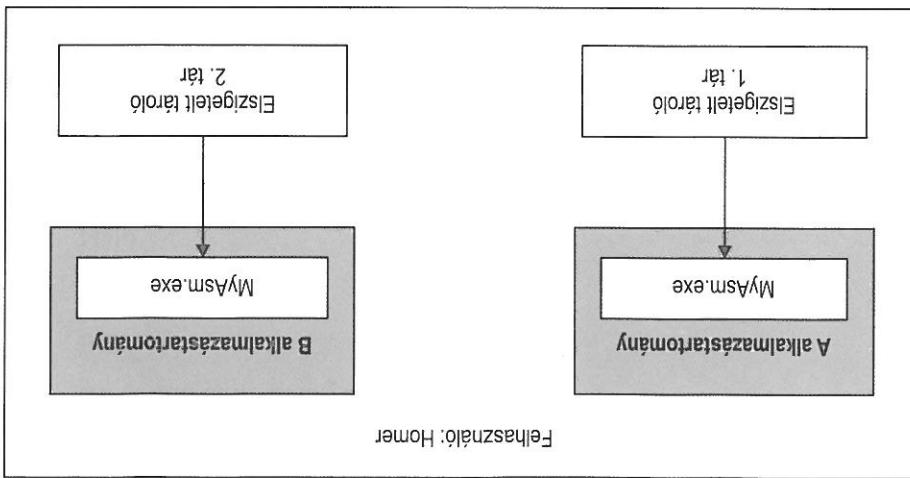
Egy Windows XP számítógépen viszont:

C:\Users\<felhasználó>\AppData\Local\IsolatedStorage

Az elszigetelt taroló nem más, mint a .NET-ét futtató számítógép fájlrendszere
réneke erre a célra szánt része, vagyis nem különbozik a C:\Windows, C:\Prog-
ram Files vagy barátomról más, a merevlemezről levő könnyvátról. Mindez azáltal
az elszigetelt taroló pontos helye az operációs rendszertől is függ. Egy adott ta-
roló gyökereinek a helye egy Vista futtatás számítógépen:

Az elszigetelt taroló helye

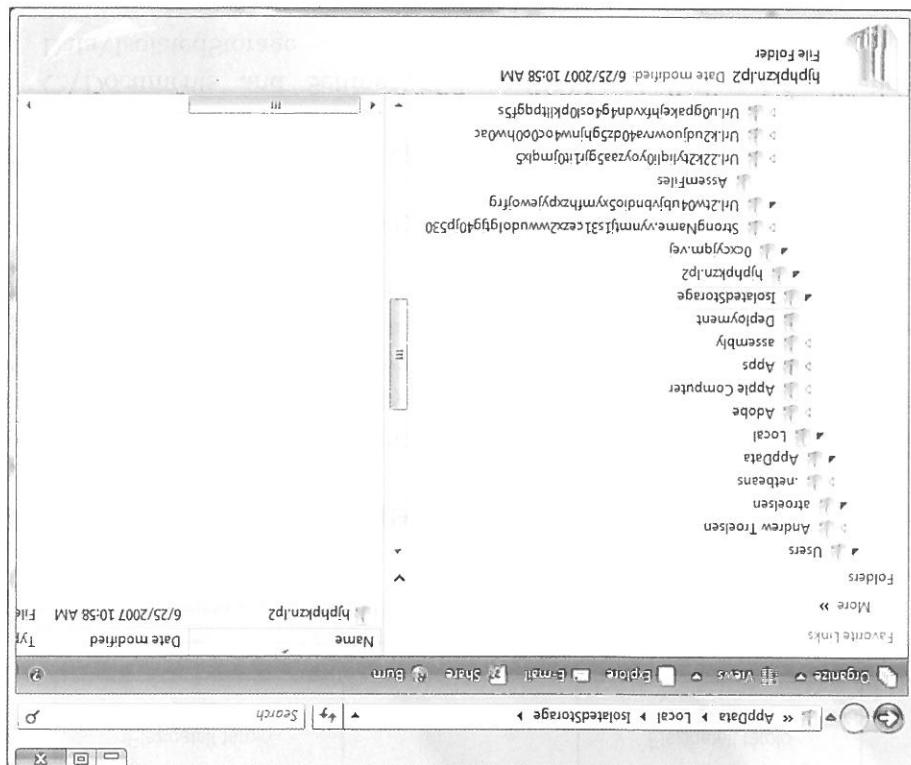
20.21. ábra: Felhasználó-, szerelemy- és alkalmazásstartomány-szintű izoláció



Elszigetelés felhasználó, szerelemy és alkalmazásstartomány alapján

Az elszigetelt taroló

20.22. abra: Elszigetelt törölkögy Vistai fürtához számtalogszer



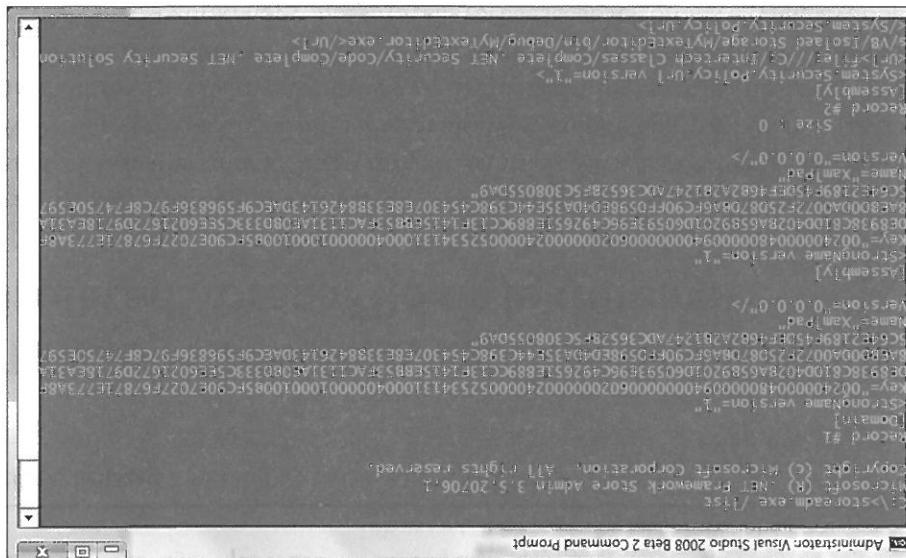
- Zöna
 - Hely
 - URL
 - Erios név
 - Kibocsátó tanúsítvány

A bizonyteleneket gyűjtésre sorban a CLR az alábbi lista tart ellenörzi (a legspeciálisabb bizonyteleneket kezelve), ebben a sorrendben:

alapvető típusával együtthatóan.

A System. 10. Isolated storage tipusa

20.23. ábra: Az aktuális felhasználó tárrolt minták megjelenítése a storeadmin.exe programmal



A NFEI frameworkk 3-5 SDK taratalmaz egy storaedam, e xe nevu paranccsottnak eszkozt is, amellyel a szamtogep alkutatis tarollorendszereket kezelhetiuk. Az eszkoz segitsegevel megtetintethetik az epben bejelentkezett felhasznalo tartoziot (a /1 ist parametrel), es torolhetiuk az aktuialis felhasznalo tartoziot (a /remove argumentummal). A 20.23. abran a /1 ist kapcsolto kiemelete lathato.

Mint azt az elozo abra is mutata, ez az eszkoz megjeleniti az izolacio szintjet (szerelveny, alkalmazastaromany) es a tarolo letrehozasahoz haszszoktaratmata. Ez a new mutataja meg az egys eszkoz tartoziheleyeket a Windows mezozolevel, vagy programozottan kell beolvasnunk az adatokat.

Az elszigetelt tárlo kezelése a storeadm.exe segítségével

AZ elszigetelt táró

```

    IsolatedStorageFile.GetUserStoreForDomain();
    IsolatedStorageFile store = new IsolatedStorageFile();
    // Alkalmazásstartomány-azonosító lapján (rövid).
    // Elszigetelt tároló megnyitása szerelvénnyé - és
    {
        static void GetAppDomainStorageForUser()
    }
}

```

Vetkésző példákak:

GetUserStorageForAssembly() metódusai is használhatók. Vizsgáljuk meg a Ko-tárolókban lévő alkalmazásokban. Az IsolatedStorageFile típus statikus GetUserStorageForDomain() vagy IsolatedStorageFile típus statikus elérésétől függetlenül, hogy az IsolatedStorageScope felismeréséig elérhetővé válik a környezetben. A.NET a tárolókat legalább felhasználónként mindenkorábban elérhetővé teszi a tárolókat, de bemeneti paramétereket nem lehet használni. Kent el kell dönteniink, hogy milyen szintű izolációt szeretnünk. Ha az elszigetelt tárolóban szeretnénk alkalmazásadatot tárolni, elso lépés-

Tároló letröhözása az IsolatedStorageFile objektummal

20.14. táblázat: A System.IO.IsolatedStorage típusai

Szabvány	Jelentés	típus
IsolatedStorageException	Előfordulhatnak a hatalmasított kivételek.	Exception
IsolatedStorageScope	Ez a típus határozza meg, hogy milyen kivételeket kell elszigetelni a programból.	IsolatedStorageScope
IsolatedStorageException	Ez a típus határozza meg, hogy milyen kivételeket kell elszigetelni a programból.	IsolatedStorageException
IsolatedStorageFile	Ez a tag jelezte azt a területet, ahol az elszigetelt tárolókban található fájlok olvasását, írását, letröhözést teszik lehetséges.	IsolatedStorageFile
IsolatedStorageFileStream	Ez a típus az elszigetelt tárolóban található fájlok rolja a fájlokat és környvtárakat tárja.	IsolatedStorageFileStream
IsolatedStorageFileScope	Ez a tag jelezte azt a területet, ahol az elszigetelt tárolókban található fájlok olvasását, írását, letröhözést teszik lehetséges.	IsolatedStorageFileScope

20. fejezet: Fájlműveletek és elszigetelt tárolás

Tag	Jelentés
CurrentSize,	Ezek az irányzásoknak megfelelően tárolhatók az el-
MaxiumSize	szígeire a tárolási kapacitásnak megfelelően.
Scope	Megadja az izoláció hatókörét (félhasználó, szerele-
CreateDirectory()	Létrehoz egy új könyvtárat a tárolóban.
DeleteDirectory()	Töröl egy könyvtárat a tárolóból.
GetDirectoryName()	Ez a metódus lehetővé teszi, hogy végigterajunk a nevezetű könyvtárat.
GetFile()	Visszaadja a tárolóban található fájlok listáját.
GetStore()	Ez a titkosszerű módszertámogatás a megalapozott alkalmazásoknak és az izolációs hatókörnek megfelelő elszigetelt területen adja a vissza-

Bármeilyik módszert is választhatunk, az eredményként egy IsolatedStorageFile osztályt kapunk. Ennek a típusnak a legtisztábban a tárolóba adatokat ír-objektumot kaphatunk. Tárolókészítésével a tárolóba adatokat ír-objektumot kaphatunk, amiből olvashatunk, vagy létrehozhatunk tetszőleges egységi könyvtáratunk, a tárolóban minden objektumot kaphatunk. Ezek az irányzásoknak megfelelően tárolhatók az el-

```
{
    IsolatedStorageScope.Assembly, null, null);

    IsolatedStorageFile store = IsolatedStorageFile.GetUserStoreForAssembly();

    // Megnyitunk egy tárolót
    // szereleványazonosító lapján (rovivid).

    static void GetAssemblyStorageForUser()
    {
        IsolatedStorageScope scope = IsolatedStorageScope.Domain, null, null);

        IsolatedStorageFile store = IsolatedStorageFile.GetUserStoreForAssembly();

        // Vagy egyszerűbben a jelenetet, es meghívjuk a GetStore() metódust.

        IsolatedStorageFile store = IsolatedStorageFile.GetUserStoreForAssembly();
    }
}
```

Tároló létrehozása az IsolatedStorageFile objektummal

```

    {
    }
}

sw.WriteLine("This is my data.");
sw.WriteLine("Cool, huh?");

using (StreamWriter sw = new StreamWriter(stream))
{
    // Az adatfolyamot "becsomagoljuk" egy streamwriter
    // objektumba, és kírunk valamityan szöveget.
    // Az adatfolyamot "becsomagoljuk" egy streamwriter
    // használható az IsolatedStorageFileStream trükk.
    // Létrehozzuk az IsolatedStorageFileStream trükket,
    // újrahasználjuk bizonnyitáskok alapján.
    // Elszigetelt tároló megnyitása szerelvénnyel
    // static void WriteToFile()
    // {
    //     using (IsolatedStorageFile store =
    //         new IsolatedStorageFileStream("MyData.txt",
    //             FileMode.OpenOrCreate, store))
    //     {
    //         store.WriteLine("This is my data.");
    //         store.WriteLine("Cool, huh?");
    //     }
    // }
}

```

alábbi segédmetódusát hívja meg:

Miután létrehoztuk a tárolót, következő lépésekkel létre kell hozunk az Iso-
latedStorageFileStream típusú objektumot. A többi IO-adatfolyamhoz hasonlóan ez a típus is
náll fajlt jelenít a tárolóban. A többi IO-adatfolyamhoz hasonlóan ez a típus is
beállítható a korábban ismertetett típus. IO.FileMode felisröléssel. Ennek
bemutatásához hozzunk létre egy konzolalkalmazást StreamHello. Újratölteni
amelybe importáltuk a System.IO-t. Ezután minden másztatás után elszigetelt tárolót
rekeret. Ezután modositunk úgy a Main() metódust, hogy a program osztály-
ellen, amelybe importáltuk a System.IO-t. Ezután a szintetikus elszigetelt tárolóval
kezelhetővé válik a rendszer. Íme a kód:

Adat irása a tároloba

20.15. táblázat: Az IsolatedStorageFileStream tagjai

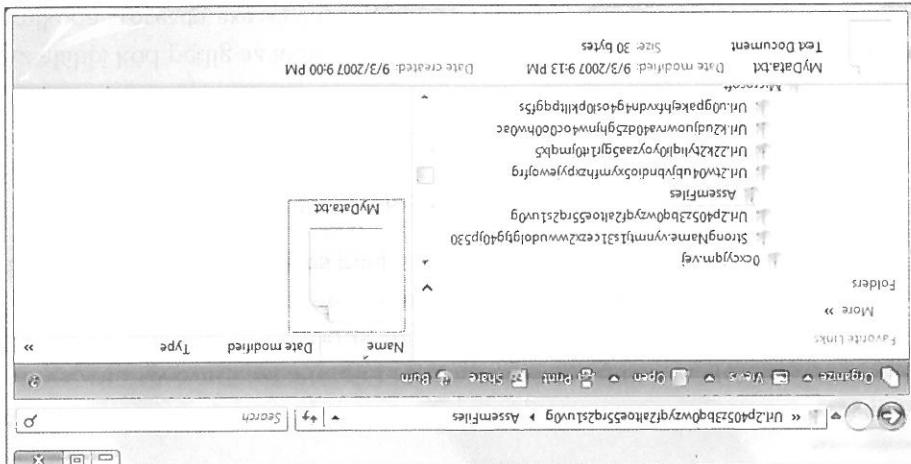
Tag	Jelentés
GetUserStoreForDomain()	Ez a metódus megszéri az alkalmazásdomain-ét
GetUserStoreForAssembly()	Ez a metódus megszéri a hívó szerevénnyazonosító- jának megfelelő elszigetelt tárolót.
Remove()	Ez a metódus eltörli a tárolókat.
SerializeXmlNode()	Szerelvénnyazonosítónak megfelelő elszigetelt tárolót.
WriteToFile()	Jelentés

20. fejezet: Fájlirányeltek és elszigetelt tárolás

A felhasználó töröljégekben elhelyezett adat olvasása is hasonlagon egyszerű. Nézük meg a következő metódust (amelyet célszerű szintén a Main() függvényből meghívni a WriteTextStorage() metódus után), amely a MyData.txt fájlban található információt törlésse, és megjeleníti a konzolon:

Adat olvasása a tárrolóból

20.24. ábra: Az elszigetelt töröloba helyezett szővegjárat



a BinaryWriter osztályt.

1. remezesetesen az ISO 14971-es tervezési követelményeket teljesítő termékekkel szemben a gyártóknak meg kell bizonyítani, hogy a termék megfelel a követelményeknek. Ez a követelményeket teljesítő termékekkel szemben a gyártóknak meg kell bizonyítani, hogy a termék megfelel a követelményeknek.

Kürt ket soft latjuk.

Tarolo leterhezésére az IsolatedStorageFile objektummal

Az előző példakban nem hoztunk lete az egységi hierarchiát a különbségek adat-
fajtól különbségek adat- fajt közvetlenül a tránszformálásra. Ez helyett a mydata.txt fájlhoz szintű CreateFactory() metódussal hozhatunk letre. Az
előző példányszámú CreateFactory() metódussal hozhatunk letre. Ez

Egyedi konyvtársztuktúra letrehozása

AZ alábbi kod példáig az aktuális fejleszsnáló oszszes trólogiat törli (Ugyanilyen mint amikor a storeadm.exe segédprogramot a /remove paramétereit indítjuk):
// Töröljük a fejleszsnáló oszszes trájolját.
IsolatedStorageFile.Remove(IsoLatedStorageScope.User);

```
az IsolatedStorageFile.Delete() metódus két modoszter biztosít a törölök törlésére. A Del-  
danyiszintű Remove() azt a törölt törli, amelyik meghívja. A statikus Isolated-  
StorageFile.Remove() esetén a törölt törli, amelyik megfelelő törölhetőnek vélez-  
ni, és törli a fájlt a felhasználó összes törölőjét. User erőfeszítés vezeti  
a dual torlás aktív törölést, es megsemmisítőt annak tartalmában:  
// Töröljük az aktuális felhasználó aktuális táróhoz járőr adatait.  
IsolatedStorageFile store = IsolatedStorageFile.GetUserStoreForAssembly();  
store.Remove();
```

Felhasználói adat törlése a tárrobbal

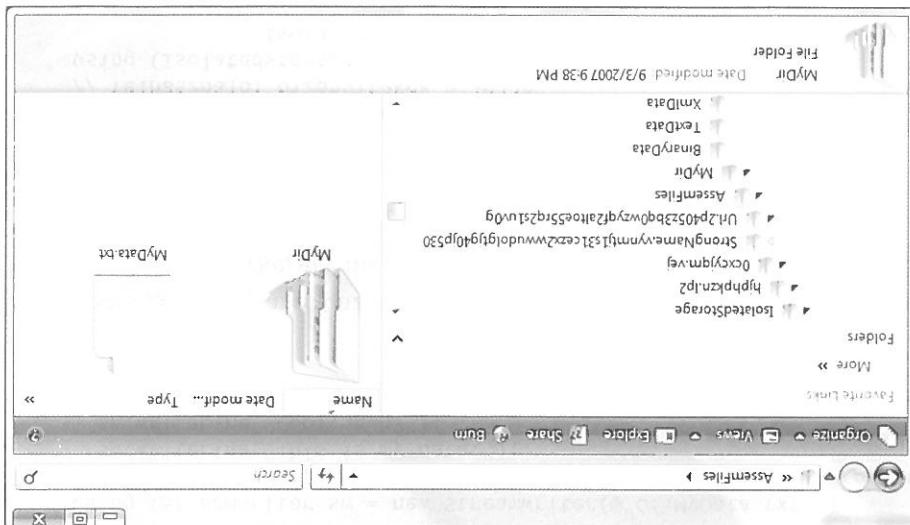
```

private static void ReadTextFromISOStorage()
{
    using (IsolatedStorageFile store =
        IsolatedStorageFile.GetUserStoreForAssembly())
    {
        using (IsolatedStorageFileStream stream =
            store.CreateFile("MyData.txt"))
        {
            new FileStream("MyData.txt", FileMode.Open,
                FileAccess.Read, FileShare.Read);
            new StreamReader(stream);
            string becsomagoljuk = sr.ReadToEnd();
            // "Becsomagoljuk" StreamReaderbe.
            using (StringStream sr = new StreamReader(isStream))
            {
                string alltheData = sr.ReadToEnd();
                Console.WriteLine(alltheData);
            }
        }
    }
}

```

Förträskad A Simplesztőrőlage projekteket a förträskokhozváttarban a 20. fejjezet alkonyváttara taratálmazza. A forrásokhozváttarral csak a Bevezetés xli., oldalát.

20.25. ábra: Környezetstruktúra leírására megadott tárolóban



Ha a jelen metodust megelvihjuk a Matinó húgsvényből, megtalálhatunk a Leter-hozott konyvtársstruktúrát az elszigetelt tárlohoz használva (lásd a 20.25. ábrát).

```

private static void CreateStorageDirectories()
{
    // Hagymányos es Fordított perjellet is használhatunk.
    using (IsoLabeledStorageFile store =
        IsoLabeledStorageFile.CreateUserStoreForAssembly())
    {
        // Hagymányos es Fordított perjellet is használhatunk.
        store.CreateDirectory("MyDir\BinaryData");
        store.CreateDirectory("MyDir\TextData");
        store.CreateDirectory("MyDir\XMLData");
        store.CreateDirectory("MyDir\TextData");
    }
}

```

Elszüggetett törölőben az alkonyvatrakat nem kepezhetik le objektumokra, hanem olyan sztringeket kell átdununk, amelyek a leírásban környvtárhoz nyújt hozzáférését. Nezzük meg a következő kódot:

I aralo leterehozasa az isolatedstoragefile objektummal

```

    // objektumba, és kírunk valamityan szöveget.
    // Az adattfolymatot "becsomagoljuk" egy streamwriterre
    {
        new IsolatedStorageFileStream("MyData.txt",
            FileMode.OpenOrCreate,
            FileAccess.ReadWrite,
            FileShare.ReadWrite) = using (IsolatedStorageFile file =
        // Letrehozzuk az isolatedstoragefilistem trüpszt.
        IsolatedStorageFileStream fileForAssembly = IsolatedStorageFile.GetUserStoreForAssembly()
    }
    // Elszígegett táróhoz megnyitásra szerelelveny - és
    // fehásználó bizonyítékok alapján.
    using (IsolatedStorageFile store =
        new IsolatedStorageFile("C:\MyData.txt",
            FileMode.OpenOrCreate,
            FileAccess.ReadWrite,
            FileShare.ReadWrite) = using (IsolatedStorageFile file =
    private void btnIsolatedStorage_Click(object sender, EventArgs e)

```

A második gomb úgyanazt az adatot egy elszígegett táróban levő fájlba írja ki. A Click eseménykezelő meghívására a korábbi projektben már megvolt a WriteToFileStorage() metódusozás.

```

    {
        sw.WriteLine("This is my data.");
        sw.WriteLine("Cool, huh?");
    }
    using (StreamWriter sw = new StreamWriter("C:\MyData.txt"))
{
    private void btnFileIO_Click(object sender, EventArgs e)

```

Az eggyik gomb a helyi merevlemezre próbál adatot menteni hagyományos IO-megoldásokkal (nefelejtse ki a importált a szintet. Io és a system.

Tegyük fel, hogy van egy Windows Forms alkalmazásunk (neve: FileOr-IsostorageInApp), amely egy két gombot tartalmazó ultralából áll. (A Windows Forms API-ról részletekben a 27. fejezetben lezyszűrő, de az egyséses Button típusa sok kattintási eseményt mar itt is le fogunk kezelni).

Mégjelent, hogy az elszígegett táróhoz ez időig puaszán a fehásználósztintű adattárolás egyszerű módjának tunt (ez önmagaiban is hasznos dolgoz). Am az egyik probléma, amit ez az API orvosol, az az, hogy a nem Full Trust jogosultsággal futható alkalmazások is biztonságosan tárolhatassanak adatokat.

Az elszígegett táróhoz működés közben:

futtatható alkalmazást a Végelehasználó számítógépéről. ClikO nincs száll telepítőjük. A ClikO nincs segítségevel telepíthetünk tiszavilági webkiszolgáloval - gedeleykészlettel fússzon. Ennek erdekeben az alkalmazásunkat a ClikO nincs Telepítésük úgy az alkalmazást, hogy az a jövől korlátozott Internet-Zone-en.

A biztonságí zóna korlátzása

töltük be, amely a Full Trust jogosultságot biztosítja a szerelvénny számára. Ennek oka az, hogy az alkalmazást a My-Computer-Zone kedcosportból tölt. Gombra is kattintunk, letrejön a megfelelő szöveges adatot táralomhoz íj fel. Kombinációval) fordítjuk le, és futtatjuk, azt tapasztaljuk, hogy bárminelyik Ha a programot közvetlenül Visual Studio 2008-ban (a Ctrl + F5 billentyű-

UsageAllowed = IsolatedStorageContainment.AssemblyIsolationUser)] IsolatedStorageFilePermission(SecurityType.Allow, RequestMinimum, [assembly]:

leszármazott típusunk kódjaihoz: érhejük el, ha a következő szerelvényszámú attribútumot hozzáadjuk a Form-venyszámú, elszigetelt tárolási izolációs jogosultsággra van szüksége. Ez úgy díabán arról ertesítjük a CLR-t, hogy az alkalmazásunknak legálabb szerel-lyen biztonságí beállításokra van szükség (az egyséb részletek mellé). A Példában a nevezetű attribútum található, amelyek segítségevel erősítetteti a biztonságát alrendszer arról, hogy az adott szerelvénny megfelelő működéséhez min-használható attribútum hozzá, amelyek segítségevel osztályt definíáló C#-fajlhoz:

using System.Security.Permissions;

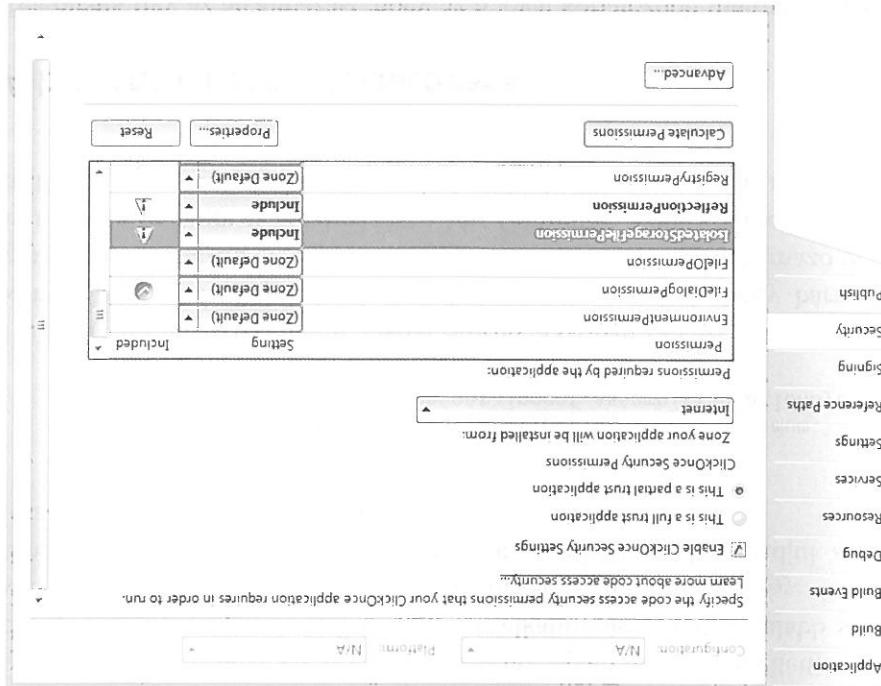
Mielőtt kiírunk az alkalmazást, adjuk hozzá az alábbi using direktívát a kezdett Form-leszármazott osztályt definíáló C#-fajlhoz:

Az IsolatedStorageFilePermission attribútum

```
using StreamWriter sw = new StreamWriter(stream);
sw.WriteLine("This is my data.");
sw.WriteLine("Cool, huh?");

sw.WriteLine("This is my data.");
sw.WriteLine("Cool, huh?");
```

20.26. ábra: Az alkalmazás biztonságízonyítánság korlátozása



Eloszor nyissuk meg a projekt Properties lapját úgy, hogy a Solution Explorer mezhez hagyományos ICO-műveletek felhasználásával. Ólyan telepítőszkriptet kell írunk, amely arra kényszeríti a programot, hogy az Internet Zoneban futson, amely nem enged hozzáférés a merevlemezhez. Íme a hagyományosan telepített programok.

Dejjel telepíthetek, ezért úgyanolyan biztonságízonyításokat fogosultak rendelkezni. Az alapértelmezés szerint a ClickOnce alkalmazások Full Trust engedélyt kapnak minden esetben duplán kattintva a Properties ikonra. Majd kattintsunk a Security részben az URL-ben megadott számra. Íme a teljesen általánosan elérhető ClickOnce Security Settings.

Megjegyzés: A ClickOnce teljes ismeretese túlmutat ennek a fejezetnek a keretében. Ha valaki meg nem telepítette illy módon alkalmazását, csak kivessze az alábbi utasításokat (és igezen szerint olvassa el a .NET Framework 3.5 SDK dokumentációját).

A tavolti alkalmazások esetében a könyvtárban található, aholnan egy szerveren az URL-ben megadott számra megegyező a könyvtárban található, es telepített.

Mivel az alkalmazásról úgy állítottuk be, hogy korlátoszt környezetben fússzon, ha arra a gombra kattintunk, amely a system. 10 hüposok használataval próbál adatot menteni, akkor a 20.27. ábrán levő biztonságkívánt fog megjelenni. Elleneben, ha arra a gombra kattintunk, amely az elszigetelt táróhoz segítsé- gével ment adatot, az alkalmazás sikeresen lefut.

Az eredmény megtékinthese

<http://localhost/MyisolateStorageApp/>

Kattintson a tulajdonosszámhoz a Publish tulajdonságokban található részbeni szolgáltató részén. Majd kattintson az oldal alsó részében található Publish Wizard gombra, és a varázsló eljövő lapján írjon be az alábbi URL-t, hogy letrehozzuk az alkalmazást tárolt új LS virtuális konnyvtárat a helyi gépen:

Az alkalmazás kozmetikai webkiszolgálón

Ehhez kattintásunk az Enable ClickOnce Security Settings lejelölőnégyzetre, vagy a szasszuk a This is a partial trust application radiógombot, majd válasszuk az internet lehetőségeit a zónákat felosztó legörbüle listamezőből. Végül, de nem utolsósorban kattintsunk a Calculate Permissions gombra a biztonságigényelt végső engedélykészletet (lásd a 20.26. ábrát).

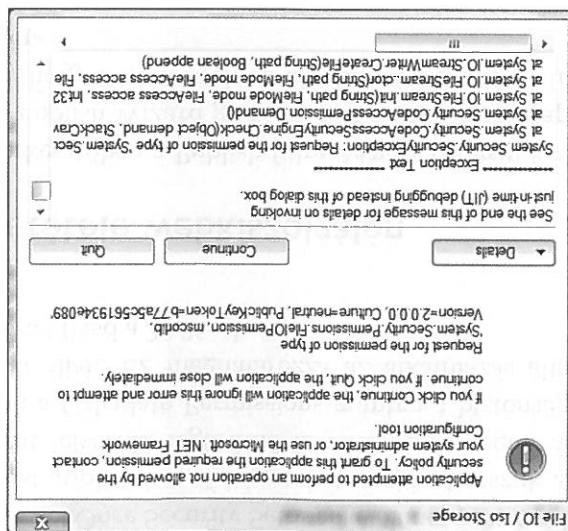
A fejezet második része az elszigetelt tárrolat mutatta be. Ez az API lehetővé teszi a programok számára, hogy egy biztonságos virtuális környezetet hoz létre a fizikai számítógépen.

A fejezet második része az elszigetelt tárrolat modellben korlátolt biztonsági környezeteket körülöznek. Bár a programozási modellekben végrehajtott olvasási és írási műveleteket még akkor is, ha az alkalmazás szorán a drívtypel szerződött funkcióinak között van különbség, ezeket rendelkezésükre bocsátják, amelyek megkönnítenek ez a folymatot. Ezek számos olyan olvasó/típus (stremwritter, stringwriter, binarywriter) használhatók nyírás bátye-folyamokkal végzések műveleteket, a szintet. Az írószolgáltatók által nyújtott funkcióinak között van különbség, ezeket a stream leszámítva. Mivel a stream-leszámítás több, az absztrakt stream osztályból származik írásossal, amelyek közül a található fizikai fájlok és környvtárak kezelését. Ezután megismertük a mérévlemezben található. Meg tudunk, hogy ezek az osztályok lehetővé teszik a mérévlemezben a fejezet előtt meghismertről a direktory(Info) és File(Info) típusok használatát. Még tudunk, hogy ezek az osztályok származnak a Stream-leszámításból, az absztrakt Stream osztályból származik írásossal, amelyek közül a található fizikai fájlok és környvtárak kezelését. Ezután megismertük a mérévlemezben található. Meg tudunk, hogy ezek az osztályok lehetővé teszik a mérévlemezben a fejezet előtt meghismertről a direktory(Info) és File(Info) típusok használatát. A fejezet előtt meghismertről a direktory(Info) és File(Info) típusok használatát.

Osszefoglalás

Forrásokból A FileOrIsolateWiApp projektet a forrásokból nyertünk a ZD. fejezet alkonytvátra tartalmazza. A forrásokból nyert tárrolat lásd a Bevezetés XLV. oldalán.

20.27. ábra: Biztonság! veszély. Nem lehet hozzáérni a helyi felhőrendszerhez az internet zónához



20. fejezet: Fájlítműveletek és elszigetelt tárrolás

meglehetősen egyérfelmi (ha megérthük az alapvető fajlmivéleteket), a kapcsolodo temakörök valamelyest megbonyolítják. Ennek fenyegeben roviden áttekinthetünk a kódhozszármaztatásról.

Ebben megtudtuk, hogy a szervizények alkalmazásától többnyire a CLR részére bizonyíthatókatt szolgáltatnak. Ez alapján kerüljük során a folyamatokat szolgáltatni. Ez a szolgáltatásnak belé az alaperelmezett engedélyezésétől rendelkező kodcsupportokba. A CAS fajlmivéletek szempontából fontos tudnunk, hogy ha az alkalmazás nem kap Full Trust jogosultságot, vagy ha az alkalmazás nem rendelkezik minden jogosultsággal, a hagyományos IO-műveletek bizonsgában kiütemezhetők. Ezáltal szemben az elszigetelt táróhoz segrisztázott adatokat.

A sorosítás kifejezés az objektumok folyamba (fájlfolyamba, memóriaobjektumra) beírása, a sorozat minden részét (az elérhető szolgáltatásokat) az objektumokba írásával tennének. Ez gyakran szabványos (NET, XML, SOAP) vagy speciális (SOAP, REST) szemantikai rendszerekkel történik, amelyek lehetővé teszik a folyamat tesztelését. A szolgáltatók minden részét (az elérhető szolgáltatásokat) az objektumokba írásával tennének. Ez gyakran szabványos (NET, XML, SOAP) vagy speciális (SOAP, REST) szemantikai rendszerekkel történik, amelyek lehetővé teszik a folyamat tesztelését.

Az objektumsorosítás

A sorosítás kifejezés az objektumok folyamba (fájlfolyamba, memóriaobjektumra) beírása, a sorozat minden részét (az elérhető szolgáltatásokat) az objektumokba írásával tennének. Ez gyakran szabványos (NET, XML, SOAP) vagy speciális (SOAP, REST) szemantikai rendszerekkel történik, amelyek lehetővé teszik a folyamat tesztelését. A szolgáltatók minden részét (az elérhető szolgáltatásokat) az objektumokba írásával tennének. Ez gyakran szabványos (NET, XML, SOAP) vagy speciális (SOAP, REST) szemantikai rendszerekkel történik, amelyek lehetővé teszik a folyamat tesztelését.

A sorosítás kifejezés az objektumok folyamba (fájlfolyamba, memóriaobjektumra) beírása, a sorozat minden részét (az elérhető szolgáltatásokat) az objektumokba írásával tennének. Ez gyakran szabványos (NET, XML, SOAP) vagy speciális (SOAP, REST) szemantikai rendszerekkel történik, amelyek lehetővé teszik a folyamat tesztelését.

Sorosítás világába Bevezetés az objektum-

HUSZONEGYEDIK FEJJEZET

A .NET-objektumsortással az objektum állapotának mentése meg lehető-
sen egy szérsűrű mondháto, am a hatterben zájba folymatok elégé összette-
tek. Ha egy objektumot például egy folyamba mentünk, minden társított adat
(összefüggő adatok, tárolt objektumok stb.) sorosításra is automatikusan megtör-

```
// Az objektumot egy helyi fajlban tárolja.
using(Stream fsream = new FileStream("User.dat", FileMode.Create, FileAccess.Write, FileShare.None));
{
    BinaryFormatter binformat = new BinaryFormatter();
    Userrefs userrefs = new Userrefs();
    // Tételezzük fel, hogy a Userrefs az alábbi tulajdonságokat
    {
        static void Main(string[] args)
        {
            // Tételezzük fel, hogy a Userrefs az alábbi tulajdonságokat
            {
                // A BinaryFormatter az általapottadatokat bináris formátumban menti.
                // A BinaryFormatter az általapottadatokat bináris formátumban menti.
                // Ez az objektum teljes állapotát csupán néhány soros forráskódossal is elment-
                //hetők. Nezzük meg az alábbi Main() metodusát:
                {
                    public class Userrefs
                    {
                        // Különösöző adatok...
                        {
                            static void Main(string[] args)
                            {
                                Console.WriteLine();
                            }
                        }
                    }
                }
            }
        }
    }
}
```

Egy az objektum teljes állapotát csupán néhány soros forráskódossal is elment-

meg saját magunknak, ha a Userrefs osztályt egy szérsűrűen a [Serializable]-
Bar a fenti megalás teljesen működőképes, rengeteg időt spórolhatunk
beolvásni minden egyes ertéket.
Attribútummal jelöljük meg:
Bar a fenti megalás teljesen működőképes, rengeteg időt spórolhatunk
új Userrefs objektum újratöltöttére, és az
nánk tölteni a memoriából, szükseg lenne a rendszerekkel vissza akar-
talan keletkezni. Ugyanilyenkor, ha az adatokat a fájlokból vissza akar-
talanírni, amely korábban 20 adattérmezőt foglal magába. Ha a rendszerek
es lehetsége szeretnének tenni, hogy a felhasználók elmenhessék a beállításokat
(ablak színe, betűméret stb.). Ebben definiálhatunk egy Userrefs nevű osz-
tályt, amely korábban 20 adattérmezőt foglal magába. Ha a rendszerek
es lehetsége szeretnének tenni, hogy a grafikus asztali alkalmazásokhoz tartozók letre,

Ennek egyeszerű szemleteresre tettezzük fel, hogy olyan osztálykészleket hozunk létre, amely néhány gépkocsit modellez. Van tehát egy car nevű osztály kibövítő a car alaptpust. A 21.1. ábra ezeket a lehetséges kapcsolatokat összefoglalja, amelynek „van egy” Radio osztálya. Egy másik, JamesBondcar névű osztályt is alkottunk, de nem használtuk.

Az objektumgráfokban található objektumok egyéni numerikus értékkel rendelkeznek. Az objektumgráf tagjaihoz rendelt számok tetszőlegesek, és semmilyen kapcsolatban nem állnak a külvilággal. Ha minden egyes objektumhoz hozzárendeljük egy numerikus érték, az objektumgráf minden egyes objektum függőséghalmazát képes elmenteni.

Az objektumgráfokban található objektumok minden objektumot száma-

leneré, hogy ezt a paradiigmát is megelhetősen jól modellezik.

Az objektumgráfok egyeszerűen azt dokumentálják, hogy az objektumok működését követően milyen objektumokat kepezik le a klasszikus OO kapcsolatokat (mint például az „az egy” vagy a „van neki egy” kapcsolatokat) annak el-

szemben, hogy a különleges objektumokat kepezik le a halmozat objektumok működését. A különleges objektumokat ez a halmozat objektumokat nevezzük.

Ha egy objektumot soroztunk, a CLR valamennyi kapcsolódó objektumot szá-

Az objektumgráfok szerepe

Az objektumgráfokat baromiljen system, IoT, stream API, származó típusba való átvittében használjuk. Az előző példában a userfejs objektumot a Filestream típus elmentethetők. Az objektumokat helyi fizálba. Ha azonban az objektumokat inkább segítségével mentettük a helyi fizálba. Ha azonban az objektumokat inkább elmentethetők. Az objektumokat más típusokat használó SOAP-vagy XML-formátumban is elmenthetők. Ezek a formátumok megelhetősen hasznosnak bizonyulnak, ha biztosítani akarjuk, hogy az elmentett objektumokat zavarthatatlanul lehessen használni a kilombázó operációs rendszerekben, nyelvezetben és architektúrákban.

A .NET-sorozatos logikai részben a objektumgráfok kilombázó formátumú mazasai lincsben levő adat tárolják. Az egymassal kapcsolatban levő objektumok egy objektumgráf alárazolhatók.

teknik. Ezért ha származtatott osztályt szerelemek elmenteni, minden, a szár-

Megjegyzés Az XML Serializer típus (lásd Kézirat) nem az objektumgráfokkal minden el az alapötöl, hanem tövábbra is prediktív módon sorosítja és általja vissza a kapcsolódó objektumokat.

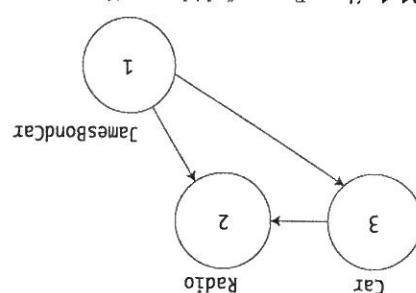
A sorosítási folyamatban az objektumok közötti kapcsolatokat ábrázoló graf automatikusan, a hatterben jön létre. A későbbiekben is látni fogunk, hogy lehetősége van arra, hogy irányítsuk az objektumgráf letrejöttét a sorosításnál.

Egy példányát, az objektumgráf biztosítja, hogy a radio es a car típusok is megfelelően sorosítják egymást. Mindenesetre ha sorosítjuk vagy viszazzállítjuk a Jamesbondcar osztályt is, akkor a radio típusnak a 2-es objektum-jamesbondcar() függvényét viszonyban áll minden 3-as, minden 2-es objektum-magános funkcióval, nincs szüksége semmiről. Végül pedig a 1-es objektum (a "magános funkciók", amelyeket a radio típusnak a radio típusnak a 2-es objektumban ábrázol) minden osztálytól eltérően ábrázolja.

[Car 3, ref 2], [Radio 2], [Jamesbondcar 1, ref 3, ref 2]

Az objektumok olvasásakor a nyílik összekapcsolásba használhatók a formulákban dokumentált kapcsolatok ábrázolásához. Az elöző diagramban dokumentált kapcsolatot írtunk röviden a memoriába a kapcsolódó objektumok között, valamint a radio osztályra (mivel az öröklődik ez a védett tagvaltozó). A CLR tervezetben nem rajzol ki a memoriába a kapcsolódó objektumokat, csak a radio osztály hivatalosan radio osztályra (tekinthető a "van neki egy" kapcsolatnak). A Jamesbondcar hivatalosan radio osztályra (mivel adott az "az egy" címzésű). A radio osztály hivatalosan radio osztályra (mivel az "az egy" radio osztály hivatalosan radio osztályra), tehát a 2.1. ábrán azt láthatunk, hogy "függőleg" vagy a "hivatalos", kifejezésekkel. Tehát a 2.1. ábrán azt láthatunk, hogy a car osztály hivatalosan radio osztályra (tekinthető a "van neki egy" kapcsolatnak).

21.1. ábra: Egyeszerű objektumgráf



21. fejezet: Bevezetés az objektumsortolás világába

```
{
    public void isHatchback();
    public Radio theRadio = new Radio();
}
public class Car
[Serializable]
```

Ezit követően adjunk hozzá két osztályt (`JamesBondCar` és `a car`), lassuk el őket a `[Serializable]` attribútummal, és definíáljuk az alábbi tagokat:

```
{
    public String radioID = "XF-552R6";
}
public class JamesBondCar
[Serializable]

{
    public boolean hasSubwoofers;
    public boolean hasTweeters;
    public boolean isHatchback();
}
public class Radio
[Serializable]
```

Kiinduláskenet hozzunk letező egy új konzolalkalmazást SimpleSerialise névvel. Szűrjunk be egy új osztályt `Radio` nevvel, amelyet pedig a `[Serializable]` attribútummal jelölünk, kivéve egy tagvaltozót (`radioID`), amelyet pedig a `NonSerialized` attribútummal jelölünk, ezért ez nem mentődik el a meghatározott adattörzsen. Ez akkor lesz aktív, ha a `JamesBondCar` osztályban:

Sorosítások tipusok meghatározása

Amma� érdekeben, hogy adott objektum a .NET-sorostoszolgáltatások számára elérhető legyen, nem kell mászt tennünk, mint minden egyes kapszon belőle osztályt (vagy struktúrát) fel kell rögzítenünk a `[Serializable]` attribútummal. Ha egy adott típus olyan adattagokkal rendelkezik, amelyeknek nem kellene szereljük meg ezeket a mezőket a `[NonSerializable]` attribútummal. Ez akkor lehetsz, ha olyan tagvaltozók vannak egy sorosítási osztályban, amelyeket lyíjük meg ezeket a mezőket a `[NonSerializable]` attribútummal. Ez akkor lesz aktív, ha a `JamesBondCar` osztályban a `radioID` attribútum a `String` típusú. Ha ez a típus olyan adattagokkal rendelkezik, amelyeknek nem kellene szereljük meg ezeket a mezőket (vagy nem szerelhetők) a sorosítási semmiben, íe-

Objektumok konfigurálása sorosításban

Eltérkintve az OO tervezési elvétől, kérdes lehet, hogy a különöző források hogyan válik a típusok adatmezője vonalakozó hozzáférésre modosítottan kialakítva. Ha a binaryformatterrel vagy a soapFormatterrel mentjük el az objektumok állapotát, abszolút nem számít, milyen hozzáférés-módostól használunk. Ez a két típus kepes sorosítani minden sorosítható mezőt attól függően, hogy ezek nyilvános mezők, privát mezők vagy nyilvános tulajdonságokat használó privat mezők. Vannak azonban olyan adatok, amelyeket nem szereznének az objektumgráfba menteni, ekkor a nyilvános és a privat mezőket selektíven a [Nonserialized] attributummal jelölhetjük, ahogy azt a rádió típus sztringmezővel is tettek.

Az egyszerűség kedvéért ezekben az osztályokban az adatmezőket nyilvánosként határozta meg. A nyilvános tulajdonosokkal felrakozott privat adatok természetesen jobbak lennének az OÖ szemponjához. Az egyszerűség kedvezett azonban ezekre a típusokra nem határozott meg semmilyen előírás konstruktor, ezért azok az adatmezők, amelyeknek nem adtunk előíret, az ilyenkor vátható alapértelmezett eretkeket Kajpálik.

Nyilvános mezők, privat mezők és nyilvános tulajdonosok

Megjegyzés Mivel a XML Serializálási típus nem használja az objektumgráfot, lechimikálálgathatunk minden elemet. A típusokat a `[Serializable]` attribútummal megjelölünk annak érdekében, hogy az objektum általában XML-kent mentenek el. Ha azonban azt szeretnénk, hogy a típusok minden eleme formátumban mentenek el, a típusukhoz minden attribútum sorosítandó tipust a `[Serializable]` attribútumkal meg kell írni.

A [Serializable] attributum nem orokoldik a szülőosztályból. Ha lehet egy osztályt egy [Serializable] attributummal jelölünk típusból származtatunk, a gyermekosztályt is a [Serializable] attributummal kell megjelölünk, különben nem tudjuk elmenteni. Valójában az objektumról minden tagját a [Serializable] attributummal kell sorolni. Ha a binaryFormattal vagy a soapFormattal segítségevel nem szerithető objektumot probálunk sorolni, futásidőben a SerializationException jelenik meg.

```
{  
    public class Car  
    {  
        String licensePlate;  
        JamesBondCar canSubmerge;  
        boolean canFly;  
        boolean isPublic;  
    }  
}
```

- BinaryFormatter
 - SoapFormatter
 - XmlSerializer

Ha a szürkeges attribútumokkal konfiguráltuk a .NET sorosítási semálában szereplő típusokat, a következő lepésben ki kell valasztanunk, melyik formátumot (bináris, SOAP vagy XML) használjuk az objektumok állapotának elmentésére. Az egyes lehetőségeket az alábbi osztályok mutatják:

A sorosító formaző kiválasztása

Hab a BinaryFormatter vagy a SoapFormatter tippst használhatik, mégígylehet-e jílik, hogy az isalíve, a personage és a name mezők mindenjárt két elmentettük a kijelölt folymaba. Az XMLSerializér azonban nem ment el a personage értelemezt, ugyanis ezt a privat adaptálémet nem ajánlottuk ki nyilvános típusfülfelhasználóknak. Ha személy életkorát az XMLSerializérrel szeretnék elmenteni, dönésággal. A mezőt nyilvánosként kell megfaztatraozunk, vagy be kell ágyazunk a privat tagot egy nyilvános tulajdonaságba.

```

    {
        set { FName = value; }
        get { return FName; }
    }

public string FirstName
private string FName = string.Empty;

// Nyitvános tulajdonság/privát adat.
// Nyitvános mező.
private int personage = 21;
// Nyitvános mező.
public bool isAlive = true;
// Nyitvános mező.
public class Person
{
    [Serializable]
}

```

Az Informatter es az RemotingFormatter interfeszek

/// DetinálaVa a System.Xml.dll szerelevenybén.

Végül pedig, ha egy objektumról XML-dokumentumként kívánunk elmeneti, használjuk a XMLserializér tipust. Ebben csak határozunk meg a névteret, ami, használjuk a XMLserializér tipust. Ez a névteret, ezért nem kell minden XML-objektumról kivánnunk elemet. Vagyis az XML-objektumról kivánnunk minden XML-objektumról kivánnunk elemet. Ezért nem kell használnunk:

/// Kotelezo reprezentacije
/// a system.Runtime.Serialization.Formatters.Soap.dll szereLVenyre.
using System.Runtime.Serialization.Formatters.Soap; Soap.
using System.Runtime.Serialization; Serializatⁱon. Formatters. Soap. dll szereLVenyre.

A SOAPFormatter típus az objektumok állapotát SOAP-üzeneteknél ment el, ezáltal a különálló szerelemei közben definíált szabványokat megőrizte. A SOAPFormatter tipus a különálló szerelemei közben definíált szabványokat megőrizte. Ezáltal a szerelemei közötti kompatibilitásról gondoskodhatunk.

```
// hozzájárult a Microsoftnak szerevben gyűjtött rövidítésekkel, amelyeket a binaryFormatter típusú hozzájárult a runtime-szerűen. Ezáltal a runtime-szerűen elérhetők lesznek a formátumok.
```

```

    static void SerializeObjectGraph(IFormatProvider format,
    Stream destination, object graph)
    {
        if (format is IFormattable)
            ((IFormattable)format).ToString();
    }
}

```

Bár a legtöbb sorosztási feladatban valósztányleg nem lesz közvetlenül dol-
gasználásaval sorosítja az objektumgrafot, a következőket figye-
ljen minden metódust kivánuunk éppen, amely ezben osztályok valamelyikének fel-
tartskára binaryFormatter vagy SoapFormatter egy példányát. Tehát ha
morfológiai lehetségekkel, eredményekkel, hagyományos módon, hogy az interfészalapú poli-
gunk ezekkel az interfészeken, minden, hogy az interfészalapú poli-

```

    public interface IRemoteFormatter : IFormatter
    {
        object Deserialize(Stream serializationStream, object header);
        void Serialize(Stream serializationStream, object graph, HeaderHandler[] headers);
    }
}

```

A rendszertől származókhoz használható interfész (ameleyet a
.NET-remoting a hatérválasztáson használ) a `Serialize()` és a `Deserialize()` tagokat
az elosztott sorosztások szempontjából sokkal hasznosabban terheli túl. Az
IRemoteFormatter a sokkal általánosabb IFormatter interfészhez használható:

```

    public interface IFormatter
    {
        void Serialize(SerializationBinder binder, object graph);
        object Deserialize(SerializationBinder binder, Stream stream, object graph);
    }
}

```

A rendszerműködésben a hatérválasztásra vonatkozó meglévő interfész (ameleyet a
Fontosabban `Serialize()` és `Deserialize()` metódusokat, amelyek az objektum-
gráfolását egy adott objektumba helyezik, vagyonnan kivétséssel. Ezekben a tago-
kon kívül az IFormatter megfoghatózza még azt a párt tulajdonoságot is, ame-
lyeket az implementáló típus a hatérválasztával.

A SOAP-formatter az eredeti szerelelveny nyomáit egy XML-nélküli segrítelevevel tárolja el. Csondoljunk a korábban említett Person típusra. Ha a típusnál minősítene. Nezzük meg az alábbi definíciót részleteztet, s közben figyelemünk az SOAP-tízeneket menterülök volna el, a Person készöd elemeit a generált XML-szabványban.

```
<xs:element name="Person">
    <xs:complexType>
        <xs:sequence>
            <xs:element name="id" type="xs:string" />
            <xs:element name="name" type="xs:string" />
            <xs:element name="age" type="xs:int" />
            <xs:element name="isAlive" type="xs:boolean" />
            <xs:element name="notes" type="xs:string" />
        </xs:sequence>
    </xs:complexType>
</xs:element>
```

Az XML-sorai között azonban nem törekzik a teljes típusponthosság megtartására, ezért nem tárolja el a típusok teljesen meghatározott nevét, sem az eredeti szerelelvenyét. Bár ez első ránézésre körülözésnek tűnhet, az ok az XML-adatok megjelenésének nyilt természetében keresendő. Nezzük meg a Person típus egyszerűbb XML-ábrázolását:

```
<xs:element name="Person">
    <xs:complexType>
        <xs:sequence>
            <xs:element name="id" type="xs:string" />
            <xs:element name="name" type="xs:string" />
            <xs:element name="age" type="xs:int" />
            <xs:element name="isAlive" type="xs:boolean" />
            <xs:element name="notes" type="xs:string" />
        </xs:sequence>
    </xs:complexType>
</xs:element>
```

A harmón formázó kozottal legyülvárolokban keresendő (bináris, SOAP vagy XML). Létezik lyamba mentesítések módjában különbségek az objektumgráfakhoz köztük. Különbségek az objektumgráfakhoz köztük a következők:

- Még néhány apróbb pont is, amelyben különbségek egy mástól, ezek közül ki-emelkedik a formázók *tipusprotosság* szempontjában. A binaryFormatter tipus nemcsak az objektumgráfában található objektumok adattípusát menti el, hanem minden egységes típus teljesen meghatározott nevet, valamint a meghatározó szerelvényt teljesen megadja.
- Nem minden minden objektumról teljesen meghatározott nevet, valamint a meghatározó szerelvényt teljesen megadja.
- Ha az objektumokat extra adaptációkkal kiegészítik, mint a *BinaryFormatter* osztályban, akkor a *BinaryFormatter* osztályt nem kell használni, hanem a kiegészítő adaptációt használva.

A formázók közötti típusos tösszág

```

// Feltetlenül importáljuk
// a System.Runtíme.Serialization.Formatters.Binary
// es a System.IO névtereket.
// a System.Runtíme.Serialization.Formatters.Binary
// static void Main(string[] args)
// Konsole.WriteLine("**** Fun with object serialization ****\n");
}

// Hozzunk létre egy JamesBondcar osztályt, és határozuk meg
// az állapotát.
// JamesBondcar jbc = new JamesBondcar();
jbc.CanFly = true;

```

Tettelezzük fel, hogy leterhözük a jámebsondcar egy példányát, módosított-túrnak néhány allapotadatot, és egy „.dat” fájlba szeremek elmenteni. Először mindenekkel szemben a sorosításban a körvonalozó módot használjuk meg. Vizsgáljuk meg a körvonalozó módot!

- `serializable()`: Az objektumgratokat byte-ok sorozatakkent egyszerűen elolvashatjuk.
 - `deserializable()`: Az elmentett byte-sorozatot objektumgrattra konvertezzük.

Azinkak szemellettessére, hogy miúlyen egyszerűen mentsétek el a jamebondcar méret a binaryFormatter típus két legfontosabb metodusáról, a serialize() és a deserialize() metódusra kell figyelniük:

Objektumok sorosítása a BinaryFormatterrel

Ha az objektumok állapotát úgy szeretnénk elmenteni, hogy barmely operációs rendszer (Windows XP, Mac OS X és a különöző Linux-distrok), alkalmazási keretrendszer (NET, J2EE, COM stb.), vagy programozási nyelv használható, ne ragaszkojdunk a teljes típusossághoz, hiszen nem tetelezhetünk fel, hogy minden lehetséges befejezés ismeli a.NET-specifikus adattípusokat. Ennek alapján a SoapFormatter-ot az XMLSerializeral legegyszerűbb választás, ha az elemeket objektumformának a lehető legszélesebb elérőhöz köthetjük.

21.2. **abra:** A BinaryFormatter használataval sorosított JamesBondCar

Cardholder	Cards	Badloans	Programme	Start Page
0000000000	00 0C 02 00 00 0F FEE FEE	00 00 00 00 00 00	Start Page	x
0000000020	00 0C 02 00 00 0F FEE FEE	00 00 00 00 00 00	Start Page	
0000000030	31 2E 00 30 2E 00 56 52 0C	95 65 27 23 95 6F 3D	Start Page	
0000000040	3D 6E 65 75 74 22 61 6C	2C 20 50 75 62 6C 69 63	Start Page	
0000000050	44 B5 65 75 74 22 61 6C	2C 20 50 75 62 6C 69 63	Start Page	
0000000060	00 00 00 1C 95 61 7A 00 00	95 65 27 23 95 6F 3D	Start Page	
0000000070	7A 55 6E 4A 61 6D 65 75	65 65 27 23 95 6F 3D	Start Page	
0000000080	00 00 00 06 35 61 6D 65 75	65 65 27 23 95 6F 3D	Start Page	
0000000090	69 73 67 65 68 74 68 65	64 61 63 6B 65 00 04 00	Start Page	
00000000A0	69 73 67 65 68 74 68 65	64 61 63 6B 65 00 04 00	Start Page	
00000000B0	01 15 69 66 65 68 42 61	65 53 72 69 61 6C 69 7A	Start Page	
00000000C0	65 72 67 65 68 42 61 63	65 53 72 69 61 6C 69 7A	Start Page	
00000000D0	01 00 00 09 03 64 69 65	00 00 00 05 00 00 01 00	Start Page	
00000000E0	64 69 66 65 68 42 61 63	65 53 72 69 61 6C 69 7A	Start Page	
00000000F0	64 69 66 65 68 42 61 63	65 53 72 69 61 6C 69 7A	Start Page	
0000000100	65 72 67 65 68 42 61 63	65 53 72 69 61 6C 69 7A	Start Page	
0000000110	73 2E 00 00 00 00 00 00	00 00 00 01 00 00 00 00	Start Page	
0000000120	00 00 00 04 00 00 00 00	00 00 00 01 00 00 00 00	Start Page	
0000000130	53 56 66 66 66 66 66 66	00 00 00 00 00 00 00 00	Start Page	
0000000140	46 58 40 60 66 66 66 66	00 00 00 00 00 00 00 00	Start Page	
0000000150	46 58 40 60 66 66 66 66	00 00 00 00 00 00 00 00	Start Page	

Visual Studio 2008-ban megy nyitott fájlt ábrázolja.

```

        static void SaveAsBinaryFormat() methodust az alábbiak szerint valósítjuk meg:
    // Mentésük az objektumot eggy CarData.dat nevű bináris fajlba.
    BinaryFormatter binariformat = new BinaryFormatter();
    using(Stream fsream = new FileStream(filtername,
    FileMode.Create, FileAccess.Write, FileShare.None))
    {
        filter.Serialize(fstream);
    }
    // Konzolra írunk a "Saved car in binary format!" üzenetet.
    Console.WriteLine("=> Saved car in binary format!");
}

```

```

jbc.canSubmerge = false;
jbc.canSummerge = false;
jbc.theradio.stationPresets = new double[]{89.3, 105.1, 97.1};
jbc.theradio.hastwometers = true;
// Most menstuk el az autot egy adott, bináris formátumú fajlba.
savEBinaryFormat(jbc, "Cardata.dat");
Console.ReadLine();
}

```

A Körvetkező vállalatokat formázó a soapFormatter típus. A soapFormatter az objektumgrafikat SOAP-üzenekkent menti el, és ez jó vállasztabanak tűnik abban az esetben, ha az objektumokat tavolíthatjuk hélyükön. Különösen környezetbarát lehet, ha az objektumokat minden típusról eltávolítjuk, mert a SOAP-üzenekben minden típusról van szó.

Objektumok sorosítása a SoapFormatterrel

A **deserializálás** hívásakor az elmentett objektumokat helyett írjuk Streamba! Ez megfelelő típusba, ítható, hogy az állapotadatok az utolsó metszi állapotot származtatott típuszt adunk meg. Minutan az objektumot visszakaszoltuk a tárkörözik.

```

        static void LoadFormatBinaryFilter(string fileName)
        {
            BinaryFormatter binFormat = new BinaryFormatter();
            FileStream fileStream = new FileStream(fileName, FileMode.Open);
            object obj = binFormat.Deserialize(fileStream);
            fileStream.Close();
            return obj;
        }
    }
}

```

Objektumok visszatállítása a BinaryFormatterrel

21.3. ábra: A SoapFormatter függvénytől származtatott JaxBSoapBondCar

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <?xml-stylesheet type="text/xsl" href="JaxB.xsl"?>
3 <SOAP-ENV:Envelope xmlns:xsd="http://www.w3.org/2001/XMLSchema"
4   xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
5   xmlns:tns="http://www.w3.org/2001/XMLSchema-namespace" xmlns:a1="http://schemas.xmlsoap.org/soap/encoding/">
6     <SOAP-ENV:Header>
7       <a1:Raido id="ref-3" xmlns:a1="http://schemas.xmlsoap.org/soap/encoding/">
8         <ns0:Raido>
9           <ns0:id>Raido</ns0:id>
10          <ns0:hasSudomotorszam>
11            <ns0:hasTwenteletszam>
12            <ns0:hasSudomotorszam>
13          <ns0:id>Raido</ns0:id>
14        </a1:Raido>
15      </SOAP-ENV:Header>
16      <tns:Raido id="ref-4" xmlns:a1="http://schemas.xmlsoap.org/soap/encoding/">
17        <ns0:id>Raido</ns0:id>
18        <ns0:szamjegyekSzamara>
19        <ns0:szamjegyekSzamara>
20      </tns:Raido>
21    </SOAP-ENV:Envelope>

```

Ahogy azt az előbb is tettük, használjuk a `serialize()` és a `deserialize()` metódusokat az objektumgráf felügyelésére és az összefüggések kezelésére. Ha a metódust a `Main()` metódusból hívjuk meg, és futtatjuk az alkalmazást, megnyithatuk az eredményként letöljtött .soap fájlt.

```

1   static void main(String[] args) {
2     SoapFormat soapFormat = new SoapFormat();
3     Filestream fs = soapFormat.createFilestream("CardataSoap");
4     Cardata car = new Cardata();
5     car.setRaido("Raido");
6     car.setSudomotor("Sudomotor");
7     car.setTwentelet("Twentelet");
8     car.setSudomotor("Sudomotor");
9     car.setRaido("Raido");
10    soapFormat.serialize(car, fs);
11    System.out.println("Cardata object saved to file " + fs.getName());
12    fs.close();
13    System.out.println("File closed.");
14    Filestream fs2 = soapFormat.createFilestream("CardataSoap");
15    Cardata car2 = soapFormat.deserialize(fs2);
16    System.out.println("Cardata object loaded from file " + fs2.getName());
17    soapFormat.serialize(car2, fs2);
18    System.out.println("File closed.");
19    fs2.close();
20  }
21}

```

Ha bárhányi objektumot importálunk a `System`, `Runtime`, `Serialize`, `Formatters`, `Formatatters`, `Soap`, `Filestream` és `SavasSoapFormat` nevű csomagból, új metódusát, amely az objektumokat helyi fájlba sorolja:

21. fejezet: Bevezetés az objektumsortsással

A XMLserializér típus alkalmazásakor felmerülő leglenyegesebb különbség, hogy ehhez meg kell határozunk minden egyes, a gyökerbe aggazott alelemet. A típusra vonatkozó információt. Az XMLserializér elso konstruktorargumen-

abban láttható XML-adatokat találjuk.

hogy ezt az új metoduszt valóban a Main() metódusban hívük meg), a 21.4. tartalmazza. Ha belenézzük az újonnán generált XML-fájlbba (felfelezve,

a system.Type típusok tömbje, amely az alelemekre vonatkozó metadatokat tuma meghatározza az XML-fájl Gyökerelmelet, míg második argumentum típusára vonatkozó információit. Az XMLserializér elso konstruktorargumen-

```

    }

    Console.WriteLine("=> Saved car in XML format!");

}

XMLFormat.Serialize(Fstream, objGraph);

}

FileMode.Create, FileAccess.Write, FileShare.None);

using(Stream Fstream = new FileStream(fileName,
    FileMode.Create, FileAccess.Write, FileShare.None));

new Type[] { typeof(Radio), typeof(Car) });

XMLSerializer XMLFormat = new XMLSerializer(typeof(JavaScriptCar),
    // new fajlba.

// Mentésük az objektumot XML-formátumban egy Cardata.xml

static void SaveAsXMLFormat(object objGraph, string fileName)

ret műr importálunk:
```

A SOAP- és a bináris formázók mellett a system.XML.dll szerevénnyel egy hár-
madik formázót is biztosít: a system.XML.Serialization.XMLSerializér egy adott objektum nyilvános állapotát SOAP-izennelbe aggazott XML-adatok he-
lyett írja XML-kent trófja. Ennek a típusnak a használata nemképp ki-
lönözik a SoapFormatterrel vagy a binaryFormatrel. Nézzük meg az-
alábbi forrásuktat, amely felfelezzi, hogy a system.XML.Serializer műg-
yűtött típusa XML-kent tárolja. Ennek a típusnak a használata nemképp ki-
adott objektum nyilvános állapotát SOAP-izennelbe aggazott XML-adatok he-
lyett írja XML-kent trófja. Ennek a típusnak a használata nemképp ki-
lönözik a SoapFormatterrel vagy a binaryFormatrel. Nézzük meg az-

Objektumok soroztása az XMLserializere

Itt találhatók azokat az XML-élémeket, amelyek az aktuális JavaScriptCar al-
lapötörő vonatkozó értékeket, valamint a grafban lévő objektumokat a #ref-
tokneken keresztül jelenílik (lásd 21.3. ábra).

Az alapfélémezes szerint egy [Serializable] típus minden nyilvános adatát allelémként és nem XML-attribútumként formázunk meg. Ha szeretnék megőrizni, hogy az XMLserializér hogyan generálja az eredményt, számta tövábbi, a system.xml.Serializablenevű osztályt, amelyek befolysósítják az XML-adatok folyamba törtenő származó attribútumokat. Kent letejővő XML-dokumentumot, a [Serializable] típusosakat leíróként számos XML-tulajdonságot, amelyek befolysósítják az XML-adatok folyamba törtenő tulajdonságait. A 21.1. táblázat néhány (de nem minden) olyan attribútummal rendelkezik, amelyeket a szerzők által vezérelt származó attribútumokkal szemben használunk fel.

Egy DTD- (dokumentum-típusdefinicció) fájlban határozunk meg.

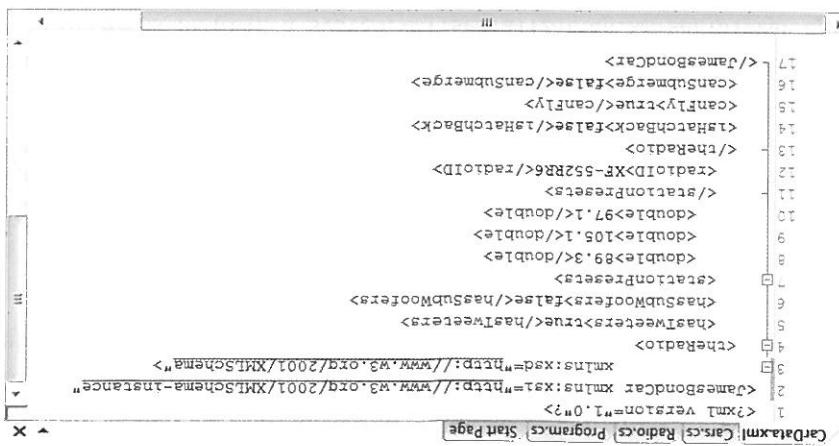
„erőnyes” XML-dokumentumoknak inkább olyan – megegyezésen alapuló – formát használjunk, amelyeket jellemezhet a dokumentumtípusa. Az „erőnyes” XML-dokumentumoknak semmi köze az XML-elémek szintaktikai jogelöléshez (pl. minden nyitóelémhez tartoznia kell egy záróelemnek is). Az „erőnyes” XML-dokumentumoknak inkább olyan – megegyezésen alapuló – formát használjunk, amelyeket a dokumentumtípusa, hogy az XML-dokumentum adattá megfelelően leírhatunk. Túdjuk, hogy az „erőnyes” XML-dokumentumoknak inkább olyan – megegyezésen alapuló – formát használjunk, amelyeket a szerzők által vezérelt származó attribútumokkal szemben használunk fel.

Generált XML-adatok szabalyozása

Megjegyzés Az XMLserializér vonatkozo hatterismeretekból tudjuk, hogy gyakran található sorosított típus támogasson egyik feltételle, hogy minden, az objektumgráfrban található sorosított konstruktor megfelelően működjen. Ha ez nem történik meg, futásidőben juk hozzá az egyszeri konstruktor megfelelőt (tehát mindeneképpen ad-

invalídoperatőreket is elkerülve).

21.4. ábra: A XMLserializér használatai sorosított JamesBondCar



21. Fejezet: Bevezetés az objektumsortosítás világába

Először egy származtatott XML-objektumot hozunk létre, amely a JamesBondCar osztályt implementálja, majd a JamesBondCar osztályt, valamint a CanFly esetén a CanSubmerge merge eredményét adja vissza.

```

    {
        public bool CanSubmerge;
        [XmlAttribute]
        public bool CanFly;
        [XmlAttribute]
        public Class JamesBondCar : Car
    }
    XmlRoot(Namespace = "http://www.interotech.com")]
    [Serializable,
    szerint:
    attributumkent kódolja, módosítva a JamesBondCar C#-definícióját az alábbiak
    JamesBondCar osztályt, valamint a CanFly és a CanSubmerge eredményét XML-
    Ha olyan egyszerű XML-nevteret szeretnénk meghatározni, amely megfelelően
    <JamesBondCar>
        <CanSubmerge>false</CanSubmerge>
        <CanFly>true</CanFly>
        ...
        xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <JamesBondCar xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
        <xmVersion="1.0" encoding="utf-8"?>
    
```

Először egy származtatott XML-objektumot hozunk létre, amely a JamesBondCar osztályt implementálja, majd a JamesBondCar osztályt, valamint a CanFly esetén a CanSubmerge merge eredményét adja vissza.

Attributum	Jelentés
XmlAttribute	A tagot XML-attributumként soroljuk.
XmlElement	A mezőt vagy tulajdonosátot XML-élémekként soroljuk.
XmlAttributeAttribute	Meghatározza, hogy egy felismerő típus egy tagához milyen nevű elem tartozzon a sorosítás során nyílzához.
XmlRootAttribute	Ez az attributum határozza meg, hogy a gyökérrel együtt milyen szerkezetű legyen (nevter és elmenye).
XmlTextAttribute	A mező vágy tulajdonosát sorosításnak XML-szöveg-
XmlAttributeType	Az XML-típus neve és nevtere.

21.1. táblázat: A System.Xml.Serialization névterrel rendelhető attribútumai

Attributum	Jelentés
XmlAttribute	A tagot XML-attributumként soroljuk.
XmlElement	A mezőt vagy tulajdonosátot XML-élémekként soroljuk.
XmlAttributeAttribute	Meghatározza, hogy egy felismerő típus egy tagához milyen nevű elem tartozzon a sorosítás során nyílzához.
XmlRootAttribute	Ez az attributum határozza meg, hogy a gyökérrel együtt milyen szerkezetű legyen (nevter és elmenye).
XmlTextAttribute	A mező vágy tulajdonosát sorosításnak XML-szöveg-
XmlAttributeType	Az XML-típus neve és nevtere.

kívánt folyamhoz.

A system objekt valójában az objektum teljes fejlődéséhez. Ez alapján, ha átadunk egy [serializable] attribútummal megjelölt objektumot, amely többi [serializable] objektumkattartalmaz, akkor az objektumok teljes halmaza egylelén metodushívásval rögzül. A systemコレクションは、sys-temコレクションが各オブジェクトを保持するメソッド呼び出しを記録する。コレクション内の各オブジェクトは、自身のコレクションを保持する場合、そのオブジェクトもまた記録される。このようにして、システムはオブジェクト構造全体の状態を記録する。コレクションの構造は、各オブジェクトの属性によって定められる。

```
public interface IFomatte{  
    ...  
    object Deserialize(Sytem.IO.Stream serializationStream);  
    void Serialize(Sytem.IO.Stream serializationStream);  
}
```

A továbbiakban azt vizsgáljuk meg, hogyán mennyire hatékony az objektumok egységei halmozat. Az információk interfész serializáció metódusa nem teszi lehe- tővé, hogy inputként tetszőleges számú objektumot adjon meg (csak egy szövegben), hogy inputként számít objektumot adjunk meg (csak egy szövegben). Ennek megfelelően a Deserializáció () és Serializáció () függvényeket szintén csak egy önálló system-object (az XMLシリализエク) között szállítjuk. Ezáltal minden objektumnak saját szerepe van a szabványos interfészben.

Objektumgyűjtemények sorosítása

Lérmeszetesen számos attribútumot használhatunk annak vezetésre, hogy az XML-szeríalizér hogyan generálja a leterjelvű XML-dokumentumokat. To-
ban a system.Xml.Serializatión nevétre vonatkozó információkat.

```
<?xml version="1.0"?>
<jamesbondcar xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <canFly="true" canSubmerge="false" xmlns="http://www.jntechtraining.com">
    <name>James Bond Car</name>
  </canFly>
</jamesbondcar>
```

Mintehogy az `xmlSerializer` a `TextReader` tipusú `hasznaltuk`, míg `Kell` határozunk a `tipusunk` a `hasznaltuk` típusú `hasznaltuk`, míg `egyéretlmeübő` logikai uton `haladunk`, például:

Igy most már az alábbiak szerint akárholán sem ismertetni lehet.

```

[Serializable]
public class JamesBondCar : Car
{
    public RootNamespace = "http://www.interteach.com"]
    [XmlElement]
    public string Serializable, 
    [XmlElement]
    public bool CanFly = skyworthy;
    [XmlElement]
    public bool CanSubmerge = seaworthy;
    [XmlElement]
    public string PublicJamesBondCar(bool skyworthy, bool seaworthy)
    {
        // AZ XMLSERIALIZER megkötetel egy alapértelmezett konstruktort!
        // AZ XMLSERIALIZER megkötetel egy alapértelmezett konstruktort!
        public JamesBondCar() {}
    }
}

```

Itelezzük fel, hogy néhány állapotaikat megállítószaboz kérlelhetőek, amelyeket a következőkben az alábbiakban részletezzük:

Típus	Jelentés	Iseriálizáló	ObjectIDGenerator	Típus generálja az objektumgráf tagjainak azonosítóját.
es.visszazállításat.	Ennek az interfésznek a [Seriálizáló] típusra törté-	nő implementációval vezérelhetők annak sorosítását	Ez a típus generálja az objektumgráf tagjainak azonosi-	ObjectIDGenerator
ez visszazállítását.	Ennek az interfésznek a [Seriálizáló] típusra törté-	nő implementációval vezérelhetők annak sorosítását	Ez a típus generálja az objektumgráf tagjainak azonosi-	ObjectIDGenerator
ez visszazállítását.	ez visszazállítását.	ez visszazállítását.	ez visszazállítását.	ez visszazállítását.

A sorosítási folyamat teszteléséhez

Főterekről A simpelesztető projektek a forraszokdókonyvtárral csak a Bevezetés XI. oldalán.

```

        static void SaveListofCarsBinary()
        {
            // Mentisk e[ az ArrayList objektumot (mycars) binaris
            // objektkument.
            // List<JamesBondCar> mycars = new List<JamesBondCar>();
            // binaryFormater biniformat = new BinaryFormatter();
            // usiing(Stream fsream = new FileStream("AllMyCars.dat",
            // FileMode.Create, FileAccess.Write, FileShare.None));
            // biniformat.Serialize(fsream, mycars);
            // console.WriteLine("=> Saved List of cars in binary!");
        }
    }
}

```

A BinarýFormatter a viszszállítás során az objektum egyazonos masolatának leterheléséhez ugyanazt az információt használja, amelyet az alatta levő folyamboi nyír ki. A SoapFormatter által alkalmazott folyamat meghetősen hasonló.

- a grafikban található objektum teljesen meghatározott nevet (pl. MyApp).
 - az objektumról megfogható szerelemei szerelvénnyel nevezet (pl. MyApp.exe),
 - az objektumtípusa osztály nevén (pl. MyApp),
 - az objektumtípusa minden példányát, amely minden, az objektumról tagjai által fenntartott állapotadatot tartalmaz.

Mielőtt megvizsgálunk a sorosítási folyamat részleteit, először el kell tennünk a sorozatok különböző módjait, erdemességeit és foglalkozni. Amikor a binaryFormat sorozatot meghívunk, az objektumról, az alábbi információkat továbbítja az adott felülyamba:

AZ objektumsortosítás hárterreszletei

21.2. **tablázat:** A System.Runitime.Serializáció neutrális típusai

Megjegyzés A .NET 2.0 megjelenésére óta a sorosítási folyamat teszszabásának preferált módszera a sorosítási attribútumok használata (ennek leírását lásd alább). A megelő rendszerekben azonban fontos, hogy ismerjük az *serializable* interfészet is.

A [serializable] attributummal megjelölött objektumoknak lehetősége van implementálni az Iserializable interfejszt. Ezzel „részvét vallatunk” a sorozatbiztonsági követelményeknek.

Sorostások teszteléséhez az ISErházzal használataval

Az azon kívül, hogy meghatározza, hogy a típus tamogatja-e az iserializábel interfejszt, a formázók azért is felleszk, hogy felléjelek, vajon a kérdezes típusok tamogatja-e azokat a tagokat, amelyeket az [Onserializing], az [Onserializing] és [Onserializing] meglévő attribútumoknak a szerepe, nezzük meg közélebbrol az iserializable funkciókat.

- Ellenorizik, hogy az objektumok még vaninak-e jelölve a [Serializable] attribútummal. Ha az objektum nincs megjelölve, serializációexception jelenik meg.
 - atttribútummal, hogy az objektumok még vaninak-e jelölve a [Serializable] attribútummal. Ha az objektum nincs megjelölve, serializációexception jelenik meg.
 - Ha az objektum megek van jelenve a [Serializable] attribútummal, ellenörzik es meghatározza, hogy az objektum implementálja-e a ISerializable interfész. Ezután az objektum a Getobjetcdata() metódust hívja az objektumra.
 - Ilyenkor az objektum a Getobjetcdata() metódust hívja az objektumra. Azt követően a Getobjetcdata() metódus az objektum implementálja-e a ISerializable interfész, ha nem, akkor nem fog eljárni a fenti lépésekkel.

Azokon túl, hogy a szakségek adatokat a tolyamba helyezik, illetve onnan kivéznek az objektumrólban található tagokat:

Megyégegyezés Az Xmisennizer, annak erdekeben, hogy a lehető legnagyobb mértékben meg-
tartásra az objektum állapotaibanak horodzhatóságát, nem menti el a típusok teljesen meghatáro-
zott nevet, sem a meghatározó szerelvénnyet. Ez a típus csak a megléntető nyilvános

Az `ISerializable` interfejsz megjelenítésen egy szérető, hiszen csak egy metodust hataroz meg, ez pedig a `Getobjetcdata()`:

```

    void Getobjetcdata(SerializationInfo info,
    StreamContext context);
}

// Ha a sorosítási folyamat finomhangolásra törekszünk,
// implementáljuk az ISerializable interfeszet.
public interface ISerializable
{
    public void Getobjetcdata(SerializationInfo info,
    StreamContext context);
}

```

Az `ISerializable` interfesz meghatároznak, hogy a fürtátmotor meg tudja határozni // az objektum allapotát.

```

    // annak érdekében, hogy a fürtátmotor meg tudja határozni
    // az egységi konstruktor el kezeti Látni ezzel az aláraszt
    // Az ISerializable interfész implementáció típusoknak ilyen, az alábbi szigna-
    turával rendelkezik, speciális konstruktor is meg kell határozni:
```

Az `ISerializable` interfész implementáció típusoknak ilyen, az alábbi szigna-

```

    {
        ...
        protected SomeClass (SerializationInfo si,
        StreamContext ctx) {...}
    }
}

class SomeClass : ISerializable
{
    [Serializable]
    // az objektum állapotát.
    // annak érdekében, hogy a fürtátmotor meg tudja határozni
    // az egységi konstruktor el kezeti Látni ezzel az aláraszt
    // Az ISerializable interfész implementáció típusoknak ilyen, az alábbi szigna-
    // turával rendelkezik, speciális konstruktor is meg kell határozni:
    ...
    public void AddValue(string name, int value);
    public void AddValue(string name, uint value);
    public void AddValue(string name, short value);
    public int MemberCount { get; set; }
    public string FullName { get; set; }
    public AssemblyName { get; set; }
    public string AssemblyName { get; set; }
    public int MemberCount { get; set; }
    public void AddValue(string name, long value);
}
```

mat. Nezzük meg egy kódreszletet:

Object számaiban hívja. Az implementáció felülírta a bemeneti `SerializationInfo` paramétert név/értek párok sorozatával, amelyek (jellemezőn) az elmenten- dő objektum adatmezőire képződnek. A `ISerializable` interfészben minden a számos variá- ciót hataroz meg a `FinalizeAddValue()` metódusra, valamint a tulajdon- saágok egy kis gyűjtmeményre, ezek lehetővé teszik, hogy az adott típus lekér- dézze és módosítsa a típus nevét, a megújításon szerevénnyel lesz a tagok szá- dzában. Ez a sorosítási folyamat a bemeneti `SerializationInfo` interfészben,

```

    public string dataItemOne = "First data block";
    public string dataItemTwo = "More data",  

    }  

    class StringData : ISerializable  

    {  

        [Serializable]  

        terfezett (ne felejtse ki el importálni a szintet. RunTime. Serialization névteret):  

        lyoknak az erdekelben a körvonalazásban implementáltak az ISerializable-tin-  

        tivel kell sorosítani, és csupa kisbetűvel kell viszazzállítani. Ezekenké a szabá-  

        határoz meg. Továbbá tegyük fel, hogy a sztringobjektumokat csupa nagybe-  

        lusztásihoz telezzük fel, hogy olyan új konzolalkalmazást hozunk létre  

        A ISerializable interfészthoz használó sorosítási folyamat tetsreszabásának íl-
    }
}

```

```

    }  

    All  

    CrossAppDomain,  

    Clone,  

    Other,  

    Remoting,  

    Persistence,  

    File,  

    CrossMachine,  

    CrossProcess,  

    }  

    public enum StreamImageContexts
    {

```

(a részletek a .NET Framework 3.5 SDK dokumentációjaiban találhatók):

ellenére mégadájuk a streamImageContexts típus lehetőségeit erőteljesen szolgáltatás, alegyha kell közvetlenül foglalkozunk ezzel a típusossal. Ennek Ha csak nem implementáltunk nehezen alacsony szintű, egyszerű remotinguvisel. A részről típus erőteljesen aktuális folyam alapossza látható.

amely egy, a streamImageContexts részről típusbeli származó erőteljesen formációit. Ennek a típusnak a leginformatívabb tagja a state tulajdonoság, amely a bármely formással vagy rendeltetési helyével kapcsolatban tartalmaz információt. Ezek a speciális konstruktor második paramétere egy streamImageContext típus, amely a bármely konstruktor második paramétere az aktuális folyam alapossza látható.

Info típus egy példánya (ahogy azt követhetünk).

Ennek konstruktorának a bárhatoságától függetlenül hozzáérhetők a streamImageContexts, mert a formázó a bárhatoságától függetlenül hozzáérhetők a taghoz. Ezek a speciális konstruktorok általában védektí (vagy privat) jelölést kapnak, hogy megakadályozzák, hogy az alkalmi objektum meghosszabbításának minden modon hozzáérhető objektumokat. A konstruktor elso paramétere a serializációhoz.

Ha megnezzük az így Leterjővő * , soap fájt, Latfúk, hogy a sztringmezőköt nagybétesen taroljuk (lásd a 21.5.ábrát).

```

    // Töltésük fel a serializációtól információkat
    // a formázott adatokkal.
    // A Gétobjektdata() metódusbeli töltésük fel a serializációtól információkat
    // az adatoknak nem kell úgyanazt a nevet adniuk, mint a típus belső tagjaihoz
    // amikor a Gétobjektdata() metódustól a serializációtól információkat
    // a testreszabás teszteléséhez tölthetünk fel, hogy elmeneteltük a myStrin
    // gban függelékenyitni szeretnénk.
    // Ekkor hasznos, ha a típus es a perszisztalesi formátumot m
    // mindenkor a testreszabás teszteléséhez tölthetünk fel, hogy elmeneteltük a myStrin
    // gban függelékenyitni szeretnénk.
    // Ez akkor hasznos, ha a típus es a perszisztalesi formátumot m
    // mindenkor a serializációhoz szükséges karaktereket hozzáadunk, mint a típus belső tagjaihoz
    // a testreszabás teszteléséhez tölthetünk fel, hogy elmeneteltük a myStrin
    // gban függelékenyitni szeretnénk.
    static void Main(string[] args)
    {
        Console.WriteLine("***** Fun with Custom Serialization *****");
        // Emleksznak, hogy ez a típus implementálja az ISerializable
        // interfészét.
        // Emleksznak, hogy ez a típus implementálja az ISerializable
        // interfészét.
        string myData = new StringData();
        // Mennek el SOAP-formátumban egy helyi járiba.
        // Mennek el SOAP-formátumban egy helyi járiba.
        SoapFormatter formatter = new SoapFormatter();
        using(Stream stream = new FileStream("MyData.soa", FileMode.Create, FileAccess.Write, FileShare.None))
        {
            formatter.Serialize(stream, myData);
        }
        Console.ReadLine();
    }
}

```

```

public StringData() {
    protected StreamingData(serializerInfo si, streamingContext ctx)
    { }
    // Tolušek fel újra a tagvaltözököt a Foleyamboit.
    datatitemone = si.getString("FirstItem").Tolower();
    datatitemtwo = si.getString("SecondItem").Tolower();
}

```

```

    public string datatitemwo = "More data";
    public string datatitemone = "First data block";
}

[Serializable]
class Moredata
{
    public string strin gdatatitemtwo = "More data";
    public string strin gdatatitemone = "First data block";
}

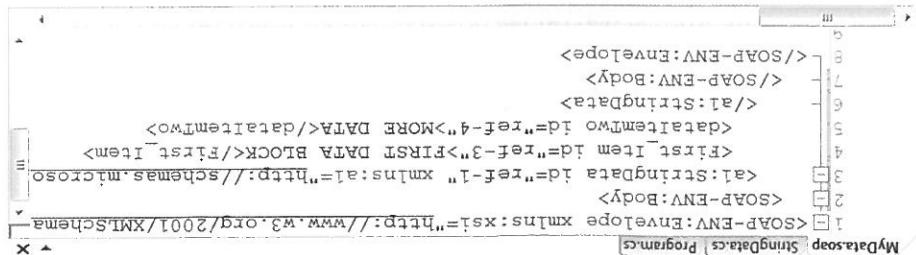
```

Megjegyzés Ezek a sorosítási attribútumok a system Runtime serialization névteren belül vanakk definiálva.

Bár az ismerőszakba interfezsz implementációja a sorosítás törlyama tetszés szabányanak egy lehetséges modja, a .NET 2.0 megjelenése óta a sorosítás folyamat tesztelészabánya inkább az új sorosítás-központhoz attribútumok segítségevel: [onserializing], [onserialized], [ondeserializing] vagy [ondeserialized] mezőkkel. Ehezett, miközben a formázó elvezeti serializációt a kapcsolatos nem kell manuálisan közreműködőnek a bemeneti serializációhoz para-menterekkel. A többi részben a formázó elvezeti a típusos adatokat.

Sorosítások teszteléséhez a [Attributumokkal](#)

21.5. ábra: A sorosítás teszteléséhez használt általánosizálható környezetmodellek



22. fejezet: Bevezetés az objektumsortsítás világába

Ebben a fejezetben az objektumsortosító szolgáltatások temátikát tárgyalunk. A .NET platform az objektumgráfikat használja azon kapcsolódó objektumok teljes halmozának fogyelmeztetére, amelyeket egy folyamba kell elmenteni. Mindaddig, amíg minden objektumgráfika [Serializable] attribútummal jelölünk, megvalósíthatunk az adatok sorosítással a formátumát bináris, SOAP vagy XML). A nem egyszeri sorosítási folyamatoknak kettelében a fejezetben az objektumsortosító szolgáltatások temátikát tárgyalunk.

Osszefoglalás

Az előző példa segítségével megtismerhetük az objektumsortosítás legrébbi szolgáltatásait, köztük a folyamatos teszteléshez szükséges modjait. A sortosítás a viszazzállítási folyamat részén nagy ményisége a datot is egyszerűen elmenhethetők, ez sokkal kevesebb munkát igényel, mintha a rendszert. Ez a szerelemnek ismerkedni a .NET platformnak ezzel az aspektusával, olvasunk el a BinaryFormatter, a SoapFormatter és az XmlSerializer szolgáltatót információkat a .NET Framework 3.5 SDK dokumentációjából.

Forráskód A CustomSerialization projektet a forráskódoknnyvtárban a 21. fejezet alkönyvtára tartalmazza. A forráskódoknnyvtárról lásd a Bevezetés Xiv. oldalát.

Ha az új típusit sortosítani szeretnénk, ismét azt tapasztalnánk, hogy az adott sortosítása nagybetűs, még a viszazzállítása kisbetűs.

```

private void OnSerialize(OnDeserializationContext context)
{
    datatitemone = datatitemtwo.Toupper();
    datatitemtwo = datatitemone.Toupper();
}

// Hívás a viszazzállítási folyamat befejezését követően.
private void OnDeserialization(OnDeserializationContext context)
{
    datatitemone = datatitemtwo.Toupper();
    datatitemtwo = datatitemone.Toupper();
}

// Hívás a sortosítási folyamat alatt.
private void OnSerialize(OnSerializeContext context)
{
    datatitemone = datatitemtwo.Toupper();
    datatitemtwo = datatitemone.Toupper();
}

```

Eloszor megneztük az `Iserializable` interfejsz implementálásának (és egy speciális privat konstruktor támogatásának) a modját arra, hogy nagyobb befolysással lehessünk arra, hogy a megalapozott adatokat hogyan mentse el a formázók. Ezt követően megismertünk néhány olyan .NET-attribútumot, amelyekkel leegyszerűsíthető a sorosítás teszteléséhez. A `StreamIngContext` paraméterrel rendelkező tagoknál az `[Onserializing]`, az `[Ondeserializing]`, az `[Ondeserialized]` attribútumot alkalmazza, a formázók pedig ennek megfelelően fogják elket megírni.

Ha ismeriük a Microsoft köröbbi, COM-alapú adatleírasi modelljét (Active Data Objects vagy ADO), tudjuk, hogy az ADO.NET-nek megfelelősen kezűlhető tártalemmel rendelkezik a kapcsolat - és utáztással körülölelik fogalmát), néhány rész közé van az ADO-hoz. Bar van kapcsolat a rendszer között (pl. minden Data Objecttől vagy ADO), még a szabványnak is van a közelében.

Az ADO.NET magas szintű meghatározása

Olyan egyszerű adathozzáírásokat fogunk tudni kezdeni (autolóton végre-
dal. díj), amely különöző, egyéni adatbázisra, az Autolóton végre-
hajtható adatbázis-műveleteket valósít meg. A könnyvtárat a következőké-
zében tüvesszük, és a tövábbiakban többször használjuk is fogunk. A
fejezet végén megvizsgáljuk, hogyan lehet a Microsoft SQL Serverrel azon-
száműzetésben tüvesszük, valamint bevezetjük az adatbázis-tranzakciók fogalmát.

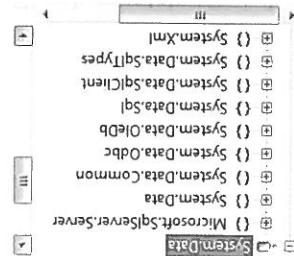
A különöző adatszolgáltatók közül funkciót megvizsgálva megnezzük a
data provider factory minta működését. A system.Data.Common névter hozzá-
nálatával (és a hozzártozó app.config fájl segrészével) elkezdtethetünk egy
olyan egyszerű kódot, amely dinamikusan képes választani és használni a
mögöttes adatszolgáltatókat közül annak, hogy az alkalmazás kódja tűrje ki-
lene fordítani vagy telepíteni.

Kommunikációra optimalizáltak.
A .NET platform számos adatszolgáltatót támogat, ezek mindenegyiket adott adatba-
zis-kézeli rendszerrel (Microsoft SQL Server, Oracle, MySQL stb.) történő
alátalansos szerepet, majd az adatszolgáltatók temakörét targyaljuk. A .NET
nemrémek a gyakran használt ADO.NET. Először körülönállazunk az ADO.NET
látható lokális vagy távoli relációs adatbázisokat kezeli tudjuk. Ezeket a
platform számos nemrémeket definiál, amelyek segítségevel a gépen ta-

Az elő kapcsolat ADO.NET, 1. rész:

HUSZONKETTEDEIK EJESZET

22.1. ábra: A System.Data.dll a legalapvetőbb ADO.NET szerelemei



Programozási szemszögből nézve az ADO.NET lenyegére egy alapvető szerepet játszik a **DataSet** osztály, amelynek definíciója a következő:

Talán a legalapvetőbb különbség a klasszikus ADO és az ADO.NET között az, hogy az ADO.NET nem más, mint egy felügyelt kodkonyvtár, így pontosan felügyelt környvatárak szabályai szerint működik. Az ADO.NET-t alkotó tisztségeket alkalmazzák (osztályokat, interfészeket, felsorolásokat, struktúrarendszereket) CLR memoriájához kötött protokollokat használják, úgy amazokat a típuspusok a CLR meghívásaként elérhetők. Az ADO.NET-t alkotó tisztségeknek a rendszerekkel való kommunikációja általában a.NET-nyelvű alkalmazásokat, struktúrakat és metodusrétegeket, es bármihez.NET-nyelvű alkalmazásokat.

További leányeges különbsége a klasszikus ADO és az ADO.NET között az, hogy az ADO.NET származtatott támogatja az XML-adatok megjelenítését. Tájédonképpen az adattárolóból kinyerhet adatot (az alapértelmezés szerint) XML-formátumban sorosítja a rendszert. Mivel az XML-adatokat gyakran szabványos HTTP protokolloon keresztül továbbítjuk a rettegék között, így az ADO.NET nem utkozik semmilyen tűzfal általi okozott korlátozásba.

ADO.NET-típus (mint pl. a Recordset) már nem letérzik. Továbbá számos olyan ADO-típus van, amelyhez nem találunk a klasszikus ADO-ban megfele-

A kapcsolat nekkti modell (lásd a 23. fejezetben részleteSEN) lehetősegeit meglép. Ha lekérünk egy dataset objektumot a megfelelő adattípusztól objektumot a dataset tárójában, amelyek a különböző adatok környezetében a másolatának felélnökek közötti arány, hogy dataset objektumoknak azt a halmozatot kezeliük (ezeket munikálnak az adattárolóval).

Az ADO.NET parancsobjektumokon és adatlással objektumokon keresztül kommunikálni, parancsobjektumokon és szinten expliciten bontha is a kapcsolatot. Ha ílyen meglévő adattárolóhoz, és szintén expliciten csatlakozik az alatta található explicit használjuk, akkor a forrásbólunk expliciten csatlakozik az elérhető használához, amely a kapcsolat-alapú es kapcsolat nekkti módon. Ha a kapcsolat-alapú módnál: elérhető kapcsolat-alapú es kapcsolat nekkti módon lehet használni: elérhető kapcsolat-alapú es kapcsolat nekkti módon. Az ADO.NET konnyvtárákatt köt köncsionalisan egyedi módon lehet használni:

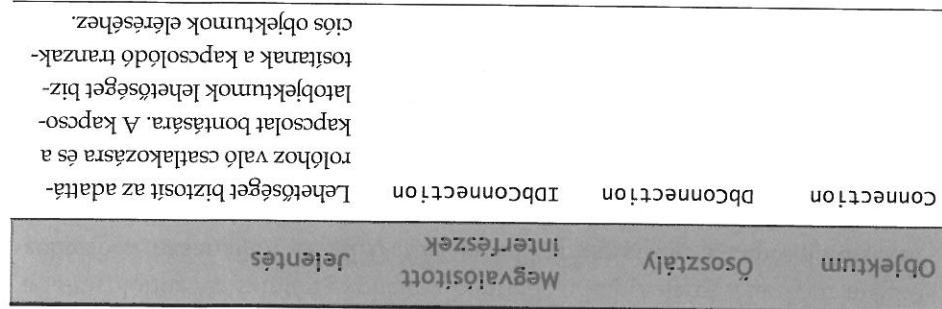
AZ ADO.NET KET ARCA

Megjegyzés A .NET 3.5 megléne sével az ADO.NET számos további szerelvénnyel és névre elérhető, amelyek az ADO.NET/LINQ integrációt segítik. A 24. fejezetben részleteSEN meglép. Vizsgáljuk az ADO.NET LINQ-központú jellemezőit.

Az említett System.Data.dll szerelvénnyen kívül Jelenleg más ADO.NET-centrikus szerelvénnyek (mint pl. a System.Data.OracleClient.dll), amelyekre manuálisan kell hivatkozunk az aktuális projektünkben az Add Reference parbeszedőpanel segítségével.

```
nameSpace MyApp
{
    class Program
    {
        static void Main(string[] args)
        {
            using System.Data.SqlClient;
            // Vezessünk be néhány ADO.NET-nevetet!
        }
    }
}
```

A legtöbb Visual Studio 2008 projektablóin automatikusan hivatkozik erre a külcsfonsosságú adattárolási konnyvárra. Ennek ellenére módosítani kell a forrásokbólunkat, hogy azokat a névtereket is importáljuk, amelyeket használni szereznünk. Például:



Az ADO.NET-adatszolgáltatók működése

túmával, a kapcsolat automatikusan megnyilik, majd lezáródik. Ez a megkötélezetűkkel a károkat elkerülve rövid időn belül visszatérítik a rendszert a normál állapotba. A rendszer a hibás csatlakozásról észrevétlenül informálja az ügyfélnek.

Megjegyzés az ADO.NETben, ha „kapcsolatobjektumról” beszélünk, akkor valóban egy konkréten szereplő objektumról számaztatott típusra hivatkoznunk, és nincs olyan osztály, amelynek a része az DBConnectionból származtatott lesz. Ugyanez érvényes a „parameterekkel” is, az adattípusról objektumra” esik igy tövább. A nevűdási konvencióinknak megfelelően egy adott adatszolgáltató objektumról szemben „Connection” lenne. Ugyanez érvényes a „parameterekkel” is, az adattípusról objektumról szemben „ConnectionString” lenne.

Noha az alapvető objektumok konkréttel néve elérhet az egyes adatszolgáltatónak, mégis minden adatszolgáltatókat használni. ne, hogy ha egy adatszolgáltatót kezelését ismerjük, már nem lesz nehéz más azonos interfészkeket válosít meg. Egyetlen megfelelően biztosak lehetünk benne, hogy a környezetben a kapcsolatobjektumok esetén), ez pedig teljességekkel származik (dbconnection a mySqlConnection viszonylatában), minden objektum ugyanabboldal az osztályból esetén (pl. az SqlConnection, az OracleConnection, az OleDbConnection) és a további esetben (pl. az SqlCommand, az OracleCommand, az OleDbCommand).

22.1. táblázat: Egy ADO.NET adatszolgáltató alapvető objektumai

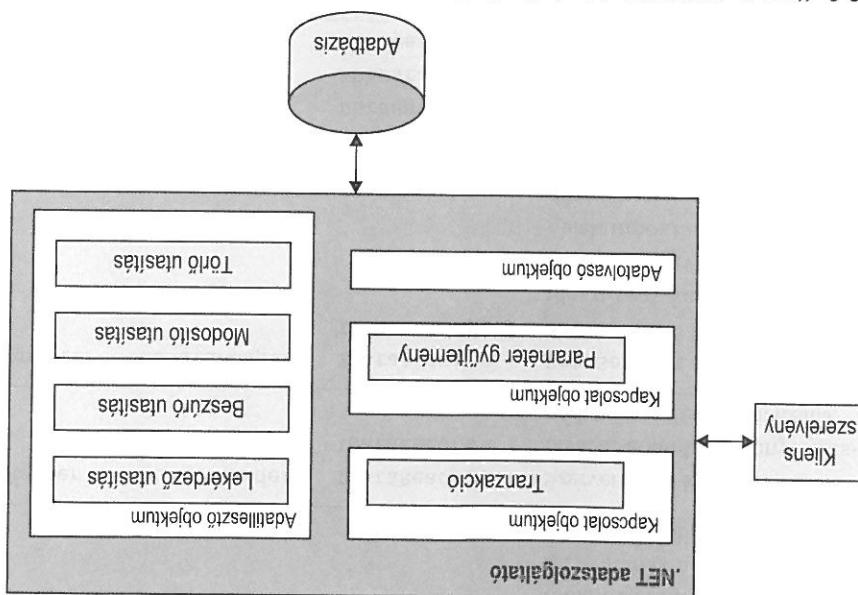
Objektum	Összefüggés	Megvalósított	Jelentés
Command	DbCommand	DbCommand	Egy SQL-lekérdezést vagy egy
DataReader	DbDataReader	IDataReader	Tárol az adatok egyszerű, táskaformában, ahol a felhasználó lekérheti.
DataAdapter	DbDataAdapter	IDbDataAdapter	Az adattárolói között. Az adat-
DataReader	DbDataReader	IDataReader	Szervertoldali kurzor használata.
Transaction	DbTransaction	IDbTransaction	Beszterkezés az adatszolgáltatóhoz.
Parameter	DbParameter	IDbParameter	Egy névvel ellátott paraméter.
Interfész			Lekezdésekben.
Attribútum			Paraméter.
Transzakció			Beágyaz az adatbázis-tran-

A Microsoft .NET-distribúciója számos adatszolgáltatival kerül a piacra, amelyek között találunk szolgáltatást az Oracle felé, az SQL Server felé és az OLE DB/ODBC összekapcsolhatóság felé is. A 22.2. ábrázlat a Microsoft ADO.NET-adatszolgáltatásokhoz tartozó nevűről es a beöglalo szerelevenye-ket sorolja fel.

A Microsoft által szállított ADO.NET-adatszolgáltatók

Termezetesen egy adatszolgáltatót az ábrán látható objektumokon kívül más típusokat is tartalmaz. Am ezek az objektumok egységesek a lapot képeznék az összes adatszolgáltatoban.

22.2. ábra: Az ADO.NET-adatszolgáltatók adott adatháztartásihoz közelebbi részt biztosítják



A 22.2. ábra bemutatja az ADO.NET-adatszolgáltatók hatterét. A kepen a „Kleinens szereleveny” baromlányen .NET-alkalmazás lehet: konzolalkalmazás, Windows Forms-alkalmazás, ASP.NET-weboldal, WCF-szolgáltatás, .NET-

kodkönyvtár es igy tövább.

Megjegyzés Egy olyan eset van, amikor a rendszer a Microsoft SQL Serverről nem tud kommunikálni.

Az OLE DB adatszolgáltatóján keresztül, amelyet a system. Data. OLEDB névvel tölts ki az alkalmak, bárminyien adathoz hozzáérhetünk, amely a klasszikus COM-alapú OLE DB protokoll támogatja adattában található. Ennek a szolgáltatónak a használatával bármiely OLE DB-vel kompatibilis adatbázissal kommunikálhatunk, ha a kapcsolatstríming Provider szeménetet beállítjuk.

Az OLE DB szolgáltató számos COM-objektummal működik együtta a szinírfalak mögött, és ez befolyásolhatja az alkalmazás teljesítményét. Általa elmondható, hogy csak akkor eredményes OLE DB adatszolgáltatót használunk, ha olyan adatbázis-közlelővel szeretnénk dolgozni, amelyhez van specifikus.NET-adatszolgáltató. Minthogy napjainkban minden valamire valós relációs adatbázis-közlelőneknél rendelkezünk saját letöltésre ADONET-

adatszolgáltatóval, így a system. Data. OLEDB olyan korábbi névter, amelyet még a .NET 3.5 világában (és ez egyre inkább így van a data provider factory modell megjelenésével, amely a .NET 2.0 megjelenése után kihagyott).

Megjegyzés: Nincsen olyan adatszolgáltatás, amelyet kifejezetten a jét motorhoz lehetségesen (esetlegesen a Microsoft Accesshez). Ha egy Access adatbázisfájlt szereznél kezelni, akkor ezt az OLE DB vagy az ODBC adatszolgáltatón keresztül tudjuk megteenni.

22.2. tablázat: A Microsoft ADU.NE!-adatszolgáltató

Adatcsövlegéltető	Névtér	Szerelvény
OLE DB	System.Data.OleDb	System.Data.dll
Microsoft SQL	System.Data.SqlClient	System.Data.dll
Microsoft Mobile	System.Data.SqlServerCe	System.Data.SqlServerCe.dll
ODBC	System.Data.Odbc	System.Data.dll
Oracle	System.Data.OracleClient	System.Data.dll

Az ADO.NET-adatszolgáltatók működése

A Microsofttól kinált adatszolgáltatónak kívül számos harmadik fejlesztő származó adatszolgáltatás létezik számos nyílt forrásrendszer és pláci adatbázis-készlethez. Noha valósztinűleg közvetlenül az adatbáziskezelő szállító cégtől fogjuk beszerezni az ADO.NET-adatszolgáltatót, erdemes megemlíteni a következő oldalt is (az URL változatával jogosult a tulajdonos felügyeletre): www.sqlsummit.com/DataProv.htm.

Ez a webhely egy azok között a webhelyek közül, amelyek minden ismert ADO.NET-adatszolgáltatot dokumentálnak, valamint hivatkozásokat biztosítanak tövábbi információkhoz és letöltésekhez. Számos ADO.NET-adatszolgáltató megtalálható itt, többek között az SQLite, a DB2, a MySQL, a PostgreSQL és a Sybase.

Mivel nagyon sok ADO.NET-adatszolgáltató létezik, a feljelzében található minőségekkel a Microsoft SQL Server adatszolgáltatot (System.Data.SqlClient) használja.

Harmadik félidő származó Ado.NET-adatszolgáltató beszerzése

A Microsoft SQL Server adatbázisokat kiszolgáló szolgáltatót (szolgáltatót) definiáljuk, amely számos adatátrolóhoz szolgálhat hozzáférésre. A Microsoft SQL Server adattárakhoz, és csak az SQL Server adattárakhoz (7.0 vagy későbbi verziókhoz). A számítási rendszertől függetlenül, a szolgáltatót a Microsoft SQL Server adatbázisokhoz, mint az OLE DB szolgáltatót. A legfontosabb különbsége az, hogy az SQL Server szolgáltató környezeti szolgáltatót a Microsoft SQL Server adatbázisokhoz számítási rendszereken belül elönnyel tessek. A Microsoft SQL Server adatbázisokhoz számítási rendszereken belül elérhetővé teszik a SQL Server Mobile DBMS kiadását (alatta Oracle-adatbázisokhoz, biztosítják az ODBC-kapcsolatokkal történő komunikációt, és elérhetővé teszik a SQL Server Mobile DBMS kiadását nyújt az Oracle-adatainakhoz, mint pl. a Pocket PC). A számítási rendszereken belül minden területen használható, amelyben csak olyan esetben hasz-

22.3. táblázat: További ADO.NET-tel foglalkozó névterek

Névtér	Jelentés
Microsoft.SqlServer.	Ez a névter olyan típusokat szolgáltat, amelyek segítenek a CLR és az SQL Server 2005 integrációs szolgáltatásait.
System.Data.	Ez a névter definíálja az ADO.NET-nek azokat az alapvető típusait, amelyeket minden adatszolgáltató használ, beleértve a vétő típusait, amelyeket minden adatszolgáltató használ, beleértve a kódos absztrakt összefüggéseket.
System.Data.Common	Ez a névter azokat a típusokat tartalmazza, amelyeket minden ADO.NET-adatszolgáltató használ, beleértve a közös absztrakt összefüggéseket.
System.Data.SqlClient	Ez a névter lehetővé teszi, hogy feltérképezzük az Ez a részt a Microsoft SQL Serverrel szembeni kommunikációt, amelyek a kapcsolat nemkülönösen komplex (Data-set, DataTable stb.).
System.Data.SQLite	Ez a névter azokat a típusokat tartalmazza, amelyeket minden ADO.NET-adatszolgáltató használ, beleértve a ver Példányokat.
System.Data.SqlTypes	Ez a névter a Microsoft SQL Serverrel szembeni kommunikációt, amelyek a részt a Microsoft SQL Serverrel szembeni kommunikációt, amelyeket minden ADO.NET-adatszolgáltató használ, beleértve a használható CLR-adattípusokat, az SqL típusai kimondottan az SqL Serverrel működnek optimálisan.

A .NET-névreken kívül, amelyek az egyes adatszolgáltatok típusait definiálják a .NET alapsztruktúráit számos további ADO.NET-tel foglalkozó nevteret kiállít, amelyeket minden adatszolgáltató használ, beleértve a .NET-alapozott környezetet a 22.3. táblázatban láthatunk.

További ADO.NET-névrekek

Ez az adatszolgáltató a Microsoft SQL Server 7.0-val és a későbbi verziónak készült biztosítja. Ha az ADO.NET-et másmilyen adatbázis-készlethez kötött környezetben használjuk, akkor sem lesz ezzel problémánk a kovetkező tudnivalók ismereteiben.

22.4. tablázat: A System-Datá neotér alapjai

Tipus	Jelentés	Constaint	A adott DATAColumn objektum megszorítását kepviseli.	DATAColumn	A DATAColumn objektum belül egy sort kepvisel.	DATASET	A adatok memoriában gyorsítókárát kepviseli, amelyben bár- menyit egymással összefüggő DATATABLe objektum található.	DATATABLe	A memoriában található adatok tablázatos formáját kepviseli. Lehetővé teszi, hogy a DATATABLe objektumot egy tömörkuri- zoként kezeljük (egyirányú, részvétel nélküli).	DATAVIEW	A datatable objektum egyetlen szabott hozzáírás készítéséhez. rendezés, szűrés, keresés, módszerek és navigálás céljára.	IDATADAPTER	A adattípuszt objektum alapvetőkéedését definíálja.	IDATAPARAMETER	A paraméterobjektum alapvetőkéedését definíálja.	IDBCOMMAND	A parancsobjektum alapvetőkéedését definíálja.	IDBDAADAPTER	Az adatolvasó objektum alapvetőkéedését definíálja.	IDATAREADER	Az adatolvasó objektum alapvetőkéedését definíálja.	IDBTRANSACTION	A tranzakcióobjektum alapvetőkéedését definíálja.
-------	----------	-----------	--	------------	--	---------	---	-----------	---	----------	---	-------------	---	----------------	--	------------	--	--------------	---	-------------	---	----------------	---

A system. Data never tipusai