Microsoft

Ľuboslav Lacko

Nová vlna technológií pre Visual Studio 2008 a .Net Framework 3.5

Microsoft Visual Studio 2008

Nová vlna technológií pre Visual Studio 2008 a .Net Framework 3.5

Ľuboslav Lacko

Obsah

Nová vlna technológií pre Visual Studio 2008 a .NET Framework 3.5

Po troch rokoch prichádza z dielne spoločnosti Microsoft nová verzia vývojového prostredia s označením Visual Studio 2008. Prináša mnohé nové črty a vylepšenia, reagujúce na nové trendy ako napríklad Web 2.0 alebo rastúci význam orientácie na služby, komfortu používateľského rozhrania, do hry vstupuje aj modelovanie obchodných procesov a správa digitálnych identít a podobne.

S niektorými novými črtami sme už mali možnosť sa stretnúť vo forme rôznych doplnkov a servisných balíčkov na konci pontifikátu predchádzajúcej verzie Visual Studio 2005, iné sú úplne nové. Novinky sa týkajú hlavne vývoja webových, intranetových a „smart" aplikácií pre Office 2007. Nová verzia už tradične ponúka efektívnejšie nástroje pre vizuálny návrh používateľského prostredia a vylepšené možnosti ladenia, takže si oprávnene zaslúži prívlastok „RAD" (Rapid Application Development). Keď vývojári odhalia a budú naplno využívať nové vlastnosti prostredia Visual Studio 2008 podstatne vzrastie efektívnosť ich práce. Vývojové prostredie je úzko spojené s technologickou platformou .NET Framework 3.5. V dobe písania tejto publikácie bol k dispozícii opravný balíček SP1 pre Visual Studio 2008 aj pre platformu .NET Framework 3.5. Na úvod zosumarizujeme novinky, ktorým sa budeme venovať v tejto publikácii:

- ADO.NET Entity Framework,
- ASP.NET Silverlight controls
- ADO.NET Data Services,
- ASP.NET Dynamic Data
- Podpora pre SQL Server 2008
- LINQ
- NET Framework Client Release („Arrowhead"),
- 2007 Ribbons

Zároveň predstavíme aj možnosti WPF (Windows Presentation Foundation) pre návrh používateľského rozhrania aplikácie. Mnohí vývojári sledovali najvýznamnejšie novinky už počas ich vývoja keď boli známe pod kódovými označeniami. Pre zorientovanie sa v názvoch prinášame tabuľku s prehľadom kódových názvov.

Kódové označenie	Názov
„Astoria"	ADO.NET Data Services
„Oryx"	ASP.NET Dynamic Data
„Cicero"	ASP.NET 3.5 SP1 enhancements
„Arrowhead"	.NET Framework Client release

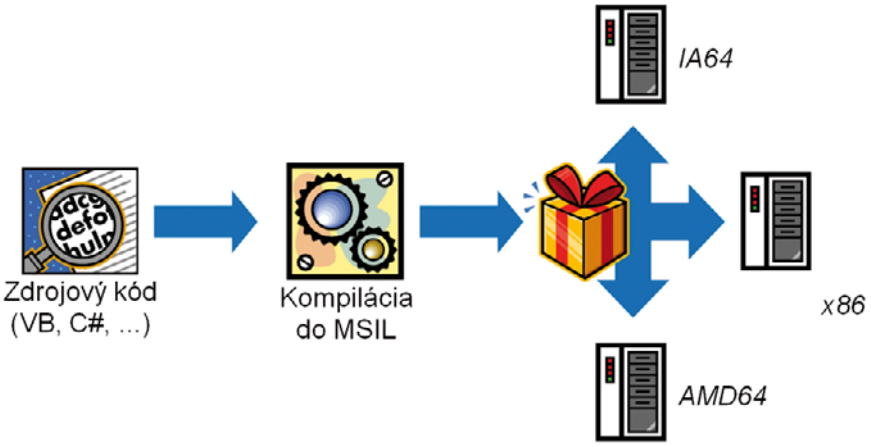
Resumé šesťročnej histórie platformy .NET Framework

.NET Framework umožňuje rýchlu tvorbu aplikácií pripojených k dátam, s ktorými sú koncoví používatelia veľmi spokojní. Poskytuje totiž stavebné bloky (prefabrikovaný softvér) pre riešenie častých programátorských úloh. Pripojené aplikácie využívajúce .NET Framework efektívne modelujú procesy podnikateľskej sféry a uľahčujú integráciu systémov v heterogénnych prostrediach. Visual Studio a .NET Framework spoločne znižujú potrebu každodenného písania programového kódu, skracujú čas vývoja a dovoľujú vývojárom sústrediť sa na riešenie vlastného problému. Vývojári nemusia znovu a znovu programovať základné elementy aplikácií, počnúc prihlasovacím dialógom, rozhraním pre administráciu aplikácie a jej používateľov zobrazením obsahu databázovej tabuľky, platbu kreditnou kartou a podobne.

Verzia 3.5 prostredia .NET Framework – vlajkovej technologickej platformy spoločnosti Microsoft nám prezradí, že .NET Framework má už za sebou pomerne bohatú históriu. Číslica 3 pred desatinnou čiarkou prezrádza, že sa ide minimálne o tretiu verziu, no číslica za desatinnou čiarkou dáva tušiť, že exitovali aj medzi – verzie. Pozrime sa teda na históriu, na postupný vývoj verzií platformy .NET Framework.

.NET Framework 1.0 (2002)

Táto verzia je spojená s vývojovým prostredím Visual Studio .NET 2002, ktoré bolo následníkom populárneho „klasického" vývojového prostredia Visual Studio 6.0. Prvý krát sa objavujú „riadené" (managed) jazyky C# 1.0 a Visual Basic .NET, dokonca aj Managed C++. Predtým bol kód napísaný v programovacom jazyku Visual Basic prekladaný v dvoch krokoch. Najskôr sa generoval medzikód, na ktorý sa aplikoval kompilátor jazyka C++. Na scénu prichádza aj platforma ASP .NET 1.0 pre vývoj webových aplikácií.



Preklad zdrojového kódu do MSIL (Microsoft Intermediate Language)

.NET Framework 1.1 (2003)

Verzia je spojená s vývojovým prostredím Visual Studio .NET 2003. Boli opravené niektoré chyby a „bezpečnostné diery" z predchádzajúcej verzie, doplnené API rozhrania.

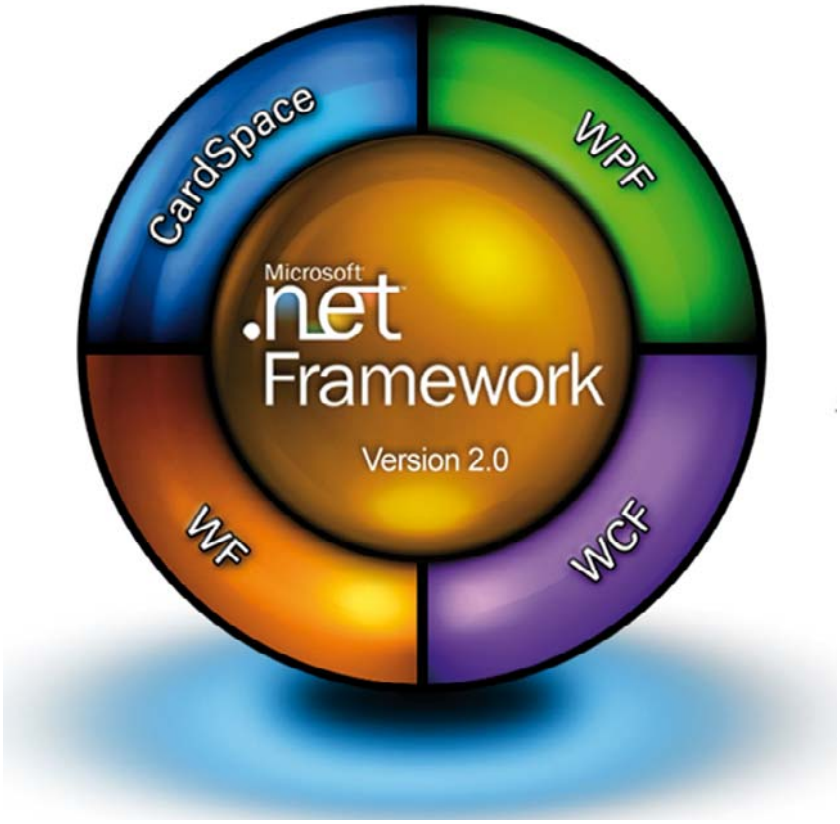
.NET Framework 2.0 (2005)

Verzia je spojená s vývojovým prostredím Visual Studio 2005. Toto technologické rozhranie v porovnaní s prvou verziou prinieslo v 2372 nových objektových tried (doterajší počet 4482 sa rozrástol 6854), pričom pribudla

veľká množina tried pre SQL Server 2005. Rozrástla sa aj paleta dostupných ovládacích prvkov či už pre ASP.NET alebo WinForms aplikácie. Významnou novinkou je aj možnosť vývoja aplikácií s využitím technológie ASP.NET 2.0, podpora sériového portu, objektové priestory, prístupové práva k súborom a práca s certifikátmi. S novou verziou frameworku prišli aj nové verzie programovacích jazykov C# 2.0, Visual Basic 2005, nové triedy v BCL (Based Class Library), podpora pre generiká v CLR... S platformou .NET Framework 2.0 je spojená aj platforma ASP.NET 2.0 pre vývoj webových a intranetových aplikácií.

.NET Framework 3.0 (2006)

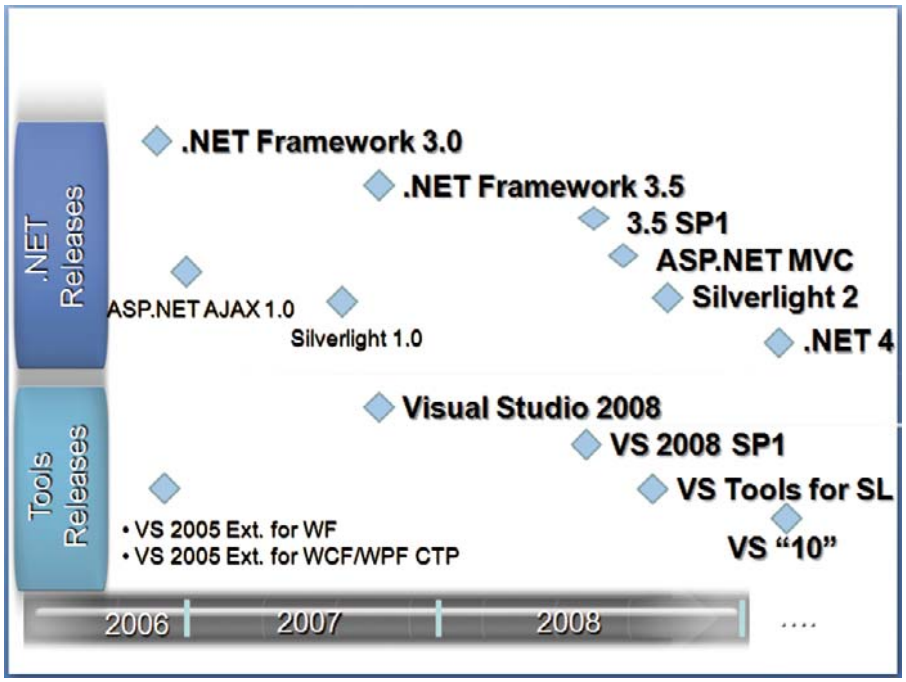
Verzia je spojená s vývojovým prostredím Visual Studio 2008. Nosnou ideou pre túto „novú" verziu technologickej platformy .NET Framework je jednoduchý a efektívny vývoj aplikácií prepojenie biznisu s ľuďmi, informáciami a procesmi. V ďalšom popise vysvetlíme prečo sme dali slovo „novú" do úvodzoviek.
Zatiaľ čo predchádzajúce inkrementálne verzie platformy .NET Framework (1.1 a 2.0) obsahovali vždy nové, úplne prepracované verzie CLR (Common Language Runtime) a BCL (Base Class Library), verzia .NET Framework 3.0 využíva CLR aj BCL zo staršej verzie 2.0. Nové technológie, teda:
– Windows Presentation Foundation (WPF, predtým Avalon),
– Windows Communication Foundation (WCF, predtým Indigo),
– Windows Workflow Foundation (WF),
– Windows CardSpace (predtým InfoCard),
sú implementované nad jadrom zo staršej verzie 2.0.



.NET Framework má implementované nové technológie na jadre CLR a BCL z verzie 2.0

Aby sme boli presnejší, pôvodný návrh operačného systému Windows Vista (v tej dobe známy pod kódovým označením Longhorn) obsahoval integrovaný grafický systém Avalon a systém pre komunikáciu Indigo. Počas ďalšieho vývoja operačného systému došlo k oddeleniu týchto blokov od operačného systému a ich začleneniu do separátneho balíka WinFx. Poslednou zmenou bolo premenovanie WinFX na .NET Framework 3.0. Jadrom novej verzie 3.0 zostáva staršia verzia .NET Framework 2.0. Jadro obsahuje dva hlavné bloky CLR (Common Language Runtime) a BCL (Base Class Library). Rozhodnutie o zachovaní jadra staršej verzie bolo prijaté hlavne z dôvodu zachovania kompatibility starších aplikácií, ktoré využívajú jadro verzie 2.0.

Teoretické hardvérové minimum pre nasadenie technologickej platformy .NET Framework 3.0 je procesor Pentium taktovaný na 400 MHz a 96 MB RAM. Pre praktické nasadenie sa odporúča 1 GHz Pentium a 256 MB RAM. Ak chceme využívať prezentačnú vrstvu WPF, musíme počítať aj s výkonnou grafickou kartou. .NET Framework 3.0 je vo verzii operačného systému Windows Vista už priamo integrovaný. Môžeme ho doinštalovať aj pre operačné systémy Windows XP SP2 a Windows Server 2003 SP1 alebo R2.
Inými slovami – aplikácie využívajúce WPF, WCF, WF a CardSpace nepotrebujú ku svojmu behu nutne operačný systém Windows Vista, no vyžadujú nainštalovaný .NET Framework vo verzii 3.0.



Roadmap platformy .NET Framework

.NET Framework 3.5

Po stručnej exkurzii dejinami platformy .NET Framework sa zamerajme na najnovšiu verziu (samozrejme v dobe písania tejto publikácie, vývoj pôjde pochopiteľne dopredu). Ako vyplýva z číselného označenia, verzia 3.5 nadväzuje na .NET Framework 3.0. Obsahuje rozšírenie programovacích jazykov C# 3.0 a Visual Basic 9, integrovanú podporu technológie ASP .NET AJAX a databázového jazyka LINQ (Language Integrated Query). Vývojové prostredie Visual Studio 2008 je úzko spojené s technologickou platformou .NET Framework 3.5. Platforma .NET Framework 3.5 je koncipovaná tak aby umožňovala jednoduchý a efektívny vývoj aplikácií a modulov pre podporu obchodných procesov a pre integráciu heterogénnych systémov. Vývojári sa nemusia zameriavať na technologické detaily ale môžu sa plne sústrediť na riešenie aplikačnej logiky a biznis problémov. Rozšírenie sa týka knižnice tried a technologických vrstiev Windows Presentation Foundation (WPF):, Windows Communication Foundation (WCF):, Workflow Foundation (WF) a Windows CardSpace, ktoré sú nadstavbou jadra platformy. Jadro zostáva rovnaké ako bolo pro verzii 2.0, čo prináša výhody spätnej kompatibility.

Windows Presentation Foundation (WPF) je grafický subsystém pre tvorbu prezentačnej vrstvy s bohatou podporou grafiky a multimédií.

Windows Communication Foundation (WCF) je základným technologickým pilierom pre tvorbu aplikácií orientovaných na služby s využitím filozofie a princípov SOA.

Workflow Foundation (WF) slúži na efektívne modelovanie procesných tokov.

CardSpace je subsystém pre prácu s digitálnymi identitami a na ich správu.

Verzia 3.5 prináša vylepšenia:

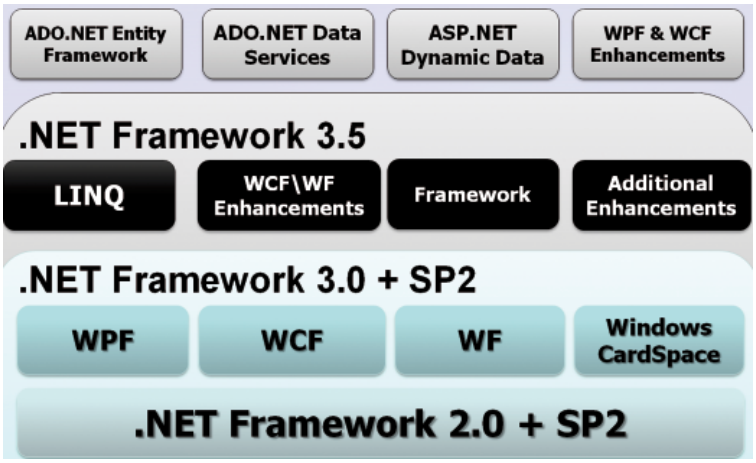
Client Application Services umožňujú využitie autentifikačných a autorizačných metód obsiahnutých v ASP. NET aj pre aplikácie typu „rich client", teda pre klasické desktopové Windows aplikácie, vytvorené na báze Windows Forms, alebo Windows Presentation Foundation.

Active Directory Managed API – vylepšená podpora riadeného API, takže už nebude potrebné písať niektoré funkcie ako COM objekty.

Application Extensibility API označovaný niekedy aj ako „add in DLL" umožňuje pridávanie prídavných modulov do aplikácií, pričom medzi aplikáciou a modulmi sa vytvárajú pomerne voľné väzby.

.NET Framework 3.5 SP1

Rýchly vývoj technológií, ale hlavne nástup finálnej verzie servera SQL Server 2008 (predtým označovaného kódovým názvom Katmai) bol dôvod, že spoločnosť Microsoft sprístupnila balíčky Service Pack (SP1) pre svoje produkty Visual Studio 2008 ako aj .NET Framework 3.5. Sú zamerané nielen na opravy chýb a zlepšenia výkonu, ale zároveň pridávajú aj nové funkcie. .NET Framework 3.5 SP1 okrem iného sľubuje vyšší výkon o 20 až 45 % v aplikáciách založených na WPF (Windows Presentation Foundation) bez nutnosti zmeny kódu. Vylepšenia vo WCF, poskytujú vývojárom zjednodušenie k dátam a službám a umožňujú viac kontroly nad spôsobom, akým aplikácie k týmto dátam a službám pristupujú. Balíček obsahuje aj novinky v oblasti dátovej platformy, ako napríklad ADO.NET Entity Framework vrátane nástroja ADO.NET Entity Framework Designer pre uľahčenie návrhu , ADO.NET Data Services a rozsiahlu podporu funkcií servera SQL Server 2008, doplnkové nástroje a komponenty pre jazyky C#, Visual Basic a Visual C++. K novinkám patrí aj optimalizácia sieťového Socket API, Peer-to-peer API. Vývojárom spoločnosti Microsoft sa podarilo významne zvýšiť výkon v niekoľkých oblastiach.



.Net Framework 3.5 SP1

Resumé histórie

. NET Framework má už osem ročnú históriu. Pripomeňme si pre zaujímavosť slová ktoré odzneli na konferencii Microsoft TechEd 2001 pri uvádzaní vtedy novej verzie .NET Framework 1.0.

„Platforma .NET umožní realizáciu vízie rovnakých informácií kdekoľvek, kedykoľvek a na akomkoľvek zariadení. Budúca generácia softvéru sa bude významne podieľať na formovaní prepojeného sveta informácií, zariadení a ľudí, ktorý bude fungovať na spoločnom základe, pri súčasnom zachovaní individuality každého používateľa. Základným stavebným kameňom novo vyvíjaných aplikácií bude .NET Framework".

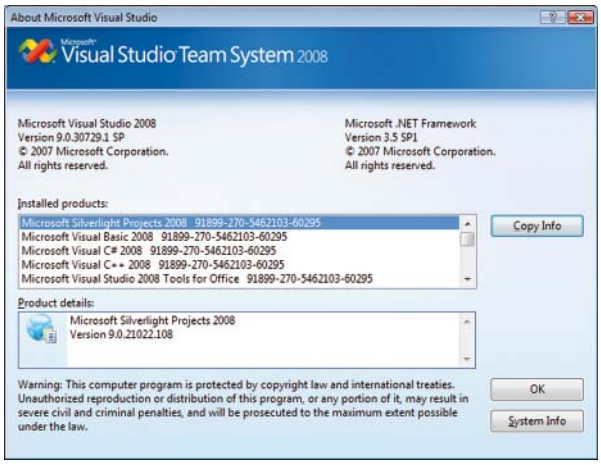
Pri posudzovaní osemročnej histórie a možností verzie 3.5, si môžete sami urobiť obraz nakoľko sa prezentované vízie podarilo splniť.

Visual Studio 2008

Produktový rad Microsoft Visual Studio 2008 ponúka nástroje pre rôzne pokročilých a skúsených vývojárov od úplných začiatočníkov v programovaní až po profesionálov a tímy hľadajúce prostredie podporujúce moderné metodiky a princípy riadenia životného cyklu vývoja softvérových aplikácií. Visual Studio je vhodné pre vývojárov, dizajnérov, architektov, testerov, projektových manažérov a ďalších účastníkov životného cyklu vývoja softvérových aplikácií.

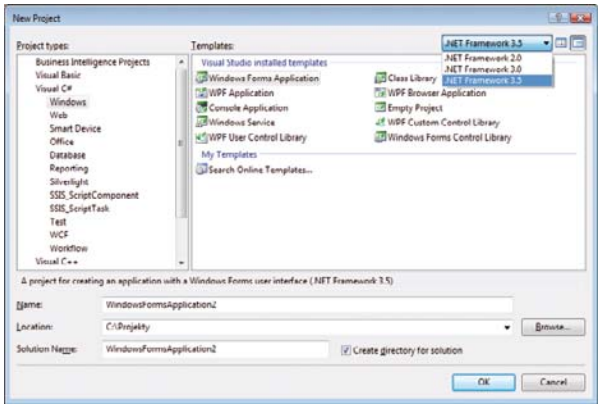
Visual Studio 2008 poskytuje pokročilé vývojové nástroje, ladiace funkcie, funkcie pre prácu s databázami a vynaliezavé novinky pre rýchlu tvorbu špičkových aplikácií rôznych typov. Prináša početné vylepšenia: vizuálnych sprievodcov pre rýchlejší vývoj širokého spektra aplikácií s využitím platformy .NET Framework 3.5, podstatné zdokonalenie webových vývojových nástrojov a vylepšenia programovacích jazykov, ktoré zrýchľujú vývoj pre dáta všetkých typov. Visual Studio 2008 ponúka vývojárom všetky nástroje a podporu frameworku, ktoré sú potrebné na vytváranie moderných webových aplikácií využívajúcich technológiu AJAX. Vďaka vysoko funkčným frameworkom na strane klienta i servera sú vývojári schopní ľahko budovať klientske webové aplikácie integrované s ľubovoľným dátovým úložiskom, bežiace v ktoromkoľvek v modernom prehľadávači a s plným prístupom k aplikačným službám ASP.NET a k platforme Microsoft. Nové vývojové prostredie pomáha pri rýchlom vývoji moderného softvéru tým, že vývojárom poskytuje vylepšené jazykové a dátové prvky, napríklad Microsoft Language Integrated Query (US) (LINQ), ktoré jednotlivým programátorom uľahčujú tvorbu riešení analyzujúcich a spracovávajúcich informácie. Visual Studio 2008 ponúka vývojárom nové nástroje urýchľujúce tvorbu aplikácií pripojených k dátam (connected applications) na najnovších platformách, ako sú web, Windows Vista, Office 2007, SQL Server 2008 a Windows Server 2008. Čo sa týka webu, ASP.NET, AJAX, Silverlight 2.0 a ďalšie nové technológie dovolia rýchly vývoj novej generácie veľmi výkonných, interaktívnych a personalizovaných webových aplikácií, spĺňajúcich náročné technologické ale hlavne biznis požiadavky na Web 2.0.

Nové vývojové prostredie prináša rozšírené a zdokonalené nástroje pre lepšiu spoluprácu vo vývojárskych tímoch, napr. nástroje, ktoré pomáhajú zapojiť do vývojového procesu databázových odborníkov a návrhárov grafiky.



Dialóg „About" s popisom aktuálnych verzií kompilátorov a technologických doplnkov

S príchodom novej verzie vývojového prostredia už tradične prichádzajú nielen očakávania ale aj určité obavy vyplývajúce hlavne z toho, ako si tvorcovia novej verzie poradili so spätnou kompatibilitou. Mnohí si ešte pamätáme časy, kedy dve po sebe idúce verzie, napríklad Visual Studio 2002 a 2003 nemohli byť súčasne nainštalované na jednom počítači. To prinášalo problémy, pretože vývojári musia nielen vyvíjať nové projekty pre ktoré zákonite chcú použiť nové efektívnejšie a technologicky vyspelejšie vývojárske nástroje, ale musia udržiavať aj staršie projekty počas celého ich životného cyklu podpory. Visual Studio 2008 prinieslo v oblasti spätnej kompatibility veľký pokrok. Nielen že dokáže koexistovať s verziami Visual Studio 2005 a Visual Studio .NET 2003, ale vývojári v ňom môžu pracovať s projektmi pre verzie prostredia .NET Framework 3.5, 3.0 aj 2.0. Požadovaná verzia platformy sa vyberá už pri zakladaní nového projektu. Visual Studio 2008 po výbere platformy ponúkne len tie typy šablón projektov, ktoré sú pre danú platformu podporované, takže nemôžeme napríklad vyvíjať Silverlight 2.0 aplikáciu nad nižšou verziou platformy .NET Framework ako 3.5 a podobne.



Verziu technologickej platformy .NET Framework, pre ktorú chceme vyvíjať projekt vyberáme už pri jeho vytváraní

Jednoduchá inštalácia, intuitívne používateľské rozhranie a množstvo prepracovaných sprievodcov pre jednoduché aj zložitejšie úkony, ako sú napríklad umiestnenie komponentov a nastavenie ich základných parametrov, vytváranie vlastných šablón, tried a objektov, vytváranie pripojení na databázy a webové služby umožní nielen efektívnu prácu ale aj ľahkú migráciu vývojárov nielen z predchádzajúcich verzií ale aj z vývojárskych nástrojov iných firiem. Za všetky črty uľahčujúce migráciu a zvládnutie syntaxe nových programovacích jazykov spomenieme interaktívnehou pomocníka IntelliSense, ktorý vo verzii 2008 funguje nielen nielen pre .NET jazyky ale aj pre skriptovací jazyk JavaScript. VSTO už nie je len doplnok, ale je integrálnou súčasťou prostredia Visual Studio od verzie Professional vyššie. Pri koordinácii vývojárskych projektov pomôže integrácia s aplikáciami Excel a hlavne Microsoft Project pre riadenie projektov.

Nové vývojové prostredie je určené pre vývoj aplikácií pre Windows XP, Windows Vista, Windows Server 2005, Office 2007 a pre Office SharePoint Server 2007. Pôvodne malo prísť prostredie Visual Studio 2008 na trh spolu s databázovým serverom SQL Server 2008 a serverovým operačným systémom Windows Server 2008 a malo tak tvoriť kompaktnú rodinku s číselnými označeniami verzií „2008". Tieto dva produkty však prišli na trh neskôr, takže pri príchode na trh vývojové prostredie Visual Studio 2008 spolupracovalo so serverom SQL Server 2005. Po príchode verzie SQL Server 2008 je potrebné doninštalovať opravný balíček SP1.

.NET Framework ako „Open Source"

Zdrojový kód platformy .NET Frameworku je uvoľnený pod licenciou Microsoft Reference Licence, takže je možné ladiť aj vnútorné procesy v zapuzdrených knižniciach. Zdrojový kód má byť dostupný na prevzatie, ale aj pomocou webovej služby, ktorú bude využívať Visual Studio 2008. Po nastavení adresy servera vo Visual Studio 2008 bude VS automaticky demandna vyžiadanie preberať potrebné zdrojové kódy počas debugovania. Výhoda tejto služby je, že vždy sa prevezmú aktuálne kódy pre používanú verziu frameworku (vo VS 2008 je možné nastaviť, voči ktorej verzii .NET sa má vyvíjať – 2.0, 3.0 alebo 3.5).

Prehľad produktov Visual Studio 2008 a ich určenie:

Produktový rad	Určené pre	Charakteristika
Visual Studio Express Editions — **Visual Studio Express**	začiatočníkov, študentov, na vyskúšanie	Jednotlivo oddelené produktové rady zamerané na výučbu jazykov C++, C# a Visual Basic prípadne web technológiu.
 **Visual Studio Standard**	príležitostných programátorov a neprofesionálov	Jednotné, komplexné prostredie pre tvorbu webových i desktopových aplikácií.
 **Visual Studio Professional**	profesionálov pracujúcich jednotlivo – samostatne	Profesionálny nástroj na tvorbu mobilných, webových desktopových a viacúrovňových aplikácií s integrovanou podporou vývoja pre MS Office.
 **Visual Studio Team System**	všetkých vývojárov, architektov, testerov, projektových manažérov a ostatných, najmä tých, ktorí pracujú v skupine	Komplexný balík toho najlepšieho, čo možno ponúknuť ozajstným vývojárom a celým vývojovým tímom, vrátane tímovej infraštruktúry.

Verzie Standard a Professional
Pri predstavovaní verzií začneme edíciami Visual Studio 2008 **Standard** a Visual Studio 2008 Professional. Verzia **Professional** je určená pre podporu projektov vyvíjaných pre web (vrátane ASP.NET a AJAX), Windows Vista, Windows Server 2008, systém Microsoft Office 2007, SQL Server 2008 a zariadenia Windows Mobile. Umožňuje programovanie a ladenie viacvrstvových serverových aplikácií kompletne vo vnútri vývojového prostredia Visual Studia vrátane "unit testing", plnú podporu všetkých typov nasadení (Click Once, tvorba MSI inštalačných balíčkov), dostupnosť najvyššej plnohodnotnej verzie SQL Servera v podobe produktu SQL Server Developer Edition a vytváranie tlačových zostáv.

Visual Studio Team System 2008
Najrozsiahlejšia je podskupina verzií Visual Studio Team System 2008 určená pre podporu efektívnej tímovej spolupráce, nielen vývojárov ale aj grafických dizajnérov a databázových špecialistov. Do tejto rodinky patrí:

Team Foundation Server koordinuje spoluprácu, zaisťujúci čo najúčinnejšiu komunikáciu vo vnútro celého IT tímu a jednoduchú správu a sledovanie vývoja a stavu projektov.

Team Suite – je komplexný integrovaný balík nástrojov pre vývoj všetkých typov programov, obsahujúci všetky dostupné technológie na podporu životného cyklu softvérových aplikácií od projektového návrhu, architektúry, cez vývoj až po testovanie vrátane potrebnej infraštruktúry.

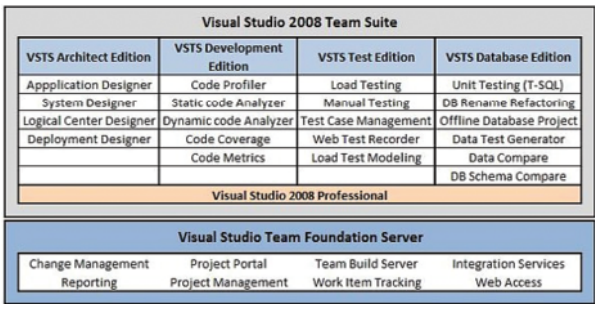
Architecture Edition – verzia určená pre softvérových architektov poskytuje nástroje pre vizuálny dizajn architektúry riešení orientovaných na služby.

Database Edition – verzia pre databázových profesionálov poskytuje nástroje na správu zmien a zdrojových kódov, testovanie a nasadzovanie databázových riešení založených na serveri SQL Server. Umožní aktívne a efektívne zapojenie databázových špecialistov do vývojového tímu.

Development Edition – ponúka pokročilé nástroje a možnosti ktoré posúvajú kvalitu vývojárskej práce v tímoch na vyššiu úroveň bez nutnosti prebytočných úkonov vyžadovaných od vývojárov.

Test Edition – je balík nástrojov integrovaných do prostredia Visual Studio, pre zaistenie kvalitného testovania aplikácií. Výsledky testovania je možné zdieľať s celým tímom aj s koncovými používateľmi.

Test Load Agent – pokročilý testovací nástroj, ktorý umožňuje simulovať výrazne vyššie množstvo užívateľov ako VS Test Edícia, to všetko s väčšou presnosťou, výkonom a stabilitou webových aplikácií a serverov.



Prehľad verzií Visual Studio Team System 2008

Express Edition
Samostatné postavenie má voľne šíriteľná verzia **Visual Studio 2008 Express Edition** určená pre nekomerčné projekty, teda pre študentov a „hobby" vývojárov. Rad produktov Visual Studio 2008 Express Editions obsahuje vývojové nástroje základnej úrovne, ktoré sú zadarmo, sú jednoduché a ľahko zvládnuteľné. Sú určené začiatočníkom a hodia sa na vytváranie jednoduchých aplikácií pre Windows a jednoduchých interaktívnych webových stránok. Nástroje nie sú svojim charakterom, i svojimi vlastnosťami určené na vývoj zložitejších aplikácií alebo kritických prevádzkových aplikácií, hoci ich tvorba v Express nástrojoch ani ich následná distribúcia nie je licenčne obmedzená.

V rámci rodiny „Express Edition" sú k dispozícii nasledujúce produkty:

Visual Basic 2008 Express Edition – kladie dôraz na produktivitu, je ideálny pre prvé programy v novom prostredí Visual Studio alebo pre príležitostných programátorov vo Windows a najmä pre všetkých, ktorí poznajú Visual Basic v akejkoľvek jeho podobe.

Visual C# 2008 Express Edition – prináša vyváženú kombináciu výkonu a produktivity pre študentov, nadšencov a fanúšikov programovania v tomto veľmi príjemnom jazyku založenom na „céčku".

Visual Web Developer 2008 Express Edition – obsahuje podporu programovacích jazykov Visual Basic a C# a slúži, ako už z názvu vyplýva, na k vytváranie webových projektov určených pre Internet alebo intranety s použitím ASP.NET 2.0.

Visual C++ 2008 Express Edition – jeho zvládnutie síce zaberie viac času než pri ostatných produktoch Express, ale v porovnaní s nimi zase ponúka vyšší výkon a presnejšiu kontrolu. Táto verzia je určená napríklad pre amatérov, ktorí vyvíjajú rôzne ovládače, hry a podobne.

Edíciu dopĺňajú databázové servery **SQL Server Express Edition a Compact.** Verzia Express býva súčasťou plnej inštalácie Express nástrojov, SQL Compact je určený pre malé desktopové aplikácie.

M**icrosoft PopFly a Popfly Explorer** – jednoduché prostredie na tvorbu webových aplikácií, umožňuje vytvárať vizuálne vyspelé webové prezentácie pomocou mixovania rôznych existujúcich webových aplikácií a webových služieb bez nutnosti písať kód a bez hlbokých technických znalostí v oblasti tvorby webu.

Integrovaná verzia „unit testovania"

Toto testovanie bolo k dispozícii aj v predchádzajúcej verzii prostredia Visual Studio 2005, však len v najvyššej edícii „Team Foundation". Vo verzii 2008 je toto testovanie dostupné už od verzie „Professional".

Testovacie projekty sú generované automaticky. Pre čo najvyššiu účinnosť testovania sa odporúča umiestniť testy do samostatných projektov, čiže ku každému projektu, ktorý chceme testovať vytvoríme jeho testovací projekt. Dôležité je nielen testovací projekt vytvoriť, ale ho aj udržiavať. To znamená, že ak sa v životnom cykle vývoja projektu do kódu pridá nová metóda, je potrebné do testovacieho projektu zahrnúť aj kód pre otestovanie tejto novej metódy. Podobne ako nie je dobré aby programátori písali ku kódu ktorý vytvorili návody, nemali by vytvárať ani testy. Ideálna koncepcia pre tvorbu testov je koncepcia „čiernej skrinky", kedy nás nezaujíma implementácia, ale požiadavky na funkcionalitu. Testy vytvárame na základe zadania a jeho analýzy a to najlepšie ešte v etape analýzy zadania a návrhu riešenia. Takéto testy sú oveľa účinnejšie ako testy, ktoré napíšeme po ukončení vývoja, kedy sa podvedome snažíme prispôsobiť testy implementácii zadania. Testy by mali byť prehľadné, to znamená, že každý test by mal testovať len jednu funkciu alebo metódu a mal by byť nezávislý na ostatných testoch. Aj napriek odporúčaniu nevytvárať testy podľa konkrétnej implementácie spravidla dokážeme definovať hraničné podmienky a oblasť zakázaných hodnôt, napríklad podľa matematickej teórie, alebo princípov fungovania objektívnej reality, podľa ktorej vytvárame model aplikácie. Aj napriek tomu, že pre testovanie je oveľa dôležitejším kritériom kvalita testovania, je potrebné pokryť čo najväčší rozsah kódu, pričom si ale netreba klásť nerealistické ciele, pretože stopercentné pokrytie zložitejšej aplikácie kvalitnými testami je prakticky nedosiahnuteľné. Pokrytie kódu testovacími unitami je možné merať pomocou kritéria „code coverage". Pri plánovaní a realizácii pokrytia kódu testovacími unitami postupujeme zhora nadol v zmysle hierarchického poradia.

systém -> riešenie -> projekt -> trieda -> metóda...

Visual Basic 9 a C# 3.0

Významnou novinkou pre túto dvojicu najpoužívanejších programovacích jazykov pre .NET sú anonymné používateľské dátové typy. Nepomenované typy môžeme vytvoriť pomocou operátora „new". Princíp tohto triku je veľmi jednoduchý – kompilátor si nepomenovaný typ vnútorne pomenuje, a toto pomenovanie platí len pre jeden cyklus prekladu. Podobný trik je použitý aj pre premenné v programovacom jazyku Visual Basic, pri ktorých vývojár neuvedie pri deklarácii explicitne ich dátový typ. Kompilátor pridelí premennej dátový typ podľa toho, s akými údajmi sa v premennej pracuje. V predchádzajúcich verziách bola takáto premenná ponímaná ako typ „Object" alebo „Variant". Takáto premenná síce fungovala s rozličnými dátovými typmi ale bola okolo toho veľká vnútorná réžia, čo pri častom používaní takýchto premenných neprispievalo k optimálnemu výkonu aplikácií. Pomocou kľúčového slova „var " je možné deklarovať premenné bez explicitne definovaného dátového typu aj v typovo prísnejšom programovacom jazyku C# 3.0. Najdôležitejší fakt sme si nechali nakoniec. Obidva programovacie jazyky sú stopercentne kompatibilné s predchádzajúcimi verziami.

C# (vyslovuj sí-shárp) je objektovo orientovaný programovací jazyk vyvinutý spoločnosťou Microsoft. Jeho „otcom" je Anders Hejlsberg (Turbo Pascal, Borland Delphi...). Vychádza z toho najlepšieho, čo poskytujú programovacie jazyky C++ a Java. Všetky dobré a pokrokové črty jazyka C++ zostali zachované. Zanikli však črty jazyka, z ktorých mali začínajúci céčkari doslova nočnú moru. Na prvom mieste to boli (a v C# už našťastie nie sú) smerníky – pointre. Skúsení céčkari na ne nedali dopustiť, pretože im umožňovali realizovať veľmi efektívne a rýchle programové konštrukcie. No stačilo trochu nepozornosti a mohlo dôjsť k veľmi nepríjemným efektom, ktoré snáď ani inak ako chybovým ukončením aplikácie ani nekončili. V prípade operačných systémov MS DOS, alebo Windows 95 „padajúca aplikácia" vzala so sebou spravidla aj operačný systém. Namiesto pointerov sa v C# používajú odkazy. Programátorom uľahčia život aj ďalšie črty, napríklad Garbage Collection. To znamená, že sa nemusíme starať o upratovanie nepotrebných objektov z pamäte. Jazyk a jeho implementácia vo prostredí Visual Studio poskytuje podporu pre rýchle programovanie „rapid application development", silnú typovú kontrolu, kontrolu ohraničenia polí, detekciu pokusov na využitie neinicializovaných premenných a automatickú správu pamäte. Dôležitými vlastnosťami je tiež robustnosť, odolnosť a produktivita. C# bol vytvorený so zreteľom na vývoj softvérových komponentov, ktoré sú vhodné pre nasadenie v distribuovaných prostrediach. Je zameraný aj na tvorbu aplikácií pre hostované, ako aj embedded systémy s ohľadom na veľkú škálovateľnosť od serverov až po mobilné zariadenia. Napriek tomu, že jazyk C# je navrhnutý s ohľadom na ekonomické využívanie procesora a pamäte, nie je až natoľko zameraný na výkonnosť a veľkosť výsledného binárneho kódu, ako jazyky C alebo assembler. Jazyk C# využíva vlastnosti vrstvy CLI

(Common Language Infrastructure), ktorá leží pod ním. Väčšina typov zavedených v jazyku C# priamo korešponduje s hodnotovými typmi implementovanými v CLI frameworku, špecifikácia jazyka C# však neurčuje podmienky, ktorými sa má generovať kód z kompilátora. To znamená, že kompilátor jazyka C# nemusí mať za cieľovú podpornú platformu priamo CLI, respektíve vôbec nemusí generovať medziprekladový jazyk MSIL (Microsoft Intermediate Language), ani žiaden iný formát. Teoreticky je možné vytvoriť kompilátor jazyka C#, ktorý bude prekladať priamo do strojového kódu ako tradičné kompilátory jazyka C++, Fortran a podobne.

Klasický učebnicový príklad „Hello World" sa v jazyku C# môže naprogramovať napríklad takto:

```
using System;
class Hello
{
  static void Main() {
   Console.WriteLine("Hello, world");
  }
}
```

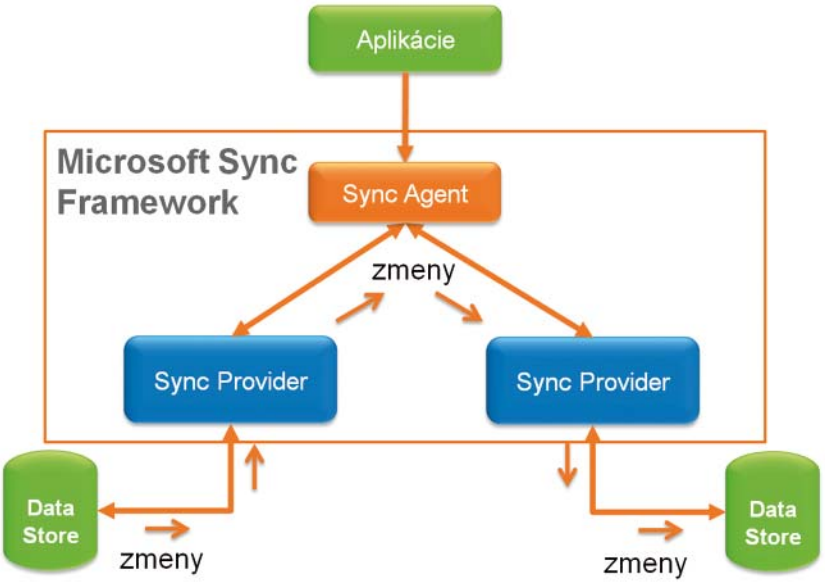
Popfly

Spoločnosť Microsoft dala k dispozícii zadarmo webovú službu Popfly **http://www.popfly.com** ktorá je určená pre tvorbu aplikácií z hotových modulárnych blokov pochádzajúcich z rôznych zdrojov. Informácie z jednotlivých modulov sa spájajú pomocou intuitívneho grafického rozhrania pre grafický návrh bez programovania, takže je vhodná aj pre úplných začiatočníkov. Pomáha realizovať ich vízie typu „Takto by som to chcel mať". Pre každého používateľa je vyhradenýché 25 MB. Na klientskej strane je potrebné mať nainštalovaný doplnok Silverlight 1.0. Podporované sú prehľadávače Mozilla Firefox 2.0 a Internet Explorer od verzie 6.0. Technológia Popfly je „mashup", teda umožňuje koexistenciu a súhru viacerých technológií za účelom prezentovania obsahu v graficky a multimediálne čo najbohatšej forme. Vo svojich „Popfly" aplikáciách môžete použiť HTML, XHTML, CSS, JavaScript, knižnice pre Ajax, Visual Studio Express projekty, grafiku vo formátoch JPG, PNG, GIF, audiovizuálne formáty WMV, WMA, MP3 a EXE aplikácie.

Len tak pre zaujímavosť, predtým bola domovskou stránkou PopFly **http://www.popfly.ms** , spoločnosť Microsoft ešte nedosiahla taký globálny rozmach, aby mala ako jediná spoločnosť svoju vlastnú internetovú domému MS. A ani ju nekúpila od žiadneho miništátika s diktátorským režimom (Pamätáte sa ako svojho času bola k dispozícii doména .TM patriaca Turkménsku?). Doména .MS patrí britskému územiu Montserrat v Karibskom mori. Je otvorená pre každého za ročný poplatok 35 dolárov. No a spoločnosti Microsoft sa táto doména celkom hodí.

Microsoft Sync Framework

Vo verzii CTP je dostupný balíček MS Sync Framework, infraštruktúry pre vývoj aplikácií využívajúcich synchronizáciou na rôznych protokoloch a medzi rôznymi heterogénnymi zdrojmi. Je možné vytvárať online P2P (peer-to-peer), scenáre, prípadne scenáre pre synchronizáciu s offline zdrojmi.



Architektúry MS Sync Frameworku

S uvedením VS2008 a VS2008 SDK sa zmenilo licencovanie pre partnerské firmy využívajúce IDE prostredia Visual Studio 2008 pre svoje projekty na iných platformách, napríklad pre Linux, alebo jednočipové počítače, vrátane sprístupnenia kompletných zdrojových kódov prostredia Visual Studio na úrovni partnerskej spolupráce „Premium".

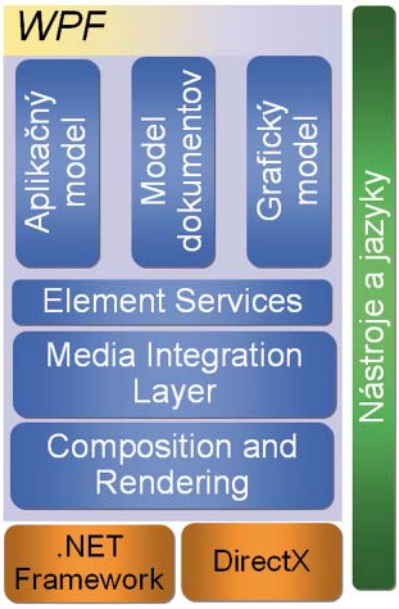
Windows Presentation Foundation

Prezentačné rozhranie WPF je koncipované ako jednotný framework pre vývoj novej generácie aplikácií, ktoré zahrňuje používateľské rozhranie aplikácií s využitím skúseností používateľov, médií a dokumentov.
Zdalo by sa, že WPF aplikácie budú náročné na výpočtovú kapacitu, no spravidla je to presne naopak.
Pri klasických WinForm aplikáciách celá aplikačná logika, vrátane grafického zobrazovania leží na bedrách procesora, zatiaľ čo najdrahšia časť klientskeho počítača – grafická karta s 3D akceleráciou nečinne zaháľa.
Pri WPF aplikáciách sa kompletné vektorové zobrazovanie realizuje hardvérovými prostriedkami grafickej karty, takže sa môže procesoru značne odľahčiť. Aplikácie je možné spúšťať na operačných systémoch Windows Vista, Windows Server 2008 a po nainštalovaní prostredia .NET Framework 3.5 aj na systémoch Windows XP SP2 a Windows Server 2003. Architektúra WPF je založená na troch základných modeloch:

Aplikačný model – pomocou deklaratívneho programovania v jazyku XAML môžeme jednoducho integrovať všetky typy médií, pomocou databindingu ich previazať s údajmi, pomocou štýlov a tém nastavovať statický a dynamický vzhľad aplikácií. Aplikácie môžeme spúšťať nielen priamo v prostredí operačného systému, ale rovnako aj v prostredí prehľadávača internetového obsahu, čo prispieva k zvýšeniu bezpečnosti a ochrany operačného systému.

Grafický model ponúka okrem rastrovej grafiky aj 2D a 3D vektorovú grafiku. Grafické prvky je možné ľubovoľne kombinovať s textovými informáciami, video a audio súbormi a taktiež aj s animáciami a obchodnou grafikou. Animácie môžu byť realizované v 2D alebo 3D zobrazovacom priestore.

Model dokumentov rozširuje možnosti spravovania dokumentov v rámci aplikácií. Textové informácie môžeme zobrazovať v pevnom, plávajúcom alebo adaptívnom rozložení pri zachovaní pokročilých typografických prvkov a metód. Na úrovni aplikácií môžeme taktiež riadiť prístupové práva k jednotlivým dokumentom.



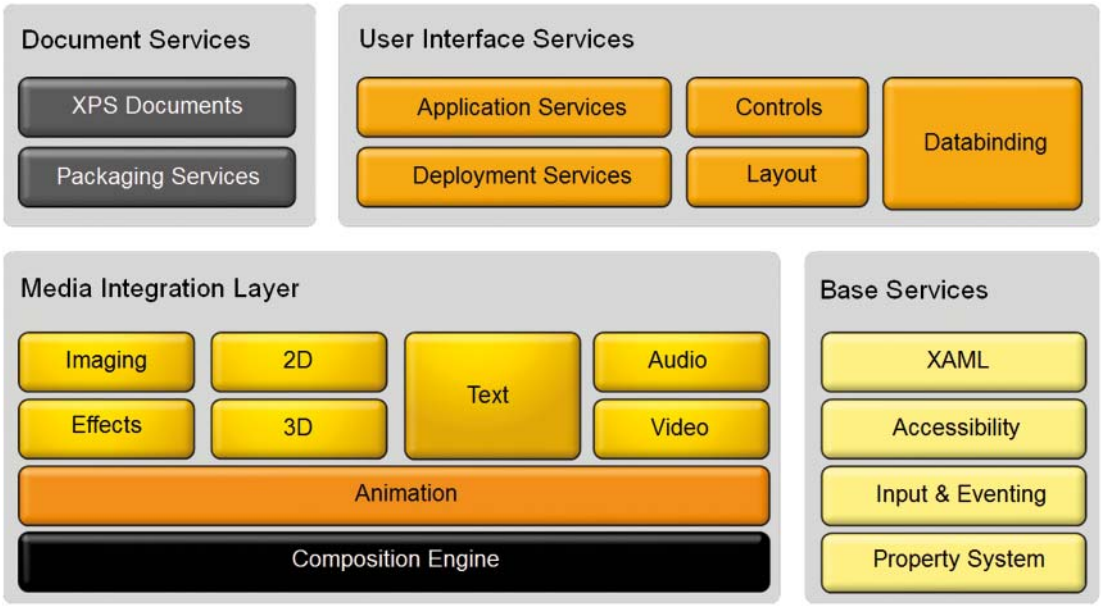
Bloková architektúra Windows Presentation Foundation

Komponenty WPF je možné rozdeliť do niekoľkých blokov. Základným technologickým pilierom architektúry WPF je blok takzvaných základných služieb, teda služieb na najnižšej úrovni. Obsahuje modul nového jazyka XAML (eXtensible Application Markup Language) pre vytváranie prezentačného rozhrania aplikácií, komponenty pre sprístupnenie ovládania našich aplikácií aj pre zdravotne postihnutých používateľov, komponenty pre vstup údajov a spracovanie udalostí ktoré v aplikáciách nastávajú, napríklad reakcie na manipuláciu s ovládacími prvkami. Do tohto bloku patrí aj podsystém pre správu vlastností vizuálnych aj nevizuálnych prvkov.

Do bloku používateľského rozhrania patria grafické ovládacie prvky vrátane rozhrania pre ich operatívne a interaktívne rozmiestňovanie, tak ako sme zvyknutí pri webových aplikáciách napríklad pri zmene veľkosti okna aplikácie. Pomocou databindingu je možné prepojiť ovládacie prvky s údajmi v databázach. Pomocou takto prepojených ovládacích prvkov vhodného typu (tabuľkové prvky, prvky pre zobrazenie zoznamov, editačné polia...) je možné zobrazovať a meniť údaje z databázových tabuliek, prípadne iných zdrojov. Do tohto bloku patria aj služby, ktoré sa starajú o riadenie aplikácie a jej nasadenie.

Blok pre integráciu médií má na starosti prácu aplikácie s médiami, či už ide o médiá statické, ako napríklad obrázky a grafy reprezentované pomocou dvojrozmernej, prípadne trojrozmernej vektorovej grafiky, prípadne médiami dynamickými, kam môžeme zaradiť text, hypertext, audio a video. V prípade potreby je pomocou animačného jadra možné statické aj dynamické grafické komponenty animovať.

Blok služieb pre dokumenty zabezpečuje ich export vo formáte XPS, prípadne ich zabalenie a ochranu pred zmenami tak, aby v ďalšom predpokladanom životnom cykle dokumentov nemohlo dochádzať k porušovaniu autorských práv.



Technologická schéma komponentov architektúry WPF

Ak chceme využívať grafický podsystém AERO (Authentic, Energetic, Reflective, Open), ktorý zabezpečuje napríklad priehľadnosť okrajov okien či 3D Flip (trojrozmerné prepínanie okien), potrebujeme výkonnú grafickú kartu s príslušnou podporou hardvérovej akcelerácie. Ak počítač klienta disponuje slabšou grafikou systém AERO nebude aktivovaný, to znamená, že jeho rozšírené grafické možnosti k dispozícii nebudú, no zvyšok operačného systému pobeží bezo zmeny a okná sa budú zobrazovať klasicky ako sme zvyknutí v szstéme Windows XP.

WPF je zamerané na používateľsky „bohaté" aplikácie, takže je možné opustiť dizajnový vizuál klasických formulárov a naplno využiť možnosti vektorovej grafiky. Vektorová grafika s využitím hardvérovej akcelerácie napríklad umožňuje graficky bezstratovú zmenu veľkosti elementov, používanie geometrických tvarov. Vizuál aplikácie pritom zostáva nezávislý od rozlíšenia, takže aplikácie vyzerajú rovnako pri rôznom nastavení rozlíšenia. Do hry vstupuje animácia vrátane trojrozmernej, jednoduchá práca s audiom a videom.

XAML ako iste tušíte, ide o programovací, alebo presnejšie značkovací jazyk, pričom kód a deklarácie sa píšu vo forme XML dokumentov. Jazyk XAML umožňuje deklaratívne zmeniť vzhľad aplikácie bez toho, aby sme museli prepisovať jej kód. Vrstva WPF prináša model postavený na deklaratívnom programovaní, kde sa snažíme oddeliť grafický návrh aplikácie a jej aplikačnú logiku. Takáto filozofia nie je nová, poznáme ju napríklad z technológie ASP.NET, ktorá je určená na vytváranie webových aplikácií. Pomocou jazyka XAML navrhneme aplikáciu. Deklaratívny návrh v jazyku XAML je následne bez nášho zásahu prevedený do zdrojového kódu a následne preložený do binárnej formy. Jazyk XAML je postavený na existencii neúplných tried (partial class). Nemusí slúžiť len k definícii grafického vzhľadu aplikácie, ako sú okná a ovládacie prvky ale aj k definícii aplikácie ako celku, definícii systémových zdrojov a podobne.

XMLNS pre XAML je na adrese: **http://schemas.microsoft.com/winfx/xaml/presentation**

Microsoft Expression Blend 2.5

Microsoft Expression Blend je flexibilné a produktívne grafické vývojové prostredie. Pomáha pri tvorbe moderných a vizuálne prepracovaných aplikácií s interaktívnou podporou 3D zobrazovania a prehrávania multimédií aj 3D zobrazením. Umožňuje vytvorenie a úpravy prezentačnej vrstvy webových aplikácií. Využíva nový druh značkovacieho jazyka XAML. V dobe písania publikácie bol tento produkt vo verzii „June 2008 Preview".

Všimnite si v kontextovom menu, že projekt, ktorý sme začali vyvíjať v prostredí Expression Blend 2.5 môžete otvoriť aj vo vývojovom prostredí Visual Studio 2008, samozrejme len v prípade, ak je toto prostredie nainštalované a pokračovať vo vývoji v tomto prostredí.

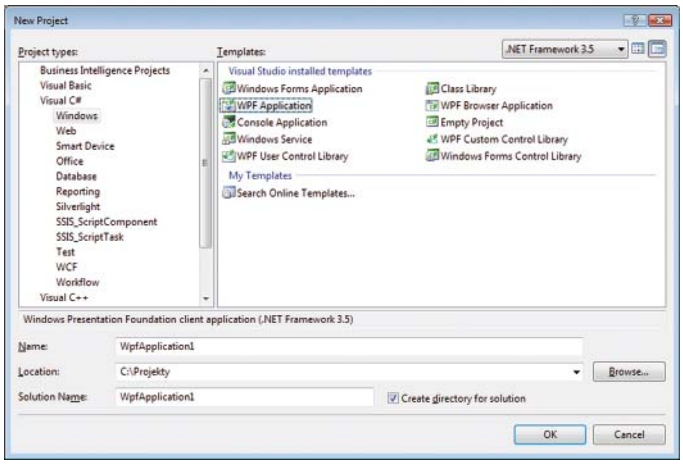
Hierarchia tried WPF

Niekedy je dôležité uvedomiť si hierarchiu tried a objektov, hlavne pri tak rozsiahlom bloku, akým je grafické prezentačné rozhranie. Triedy WPF sú v namespace System.Windows z assembly: PresentationFramework v knižnici PresentationFramework.dll. Samotná hierarchia tried je nasledovná:

```
System.Object
  System.Windows.Threading.DispatcherObject
    System.Windows.DependencyObject
      System.Windows.Media.Visual
        System.Windows.UIElement
          System.Windows.FrameworkElement
            System.Windows.Controls.Control
              System.Windows.Controls.ContentControl
                System.Windows.Window
                  System.Windows.Navigation.NavigationWindow
```

Prvá WPF aplikácia

Aby sme ukázali postup a základné možnosti WPF aplikácie, vytvoríme vo vývojovom prostredí Visual Studio 2008 novú aplikáciu podľa šablóny „WPF application". Aplikácia vypíše jednoduchý text.



Výber šablóny pre vytvorenie novej WPF aplikácie

Po vytvorení projektu je pracovná plocha rozdelená na oblasť pre návrh v grafickom prostredí v hornej časti a okno pre zobrazenie XAML kódu v spodnej časti. Vpravo sú okná „Solution Explorer " a „Properties". Vľavo môžeme zobraziť a Toolbox s ponukou prvkov pre grafický návrh. Po voľbe základných parametrov, teda názvu projektu, jeho umiestnenia a programovacieho jazyka, modul vývojového prostredia nazývaný „Sprievodca vytvorením aplikácie" vytvorí prázdnu šablónu aplikácie, čiže ľudovo povedané aplikáciu, ktorá zatiaľ nič nerobí a nič nezobrazuje.

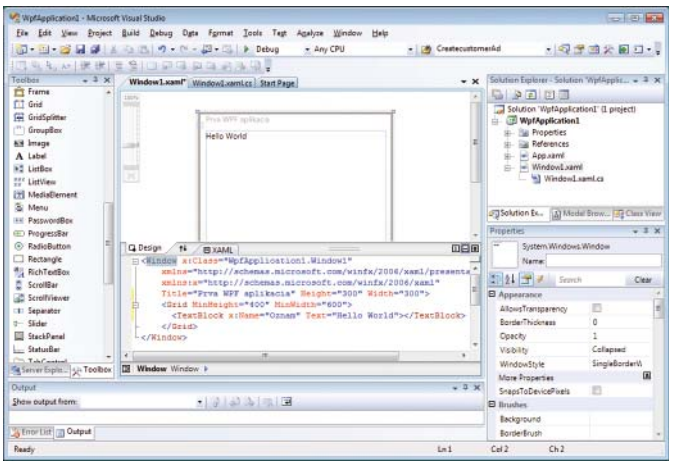
V súbore Windows1.xaml bude XAML kód definujúci grafické prvky prezentačného rozhrania:

```
<Window x:Class="WpfApplication1.Window1"
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  Title="Window1" Height="300" Width="300">
  <Grid>

  </Grid>
</Window>
```

V súbore Windows1.xaml.cs budeme tvoriť kód aplikačnej logiky. Pri vytvorení WPF aplikácie je tam len inicializačný kód:

```
namespace WpfApplication1
{
  public partial class Window1 : Window
  {
    public Window1()
    {
      InitializeComponent();
    }
  }
}
```

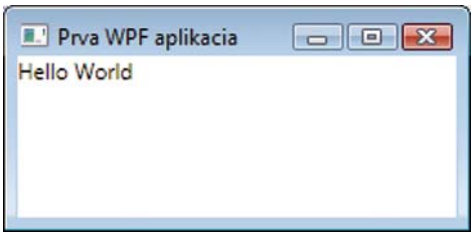
Ak by sme vo verzii Beta 2 skúsili pridať vizuálny prvok z Toolboxu priamo na návrhovú plochu, neuspeli by sme. Prvky sa pridávajú do XAML kódu. Takto môžeme pridať napríklad prvok Label dovnútra tagu `<Grid>`. Pri nastavovaní vlastností prvku môžeme využiť črtu Intellisense, ktorá nám formou interaktívneho pomocníka ponúka názvy parametrov a kostru syntaxe pre ich zápis. Tento pomocník funguje nielen pre kľúčové slová a objekty programovacích jazykov, ale aj pre XAML. Pridáme prvok TextBlock s textom Hello World.



Návrh WPF aplikácie vo prostredí Visual Studio 2008

```
<Window x:Class="WpfApplication1.Window1"
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  Title="Prva WPF aplikacia" Height="300" Width="300">
  <Grid MinHeight="400" MinWidth="600">
   <TextBlock x:Name="Oznam" Text="Hello World"></TextBlock>
  </Grid>
</Window>
```

Teraz môžeme aplikáciu spustiť.



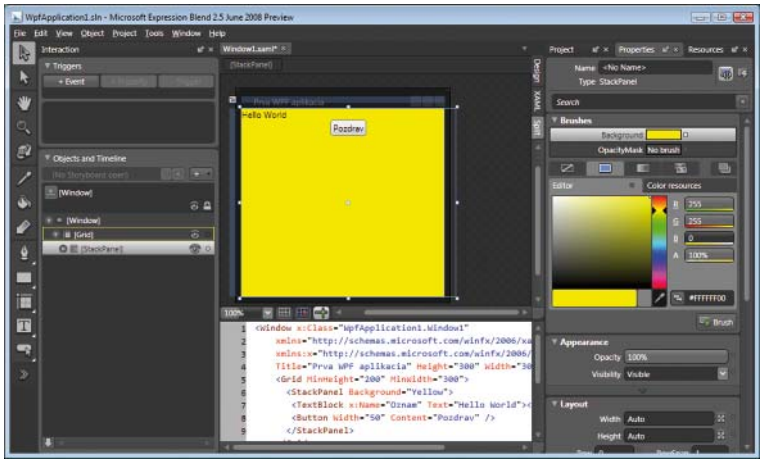
Spustenie WPF aplikácie

Ak potrebujeme, môžeme v okne „Properties" meniť rôzne vlastnosti okna a ovládacích prvkov. Napríklad môžeme zmeniť štýl okna a podobne.

Zmena štýlu okna

Návrh WPF aplikácie v prostredí Expression Blend 2.5

Projekt, ktorý sme začali vyvíjať vo vývojovom prostredí Visual Studio 2008 je možné otvoriť a pracovať s ním aj v prostredí Expression Blend 2.5 a naopak. Preto budeme v interaktívnom grafickom návrhu našej „Hello World " aplikácie v Expression Blend 2.5. Takže otvoríme projekt a doplníme do neho tlačidlo.



Návrh WPF aplikácie v prostredí Expression Blend 2.5

Aby aplikácia mala aspoň nejaký náznak úpravy a text a tlačidlo boli pod sebou, pridali sme do projektu kontajnerový objekt **StackPanel,** ktorý slúži na rozmiestnenie objektov nad sebou, alebo vedľa seba.

```
<Window x:Class="WpfApplication1.Window1"
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  Title="Prva WPF aplikacia" Height="300" Width="300">
  <Grid MinHeight="200" MinWidth="300">
   <StackPanel Background="Yellow">
    <TextBlock x:Name="Oznam" Text="Hello World"></TextBlock>
    <Button Width="50" Content="Pozdrav" />
   </StackPanel>
      </Grid>
</Window>
```

Okrem prvku StackPanel sú pre WPF k dispozícii ďalšie kontajnerové prvky:

Grid – tabuľka podobná HTML tabuľke, obsahuje riadky a stĺpce, pričom sa dá nastaviť ich dĺžka.

WrapPanel – prvky sú zalomené vedľa seba, pričom ak sa niektorý prvok už nevojde na koniec riadka, bude presunutý na nový riadok.

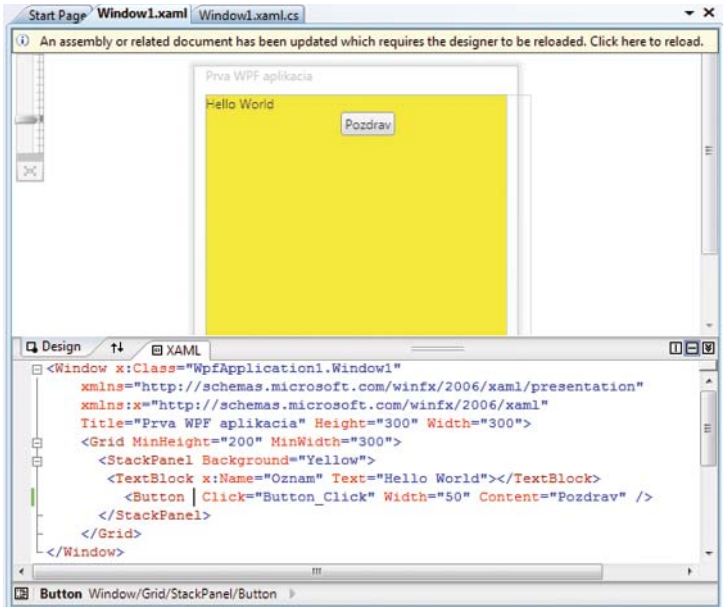
DockPanel slúži na „prilepenie" elementov k jednému zo štyroch okrajov formulára.

Canvas slúži na uloženie prvkov na absolútne pozície. Preto by sa mal tento kontajner používať len v odôvodnených prípadoch.

Zmeny vykonané návrhovom prostredí Expression Blend 2.5 sú po upozornení následne akceptované aj v prostredí Visual Studio 2008. Takže projekt zatvoríme a pokračujeme znova v aplikácii Visual Studio, kde doprogramujeme aplikačnú logiku.

Pokračujeme vo vývoji aplikačnej logiky v prostredí Visual Studio...
Každú aplikáciu tvorí aplikačná a prezentačná vrstva. Spomínané vrstvy dokážeme v aplikácii identifikovať aj tomto prípade, kedy pôjde o niekoľko málo riadkov kódu. Úlohou aplikačnej vrstvy je pracovať s údajmi a premennými a pripraviť údaje, ktoré sa zobrazia používateľovi. Úlohou prezentačnej vrstvy je výpis týchto údajov vo vhodnej forme.

Pre tlačidlo pridáme parameter Click. Po pridaní atribútu Click sa pridá do XAML kódu text Click ="".Medzi úvodzovky bude automaticky umiestnená položka kontextového menu pre vytvorenie obslužnej procedúry udalosti kliknutia na tlačidlo.



Úprava WPF aplikácie v prostredí Expression Blend 2.5 sa prejaví aj v Visual Studio

V XAML kóde bude definícia tlačidla vo forme:

```
<Button Click="Button_Click" Width="50" Content="Pozdrav" />
```

Vidíme, že Intellisense nám výdatne pomáha, ponúklo nám aj vpísanie názvu obslužnej procedúry dovnútra úvodzoviek. Navyše ak v kontextovom menu, aktivovanom pravý tlačidlom myši zvolíme možnosť „Navigate To Event Handler" vytvorí sa aj telo obslužnej procedúry. Sem následne pridáme kód pre zmenu textu:

```
private void Button_Click(object sender, RoutedEventArgs e)
{
  Oznam.Text = "Ahoj";
}
```

Fungovanie aplikácie môžeme následne otestovať.

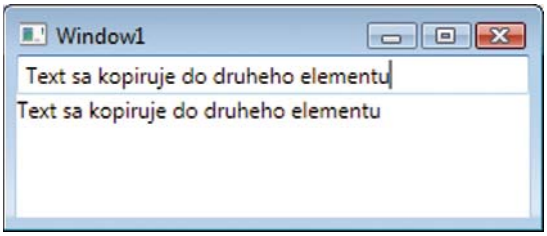
Ak si chcete pozrieť reálne možnosti WPF aplikácie, prevezmite a nainštalujte si aplikáciu Family.Show. Je k dispozícii vrátane zdrojového kódu na serveri Codeplex. Jej popis a prepojenie na prevzatie je na adrese **http://www.maxiorel.cz/sprava-rodokmenu-jako-ukazka-wpf.**

Ukážka možností WPF aplikácie

Databinding WPF aplikácií

Veľa vývojárov si pod databindingom predstavuje nadviazanie na zdroj údajov niekde v databáze. Možnosti databindingu sú oveľa širšie. Aktívne zobrazovacie prvky a prvky pre zadávanie údajov môžeme pripájať nielen k databáze a rôznym objektom typu pole a zoznam (.NET Framework 3.5 obsahuje pre tento účel nové technológie a služby, ktoré budú predstavené v ďalších kapitolách), ale aj na premenné a dokonca aj medzi sebou. Najskôr ukážeme najtriviálnejšie nadviazanie dvoch prvkov na seba. Text napísaný do prvku TextBox sa bude automaticky kopírovať do prvku TextBlock.

```
<Window x:Class="Binding.Window1"
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  Title="Window1" Height="300" Width="300">
  <Grid>
    <StackPanel Orientation="Vertical">
      <TextBox Name="tbZdroj" Text="zdroj" />
      <TextBlock Text="{Binding ElementName=tbZdroj, Path=Text}" />
    </StackPanel>
  </Grid>
</Window>
```



Databinding dvoch elementov

Smer a režim databindingu môžeme ovplyvňovať nastavením parametrov:

Mode – režim synchronizácie

TwoWay	Hodnoty sú synchronizované obojsmerne. Tento mód je nastavený implicitne.
OneWay	Hodnoty sú synchronizované iba zo zdroja do nabindovanej vlastnosti, zmeny v tejto vlastnosti sa do zdroja neprenesú.
OneTime	Hodnoty sú synchronizované len po štarte aplikácie. Neskoršie zmeny sú ignorované.

UpdateSourceTrigger – atribút pre aktualizáciu zdrojovej hodnoty

Excplicit	K synchronizácii dôjde po zavolaní metódy UpdateSource.
LostFocus	Hodnota sa aktualizuje len čo nabindovaný komponent prestane byť aktívna. Tento atribút je nastavený implicitne.
PropertyChanged	K synchronizácii dôjde po zmene nabindovanej hodnoty.

V ďalšej mini ukážke sa zameriame na databinding na objekty, konkrétne na triedu. Trieda musí byť odvodená od materskej triedy `INotifyPropertyChanged`, aby mohlo bindovanie reagovať na zmeny. Do projektu pridáme namespace:

```
using System.ComponentModel;
```

Vytvoríme novú triedu s názvom knihy. Budú nás zaujímať dva textové atribúty – meno autora a názov knihy a jeden číselný atribút, ktorým je cena. Najskôr triedu definujeme:

```
public class Knihy : INotifyPropertyChanged
{
  private string autor { get; set; }
  private string nazov { get; set; }
  private int cena { get; set; }

  public string Autor
  {
    get { return autor; }
    set { autor = value; NotifyPropertyChanged("Autor"); }
  }

  public string Nazov
  {
    get { return nazov; }
    set { nazov = value; NotifyPropertyChanged("Nazov"); }
  }

  public int Cena
  {
    get { return cena; }
    set { cena = value; NotifyPropertyChanged("Cena"); }
  }

  public event PropertyChangedEventHandler PropertyChanged;
  private void NotifyPropertyChanged(string info)
  {
    if (PropertyChanged != null)
      PropertyChanged(this, new PropertyChangedEventArgs(info));
  }

}
```

Udalosť PropertyChanged je aktivovaná metódu NotifyPropertyChanged vtedy keď sa zmení hodnota niektorého parametra triedy. Zmenu budeme simulovať tlačidlom s obslužnou procedúrou:

```
private void Button_Click(object sender, RoutedEventArgs e)
{
  kniha.Autor = "Adams Douglas";
  kniha.Nazov = "Stoparov sprievodca galaxiou";
  kniha.Cena = 299;
}
```

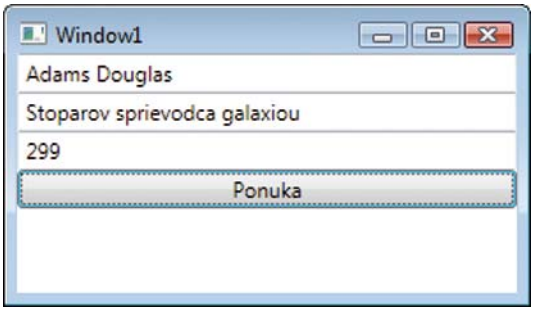
V aplikácii vytvoríme inštanciu objektu kniha:

```
Knihy kniha;

public Window1()
{
  InitializeComponent();
  kniha = new Knihy();
  spFormular.DataContext = kniha;
}
```

Používateľské rozhranie miniaplikácie bude pozostávať z troch textboxov naviazaných na jednotlivé atribúty triedy kniha a tlačidla ponuka, ktoré inicializuje implicitnú hodnotu:

```
<Window x:Class="Binding2.Window1"
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  Title="Window1" Height="300" Width="300">
  <Grid>
    <StackPanel Name="spFormular">
      <TextBox Text="{Binding Path=Autor}" />
      <TextBox Text="{Binding Path=Nazov}" />
      <TextBox Text="{Binding Path=Cena}" />
      <Button Click="Button_Click">Ponuka</Button>
    </StackPanel>
  </Grid>
</Window>
```

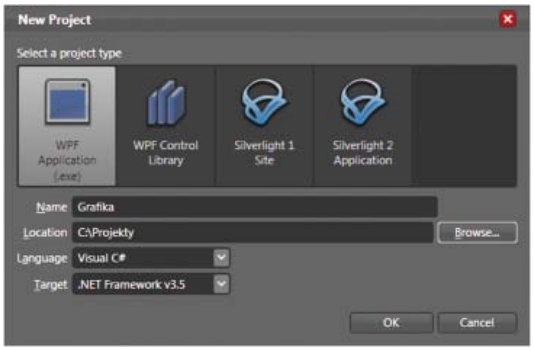
Po spustení aplikácie môžeme otestovať jej správanie, po inicializácii sa nabindujú prvky na hodnoty atribútov triedy, taktiež po zmene atribútov simulovaných tlačidlom sa táto zmena okamžite prejaví.



Databinding na objekt

Dvojrozmerná grafika cez WPF

Základným pilierom pre tvorbu dvojrozmernej grafiky je značkovací jazyk XAML. Nakoľko pri grafike platí dvojnásobne, že lepšie je raz vidieť ako dvakrát čítať, budeme princípy tvorby grafického prezentačného rozhrania vysvetľovať na praktickom príklade. Vytvorme nový projekt typu WPF aplikácia v interaktívnom návrhovom prostredí Microsoft Expression Blend 2.5. Projekt môžeme vytvoriť a realizovať aj v prostredí Visual Studio, Expression Blend je však pre tvorbu grafiky podstatne interaktívnejší. Keďže ten istý projekt môžeme vyvíjať aj v prostredí Expression Blend 2.5 aj v prostredí Visual Studio 2008, môžeme medzi týmito prostrediami ľubovoľne prepínať.

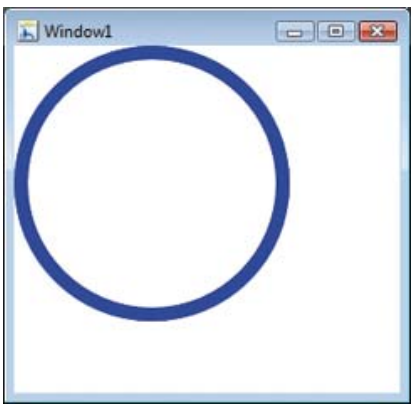


Vytvorenie WPF aplikácie v návrhovom prostredí Expression Blend

Pre názornosť budeme pracovať s absolútnymi súradnicami, preto grafické objekty zapuzdríme do kontajnerového prvku Canvas, ktorý má pevnú súradnicovú sústavu. Ako prvý útvar vytvoríme ohraničený kruh. Kruh sa vytvára ako elipsa, ktorá má obidve osi rovnaké.

```
<Window
       xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
       xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
       x:Class="WpfGrafika.Window1"
       x:Name="Window"
       Title="Window1"
       Width="640" Height="480">
  <Canvas>
   <Ellipse Width="200" Height="200" Stroke="Blue" StrokeThickness="10">
   </Ellipse>
  </Canvas>
</Window>
```

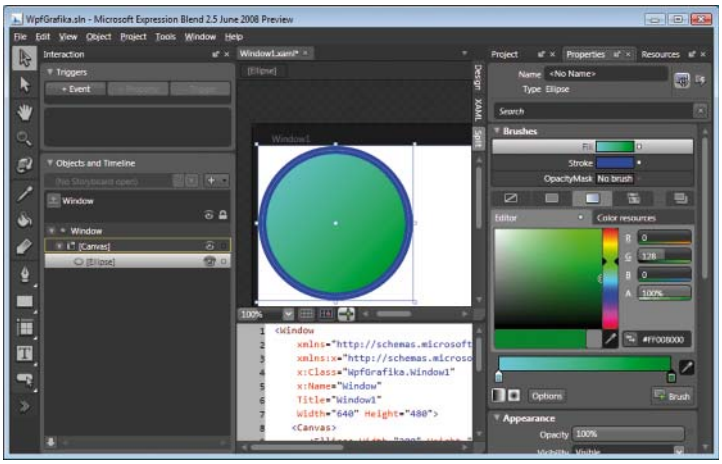
V návrhovom prostredí Microsoft Expression Blend 2.5 môžeme spustiť WPF aplikáciu pomocou klávesovej skratky CTRL F5.



WPF grafická aplikácia zobrazujúca olemovaný kruh

```
<Canvas>
  <Ellipse Width="200" Height="200" Stroke="Blue" StrokeThickness="10">
   <Ellipse.Fill>
    <LinearGradientBrush>
      <GradientStop Offset="0" Color="Cyan" />
      <GradientStop Offset="1" Color="Green" />
    </LinearGradientBrush>
   </Ellipse.Fill>
  </Ellipse>
 </Canvas>
```

Kruh môžeme vyfarbiť, pričom pomocou parametra Gradient môžeme jednoducho dosiahnuť prelínanie farieb.



Vyfarbený prvok s gradientom

Geometrické útvary sa môžu prekrývať, pričom dovnútra môžeme vložiť textový element, prípadne video. Tieto mediálne prvky môžeme vložiť aj ako pozadie prvku TextBlock. V našom príklade to všetko skombinujeme.

```
 <Canvas>
  <Path Width="800" Height="800">
   <Path.Data>
   <CombinedGeometry GeometryCombineMode="Union">
    <CombinedGeometry.Geometry1>
     <EllipseGeometry Center="200,200" RadiusX="150" RadiusY="150" />
    </CombinedGeometry.Geometry1>
    <CombinedGeometry.Geometry2>
     <EllipseGeometry Center="400,200" RadiusX="150" RadiusY="150" />
    </CombinedGeometry.Geometry2>
   </CombinedGeometry>
  </Path.Data>

   <Path.Fill>
    <VisualBrush>
    <VisualBrush.Visual>
    <TextBlock Text="(c) Moje video" FontWeight="Bold" Padding="10">

    <TextBlock.Background>
     <VisualBrush>
      <VisualBrush.Visual>
       <MediaElement UnloadedBehavior="Play" Source="C:\Projekty\Lake.wmv" />
     </VisualBrush.Visual>
    </VisualBrush>
    </TextBlock.Background>

   </TextBlock>
  </VisualBrush.Visual>
  </VisualBrush>
 </Path.Fill>

</Path>
</Canvas>
```

Možnosti WPF grafiky

Trojrozmerná grafika cez WPF

Vývojári zaoberajúci sa vývojom hier vedia, že všetky trojrozmerné objekty sa počítajú a renderujú prostredníctvom trojuholníkov. Pri listovaní parametrami grafických kariet s 3D akceleráciou určite narazíte na parameter, koľko miliónov trojuholníkov dokáže karta vyrenderovať za sekundu. My ich nebudeme potrebovať milióny ako pre zložité hry, ukážeme zobrazenie jednoduchých dvojrozmerných a trojrozmerných objektov. Aj pri dvojrozmerných objektoch, napríklad trojuholníkovej plochy, môžeme túto umiestniť ľubovoľne v priestore. Ešte raz zdôrazňujeme, že **všetky trojrozmerné grafické útvary sú v 3D WPF zložené z trojuholníkov**. Určite si dokážete predstaviť, že štvorec vytvoríme z dvoch rovnoramenných pravouhlých trojuholníkov, ktoré sa dotýkajú preponami, čitatelia podkutí v deskriptíve nebudú mať problém predstaviť si kocku zloženú z 12 trojuholníkov, veď kocka má šesť štvorcových stien, a vieme, že každá sa skladá z dvoch trojuholníkov. Z veľkého množstva maličkých trojuholníkov dokážeme poskladať aj mnohosten, ktorý je aproximáciou gule a vlastne ľubovoľne zložitý útvar.

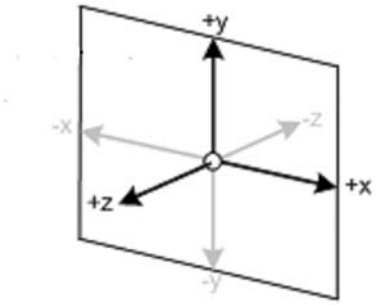
Návrh 3D prostredia môžeme realizovať v interaktívnom návrhovom prostredí Microsoft Expression Blend 2.5. Vytvoríme nový projekt typu WPF aplikácia.

Príprava scény a rozmiestnenie kamier

Aby sme mohli zobrazovať trojrozmernú grafiku, je potrebné vytvoriť trojrozmerný pohľad Viewport3D. Tento pohľad si môžeme predstaviť ako otvorený priestor „za sklom monitora" v ktorom budú neskôr podľa definovaných pravidiel rozmiestnené grafické objekty. Pre jednoduchosť si predstavme, že tam už sú. To ako sa nám budú javiť, bude závisieť od toho, odkiaľ sa na ne budeme pozerať, teda z akého smeru a z akej vzdialenosti. Takže spresníme našu predstavu tak, že na vhodné miesto umiestnime kameru, ktorá bude snímať našu trojrozmernú scénu a obraz prenášať na monitor na ktorý sa budeme dívať. Preto ďalším krokom po vytvorení trojrozmerného prostredia pomocou Viewport3D bude umiestnenie virtuálnej kamery, ktorá toto prostredie bude snímať. Pre kameru nastavíme dva parametre:

– Position – pozícia kamery,
– LookDirection – smer pohľadu kamery,

Súradnicový systém trojrozmernej grafiky vo WPF má začiatok v strede monitora v rovine zobrazovacej plochy. Súradnice x a y sú orientované tak ako sme zvyknutí, teda x vodorovne a y zvislo. Os z je orientovaná kolmo na zobrazovaciu rovinu, pričom kladný smer smeruje k používateľovi a záporný akoby dovnútra monitora. Určite to lepšie objasní obrázok:



Súradnicový systém trojrozmernej grafiky vo WPF

```
<Viewport3D Name="priestor3D">
  <Viewport3D.Camera>
   <PerspectiveCamera LookDirection="0,0,-1" Position="0,0,5"
     NearPlaneDistance="3" FarPlaneDistance="100" />
</Viewport3D.Camera>
```

V našom prípade kamera leží pred monitorom a díva sa „dovnútra". Pomocou atribútov NearPlaneDistance a FarPlaneDistance určíme hranice zobrazovania, ktoré si môžeme predstaviť ako určitú hĺbku ostrosti pri klasickej kamere až na to, že pred a za ohraničenou oblasťou sa nič nezobrazí. Takto môžeme napríklad odfiltrovať nežiaduce objekty v pozadí, ktoré nás pri danom zobrazení nezaujímajú.

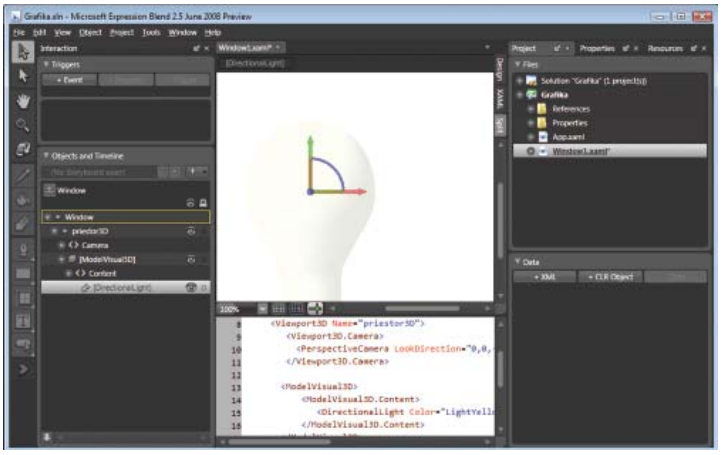
Rozmiestnenie svetiel osvetľujúcich scénu

Zatiaľ v našom príklade postupujeme ako pri príprave fotografického alebo filmového štúdia. Máme pripravenú scénu, rozmiestnené kamery..., ale zatiaľ je v štúdiu tma. Áno uhádli ste, bude nasledovať rozmiestnenie a smerovanie svetiel, čím určíme rôzne efekty bežné v trojrozmernom priestore, hlavne tiene. Osvetľovacie prvky sa zapuzdrujú do modelu ModelVisual3D. K dispozícii máme niekoľko typov svetiel:

– AmbientLight – rovnomerné plošné svetlo,
– DirectionalLight – smerové svetlo, tento druh svetla je ekvivalentom slnečného osvetlenia alebo reflektora, pretože osvetlí len jednu stranu modelu,
– PointLight, PointLightBase a SpotLight – bodové reflektory.

Pre náš model použijeme klasické smerové osvetlenie, čiže DirectionalLight:

```
<ModelVisual3D>
  <ModelVisual3D.Content>
    <DirectionalLight Color="LightYellow" Direction="0,0,-1" />
  </ModelVisual3D.Content>
</ModelVisual3D>
```



Návrh 3D priestoru v prostredí Expression Blend

Kompletný kód návrhu trojrozmernej scény bude:

```
<Window
       xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
       xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
       x:Class="Grafika.Window1"
       x:Name="Window"
       Title="Window1"
       Width="640" Height="480">
  <Viewport3D Name="priestor3D">
    <Viewport3D.Camera>
     <PerspectiveCamera LookDirection="0,0,-1" Position="0,0,5"
       NearPlaneDistance="3" FarPlaneDistance="100" />
    </Viewport3D.Camera>

   <ModelVisual3D>
     <ModelVisual3D.Content>
       <DirectionalLight Color="LightYellow" Direction="0,0,-1" />
     </ModelVisual3D.Content>
   </ModelVisual3D>
 </Viewport3D>

</Window>
```

Trojrozmerné objekty sú vytvorené z akejsi virtuálnej „drôtenej" kostry (mesh) na ktorú sa nanáša textúra. Kostra mesh je tvorená trojuholníkmi.

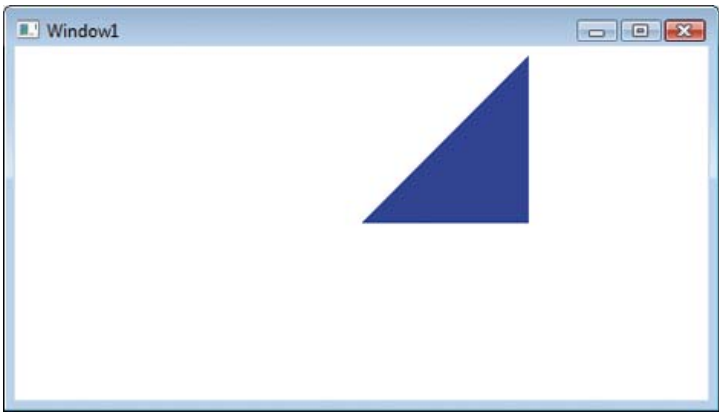
Do kódu pridáme referenciu:

```
using System.Windows.Media.Media3D;
```

V prvom príklade vykreslíme základný objekt WPF – trojuholník:

```
public partial class Window1 : Window
{
       public Window1()
       {
              this.InitializeComponent();

              Model3DGroup models = new Model3DGroup();
   MeshGeometry3D mesh = new MeshGeometry3D();

   mesh.Positions.Add(new Point3D(0, 0, 0));
   mesh.Positions.Add(new Point3D(1, 0, 0));
   mesh.Positions.Add(new Point3D(1, 1, 0));

   mesh.TriangleIndices.Add(0);
   mesh.TriangleIndices.Add(1);
   mesh.TriangleIndices.Add(2);

   mesh.TextureCoordinates.Add(new Point(1, 1));
   mesh.TextureCoordinates.Add(new Point(1, 0));
   mesh.TextureCoordinates.Add(new Point(0, 0));

   models.Children.Add(new GeometryModel3D(mesh, new DiffuseMaterial(Brushes Blue)));
            ModelVisual3D visual = new ModelVisual3D();
   visual.Content = models;
   priestor3D.Children.Add(visual);
  }
}
```

Najskôr definujeme body, ktoré tvoria trojuholník, Body sa indexujú od nuly. Poradie bodov je dané parametrom TriangleIndicies. Parameter TextureCoodrinates určuje pomocou množiny bodov ako má byť na „drôtenú kostru" mesh vyrenderovaná textúra.



Vykreslenie trojuholníka

V nadpise kapitoly sme sľúbili trojrozmernú grafiku a zatiaľ sme vykreslili len jeden trojuholník. Aj keď je trojuholník základným pilierom 3D grafiky vo WPF, predsa len sám o sebe je to len dvojrozmerný objekt, aj keď môže byť rôzne situovaný v 3D priestore. Najjednoduchším a zároveň najľahšie predstaviteľným trojrozmerným objektom je kocka. Cesta od trojuholníka ku kocke nie je zložitá. Dve pravouhlé trojuholníkové plochy spojené preponou tvoria štvorcovú plochu a štvorcová plocha je predsa stena kocky. Kocka má 6 stien, takže musíme vykresliť 12 trojuholníkov. Aby sme si to zjednodušili, vytvoríme procedúru na vykreslenie štvorcovej steny.
V prostredí Visual Studio 2008 vytvoríme nový WPF projekt, napríklad s názvom kocka. Scénu v súbore Windows1.xaml zatiaľ pre zaujímavosť ponecháme rovnakú ako v predchádzajúcom príklade. Uvidíme, že takéto umiestnenie kamery v smere osi z sa nie vždy hodí.

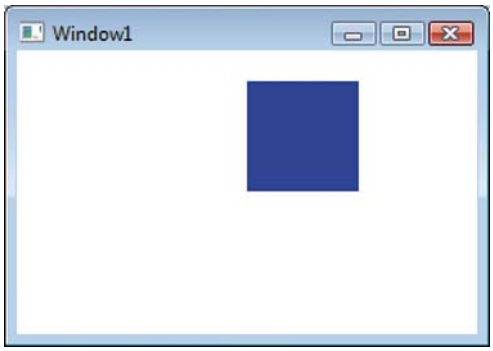
```
  <Viewport3D Name="priestor3D">
    <Viewport3D.Camera>
     <PerspectiveCamera LookDirection="0,0,-1" Position="0,0,5"
       NearPlaneDistance="3" FarPlaneDistance="100" />
    </Viewport3D.Camera>

   <ModelVisual3D>
     <ModelVisual3D.Content>
       <DirectionalLight Color="LightYellow" Direction="0,0,-1" />
     </ModelVisual3D.Content>
   </ModelVisual3D>
 </Viewport3D>
```

Najskôr vytvoríme procedúru na vykreslenie štvorca z dvoch trojuholníkov:

```
public GeometryModel3D Stvorec(Point3D A, Point3D B, Point3D C, Point3D D, Brush b)
{
     MeshGeometry3D mesh = new MeshGeometry3D();
     mesh.Positions.Add(A); mesh.Positions.Add(B);
     mesh.Positions.Add(C); mesh.Positions.Add(D);

     mesh.TriangleIndices.Add(0);
     mesh.TriangleIndices.Add(1);
     mesh.TriangleIndices.Add(2);

     mesh.TriangleIndices.Add(0);
     mesh.TriangleIndices.Add(2);
     mesh.TriangleIndices.Add(3);

    return new GeometryModel3D(mesh,new DiffuseMaterial(b));
 }
```

Doplníme aplikáciu testovacím kódom pre vykreslenie jedného štvorca:

```
public Window1()
{
   InitializeComponent();
   Model3DGroup models = new Model3DGroup();
   Model3D stvorec1 = Stvorec(
   new Point3D(0, 0, 0), new Point3D(1, 0, 0),
   new Point3D(1, 1, 0), new Point3D(0, 1, 0),
   Brushes.Blue);

   models.Children.Add(stvorec1);
   ModelVisual3D visual = new ModelVisual3D();
   visual.Content = models;
   priestor3D.Children.Add(visual);
}
```

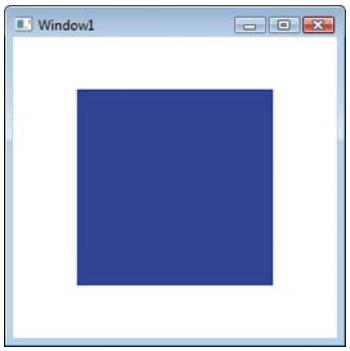


Vykreslenie štvorca pomocou dvoch trojuholníkov

Teraz už nie je ťažké nadefinovať jednotlivé plochy kocky:

```
public Window1()
{
   InitializeComponent();
   Point3D p0 = new Point3D(-1, -1, -1);
   Point3D p1 = new Point3D(1, -1, -1);
   Point3D p2 = new Point3D(1, -1, 1);
   Point3D p3 = new Point3D(-1, -1, 1);
   Point3D p4 = new Point3D(-1, 1, -1);
   Point3D p5 = new Point3D(1, 1, -1);
   Point3D p6 = new Point3D(1, 1, 1);
   Point3D p7 = new Point3D(-1, 1, 1);

   Model3DGroup models = new Model3DGroup();

   models.Children.Add(Stvorec(p3, p2, p6, p7, Brushes.Blue)); // predna
   models.Children.Add(Stvorec(p2, p1, p5, p6, Brushes.Blue)); // prava
   models.Children.Add(Stvorec(p1, p0, p4, p5, Brushes.Blue)); // zadna
   models.Children.Add(Stvorec(p0, p3, p7, p4, Brushes.Blue)); // leva
   models.Children.Add(Stvorec(p7, p6, p5, p4, Brushes.Blue)); // horna
   models.Children.Add(Stvorec(p2, p3, p0, p1, Brushes.Blue)); // dolna

   ModelVisual3D visual = new ModelVisual3D();
   visual.Content = models;
   priestor3D.Children.Add(visual);
}
```

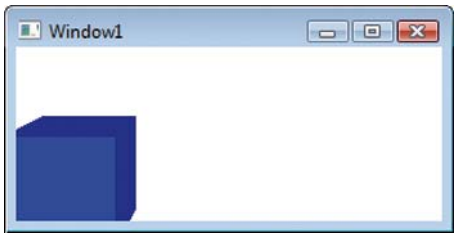
Po spustení však budeme sklamaní, zobrazí sa nám štvorec. Ak sa pozrieme na umiestnenie svetiel a kamery, je to jasné. Svetlo osvetľuje objekt naplocho v smere osi a aj kamera sa díva na kocku spredu v smere osi z, takže kocku zákonite vidíme ako štvorcovú plochu.



Vykreslenie kocky – pri nevhodnom osvetlení a umiestnení kamery sa zobrazí ako štvorec

Premiestnime teda svetlá a kameru a zobrazí sa nám kocka ako z učebnice:

```
<Window x:Class="Kocka.Window1"
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  Title="Window1" Height="300" Width="300">
  <Viewport3D Name="priestor3D">
    <Viewport3D.Camera>
      <PerspectiveCamera LookDirection="0,0,-1" Position="3,3,10"
       NearPlaneDistance="3" FarPlaneDistance="100" />
    </Viewport3D.Camera>

    <ModelVisual3D>
      <ModelVisual3D.Content>
        <Model3DGroup>
          <AmbientLight Color="#0000AA" />
          <DirectionalLight Color="White" Direction="0.2,0.5,-1" />
        </Model3DGroup>
      </ModelVisual3D.Content>
    </ModelVisual3D>
  </Viewport3D>
</Window>
```



Vykreslenie kocky pri vhodnom osvetlení a umiestnení kamery

Samozrejme trojrozmerná grafika vo WPF sa nebude používať na tvorbu hier, na tento účel sú k dispozícii špeciálne grafické engine, no určite nájde uplatnenie pri zobrazovaní „obchodnej grafiky", trojrozmerných schém rôznych procesov a podobne.

Windows Communication Foundation

Pod skratkou WCF sa skrýva jednotná technológia pre rýchle vytváranie bezpečných a spoľahlivých aplikácií využívajúcich architektúru orientovanú na služby. Táto vrstva kombinuje výhody dosiaľ existujúcich komunikačných technológií. WCF obsahuje unifikovaný programovací model pre prenos správ, založený na skriptovacom jazyku WSDL. Slúži na programovanie webových služieb. Na úrovni vrstvy WCF sa spravujú informácie o službách a operáciách ktoré služba vykonáva, vrátane odosielaných a prijímaných údajov. Aplikácie a služby sú poprepájané pomocou prepojovacích bodov „Service Endpoint", ktoré obsahujú adresu klienta a informáciu o spôsobe komunikácie medzi službou a klientom.

Windows Workflow Foundation

Táto vrstva prostredia .NET Framework obsahuje programový model, engine a nástroje pre vytváranie diagramov procesov (workflow) a riadenie procesov v aplikáciách. Väčšinu procesov v aplikáciách je možné popísať stavovým alebo sekvenčným diagramom. Určite si z výučby programovania pamätáte rôzne typy grafického znázornenia algoritmov. V prípade WF sú však tieto diagramy uložené v počítači a na základe informácií v nich zahrnutých je možné generovať vykonateľné bloky kódu aplikácií. Je potrebné si uvedomiť, že každý blok kódu má inú pravdepodobnosť zmeny. Ako príklad uvedieme kód pre faktoriál, ktorý vychádza z matematickej teórie a je teda nemenný. Opačným prípadom je napríklad algoritmus pre výpočet dane z príjmu. Tento sa môže meniť aj každý rok v závislosti od aktuálne platnej legislatívy. Proces transformácie od definície procesu ku kódu je jednosmerný, nevratný a neopakovateľný. Preto udržať v súlade definíciu procesu a jeho kód nie je až taká triviálna záležitosť. Vrstva Workflow umožňuje pohodlnú abstrakciu pre popis problémov reálneho sveta. Uľahčuje najmä riešenie technologických problémov, kedy procesy by mali byť transparentné a musia si uchovávať stav a vyžadujú flexibilnú kontrolu toku.

CardSpace

Technológia CardSpace pre správu digitálnych identít zjednodušuje a vylepšuje bezpečnosť autentifikácie v prostredí internetových aplikácií. Vývojári môžu tento spôsob autentifikácie a autorizácie jednoducho zahrnúť do svojich aplikácií. Identity sú uchované ako „kartičky" obsahujúce digitálne podpísaný XML dokument. Môže ich podpisovať sám používateľ. Pod pojmom digitálna identita osoby rozumieme množinu oprávnení, ktorou príslušná osoba disponuje pre kontakt s rôznymi inštitúciami (úrady, banky, internetové obchody....), pričom kontakt prebieha prostredníctvom elektronických komunikačných prostriedkov, najčastejšie prostredníctvom siete Internet. Technológia CardSpace je založená na technológiách WS-Security, WS-Trust, WS-MetadataExchange a WS-SecurityPolicy.

ASP.NET Model View Controller

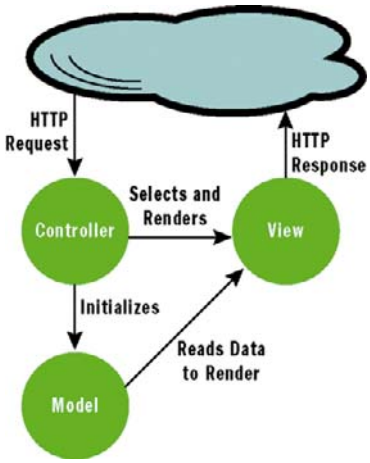
MVC (Model View Controller) je Framework pre ASP.NET. Umožňuje dôslednejšie oddelenie aplikačnej logiky od prezentačnej vrstvy. V dobe písania publikácie bol ASP.NET MVC vo verzii Preview 3. Najskôr je potrebné prevziať a spustiť inštalačný súbor AspNetMVCPreview3-setup.msi. Architektúra MVC pozostáva z troch komponentov, pričom výhodou je, že zmena ktoréhokoľvek z nich má len minimálny vplyv na ostatné, preto sa využíva hlavne pri rozsiahlych a zložitých systémoch.

Model – slúži na uchovávanie a aktualizáciu stavu aplikácie. Niekedy potrebujeme uchovávať stav len dočasne, napríklad počas komunikácie s používateľom, inokedy uchovávame stavové informácie mimo aplikáciu, najčastejšie v XML súbore, alebo v databáze. Model poskytuje prístup k dátam a umožňuje ich ukladanie a aktualizáciu. Model má aj metódy na aplikovanie pravidiel a podmienok nadviazaných na údaje. Napríklad ak naša aplikácia má na starosti evidenciu trpaslíkov u Snehulienky, úlohou modelu je zaistiť aby ich počet neprekročil sedem. Úlohou modelu je teda aj starať sa o konzistenciu a správnosť údajov.

View – ako vyplýva z názvu je to "pohľad" na údaje. Jeho hlavnou úlohou je prezentácia a formátovanie údajov z modelu. Pre jeden model môže existovať viacero pohľadov, napríklad iný „pohľad" na databázu zamestnancov má personalista, iný administrátor, iný vedúci oddelenia a podobne. Napriek tomu, že všetky pohľady používajú ten istý model, ich vzhľad a funkcionalita je iná.

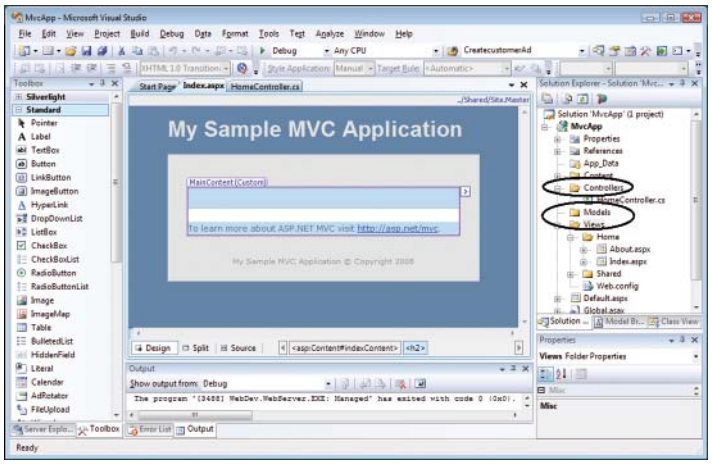
Controller – alebo po našom **„riadiaca jednotka"** tvorí akési prepojenie medzi modelom a **riadiacou jednotkou** – má na starosti riadenie aplikácie, spracováva vstupy od používateľov, vyžiada si údaje od modelu a pošle aktuálny View späť užívateľovi.

Vo väčšine prípadov MVC funguje podľa nasledujúcej postupnosti:
- Používateľ vykoná nejakú akciu prostredníctvom prvkov používateľského rozhrania (View).
- View odovzdá informácie o akcii Controllera.
- Controller na základe požiadavky z pohľadu View aktualizuje Model.
- Model požiadavku spracuje, aktualizuje dáta a odovzdá ich Controllera.
- Controller poskytne aktualizované údaje pohľadu View.
- Pohľad View zobrazí údaje.



Princíp fungovania MVC aplikácie

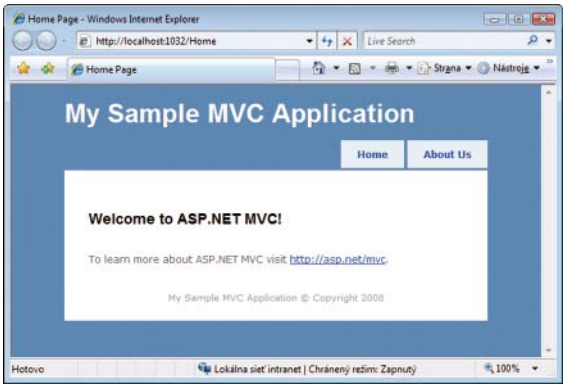
Vytvoríme nový projekt typu ASP.NET MVC Web Application. V aplikácii vzniknú tri zložky:
- Controllers
- Models
- Views



V MVC aplikácii sa vytvoria tri zložky Controllers, Models a Views

V zložke Controllers je súbor s jednoduchým ukážkovým kódom pre dva pohľady:

```
public class HomeController : Controller
{
  public ActionResult Index()
  {
    ViewData["Title"] = "Home Page";
    ViewData["Message"] = "Welcome to ASP.NET MVC!";
    return View();
  }

  public ActionResult About()
  {
    ViewData["Title"] = "About Page";
    return View();
  }
}
```



Ukážková MVC aplikácia

ASP.NET Silverlight controls

K výraznému zvýšeniu úrovne prezentačnej vrstvy a jej interaktivity môže prispieť aj technológia Silverlight z dielne spoločnosti Microsoft. Silverlight rozširuje prezentačnú úroveň prehľadávača webového obsahu o nové možnosti s využitím vektorovej grafiky. Spoločnosť Microsoft zároveň s finálnou verziou Silverlight 1.0 predstavila aj alfa verziu 1.1. Táto verzia bola následne premenovaná na **Silverlight 2.0**. Zatiaľ čo Silverlight 1.0 využíva ako programovací jazyk JavaScript, vo verzii 2.0 je možné v plnej miere využívať .NET jazyky. Predtým, v procese vývoja mala táto technológia označenie Windows Presentation Foundation/Everywhere, čo naznačuje, zámer spoločnosti Microsoft preniesť čo možno najviac čŕt nového prezentačného rozhrania WPF (Windows Presentation Foundation), ktoré je súčasťou platformy .NET Framework.

Aby sme mohli na svojom klientskom počítači prehliadať stránky využívajúce funkcionalitu technológie Silverlight 2 a robiť s ňou pokusy, je potrebné prevziať si doplnok prehľadávača na adrese (**http://www.microsoft.com/silverlight/resources/install.aspx?v=2.0**). Skript na stránke najskôr identifikuje, ktorú verziu máme aktuálne nainštalovanú a v prípade, ak je na webovej stránke k dispozícii už nová verzia, ponúkne nám ju na prevzatie a inštaláciu.

Pre aplikáciu Silverlight v ASP.NET aplikáciách sú k dispozícii dva ASP.NET serverové prvky:

MediaPlayer control – určený na prehrávanie multimediálnych súborov
Silverlight control – všeobecný Silverlight prvok

Ako príklad si môžete vyskúšať napríklad textový editor s možnosťami formátovania textu na **http://michaelsync.net/2008/05/04/silverlight-rich-text-editor-demo**

Technológii Silverlight 2 sa podrobne venuje samostatná publikácia z tejto edície.

LINQ

Dosiaľ bolo možné pristupovať k údajom v relačných databázach cez rozhranie ADO.NET a to buď v pripojenom, alebo odpojenom režime. Čoraz populárnejší je dátový formát XML, dôkazom čoho je skutočnosť, že všetky moderné databázové servery podporujú XML ako natívny dátový formát. S XML údajmi je možné na platforme .NET možné pracovať pomocou „klasického" namespace System.XML. Jeho metódy dokážu pracovať s XML údajmi buď prostredníctvom DOM (Document Object Model) alebo ich sekvenčne parsovať. Jednou z najvýznamnejších noviniek v novej verzii technologickej platformy Microsoft .NET Framework 3.5 je technológia LINQ (Language Integrated Query), ktorá slúži na jednoduchší a efektívnejší prístup k údajom, či už v objektovo-relačných databázach, alebo XML. Duchovným otcom tejto technológie nie je nikto iný ako známy „guru" objektovo orientovaného programovania a jazyka C# Anders Hejlsberg.

V porovnaní s dopytovacím jazykom SQL, dokáže LINQ v maximálnej miere zjednodušiť operácie nad údajmi v databázach. Integrácia LINQ je tak hlboká, že umožňuje vykonávať dopyty do databáz, ADO.NET datasetov, XML štruktúr, proste všade tam, kde sa pracuje s údajmi. LINQ je zakomponovaný priamo do programovacích jazykov C# 3.0 a Visual Basic 2008. Vývojárom tak stačí poznať programovací jazyk v ktorom pracujú a základné konštrukcie LINQ. Už nie sú odkázaní na pomerne zložitú a striktnú syntax jazyka SQL, hlavne pri vytváraní zložitejších vnorených dopytov. Navyše definovať objektovo – relačné väzby v jazyku SQL bolo niekedy pomerne ťažké. Podobne ako C#, je aj LINQ pomerne silno typovo závislý, takže množstvo potenciálnych chýb zachytí už kompilátor pri preklade zdrojového kódu. Prostredníctvom LINQ môžeme nielen realizovať dopyty, ale údaje aj požadovaným spôsobom modifikovať, teda pridávať nové záznamy, editovať už existujúce záznamy, prípadne nepotrebné záznamy vymazávať.

LINQ umožňuje pracovať s akýmikoľvek údajmi, pretože jeho architektúra umožňuje vytvárať jej implementácie pre rôzne dátové zdroje. Na rozdiel od ADO.NET, kde bolo možné vytvoriť provider pre prístup k údajom pre akýkoľvek databázový server, možnosti implementácie LINQ sú oveľa abstraktnejšie.
Expression Trees umožňuje implementovať LINQ dopyt pre akýkoľvek typ údajov.

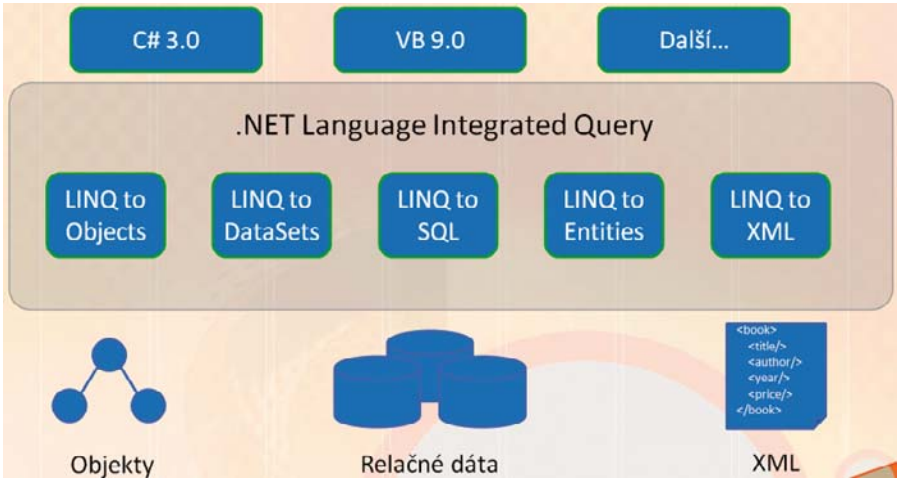
V .NET 3.5 je v namespace System.Core implementovaná statická trieda Enumerable rozširujúca interface **IEnumerable** o metódy na projekciu, selekciu, agregáciu a triedenie. Operácie nad IEnumerable sa vykonávajú prostredníctvom kódu preloženého do MSIL. Tento však nerobí operácie mimo aplikačnej domény. Ak chceme vykonávať takéto operácie, napríklad nad SQL databázou, webovou službou a podobne, je potrebné vytvoriť prostredníctvom rozhrania IQueryable požiadavku pre externý zdroj, na základe ktorej sa nám vráti výsledok. IQueryable obsahuje

Expression – objekt reprezentujúci operáciu, ktorú chceme vykonať nad externým zdrojom. Objekt je typu System.Linq.Expressions.Expression.

Provider – objekt na vytvorenie požiadavky pre externý zdroj. Provider je typu **IQueryProvider**.

Na najvyššej architektonickej úrovni je pridanie nových kľúčových slov do programovacích jazykov. Pod nimi sú moduly a komponenty pre sprostredkovanie LINQ technológie smerom k zdroju údajov.

- LINQ toObjects– dopytovanie v kolekciách údajov a rôznych typoch polí v operačnej pamäti,
- LINQ to XML – dopytovanie do XML (predtým XLinq),
- LINQ to SQL dopytovanie do databázy (predtým DLinq),
- LINQ to DataSets implementácia LINQ pre prácu s ADO.NET datasetmi.



Architektúra LINQ

LINQ to Objects umožňuje vytvárať dopyty nad objektmi z kolekcie IEnumerable.

Najzaujímavejším variantom je zrejme LINQ to SQL, pretože umožňuje vytváranie jednoduchých objektových dopytov nad databázou. Základom pre implementáciu LINQ to SQL je behové prostredie na riadenie relačných dát vo forme objektov bez straty podpory „query" jazyka. Aplikácia manipuluje s objektmi v managed alebo natívnom jazyku .NET (čo je pre objektovo orientované jazyky .NET prirodzenejšie), kým LINQ to SQL na pozadí má na starosti automatické „trekovanie" zmien v databáze. Prepojenie medzi objektmi aplikácie a reálnymi tabuľkami databázy je deklarované tzv. diagramom ORM (object relational mapping).

LINQ prináša možnosť dopytovania priamo integrovanú do .NET jazyka, pomocou množiny kľúčových slov. Ukážeme fragment kódu pre „klasický" prístup k údajom v databáze, ktorý využíva ADO.NET:

```
using(SqlConnection conn = new SqlConnection
 ("server=localhost;database=dbDokumenty;uid=sa;pwd=nbusr123");
 {
   conn.Open();
      SqlCommand cmd =new SqlCommand(@"SELECT Dokument FROM TabDoc
      WHERE ID_Dokumentu=@ID",conn);
      cmd.Parameters.Add("@ID",SqlDbType.Int);
   cmd.Parameters.Add(param);
      // …
   cmd.ExecuteReader()

 }
```

Ak si tento fragment kódu rozanalyzujeme, zistíme, že obsahuje databázovú logiku vo forme textových reťazcov. Samozrejme celý zdrojový kód v jayzku C# alebo Visual Basic je v okamihu písania, alebo vizuálnej tvorby textový reťazec, no následne sa kompiluje do medzikódu a prechádza ďalšími vrstvami architektúry .NET. Príkazy jazyka SQL však stále zostávajú v podobe textových reťazcov. Aj keď vhodným využitím možností ADO.NET dokážeme do značnej miery zabrániť „SQL Injection", teda situácii kedy hackeri vložia do reťazca SQL kódu iné príkazy, ktorými môžu napríklad poškodiť databázu, takéto príkazy nie je možné v etape kompilovania dôkladne skontrolovať, nakoľko stále zostávajú v textovej podobe a tak aj úspešne skompilovaný kód obsahujúci reťazce chybného SQL kódu nebude pracovať. Ak programujete databázové aplikácie určite si spomínate koľko vás natrápili preklepy v SQL príkazoch, napríklad v názvoch atribútov.

Naproti tomu ak by boli príkazy pre prácu s údajmi integrované do programovacieho jazyka, bolo by možné vykonať kontrolu vrátane typovej už pri preklade a hlavne by sa zabránilo preklepom.

Základný syntaktický predpis pre dopyt v jazyku LINQ je:

```
from itemName in srcExpr
join itemName in srcExpr on keyExpr equals keyExpr
       (into itemName)?
let itemName = selExpr
where predExpr
orderby (keyExpr (ascending | descending)?)*
select selExpr
group selExpr by keyExpr
into itemName query-body
```

Zdá sa to na prvý pohľad zložité, ale vôbec to tak nie je. K tomu, že sme o miere nutnosti osvojenia syntaxe LINQ hovorili veľmi benevolentne nás oprávňuje veľmi dobre vyriešený pomocník typu IntelliSense aj pre tento databázový dopytovací jazyk. Keď začneme písať príkaz, prípadne názov objektu, IntelliSense nám ponúkne možnosti pre doplnenie na kompletný syntaktický výraz.

Pre porovnanie, základný syntaktický predpis dopytu v SQL je:

```
SELECT [*] [zoznam_položiek_výstupnej_zostavy]
 FROM meno_tabuľky
 WHERE podmienka_výberu
 GROUP BY položky
 HAVING podmienka_agregácie
 ORDER BY zoznam_položiek [ASC] [DESC]
```

Je to vlastne syntaktický predpis jednoduchej anglickej vety v rozkazovacom spôsobe. Povinné sú len kľúčové slová SELECT a FROM. Ostatné sú voliteľné.

SELECT keďže ide o príkaz, použijeme ako prvé sloveso v rozkazovacom spôsobe, teda v neurčitku. Za kľúčovým slovom nasleduje predpis pre výber stĺpcov. Môžeme napríklad vybrať všetky stĺpce príkazom v tvare SELECT * FROM meno_tabuľky alebo ich napríklad vymenovať v požadovanom poradí SELECT stlpec1, stlpec2... FROM meno_tabuľky.

FROM pomocou tejto klauzule špecifikujeme, kde sa požadované informácie nachádzajú. Môže to byť jedna tabuľka, viacero tabuliek, pohľad...

WHERE podmienky nasledujúce za klauzulou WHERE nám presne špecifikujú jednak podmnožinu údajov, ktorú potrebujeme vybrať, jednak pomocou podmienky môžeme stanoviť predpis pre výber údajov z viacerých databázových tabuliek.

GROUP BY určuje zoskupenie údajov z databázy podľa určitých pravidiel.

HAVING – podmienka za klauzulou HAVING určuje predpis pre výber agregovaných záznamov.

ORDER BY určuje spôsob usporiadania údajov.

LINQ výraz v porovnaní s SQL začína trochu netypicky operátorom **FROM**, ktorý špecifikuje dátový zdroj a vracia objekt typu IEnumerable alebo IQueryable. Za operátorom FROM je premenná in ukazujúca na kolekciu. Iterujúcu premennú takto deklarujeme, aby sme s ňou v ďalšej časti dopytu mohli pracovať a odkazovať sa na ňu. Iterujúca premenná je vlastne určitou analógiou aliasu databázovej tabuľky v jazyku SQL. Toto poradie, teda najskôr FROM a potom SELECT je logické, pretože najskôr zadefinujeme tabuľky a iterujúce premenné a až potom definujeme projekciu. Poradie príkazov v LINQ je dané logikou dopytovania, poradie v SQL jazyku je dané skôr tým, aby príkaz vypadal ako anglická veta v rozkazovacom spôsobe.

Operátor SELECT definuje projekciu, teda výber atribútov. Okrem atribútov môže obsahovať napríklad nový anonymný objekt, alebo iterujúcu premennú. Ukážeme jednoduchý príklad:

```
var vizitky =
  from z in zakaznici
  where z.Mesto == "Bratislava"
  select new { z.Meno, z.email };
```

Z klauzule FROM si prekladač zistí akého typu je daný objekt, v našom prípade „z". Všimnite si, že na začiatku nie je potrebné určovať typ lokálnej premennej. Kompilátor si typ premennej „vizitky" odvodí sám. Zástupné kľúčové slovo VAR nám podstatne zjednoduší tvorbu kódu.

Lambda expression

Lambda výrazy sa používajú na filtrovanie, alebo projekciu údajov. Umožňujú napísať výraz, ktorý použijeme napríklad na filtrovanie údajov priamo do metódy. Lambda výrazy vznikli prirodzeným vývojom anonymných metód, ktoré poznáme z programovacieho jazyka C# 2.0. Všimnite si konštrukcie „`z=> z.Mesto == "Bratislava"`". Pretransformovaný na výraz „`z=> z.Mesto == "Bratislava"`" označujeme ako Lambda výraz. Lambda výraz je určitým druhom anonymnej metódy. Je to výraz skladajúci sa z ľavej a pravej strany oddelenej operátorom =>. Ľavá strana obsahuje jeden alebo viac parametrov. Ak ľavá strana obsahuje len jeden parameter, stačí napísať jeho názov. Viac parametrov oddeľujeme čiarkami a uzatvoríme do zátvoriek. Typ parametra sa automaticky odvodí z typu delegáta, prípadne môžeme typ parametra určiť aj explicitne. Pravá strana obsahuje telo anonymnej metódy. Spravidla ide o nejaký výraz, ktorý vracia hodnotu. Pri zložitejších funkciách používame blokovú konvenciu. Lambda výraz je možné preložiť ako kód, ako údaje, alebo ako expression tree, teda akýsi výrazový strom. Tieto výrazy je potom možné automaticky pretransformovať do iného jazyka, v našom prípade do SQL.

Jednoduché pokusy s technológiou LINQ

Nakoľko technológia LINQ je nová a poskytuje pri správnom zvládnutí netušené možnosti, nebudeme ľutovať úsilia zoznámiť sa s touto technológiou podrobnejšie. Vytvoríme niekoľko jednoduchých cvičných projektov typu „Console Application".

Príklad pre LINQ to Objects

Pre demonštráciu jednoduchosti LINQ si skúste s nami urobiť jednoduchý cvičný príklad. Vytvorte v prostredí Visual Studio projekt typu konzolová aplikácia. My sme vytvorili projekt v programovacom jazyku C#. Námetom príkladu bude z vymenovanej množiny čísel od 0 do 9 vypísať čísla, ktoré sa využívajú pre známkovanie na školách, teda čísla od 1 do 5. Nakoľko budeme pracovať s vymenovanou množinou objektov, `(IEnumerable)` využijeme v tomto príklade, bez toho aby sme si to nejako zvlášť uvedomovali, rozhranie „LINQ to Objects".

Aby sme ukázali aj zoradenie výsledkov podľa predpísaného poradia, zoradíme známky zostupne, teda od najväčšej (a vlastne v zmysle aplikačnej logiky najhoršej známky) po najmenšiu, teda najlepšiu známku:

```
static void Main(string[] args)
{
   IEnumerable<int> cisla = new List<int>() {0, 1, 2, 3, 4, 5, 6, 7, 8, 9 };
   IEnumerable<int> znamky = from nn in cisla
                where nn > 0 && nn <=5
                orderby nn descending
                select nn;
   foreach (int nn in znamky)
   {
     Console.WriteLine(nn);
   }
   Console.ReadLine();
}
```

Podobný jednoduchý príklad môžeme vymyslieť aj pre hľadanie v množine textových reťazcov:

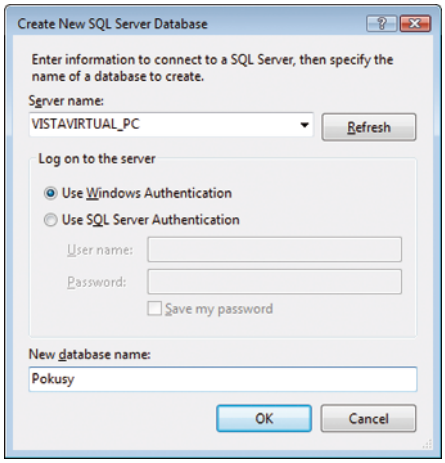
```
static void Main(string[] args)
{
   List<string> prezidenti = new List<string> {"George W. Bush", "Bill Clinton",
"George Bush", "Ronald Regan"};
   IEnumerable<string> prez = from pp in prezidenti
             where pp.Contains("Bush")
             select pp;

   foreach (string pp in prez)
   {
     Console.WriteLine(pp);
   }
   Console.ReadLine();
}
```

Príklad pre LINQ to SQL

Najskôr si pripravíme cvičné údaje s ktorými budeme pracovať. Mohli by sme využiť napríklad cvičnú databázu AdventureWorksLT2008. Aby sme pracovali naozaj od základov, vytvoríme na platforme servera SQL Server (2005, alebo 2008) novú databázu s názvom napríklad „Pokusy". Novú databázu môžeme vytvoriť prostredníctvom aplikácie SQL Server Management Studio buď prostredníctvom kontextového menu na zložke „Databases" alebo cez konzolu príkazom:

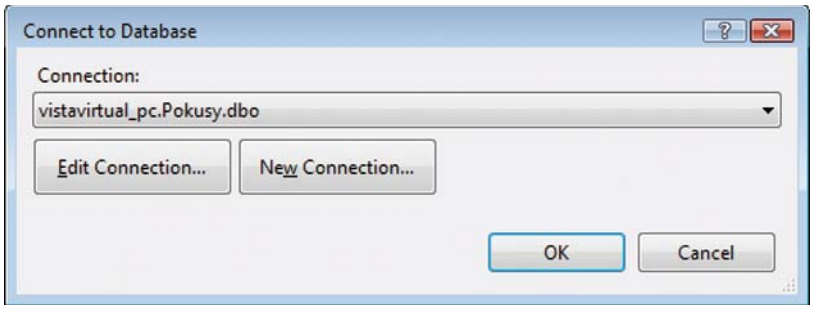
```
CREATE DATABASE Pokusy
```

Novú databázu môžeme vytvoriť aj priamo vo prostredí Visual Studio 2008. V okne Server Explorer aktivujeme položku „Create New SQL server database". Vyberieme inštanciu SQL servera a zadáme názov databázy.



Výber inštancie SQL servera pre vytvorenie novej databázy

Databáza je zatiaľ prázdna. Potrebujeme v nej vytvoriť databázovú tabuľku a naplniť ju údajmi. Znovu môžeme využiť SQL Server Management Studio alebo priamo prostredie Visual Studio 2008 prostredníctvom menu „Data" – „T-SQL Editor". Editor pripojíme k zdroju údajov, ktorým je naša novo vytvorená databáza Pokusy.



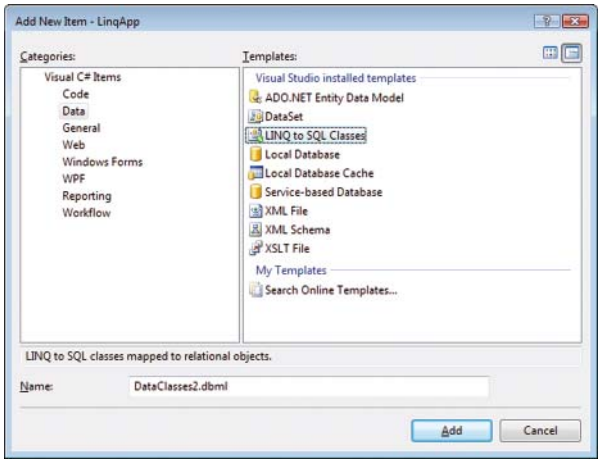
Pripojenie konzoly T-SQL editora k zdroju údajov

Teraz môžeme spustiť príkazy pre vytvorenie databázovej tabuľky a jej naplnenie údajmi. Vytvoríme jednoduchú tabuľku so zoznamom amerických prezidentov, ktorá obsahuje poradové číslo, meno a roky ohraničujúce obdobie „pontifikátu". Pre zjednodušenie zadáme len prezidentov od roku 1900. Názov tabuľky „potus" je akronym od „President of the United States".

```
CREATE TABLE potus
(
  id INT PRIMARY KEY,
  meno VARCHAR(40),
  od INT,
  do INT
);

INSERT INTO potus VALUES (25, 'William McKinley', 1897,1901);
INSERT INTO potus VALUES (26, 'Theodore Roosevelt', 1901,1909);
INSERT INTO potus VALUES (27, 'William Howard Taft', 1909,1913);
INSERT INTO potus VALUES (28, 'Woodrow Wilson', 1913,1921);
INSERT INTO potus VALUES (29, 'Warren Gamaliel Harding', 1921,1923);
INSERT INTO potus VALUES (30, 'Calvin Coolidge', 1923,1929);
INSERT INTO potus VALUES (31, 'Herbert Hoover', 1929,1933);
INSERT INTO potus VALUES (32, 'Franklin Delano Roosevelt', 1933,1945);
INSERT INTO potus VALUES (33, 'Harry S, Truman',1945,1953);
INSERT INTO potus VALUES (34, 'Dwight David Eisenhower', 1953,1961);
INSERT INTO potus VALUES (35, 'John Fitzgerald Kennedy', 1961,1963);
INSERT INTO potus VALUES (36, 'Lyndon Baines Johnson', 1963,1969);
INSERT INTO potus VALUES (37, 'Richard Milhous Nixon', 1969,1974);
INSERT INTO potus VALUES (38, 'Gerald Rudolph Ford', 1974,1977);
INSERT INTO potus VALUES (39, 'Jimmy Carter', 1977,1981);
INSERT INTO potus VALUES (40, 'Ronald Reagan', 1981,1989);
INSERT INTO potus VALUES (41, 'George Bush', 1989,1993);
INSERT INTO potus VALUES (42, 'Bill Clinton', 1993,2001);
INSERT INTO potus VALUES (43, 'George Walker Bush jr', 2001, NULL)
```

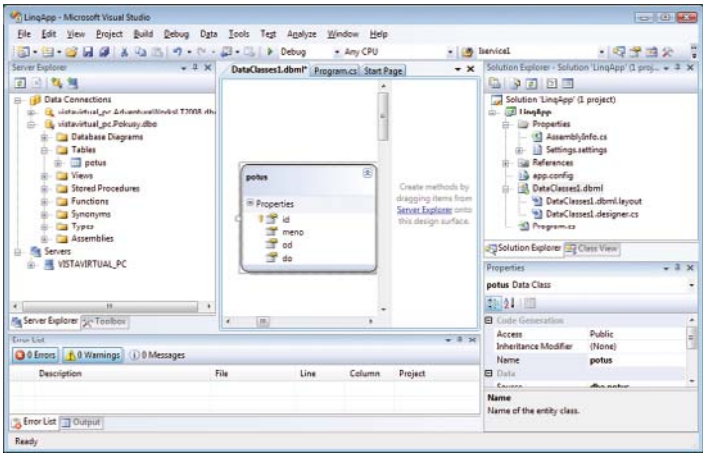
V tejto fáze máme pripravený zdroj údajov na ďalšie pokusy.

Pomocou kontextového menu „Add New Item" pridáme do projektu triedu LINQ to SQL.



Pridanie triedy LINQ to SQL do projektu

Metódou „dragg and drop" presunieme na návrhovú plochu „Object Relational Designer "symbol názvu databázovej tabuľky Potus.



Návrh databázového diagramu

Dopyt zapuzdríime do tela implementácie metódy. Najskôr vytvoríme inštanciu objektu DataContext, ktorý obsahuje informácie o pripojení na databázu ako zdroj údajov. Dopyty sú implementované nad typom IQueryable<T>. Pre prístup k údajom slúžia triedy odvodené od objektu DataContext. Pre prácu s údajmi môžeme využívať uložené procedúry. Objektová reprezentácia údajov umožňuje využívanie všetkých výhod objektov vrátane dedičnosti. Napríklad ak je objekt „zamestnanec" odvodený od objektu „osoba", je možné u zamestnanca využívať nielen jeho vlastné atribúty, ale aj všetky atribúty zdedené od objektu osoba
V našom prípade sme v klauzule SELECT vytvorili jeden anonymný dátový typ, položku Meno, ktorá má taký istý dátový typ ako atribút meno z databázovej tabuľky POTUS. Všimnite si, že ak názov atribútu koliduje s niektorým kľúčovým slovom jazyka C#, v našom prípade atribút „do", automaticky sa pred neho priradí prefix „@". Nakoniec v cykle typu foreach prechádzame jednotlivé záznamy.

```
static void Main(string[] args)
{
  DataClasses1DataContext dc = new DataClasses1DataContext();
  var dopyt = from p in dc.potus
        where p.meno.Contains("Bush")
        select new { Meno = p.meno, p.od, p.@do };

  foreach (var p in dopyt)
  {
    Console.WriteLine(p.Meno+": "+p.od+" - "+p.@do);
  }
  Console.ReadLine();
}
```

Test aplikácie

Príklad pre Lambda výrazy

Dopyty sú implementované nad typom IQueryable<T>. Pre prístup k údajom slúžia triedy odvodené od objektu DataContext. Táto trieda predstavuje zdroj údajov. Pre prácu s údajmi môžeme využívať uložené procedúry. Objektová reprezentácia údajov umožňuje využívanie všetkých výhod objektov vrátane dedičnosti. Napríklad ak je objekt „zamestnanec" odvodený od objektu „osoba", je možné u zamestnanca využívať nielen jeho vlastné atribúty, ale aj všetky atribúty zdedené od objektu osoba.

Do projektu pridáme novú triedu (kontextové menu „Add" – „New Item" – „Class"), ktorú nazveme prezidenti a definujeme pre ňu tri atribúty:

```
class Prezidenti
{
  public int Id { get; set; }
  public string Meno { get; set; }
  public int Od { get; set; }
}
```

Už v definícii triedy postrehnete významnú novinku C# 3.0 – – automatické vlastnosti (Automatic properties). Sú to vlastnosti, ktoré len zapisujú do premennej, prípadne čítajú jej obsah bez ďalšej funkcionality.

V klasickom C# bez automatických vlastností by sme definíciu triedy zapísali takto:

```
class Customer
{
  private int Id;
  public int Id {get { return id; }  set { id = value; }}

  private string meno;
  public string Meno {get { return meno; } set { meno = value; }}

  private int od;
  public ind Od {get { return od; }  set { Od = value; }}
}
```

Zoznam prezidentov naplníme niekoľkými menami priamo v kóde a vytvoríme Lambda výraz. Pomocou kľúčového slova var deklarujeme premennú, ktorej typ sa dá zistiť podľa výrazu na pravej strane, za znamienkom „=". Operátor =>, používaný v Lambda výrazoch môžeme čítať v zmysle „ide do". Pred šípkou sa deklarujú premenné a na druhej strane výraz, ktorého hodnota sa vráti.

```
  var Bushovci = p.FindAll(pp => pp.Meno.Contains("Bush"));
```

Kompletný kód miniaplikácie bude:

```
static void Main(string[] args)
{
  var p = new List<Prezidenti>
  {
    new Prezidenti() { Id=40, Meno="Ronald Reagan", Od=1981 },
    new Prezidenti() { Id=41, Meno="George Bush", Od=1989 },
    new Prezidenti() { Id=42, Meno="Bill Clinton", Od=1993 },
    new Prezidenti() { Id=43, Meno="George Walker Bush jr", Od=2001 }

  };

  var Bushovci = p.FindAll(pp => pp.Meno.Contains("Bush"));

  foreach (var pp in Bushovci)
  {
    Console.WriteLine(pp.Meno + ": " + pp.Od);
  }

  Console.ReadLine();
}
```

ADO.NET Entity Framework

ADO.NET Entity Framework (v ďalšom texte aj ADO.NET EF) je súbor technológií, ktoré zjednodušujú vývoj dátovo flexibilných aplikácií. Hlavnou myšlienkou ADO.NET EF je abstrakcia prístupu k údajom. Takže pri návrhu prístupu k údajom nemusíme prihliadať na štruktúru údajov v o fyzickej databáze, ktoré sú uložené v relačne zviazaných databázových tabuľkách a majú atribúty, obmedzenia a podobne. Aby sme vás motivovali k podrobnejšiemu zoznámeniu sa s technológiou ADO.NET Entity Framework, ukážeme dva obrázky. Na prvom je prístup k údajom pomocou ADO.NET 2.0 a na druhom prístup k údajom cez ADO.NET EF. Všimnite si, že na prvom obrázku sa dopytovanie vykonáva pomocou jazyka SQL a na druhom obrázku pomocou LINQ.

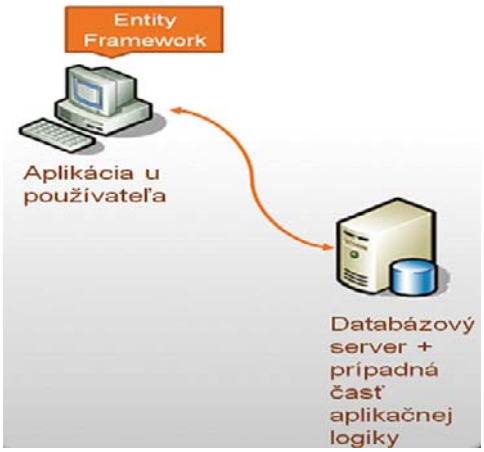
```
// Create a connection with the AdventureWorks connection string.
using (SqlConnection conn = new SqlConnection(
    ConfigurationManager.ConnectionStrings["AdventureWorks"].ConnectionString))
{
    conn.Open();

    // Create a command to join Customer and CustomerContactInfo tables.
    SqlCommand command = conn.CreateCommand();
    command.CommandText = @"
        SELECT  cust.FirstName, cust.LastName, contact.EmailAddress
        FROM    [dbo].[Customer] AS cust
        JOIN    [dbo].[CustomerContactInfo] AS contact
        ON      cust.CustomerID = contact.CustomerID
        WHERE   cust.MiddleName IS NULL";

    // Execute the command and obtain a data reader.
    using (SqlDataReader reader = command.ExecuteReader(
        CommandBehavior.SequentialAccess))
    {
        while (reader.Read())
        {
            // Write a response line using values from the reader.
            Response.Write(String.Format("<p>{0}\t{1}\t{2}</p>",
                reader["FirstName"],
                reader["LastName"],
                reader["EmailAddress"]));
        }
    }
}
```

Prístup k údajom pomocou ADO.NET

```
using (AdventureWorksModel model = new AdventureWorksModel())
{
    var query = from c in model.Customer
                where c.MiddleName == null
                select new {
                    FirstName = c.FirstName,
                    LastName = c.LastName,
                    EmailAddress = c.EmailAddress };

    foreach (var c in query)
    {
        Response.Write(String.Format("<p>{0}\t{1}\t{2}</p>",
            c.FirstName,
            c.LastName,
            c.EmailAddress));
    }
}
```

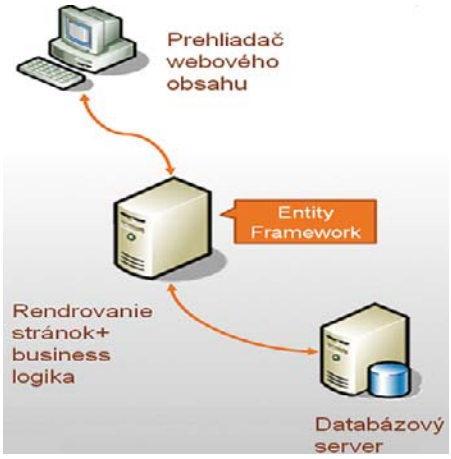
Prístup k údajom cez ADO.NET Entity Framework

Aplikácie využívajúce ADO.NET Entity Framework môžu využívať dvojvrstvovú, alebo trojvrstvovú architektúru. Pri dvojvrstvovej architektúre, klientska aplikácia pristupuje k databázovému serveru priamo, pričom v databázovom serveri môže byť implementovaná aj časť aplikačnej logiky, napríklad vo forme uložených procedúr a podobne.



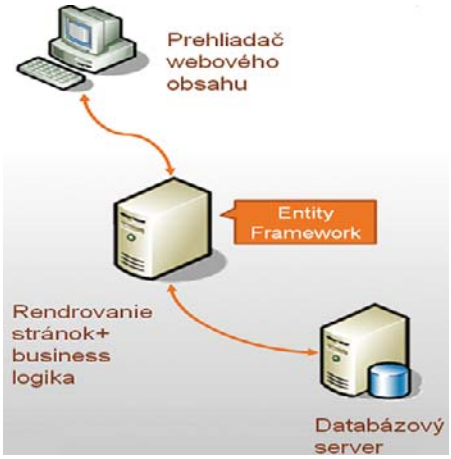
Technológia ADO.NET Entity Framework aplikovaná na dvojvrstvovej architektúre

Pri trojvrstvovej architektúre je medzi klientskym počítacom a databázovým serverom vložený ešte aplikačný server, na ktorom beží business logika a kde sa v prípade internetových a intranetových aplikácií renderujú aj stránky zobrazované v klientovom prehľadávači. Práve táto biznis logika pristupuje k údajom v databáze prostredníctvom technológie ADO.NET Entity Framework.



Technológia ADO.NET Entity Framework aplikovaná pre webové aplikácie

Komunikácia s middlewarovou vrstvou môže prebiehať priamo, alebo prostredníctvom služieb, či už webových alebo realizovaných pomocou vrstvy Windows Communication Foundation.



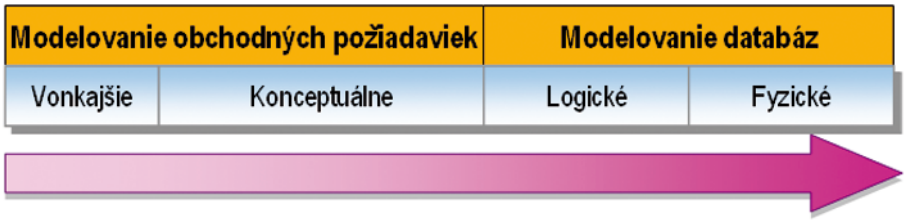
Technológia ADO.NET Entity Framework aplikovaná na trojvrstvovej architektúre

Modelovanie

Abstrakcia prístupu k údajom je dosiahnutá využitím konceptuálneho modelu Entity Data Model (EDM), takže musíme na úvod state zaradiť niekoľko viet venovaných teórii modelovania. Podstatou činnosti každého informačného systému, ktorý využíva databázy je transformácia informácií z vonkajšieho sveta na dáta. Tento proces môžeme na rôznych úrovniach modelovať. Model informačného systému by mal byť:

- **zrozumiteľný** – mal by vyjadrovať fakty a pravidlá v jednoduchom jazyku,
- **vhodný** – mal by podchytiť čo najviac pravidiel vyplývajúcich z obchodnej logiky navrhovanej aplikácie,
- **spoľahlivý** – mal by umožniť overiť si pravidlá v prirodzenom jazyku na jednoduchých príkladoch,
- **stály** – je potrebné minimalizovať dosah zmien,
- **vykonateľný** – model musí byť jednak realizovateľný na technickej úrovni pomocou dostupných hardvérových a softvérových prostriedkov a taktiež musí byť vhodný pre prevádzku.

Modelovanie databázovej aplikácie pozostáva z dvoch etáp:
- modelovanie obchodných požiadaviek, ktoré môže byť vonkajšie (analýza vonkajších vzťahov) a konceptuálne (analýza obchodnej logiky),
- modelovanie databáz, ktoré môže byť na logickej a fyzickej úrovni.



Modelovanie databázovej aplikácie

Pri modelovaní databáz prechádzame od konceptuálneho modelu, ktorý je úplne hardvérovo a softvérovo nezávislý k logickému a fyzickému modelu. Tento proces má nielen svoje hardvérové a softvérové špecifiká, ale musíme pritom akceptovať aj reálny čas, nemôžeme predsa ukladať údaje do databázy pomalšie, ako vznikajú, a podobne ako pri konceptuálnom modeli aj požiadavky obchodnej logiky. Ak napríklad manažéri požadujú súhrny vo forme určitých typov grafov, bude rozumné navrhnúť databázové tabuľky tak, aby slúžili ako podklady pre požadované typy grafov.

Modelovanie databáz

Základnú informáciu o tom, akým spôsobom sú dáta v databáze uložené obsahuje databázová schéma. Príkladom takejto schémy pre uloženie informácií o zákazníkoch môže byť schéma:

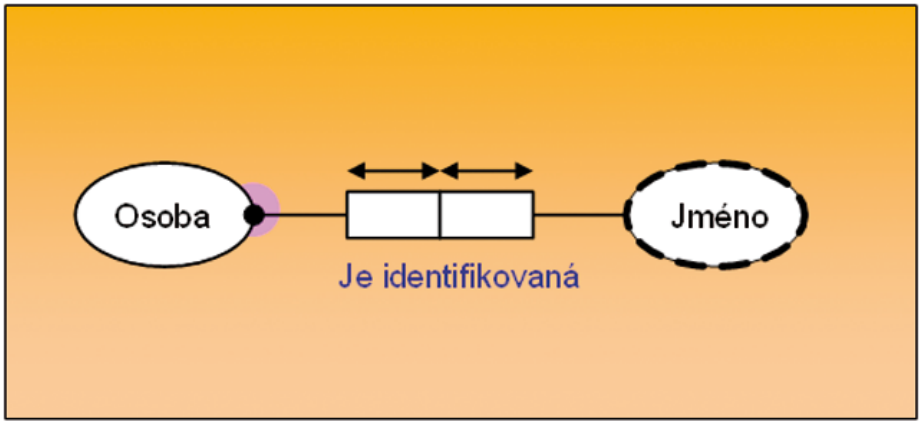
zakaznici(id_zak, firma, kontakt_jmeno, adresa, mesto, psc, stat, telefon)

pričom **zakaznici** je meno schémy a **id_zak, firma, kontakt_jmeno, adresa, mesto, psc, stat, telefon** sú mená položiek (atribútov údajov, ktoré do databázy chceme ukladať).

Pri návrhu databázovej časti projektu riešime spravidla tri okruhy problémov:
- štruktúru dát,
- manipuláciu s dátami,
- integritné obmedzenia.

Konceptuálny model

Niekedy procesu systémovej analýzy a systémového návrhu hovoríme aj konceptuálny model. Takýto model popisuje údaje v databáze úplne nezávisle od ich fyzického uloženia a pri jeho návrhu (tento proces sa nazýva konceptuálne modelovanie) sa zameriame na aplikačnú logiku ale z pohľadu človeka, nie z pohľadu neskôr použitých hardvérových a softvérových technológií. Pri tvorbe konceptuálnych modelov spravidla vnímame objekty reálneho sveta, vzťahy medzi nimi a funkcie, pomocou ktorých sa tieto vzťahy realizujú, takže konceptuálne modelovanie je v podstate objektovo orientovaný proces. Okrem objektov spravidla vstupuje do hry aj ich hierarchické usporiadanie, napríklad dedičnosť, to znamená, že objekty môžu byť vytvorené na základe iných objektov, pričom zdedia časť vlastností a podobne.

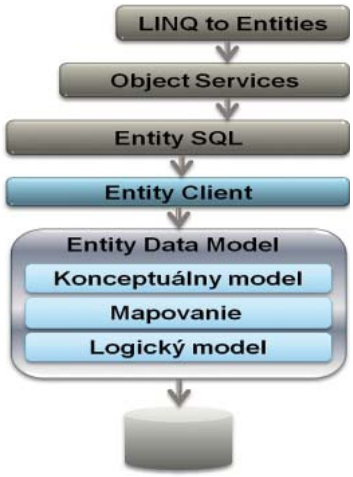


Miesto konceptuálneho a logického modelovania v schéme projektu

Namiesto klasickej databázovej schémy využívanej v ADO.NET 2.0 využívame pri ADO.NET EF konceptuálnu schému.

Entity Data Model (EDM)

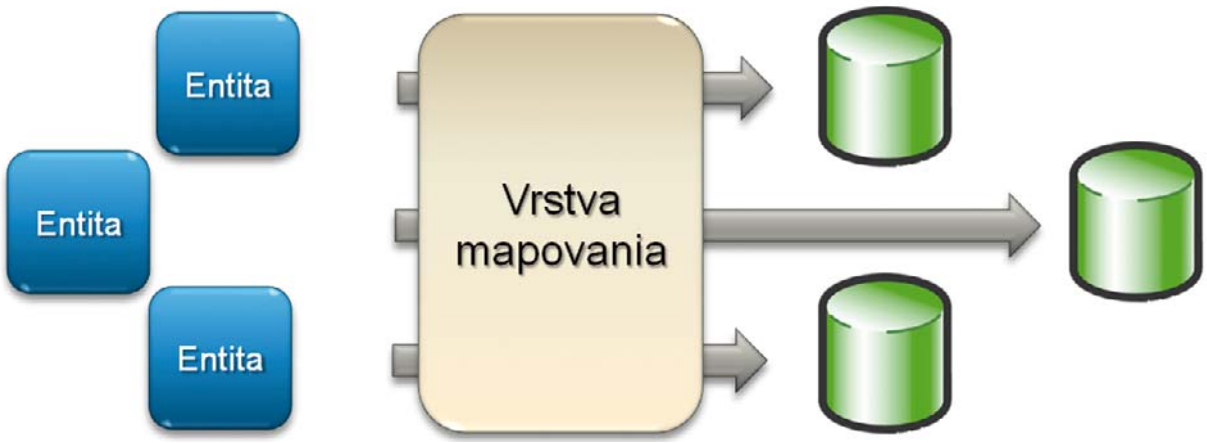
Abstrahovanie prístupu k údajom sa dosahuje pomocou Entity Data Model (EDM). Vývojár pracuje s konceptuálnym entitami reprezentujúcimi údaje, pričom tieto entity sú mapované na databázovú schému. Mapovanie je explicitné a dá sa meniť, takže nie je potrebné upravovať aplikáciu pri každej zmene medzi aplikáciou a dátovým úložiskom, prípadne pri migrácii na iný databázový server (SQL Server, Oracle, IBM DB2...



Principiálna schéma prístupu cez ADO.NET Entity Framework

Nad úložiskom údajov je definovaný Entity Data Model. Entita reprezentuje rôzne pohľady na údaje nezávisle od ich fyzickej štruktúry. Logická časť obsahuje informácie o štruktúre fyzického úložiska údajov. Vrstva mapovania mapuje fyzické objekty do entít, ktoré sú na konceptuálnej úrovni. Vo vyšších vrstvách je rozšírenie SQL tak, aby bolo možné zadávať požiadavky voči entitám. Tento pridáva EDM sémantiku do SQL jazyka. Rozšírenie sa týka hlavne prístupu k hierarchickej štruktúre entít. LINQ aj Entity SQL sú prekladané do SQL jazyka a optimalizované pre príslušný databázový server. Vrstva Object Services obsahuje kolekcie tried, pre prístup k entitám. Tieto triedy je možné využívať v aplikáciách.

Entity Data Model podporuje dátové typy Binary, Byte, Boolean, SByte, Int16, Int32, Int64, Decimal, Double, Single, DateTime, String a Guid. Nakoľko nie všetky tieto dátové typy sú natívne podporované na každom databázovom serveri, je možné mapovať dátové typy modelu Entity Data Model na databázové typy. Súvisí to s nezávislosťou konceptuálneho modelu od fyzickej štruktúry. Teoreticky by bolo napríklad možné namapovať dátové typy modelu Entity Data Model na fiktívny databázový server, ktorý podporuje len jeden číselný a jeden reťazcový dátový typ. Na úrovni EDM nie sú podporované používateľsky definované dátové typy.



Mapovanie entít na fyzické údaje

Vrstva mapovania prepája entity konceptuálneho modelu na štruktúru úložiska údajov. Model mapovania je implementovaný konceptuálnou schémou (CSDL), schémou úložiska (SSDL) a mapovacou schémou. (MSL). Mapovanie prevádza objekty z databázy, napríklad funkcie, vrátane používateľsky definovaných funkcií typu „Table-Value" a uložené procedúry na ImportFunction, tabuľky a pohľady na EntitySet, reštrikcie na AssociationSet a podobne. Entity môžu byť štruktúrované a podporujú dedičnosť.
Súčasťou modelu mapovania sú aj klientske pohľady, ktoré sú podobné databázovým pohľadom.

Architektúra

Na schéme architektúry vidíme, že ADO.NET Entity Framework spočíva na pilieri ADO.NET data provider pre jednotlivé databázy a iné typy dátových úložísk. Nad touto vrstvou je EntityClient Data Provider. Táto vrstva vracia Data Reader. Zatiaľ čo pri ADO.NET 2.0 sme dostávali výsledky v pomerne voľnej typovej forme, výsledky vrátené cez ADO.NET EF sú silno typové. Na najvyššej vrstve sú objektové služby.

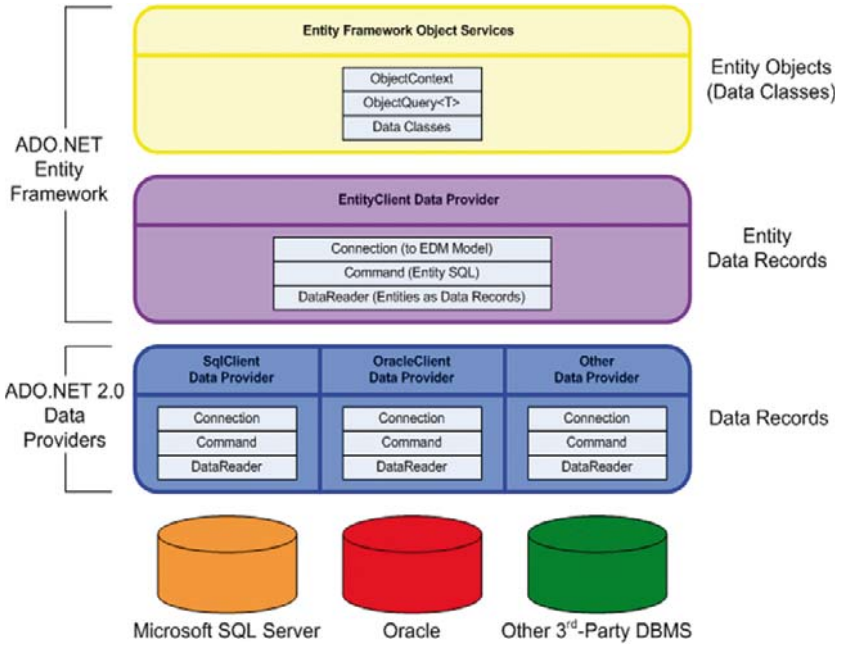
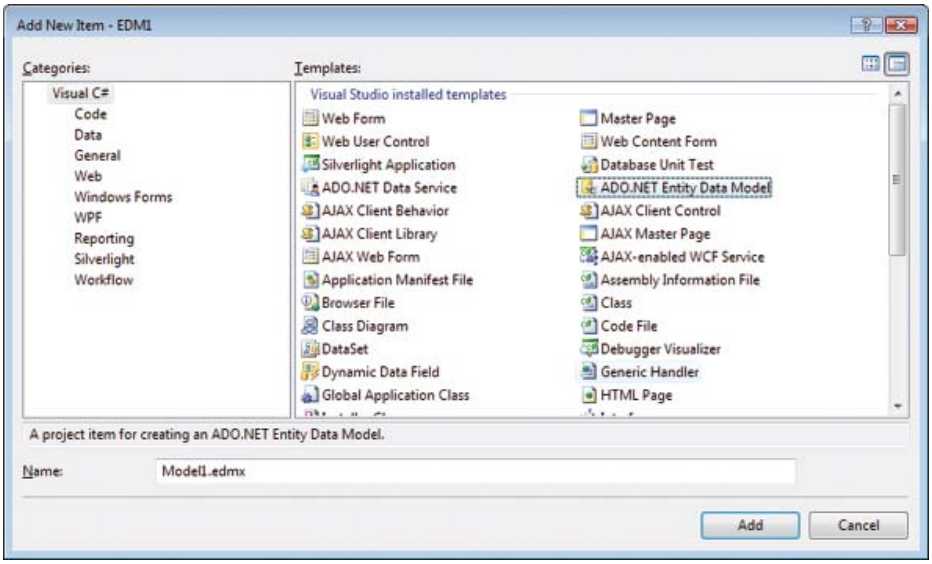


Schéma architektúry technológie ADO.NET Entity Framework

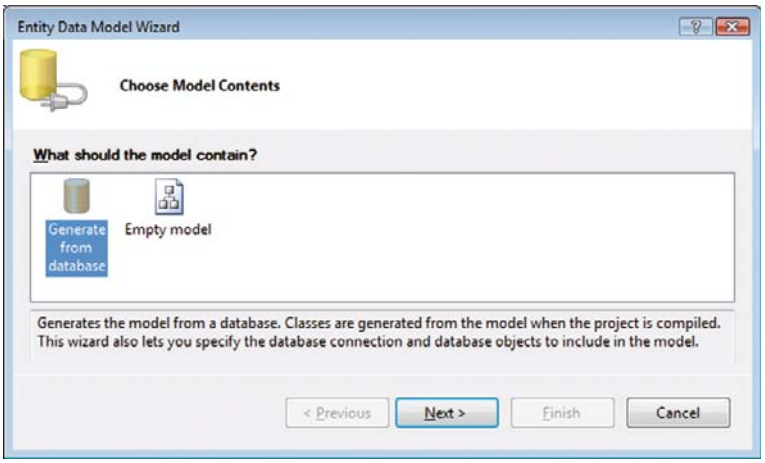
Príklad databázovej aplikácie v ADO.NET EF

Pre ilustráciu možností práce s modelom Entity Data Model a mapovania údajov vytvorme cvičnú aplikáciu, napríklad webovú. Pracovať budeme s údajmi v databáze AdventureWorks2008LT. Štruktúra a návod na inštaláciu tejto databázy je v prílohe.



Pridanie modelu ADO.NET Entity Data Model

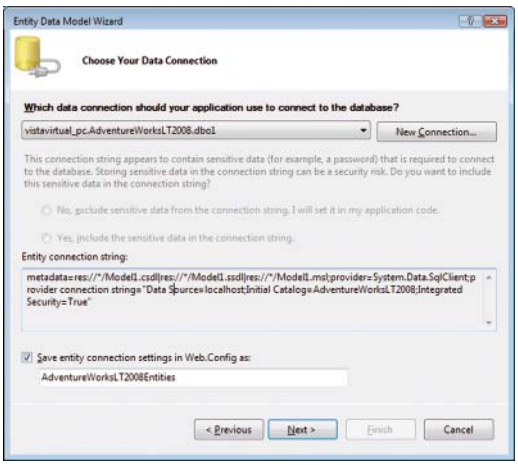
Model môžeme generovať na základe štruktúry databázy priamo, alebo začneme model vytvárať od základov, teda od prázdneho modelu. V našom príklade budeme generovať model podľa databázy.



Výber prázdneho modelu alebo vytvorenie modelu podľa databázy

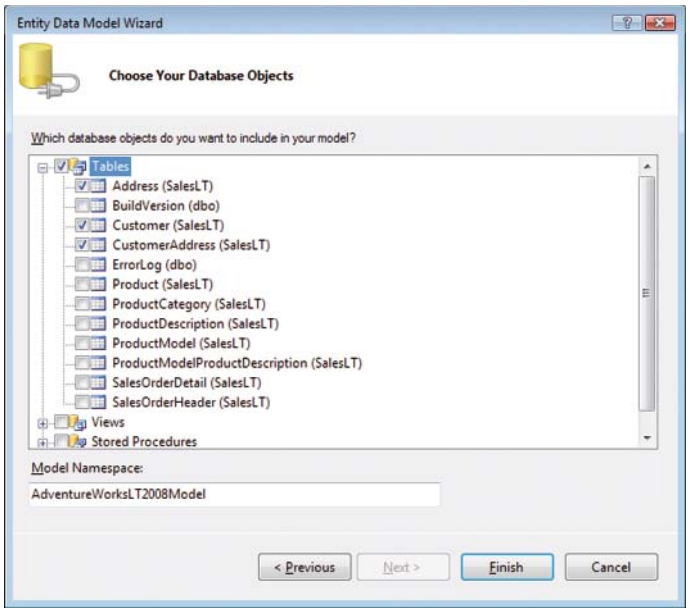
Po výbere servera a databázy sa automaticky vygeneruje Entity connection string:

```
metadata=res://*/Model1.csdl|res://*/Model1.ssdl|res://*/Model1.msl;
provider=System.Data.SqlClient;
provider connection string="Data Source=localhost;
Initial Catalog=AdventureWorksLT2008;
Integrated Security=True"
```



Pripojenie sa k databáze

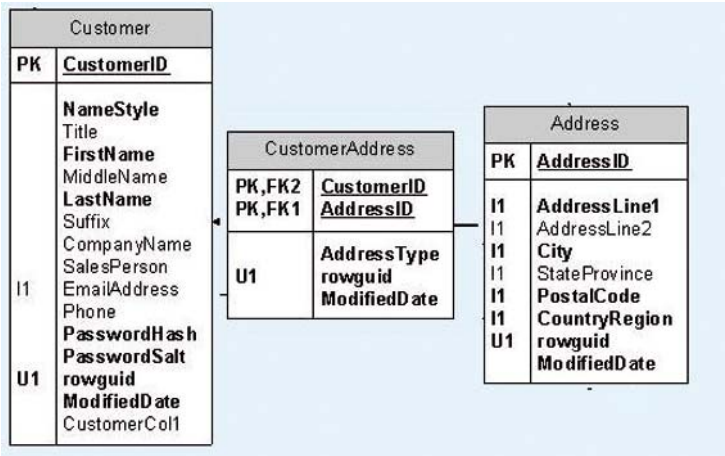
Entity connection string sa uloží do súboru web.config. V ďalšom kroku sprievodcu sa načíta a následne zobrazí fyzická štruktúra databázy pozostávajúca z tabuliek, pohľadov a uložených procedúr. Entity Data Model môžeme vytvoriť na základe kompletnej štruktúry databázy, prípadne vyberieme, ktoré tabuľky, pohľady a uložené procedúry chceme do modelu zahrnúť.



Zobrazenie fyzickej štruktúry databázy pre výber objektov

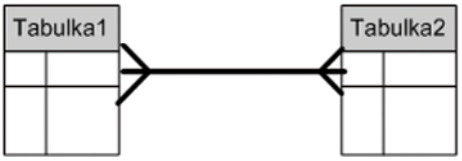
V aplikácii budeme pracovať len s databázovou tabuľko Customer, ale aby si prišli na svoje aj čitatelia, ktorí chcú pracovať s relačne zviazanými tabuľkami pridáme do modelu tri relačne zviazané tabuľky:

– Customer
– CustomerAddress
– Address

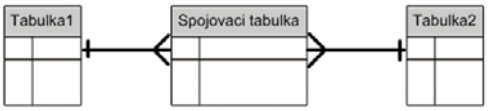


Tabuľky databázy

Aby sme rozumeli väzbám medzi týmito tabuľkami ukážeme princíp dekompozície dvoch tabuliek na relačnú väzbu pomocou spojovacej tabuľky. Väčšina databázových systémov totiž nedokáže priamo pracovať so vzťahmi typu N:M, v praxi sa používa dekompozícia, to znamená, že tento vzťah sa implementuje pomocou spojovacej tabuľky. To znamená, že vzťah N:M môžeme rozložiť na dva vzťahy typu N:1.

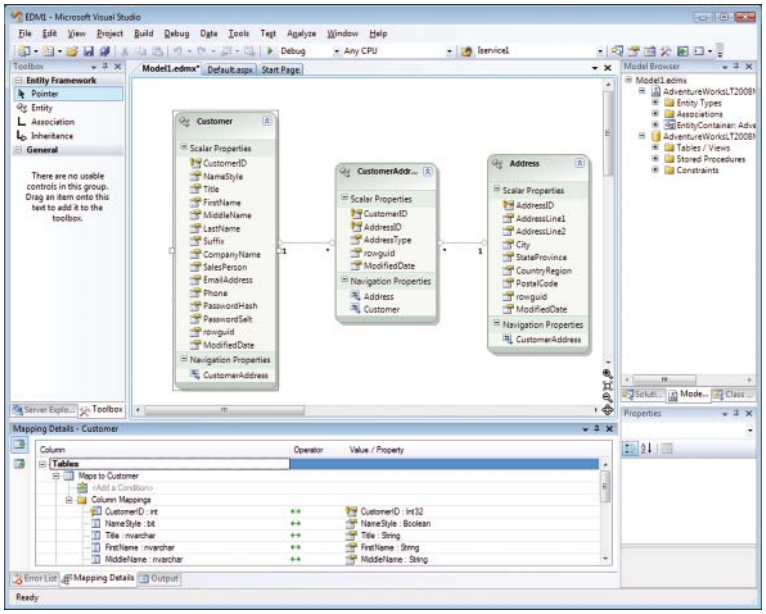


Vzťah „viaceré k viacerým"



Rozloženie vzťahu „viaceré k viacerým" pomocou spojovacej tabuľky

Pre zaujímavosť pridajme do modelu aj jednu uloženú procedúru, napríklad GetSalesOrderStatus, aby sme videli v akej forme sa do modelu ukladá. Názov namespace modelu ponecháme taký, ako bol automaticky vygenerovaný, v našom prípade AdventureWorksLT2008Model. Po potvrdení tlačidlom „Finish" bude model na základe nášho zadania vygenerovaný. Do modelu budú zahrnuté aj relačné väzby medzi tabuľkami definované pomocou cudzích kľúčov.



Príklad konceptuálneho modelu Entity Data Model

Všimnite si rozmiestnenie okien nástroja Model Designer. V ľavej častij e ponuka prvkov pre návrh, v pravej časti je okno Model Browser. Mapovanie je definované v okne Mapping Details v spodnej časti obrazovky.
V tejto etape môžeme s modelom pracovať, napríklad môžeme v modeli zmeniť názvy atribútov na slovenské a podobne.

Model je uložený vo forme XML dokumentu, pre ilustráciu jeho štruktúry a obsahu ukážeme mapovanie jednej tabuľky:

```
<EntitySetMapping Name="CustomerAddress">
  <EntityTypeMapping TypeName="IsTypeOf(AdventureWorksLT2008Model.CustomerAddress)">
    <MappingFragment StoreEntitySet="CustomerAddress">
     <ScalarProperty Name="CustomerID" ColumnName="CustomerID" />
     <ScalarProperty Name="AddressID" ColumnName="AddressID" />
     <ScalarProperty Name="AddressType" ColumnName="AddressType" />
     <ScalarProperty Name="rowguid" ColumnName="rowguid" />
     <ScalarProperty Name="ModifiedDate" ColumnName="ModifiedDate" />
    </MappingFragment>
   </EntityTypeMapping>
 </EntitySetMapping>
```

Uložená procedúra je v modeli implementovaná nasledovne:

```
<Function Name="ufnGetSalesOrderStatusText" ReturnType="nvarchar"
 Aggregate="false" BuiltIn="false" NiladicFunction="false" IsComposable="true"
 ParameterTypeSemantics="AllowImplicitConversion" Schema="dbo">
 <Parameter Name="Status" Type="tinyint" Mode="In" />
</Function>
```

Objektový model je reprezentovaný triedami v súbore Model1Designer.cs. Ukážeme fragment kódu ako je reprezentovaný objekt CustomerAddress:

```
public partial class CustomerAddress : global::System.Data.Objects.DataClasses.
EntityObject
{
public static CustomerAddress CreateCustomerAddress(int customerID, int addressID,
string addressType, global::System.Guid rowguid, global::System.DateTime modifiedDate)
 {
   CustomerAddress customerAddress = new CustomerAddress();
   customerAddress.CustomerID = customerID;
   customerAddress.AddressID = addressID;
   customerAddress.AddressType = addressType;
   customerAddress.rowguid = rowguid;
   customerAddress.ModifiedDate = modifiedDate;
   return customerAddress;
 }


 public int CustomerID
 {
   get { return this._CustomerID;}
   set
   {
     this.OnCustomerIDChanging(value);
     this.ReportPropertyChanging("CustomerID");
     this._CustomerID = global::System.Data.Objects.DataClasses.StructuralObject.
              SetValidValue(value);
        this.ReportPropertyChanged("CustomerID");
        this.OnCustomerIDChanged();
      }
    }
...
```

Tieto triedy môžeme využiť v aplikácii, pretože sa neviažu na fyzické štruktúry ale pristupujú k nim prostredníctvom mapovacej vrstvy.

ADO.NET Entity Provider

ADO.NET Entity Provider sa nepripája k databáze, ale ako zdroj údajov využíva Entity Data Model. Podobne ako ostatné „klasické" ADO.NET data providery obsahuje štruktúru tried Connection, Command, Parameter, v tomto prípade sú to konkrétne triedy EntityConnection, EntityCommand, EntityParameter...
ADO.NET Entity Provider je teda nezávislý na poskytovateľovi dát na pozadí. Pre definovanie parametrov pripojenia slúži textový reťazec „Connection string", ktorý obsahuje údaje o pripojení. Na rozdiel od klasických ADO.NET data providerov, kde trieda Command využíva príkaz pre výber údajov v jazyku SQL, ADO.NET Entity Provider akceptuje iba dopyty v syntaxi Entity SQL. Vracia triedu DbDataReaders, ktorá je odvodená od triedy DataReaders. Prostredníctvom ADO.NET Entity Provider nie je možné vykonávať príkazy CREATE, UPDATE ani DELETE. Tieto príkazy sú implementované na vrstve Object Services:

```
using (EntityConnection conn = new EntityConnection("Name=
AdventureWorksLT2008Entities"))
{
  conn.Open();
  using (EntityCommand cmd = new EntityCommand( // dopyt v jazyku Entity SQL
    "SELECT VALUE count(AdventureWorksLT2008Model.Customer.LastName)" +
    "from AdventureWorks2008LTModel.Customer "+
    "WHERE Customer.CompanyName = 'Progressive Sports'", conn))
  {
  using (EntityDataReader rdr = cmd.ExecuteReader(CommandBehavior.SequentialAccess))
  {
    // spracovanie údajov z Data readeru, spravidla sa realizuje v cykle
     TextBox1.Text = rdr[0].ToString();
  }
}
```

Dopytovací jazyk Entity SQL

Pre dopytovanie v jazyku Entity SQL využívame v princípe štandardný príkaz jazyka SQL SELECT, no namiesto k štruktúram databázy sa dopytujeme na entity. Do jazyka SQL sú pridané kľúčové slová:

Kľúčové slovo	Význam
VALUE	Hodnota entity
REF	Referencia na inštanciu entity
DEREF	Vrátenie hodnoty
NAVIGATE	Navigácia cez vzťahy medzi entitami
IS (NOT) TYPE OF	Zistenie, či výraz je daného typu alebo podtypu
TREAT	Spracovanie bázového typu ako odvodeného typu
MULTISET	Konštrukcia multisetu z kolekcie hodnôt
ROW	Konštrukcia anonymného riadkového typu
ANYELEMENT	Vrátenie ľubovoľného elementu multihodnotovej kolekcie
FLATTEN	Konverzia kolekcie kolekcií na „plochú" kolekciu

V prvom pokuse s technológiou Entity Framework umiestnime na plochu formulára len jeden TextBox do ktorého vypíšeme nájdenú hodnotu a tlačidlo, ktorým aktivujeme vyhľadávanie.

```
protected void Button1_Click(object sender, EventArgs e)
{
   EntityConnection conn = nnew EntityConnection(ConfigurationManager.
ConnectionStrings["AdventureWorksLT2008Entities"].ConnectionString);
   conn.Open();

   EntityCommand command
   = new EntityCommand("SELECT VALUE count(Customer.LastName) from
AdventureWorks2008LTEntities.Customer WHERE Customer.CompanyName = 'Progressive
Sports'", conn);

   DbDataReader reader = command.ExecuteReader(CommandBehavior.SequentialAccess);
   while (reader.Read())
   {
      TextBox1.Text = reader[0].ToString();
   }
}
```

Object Services

Objektový model entít sa v hierarchii objektov nachádza nad ADO.NET Entity Provider. Okrem možnosti vykonávať takzvané CUD (CREATE, UPDATE, DELETE) príkazov môžeme sledovať vykonané zmeny pomocou „Change Tracking". Každú zmenu vykonanú na úrovni EDM je potom možné zapísať aj do fyzického databázového úložiska. CUD príkazy sa k vytváraným entitám generujú automaticky.

Požiadavky sa v Object Services realizujú cez generickú triedu ObjectQuery. Ako jazyk požiadaviek je použitý opäť Entity SQL, Projekcia sa vykonáva prostredníctvom ObjectQuery<DbDataRecord>. Výsledky sú potom vrátené v ObjectResult<T>:

```
using (ObjectContext ctx = new ObjectContext("Name= AW2008"))
{
 ObjectQuery<Customers>
  q = ctx.CreateQuery<Customers>
    ("select value c from AW2008.Customers as c");
  ObjectResult<Customers> result = q.Execute(MergeOption.NoTracking);
}
```

Všimnite si, atribút MergeOption pre nastavenia správania objektu ObjectREsult. Môžeme nastaviť hodnotu parametra AppendOnly, NoTracking, PreserveChanges alebo OverwriteChanges.

Ak požiadavku metódou Execute nespustíme, spustí sa implicitne napríklad vtedy ak začneme prehliadať údaje prostredníctvom cyklu foreach:

```
using (ObjectContext ctx = new ObjectContext("Name= AW2008"))
{
 ObjectQuery<Customers>
  q = ctx.CreateQuery<Customers>("select value c from AW2008.Customers as c");

  foreach (Customers c in q)
   Console.WriteLine(c.CustomerID);
}
```

Zatiaľ vidíme len samé výhody, ak by sme sa zamysleli nad potenciálnymi problémami, tak príkazy v dopytovacom jazyku Entity SQL podobne ako v jazyku SQL sú let textové reťazce, ktoré nemôžeme verifikovať v okamihu prekladu. A to je jeden z dôvodov prečo na scénu vstupuje LINQ to Entities.

LINQ to Entities

LINQ umožňuje dopytovanie voči Object Services. Je zabezpečené natívnym poskytovateľom s podporu syntaktickej konvencie LINQ. LINQ to Entities podporujú podmnožinu Entity SQL, pričom môžeme použiť väčšinu operátorov štandardných LINQ konštrukcií. K údajom prisupujjeme pomocou objektov z namespace.

```
using System.Data.Entity;
using System.Data.Objects;
```

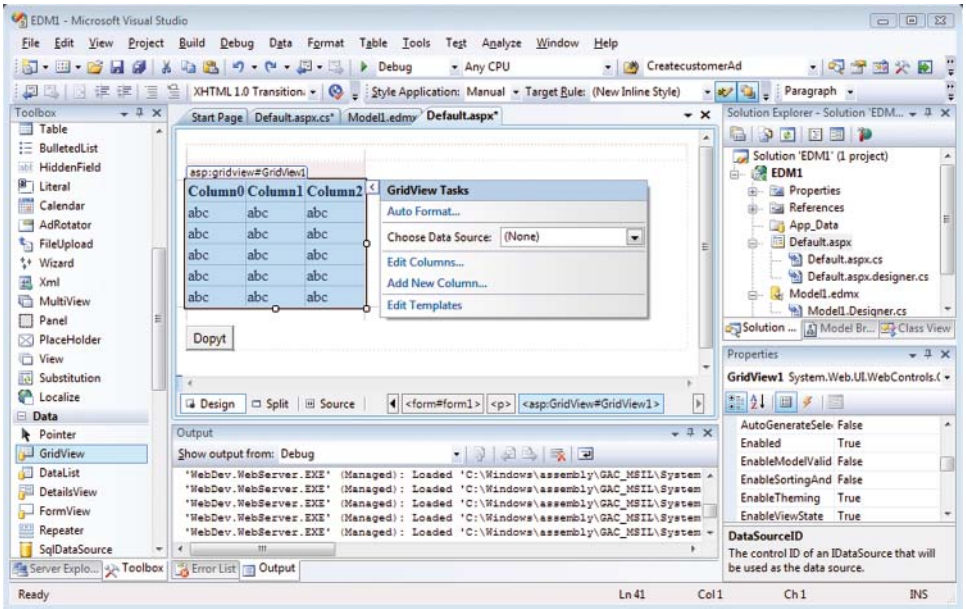
V tabuľke je stručný prehľad LINQ syntaxe:

Projekcia	**Select** <expr>
Filter	**Where** <expr>, Distinct
Definovanie rozsahu	**Any**(<expr>), **All**(<expr>)
Spájanie	<expr> **Join** <expr> **On** <expr> **Equals** <expr>
Zoskupovanie	**Group By** <expr>, <expr> **Into** <expr>, <expr> **Group Join** <decl> **On** <expr> **Equals** <expr> **Into** <expr>
Agregácie	**Count**([<expr>]), **Sum**(<expr>), **Min**(<expr>), **Max**(<expr>), **Avg**(<expr>)
Partitioning	**Skip [While]** <expr>, **Take [While]** <expr>
Množiny	**Union, Intersect, Except**
Utriedenie	**Order By** <expr>, <expr> **[Ascending | Descending]**

Možno viac napovie krátka ukážka kódu. Ak chceme vypísať údaje zo všetkých záznamov, použijeme kód v štýle:

```
AW2008 aw = new AW2008 ();
var query = from c in aw.Customers select c;
foreach (var cu in query)
   Console.WriteLine("{0} {1}", cu.LastName, cu.FirstName);
```

Pre ilustráciu možností vytvoríme nad modelom, ktorý sme prezentovali v jednej s predchádzajúcich statí zobrazenie údajov s využitím LINQ To Entities. Na novú aspx stránku, ktorú pridáme do projektu pridáme prvok GridView a tlačidlo dopyt.



Používateľské rozhranie aplikácie pre testovanie LINQ To Entities

Zaujímať nás bude ID, meno a priezvisko zákazníka a jeho mailová adresa, pričom výber zákazníkov obmedzíme na spoločnosť Progressive Sports:

```
protected void Button1_Click(object sender, EventArgs e)
{
  AdventureWorksLT2008Entities aw = new AdventureWorksLT2008Entities();
  var lq = from Customer in aw.Customer
       where Customer.CompanyName == "Progressive Sports"
          select new {Customer.CustomerID, Customer.FirstName ,
          Customer.LastName, Customer.EmailAddress};
  GridView1.DataSource = lq;
  GridView1.DataBind();
}
```

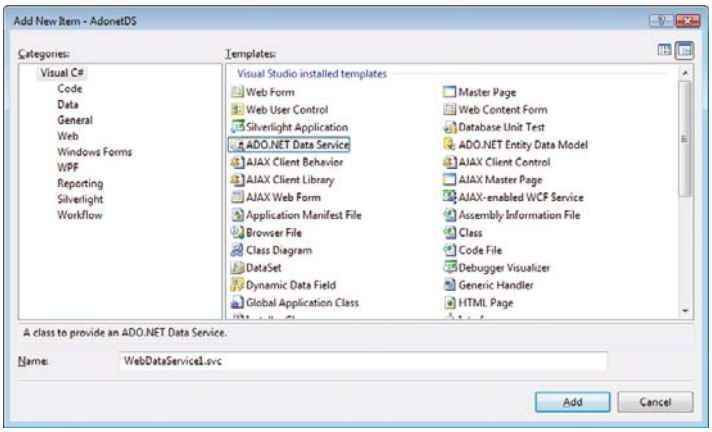
Zobrazenie údajov vybraných pomocou LINQ To Entities

ADO.NET Entity Framework umožňuje uskutočňovať štrukturálne zmeny v databáze bez nutnosti zmien v aplikáciách, ktoré k týmto údajom pristupujú. Môžeme nielen, pridávať, odoberať a meniť atribúty, ale aj zlúčiť tabuľky, prípadne dekompozíciou rozdeliť zložitejšiu tabuľku na viac tabuliek, aby vyhovovali prvej, alebo druhej normálnej forme. Dokonca môžeme migrovať na iný databázový server.

ADO.NET Data Services

ADO.NET Data Services, je nadstavba nad ADO.NET Entity Framework. Je to služba fungujúca cez http protokol, ktorá umožňuje vykonávať operácie s údajmi. Môžeme vykonávať množinu úkonov CRUD (CREATE, READ, UPDATE, DELETE). Podobne ako ADO.NET EF aj ADO.NET Data Services pracujú s konceptuálnym modelom, ktorý je abstrahovaný od fyzického úložiska údajov. Keďže ADO.NET Data fungujú cez http protokol vracajú údaje vo formáte XML alebo JSON.

Aby sme ukázali fungovanie ADO.NET Data Services, vytvoríme v aplikácii Visual Studio 2008 nový projekt typu ASP .NET Web Application. Cez Add New Item pridáme do projektu „ADO.NET Data Service".



Pridanie ADO.NET Data Service

Do projektu pribudne súbor s príponou svc. Teraz rovnako ako v príklade v stati ADO.NET Entity Framework pridáme do projektu ADO.NET Entity Data Model a vyberieme príslušné databázové tabuľky. V našom prípade sa uspokojíme s tabuľkou „Customer" z databázy AdventureWorks2008 LT.

V súbore svc.cs máme kostru kódu služby:

```
using System;
using System.Collections.Generic;
using System.Data.Services;
using System.Linq;
using System.ServiceModel.Web;
using System.Web;

namespace AdonetDS
{
  public class WebDataService1 : DataService< /* TODO: put your data source class name
here */ >
  {
    // This method is called only once to initialize service-wide policies.
    public static void InitializeService(IDataServiceConfiguration config)
    {
      // TODO: set rules to indicate which entity sets and service operations are
visible, updatable, etc.
      // Examples:
      // config.SetEntitySetAccessRule("MyEntityset", EntitySetRights.AllRead);
      // config.SetServiceOperationAccessRule("MyServiceOperation",
ServiceOperationRights.All);
    }
  }
}
```

Najskôr musíme do riadku definujúceho triedu WebDataService1 do špicatých zátvoriek namiesto komentára:

```
  public class WebDataService1 : DataService< /* TODO: put your data source class name
here */ >
```

Pridať názov triedy dátového zdroja, v našom prípade:

```
public class WebDataService1 : DataService<AdventureWorksLT2008Entities>
```

Následne odstránime komentár z riadka:

```
// config.SetEntitySetAccessRule("MyEntityset", EntitySetRights.AllRead);
```

Aby sme povolili prístup k údajom. Riadok zmeníme tak, aby sme umožnili neobmedzený prístup k údajom. V cvičnom príklade pri vysvetľovaní základných princípov nebudeme zatiaľ myslieť na bezpečnosť. V reálnej aplikácii povolíme len to, čo je pre fungovanie aplikácie nevyhnutné:

```
public static void InitializeService(IDataServiceConfiguration config)
{
  config.SetEntitySetAccessRule("*", EntitySetRights.All);
}
```

Ak teraz aplikáciu spustíme, zobrazí sa vo formáte XML zoznam entít poskytovaných službou:

```
<?xml version="1.0" encoding="utf-8" standalone="yes" ?>
 <service xml:base="http://localhost:1044/WebDataService1.svc/"
 xmlns:atom="http://www.w3.org/2005/Atom" xmlns:app="http://www.w3.org/2007/app"
 xmlns="http://www.w3.org/2007/app">
  <workspace>
    <atom:title>Default</atom:title>
    <collection href="Customer">
     <atom:title>Customer</atom:title>
    </collection>
   </workspace>
 </service>
```

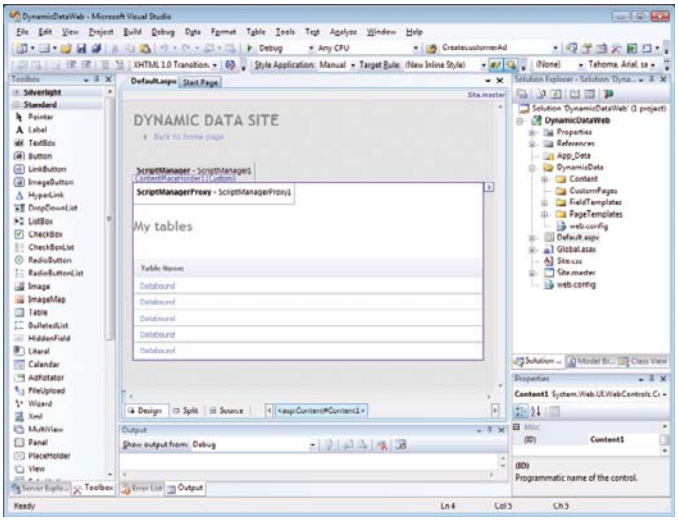
Ak pridáme do URL adresy názov tabuľky, vráti služba všetky záznamy v XML. V našom prípade pridáme do URL adresy názov Customer **http://localhost:1044/WebDataService1.svc/Customer**

Do URL adresy dátovej služby môžeme pridať parametre:

expand – pridá do výsledku asociované entity,
value – vráti hodnotu vlastnosti ako jednoduchý text bez prídavných atribútov,
top – podobne ako v príkaze SELECT vráti len požadovaný počet záznamov,
skip – preskočí požadovaný počet záznamov. Používa sa v kombinácii s top pre stránkovanie výsledkov,
orderby – parameter pre utriedenie výsledkov,
filter – umožňuje vytvárať podmienky pre filtrovanie údajov. Je určitým ekvivalentom klauzuly WHERE.

ASP.NET Dynamic Data

Dynamic Data je technológia, ktorá na základe popisu v objektovo relačnom modeli vygeneruje používateľské rozhranie bez potreby písania kódu, prípadne kód na maximálnu mieru minimalizuje. Je dôležitým pilierom pre vývoj takzvaných „data – driven" webových aplikácií.
V aplikácii Visual Studio 2008 vytvoríme webovú aplikáciu typu „Dynamic Data Web Application". Projekt vytvorený pomocou tejto šablóny je pomerne názorná ukážka, ktorú odporúčame preštudovať.



Vývoj ukážkovej aplikácie typu „Dynamic Data Web Application".

Aby sme mohli vytvoriť „data – driven" webovú aplikáciu, potrebujeme predovšetkým dáta. Buď sa pripojíme na SQL Server, v našom prípadne na databázu AdwentureWorks2008LT, alebo na databázový súbor v adresári priečinku App Data. Do projektu vložíme „LINQ to SQL classes", a na plochu nástroja Object Relation Designer presunieme tabuľku alebo tabuľky z Data Connections. V našom príklade budeme pracovať s tabuľkou Customers.

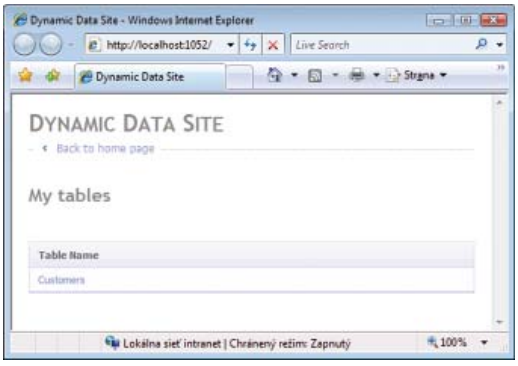
V súbore Global.asax odstránime komentár z riadka:

```
//model.RegisterContext(typeof(YourDataContextType), new ContextConfiguration() {
ScaffoldAllTables = false });
```

a riadok upravíme podľa použitého zdroja údajov, v našom prípade:

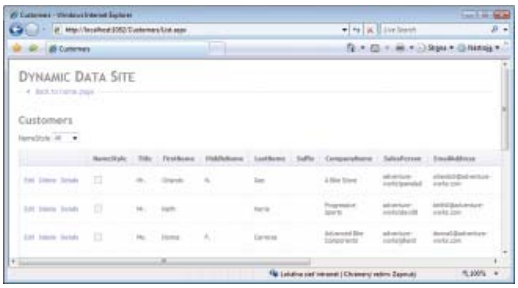
```
model.RegisterContext(typeof(awDataContext), new ContextConfiguration() {
ScaffoldAllTables = true });
```

Teraz môžeme aplikáciu spustiť. Na úvodnej stránke sa zobrazí zoznam tabuliek, ktoré boli zaradené do objektovo relačného dizajnéra.

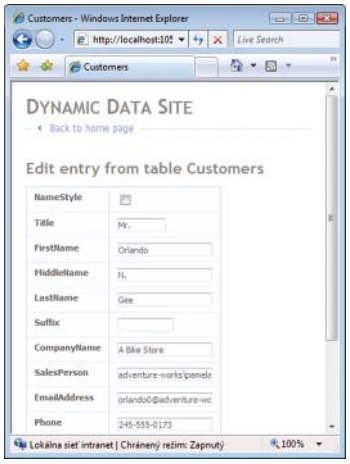


Na úvodnej stránke „Dynamic Data Web Application" sa zobrazí zoznam tabuliek

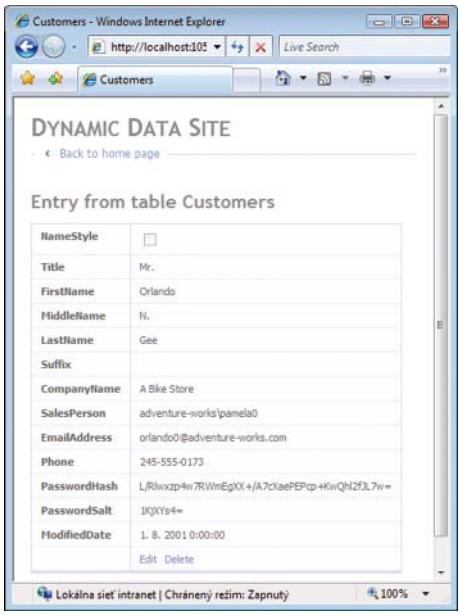
Po výbere tabuľky môžeme pracovať s údajmi, napríklad zobrazovať detaily, prípadne údaje editovať alebo vymazávať, alebo pridať nový záznam. **Podotýkame, že sme nenapísali ani jediný riadok kódu.**



Po výbere tabuľky môžeme pracovať s údajmi



Režim Edit



Režim Details

Vývoj aplikácií pre Office 2007

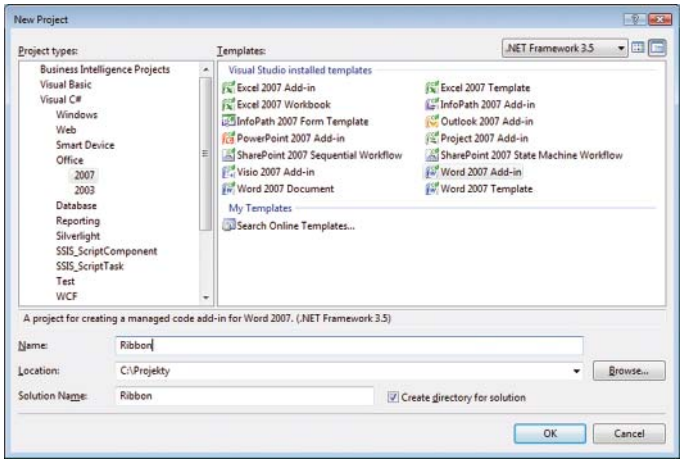
Vývoj aplikácií pre Office nadobúda čoraz viac na význame, nakoľko množstvo aplikácií pracuje s dokumentmi balíka Office. V predchádzajúcej verzii Visual Studio 2005 bolo možné vyvíjať VSTO projekty, no v novej verzii 2008 sa táto možnosť podstatne rozšírila. Prostredie kancelárskeho balíka Office je integrované priamo do vývojového prostredia Visual Studio 2008. Je možné vytvárať riešenia na úrovni dokumentu pre Word, Excel a InfoPath. Ovládacie prvky je možné umiestňovať priamo na plochu dokumentu a vytvárať tak napríklad vysoko sofistikované inteligentné formuláre. VSTO pre Office 2008 vyžaduje .NET Framework 3.5. Problematika vývoja aplikácií pre Office by zabrala celú publikáciu a tak viac menej náhodne vyberieme a predstavíme jednu novú funkcionalitu.

Office 2007 Ribbons

Kancelársky balík Office 2007 prišiel s novou filozofiou inovatívneho ovládania používateľského rozhrania „Ribbon" s pásom nástrojov. Ide o pomerne široký panel nástrojov, umiestnený v hornej časti používateľského okna. Ikony v ňom sa prispôsobujú práci. Týmto pásom sú vybavené najdôležitejšie aplikácie kancelárskeho balíka Office. Pás obsahuje v podstate tri časti. Celkom hore je ikona celého balíka Office. Tá obsahuje vypichnuté funkcie, ktoré používa väčšina používateľov. Obsahuje položky New, Open, Save, Save As, Print, Finish (nová možnosť, pozri ďalej), Send, Publish a Close. Vedľa ikony Office je potom panel nástrojov, označený ako Quick Access Toolbar, kde sú vo forme ikon znázornené dôležité nástroje, ako je Save, Print, Undo. Tento Toolbar možno jednoducho používateľsky prispôsobiť, teda doplniť vlastné ikony alebo umiestniť na iné miesto. Nasledujú jednotlivé karty pása, ktoré obsahujú typicky osem položiek. Každej karte zodpovedá iná skupina nástrojov funkcií. Základnou kartou je položka s označením Home, kde sa nachádzajú najdôležitejšie funkcie.

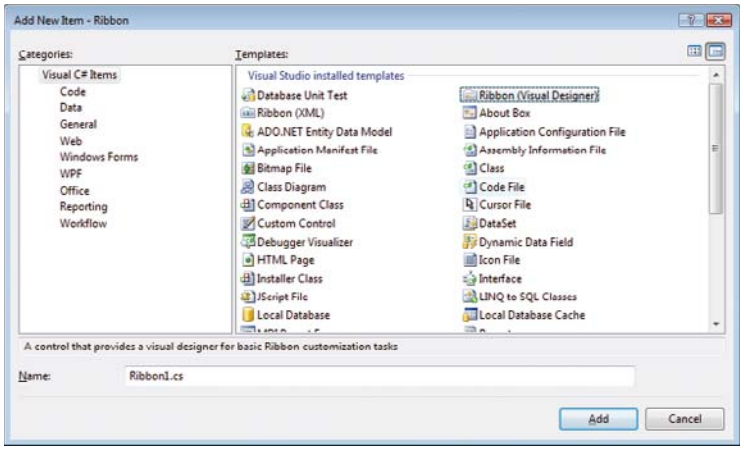
Podpora RibbonX XML bola zahrnutá už vo VSTO 2005 SE a samozrejme zostáva aj v aplikácii Visual Studio 2008. Pribudlo však nové interaktívne návrhové prostredie RibbonX Designer.

Vo vývojovom prostredí Visual Studio 2008 vytvoríme projekt typu Word 2007 Add-in.

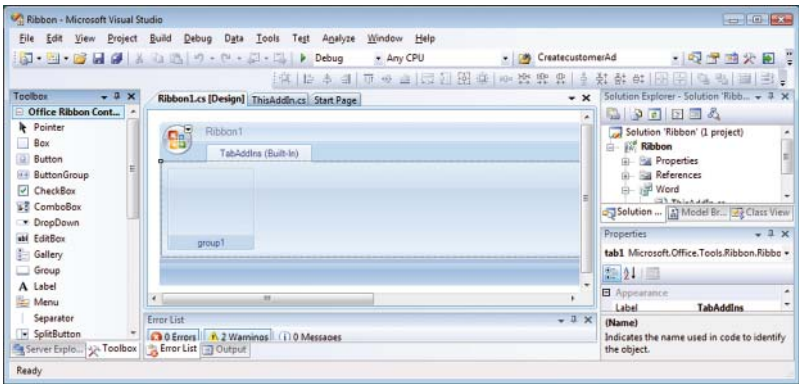


Vytvorenie doplnku Word 2007 Add-in

Zatiaľ projekt obsahuje len prázdnu kostru triedy. Cez kontextové menu Add New Item pridáme do projektu Ribbon Visual Designer.



Pridanie nástroja Ribbon Visual Designer



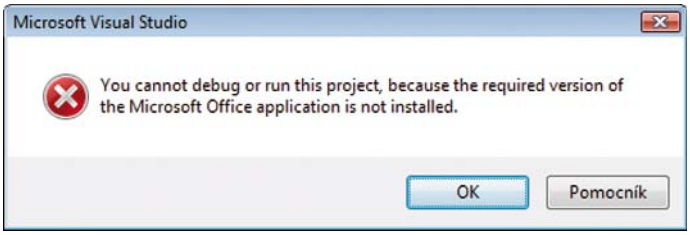
Návrhové prostredie Ribbon Visual Designer

Môžeme napríklad pridať tlačidlo pre vyvolanie nejakej akcie. Vytvorí sa kód obslužnej procedúry. V našom príklade budeme procedúru simulovať oznamom v Message boxe:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using Microsoft.Office.Tools.Ribbon;

namespace Ribbon
{
  public partial class Ribbon1 : OfficeRibbon
  {
    public Ribbon1()
    {
      InitializeComponent();
    }

    private void Ribbon1_Load(object sender, RibbonUIEventArgs e)
    {

    }

    private void button1_Click(object sender, RibbonControlEventArgs e)
    {
```

```
        //akcia
        System.Windows.Forms.MessageBox.Show("Vykonava sa akcia");
    }
  }
}
```

Pre spustenie VSTO aplikácie potrebujeme mať nainštalovanú správnu verziu Office, inak sa miesto spustenia aplikácie zobrazí chybové hlásenie.

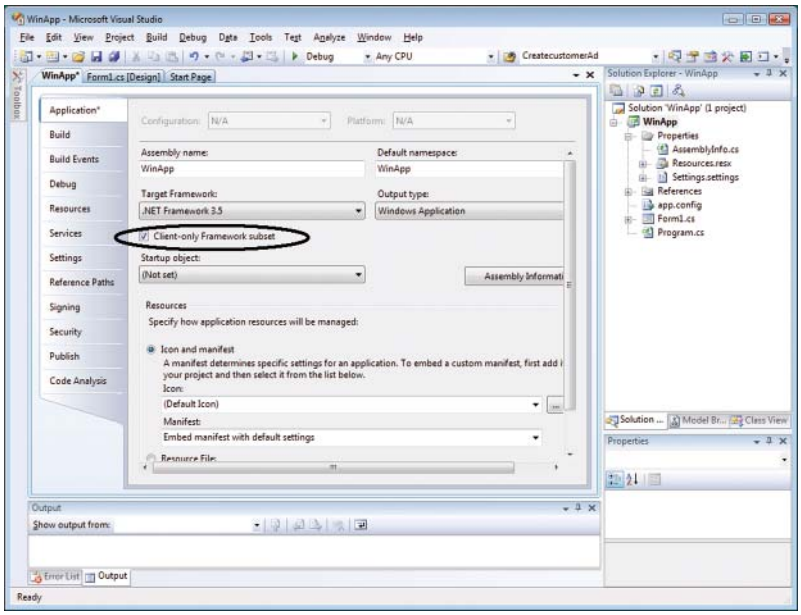


Pre spustenie VSTO aplikácie potrebujeme mať nainštalovanú správnu verziu Office

.NET Framework Client Release „Arrowhead“

Pod kódovým označením „Arrowhead“ bola vyvíjaná podmnožina prostredia .NET Framework 3.5 SP1 s názvom NET Framework Client. Tento „mini framework“ je určený pre klientske aplikácie a nainštaluje sa spolu s nimi. Plná inštalácia zaberá približne 30 MB. Obsahuje Windows Presentation Foundation (WPF), Windows Forms, Windows Communication Foundation (WCF) a Base Class Libraries.

Ak chceme vyvíjať aplikáciu aby fungovala s .NET Framework Client Release, je potrebné vo vlastnostiach aplikácie (Menu Project, NazovAplikacie Properties) nastaviť v záložke Application parameter Client-only Framework subset.



Nastavenie parametra Client-only Framework subset

Aplikáciami .NET Framework 3.5 a Visual Studio 2008 sa história nekončí, príde „Rosario“

Už v dobe uvádzania vývojového prostredia Visual Studio 2005 sa vedelo nielen kódové označenie nasledujúcej verzie („Orcas“), ale Microsoft odhalil aj niečo z pod pokrievky prác svojich vývojárskych tímov. Podobne je to aj v tomto prípade. Nástupcom verzie 2008 bude verzia zatiaľ známa pod kódovým označením „Rosario“. Bude oveľa hlbšie podporovať efektívny vývoj projektov na všetkých úrovniach a dôsledne naprieč celým životným cyklom vývoja a podpory, pričom sa zameria hlavne na kvalitu vyvíjaných aplikácií. Preto bude obsahovať pokročilejšie nástroje pre ladenie. Predpokladá sa rozsiahlejšia podpora technológie Silverlight 2.0 alebo vyššej vtedy aktuálnej verzie, vývoj aplikácií pre Business Intelligence a podobne.

Príloha – Inštalácia cvičnej databázy Adventure Works 2008 a odľahčenej verzie LT pre SQL Server 2008

V praktických príkladoch sú využité údaje z cvičnej databázy AdventureWorks LT. Cvičné databázy sú pre SQL Server 2008 k dispozícii voľne na prevzatie napríklad na adrese

http://go.microsoft.com/fwlink/?LinkId=87843.

K dispozícii sú štyri verzie cvičnej databázy. V názve každej z nich je názov fiktívnej firmy „Adventure Works" vyrábajúcej bicykle a bicyklové príslušenstvo. Dodáva ich na americký európsky a ázijský trh (fiktívny samozrejme...).

AdventureWorks OLTP – štandardná databáza fiktívnej firmy pre testovanie transakčných príkladov a scenárov. Obsahuje schémy Manufacturing, Sales, Purchasing, Product Management, Contact Management, a Human Resources.

Adventure Works DW databáza pre príklady scenárov budovania dátového skladu.

Adventure Works AS cvičná databáza pre scenáre využitia business intelligence, teda analytických služieb, dolovania údajov (datamining), integračných a reportovacích služieb.

Adventure Works LT je podstatne zjednodušenou verziou databázy AdventureWorks OLTP.

AdventureWorks OLTP

Ak by sme sa pokúsili o popis všetkých takmer sedemdesiatich tabuliek ktoré tvoria túto databázu, aj napriek veľmi jasne pomenovaným tabuľkám v takejto podobe by sme len veľmi ťažko identifikovali, aké sú vzájomné väzby a filozofia návrhovej štruktúry. Predstaviť firemnú databázu, či už reálnej, alebo v tomto prípade fiktívnej firmy sa nám pravdepodobne najlepšie podarí tak, že predstavíme firemné procesy. Podľa týchto procesov sú objekty rozdelené do schém.

Schéma	Popis objektov	Príklad tabuliek v schéme
HumanResources	Zamestnanci spoločnosti Adventure Works Cycles.	Employee, Department
Person	Mená a adresy zákazníkov, predajcov a zamestnancov.	Contact, Address, StateProvince
Production	Produkty vyrábané a predávané spoločnosťou Adventure Works Cycles.	BillOfMaterials, Product, ProductCategory, ProductSubcategory, WorkOrder
Purchasing	Dodávatelia súčiastok.	PurchaseOrderDetail, PurchaseOrderHeader, Vendor
Sales	Údaje týkajúce sa obchodu a zákazníkov.	Customer, Individual, SalesOrderDetail, SalesOrderHea der, Store, StoreContact

Inštalačný program cvičnej databázy AdventureWorksLT má približne 50 MB a umožňuje jej nainštalovanie na lokálny server. Po spustení inštalačného programu sa vytvorí priečinok:

c:\Program Files\Microsoft SQL Server\100\Tools\Samples\.

V tomto priečinku je cvičná databáza uložená vo forme záložného súboru s príponou BAK, konkrétne AdventureWorks2008.bak. Skript pre obnovu tejto databázy je v súbore RestoreAdventureWorks2008.sql Tento skript spustíme pomocou nástroja SQL Server Management Studio. Skript obsahuje v komentári priame nastavenie priečinka, v ktorom sa nachádza súbor zálohy. Aby sme sa vyhli problémom s prideľovaním prístupových práv do podpriečinkov priečinka c:\Program Files, prekopírovali sme súbor zálohy do novovytvoreného priečinka c:\Install, ktorý má štandardné prístupové práva.

```
USE master;
DECLARE @source_path nvarchar(256);
SET @source_path = 'c:\Install\'
...
```

AdventureWorksLT

Vzhľadom k jednoduchosti budeme v cvičných príkladoch využívať práve túto databázu. Ide o zjednodušenú a do značnej miery denormalizovanú databázu AdventureWorks OLTP. Zatiaľ čo pôvodná databáza mala 180 MB a obsahovala 5 schém a 70 tabuliek, zjednodušená databáza AdventureWorksLT má necelých 7 MB a obsahuje len jednu schému a v nej 12 tabuliek.
Inštalačný program cvičnej databázy AdventureWorksLT má 3 MB a umožňuje jej nainštalovanie na lokálny server. Po spustení inštalačného programu sa vytvorí priečinok.

c:\Program Files\Microsoft SQL Server\100\Tools\Samples\.

V tomto priečinku je cvičná databáza uložená vo forme záložného súboru s príponou BAK, konkrétne AdventureWorks2008LT.bak. Skript pre obnovu tejto databázy je v súbore RestoreAdventureWorks2008LT.sql. Pri inštalácii postupujeme podobne ako v predchádzajúcom odseku zadaním zdrojového priečinka do skriptu.

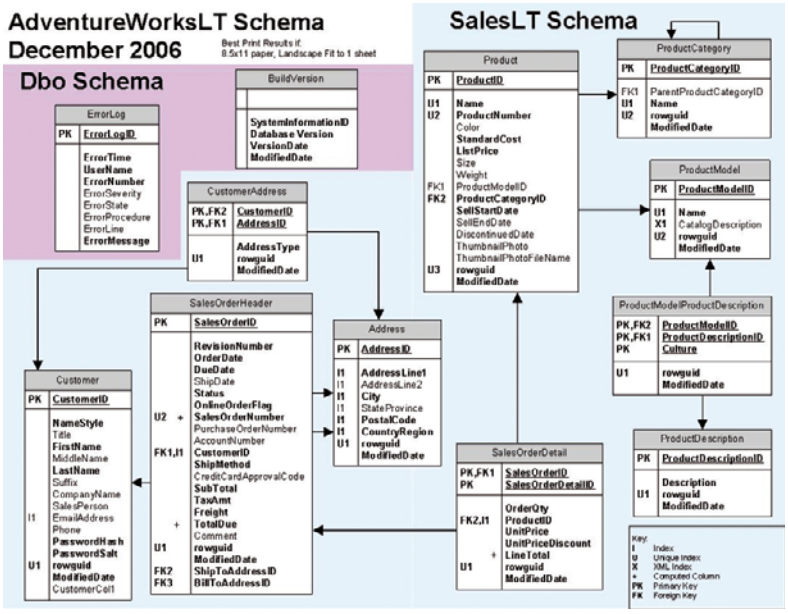


Schéma cvičnej databázy AdventureWorksLT

www.microsoft.sk