

Разработване на мобилни приложения за windows phone 7.5

Настоящото издание е оригинален превод на синята книга на автора Rob Miles от University of Hill, озаглавена „Windows Phone Programming in C#“ предназначена за разработване на мобилни приложения за платформа Windows Phone версия 7.5 Превода е извършен от колектив студенти в Центъра по информатика и технически науки на Бургаски свободен университет.

РЕДАКТОР: ГЛ.АС.Д-Р ДИМИТЪР МИНЧЕВ © 2014

За тази книга

Настоящото издание „РАЗРАБОТВАНЕ НА МОБИЛНИ ПРИЛОЖЕНИЯ ЗА WINDOWS PHONE 7.5“ е оригинален превод на синята книга на автора Rob Miles от University of Hill, озаглавена „Windows Phone Programming in C#“ предназначена за разработване на мобилни приложения за платформа Windows Phone версия 7.5 Превода е извършен от колектив студенти в Центъра по информатика и технически науки на Бургаски свободен университет.

Колектив



Галина Величкова

Галина Величкова е завършила СОУ "Любен Каравелов" гр. Несебър с профил "Информатика и информационни технологии". В момента се обучава за бакалавърска степен по "Информатика и компютърни науки" в Бургаски свободен университет. Взима участие в най-различни студентски състезания и научни конференции.



Стефани Николова

Стефани Николова е завършила ГПАЕ „Гео Милев“ през 2012г. с положени два сертификата за владение на чужд език: Advanced по английски и TestDaF по немски език. Понастоящем е студентка в Бургаския свободен университет, специалност Информатика и компютърни науки, 2.курс, степен бакалавър. Отличава се с трето място на миналогодишното си участие в Студентско научно творчество. Освен това, е провела студентска практика в реална работна среда благодарение на проекта „Студентски практики“ в „Джей Софт“ ООД. Области на научен интерес: уеб дизайн и приложения.



Бианка Оливейра

Първите две години от средното си образование, осми и девети клас, бях в Dawnview High School, Йоханесбург, където съм родена. Завърших останалите три години в Бургас, България, в Гимназия с преподаване на английски език „Гео Милев“. В момента уча Информатика и компютърни науки в Бургаския свободен университет, редовно обучение. Участвала съм в Студентското научно творчество в първи курс(2012-2013). Владея английски, португалски и български език. Компютърна грамотност със сертификат за изкаран курс Microsoft Office 2007 и Microsoft Office 2010.

Глава 1. Windows Phone

В тази глава ще се запознаете с Windows Phone платформата като средство за изпълнение на програми. Ще научите за основните характеристики на самата платформа, с начина на писане на програмите и как можете да ги продавате чрез Windows Marketplace.

1.1. Платформата Windows Phone

В тази секция ще разгледаме основния хардуер, от който се състои Windows Phone. Това е особено важно, тъй като е необходимо да се приложат способностите на телефона в контекст и да се определи резултата от физическите ограничения, наложени от употребяваната платформа.

Windows Phone като компютър

Почти всичко в днешни дни е един компютър. Мобилните телефони не са изключение. Когато достигнете нивото на Windows Phone device, е логично да го възприемате като компютър, който може да извършва телефонни разговори отколкото като телефон, който поддържа програми.

Windows Phone device има много общо с „обикновения“ компютър. Притежава мощен процесор, бързи 3D графики и много голяма памет. Освен това има собствена операционна система, която следи устройството и програмите, изпълнявани от него. Ако сте използвали PC, вероятно сте запознати с операционната система Windows, която се стартира с пускането на компютъра и дори го изключва, когато приключите.

Сериите на Windows Phone 7 представляват тотална новост в сравнение с предишните версии на устройствата на Windows Mobile. Имате възможността да пишете и изпълнявате програми на по-ранни версии, но не сте използвали Silverlight или XNA среди за тази цел. Числото 7 в името на продукта въплъщава идеята, че това е седмото прераждане на Windows Mobile платформата. Това не означава, че устройството използва подпорите на мониторните PC-та, зареждащи Windows 7. Въпреки това, както ще видим, съществува идеалната възможност да изберете програма, създадена за Windows Phone и да я заредите на Windows десктоп, както и обратно.

Ако сте запознати с компютърните спецификации, тогава тези, изброени отдолу биха били доста иновативни за преносимите устройства. Ако не сте, то помнете, че никой в света не е имал такъв компютър до преди няколко години, а сега можете да носите такъв в малкия си джоб.

Windows Phone Hardware

Преди да започнем да програмираме можем първо да се запознаем с хардуера, с който ще работим. Това няма да е текст за компютърния хардуер, но си струва да се спомене част от мобилния хардуер. Всички Windows телефони трябва да съдържат определен минимален брой спецификации, които да намерите в устройството.

Много е вероятно различните производители на мобилни устройства да добавят свой „щрих“ върху платформата, така че не е чудно да намерите апарати с повече памет, по-бързи процесори, хардуерни клавиатури и по-големи екрани.

Важно е да се отбележи, че хардуерните клавиатури не е задължително да присъстват в устройството (възможно е да е само с touch screen), защото ако сте XNA разработчик, може да се чудите къде е отишъл джойстика. Има някои основни промени в хардуера, към които трябва да се приспособите при писането за тази платформа, но също така има много интересни опции за вход на данни (особено за разработването на игри), където бихте могли да използвате акселометъра и touch screen-а за по-добър ефект. По-късно в Глава 8 ще ги разгледаме.

The Windows Phone Processor

Централният процесор (ЦП) на компютъра е мястото, където се извършва цялата работа. Всеки път, когато се изпълнява дадена програма ЦП е отговорен за предоставянето и променянето на база данни от паметта и връщането им обратно (нещо, което правят всички компютри). Най-известната скоростна машина в компютъра е тази на часовника. ЦП има часовник, който тиктака, когато се извършва работа. При всяка отчетена секунда процесорът ще извършва една част от операцията – може да взима инструкции от паметта, изпълнява изчисление и т.н.

Колкото по-бърза е скоростта на часовника, толкова по-бърз е процесорът. Модерните настолни компютри имат часовници, които тиктакат на всеки 3GHz, което прави около 3млн. пъти в секунда, което реално е невероятно бързо. Това означава, че един единствен такт на часовника ще трае една наносекунда – времето, за което светкавицата изминава 30см. Ако се чудите, защо вече нямаме големи компютри, то е защото времето, за което сигналите правят едно пълно обхождане е сериозен ограничаващ фактор в изпълнението. Колкото по-малък е компютърът, всъщност, толкова по-бърз е той.

Windows Phone има часовник, който се отчита на всеки 1 GHz. Вероятно си мислите, че ще зарежда със скорост една трета от тази на PC, но не такъв се оказва случаят. Това е така поради набор от фактори:

Първо, скоростта на часовника не е директно сравнима между процесорите. Този в Windows PC може да извършва нещо за пет тиктакания докато този на Windows Phone може да се нуждае от десет. Процесорът на настолния Windows вероятно извършва операции в хардуера (напр. аритметика с плаваща запетая), нещо, за което процесорът на Windows Phone би извикал софтуерна подчинена програма, за да го изпълни – процедура, която би била по-бавна. Може да се направи сравнение на фактора скорост на часовника с големината на двигателя в колите. Автомобил с по-голям двигател ще се движи по-бързо отколкото този с малък, но много други фактори (тегло на колата, скоростна кутия, джанти) са също от значение.

Второ, Windows PC може също така да има множество от процесори, но това не означава, че може да има по-голямо бързодействие в сравнение с два мотоциклета, които се движат по-бързо от един, макар и това да не означава, че могат да преработят повече данни за единица време

(извозват два пъти повече хора). По някое време бихме си взели телефони с множество от процесори (нещо, което операционната система на Windows Phone може да поддържа), но в един момент се оказва, че всички те имат един единствен процесор.

В заключение, Windows PC има неограничена захранваща мощност. Има възможност да поддържа нонстоп ЦП на пълни обороти, ако се наложи. Единственият реален проблем е, че процесорът трябва да се охлажда, за да не се стопи. Колкото по-бързо работи той, толкова повече енергия ще изконсумира. Ако телефонът използва процесора на максимална скорост, то животът на батерията би бил много кратък. ОС ще ускори и забави процесора в зависимост от работата, която трябва да свърши за всеки изминал момент. Въпреки че телефонът притежава бърз процесор тази скорост всъщност се използва когато се налага много бърза реакция.

Имайки предвид казаното по-горе, може да се обобщи, че когато се пише програма за Windows Phone, мощността на процесора не трябва да се възприема като неограничен ресурс. Програмистите за настолни компютри не придават голямо значение на скоростта на процесора, но тези за Windows Phone е добре да запомнят, че лошият дизайн може да има последствия, засягащи опита на потребителя или живота на батерията на телефона. Добрите новини за нас са, че притеснявайки се за тези неща, ние ще се превърнем в по-добри програмисти.

1.2. Операционна система Windows Phone

Операционната система на Windows Phone се нарича Windows CE ("Compact Edition"). Била е специално разработена за преносими компютърни системи и е много добра в извършването на работа и щаденето на живота на батерията на устройството. Както ще видим по-късно, това създава някои ограничения на програмите, но въпреки това хубавата новина е, че що се отнася до нас, основната ОС е без голямо значение. Програмата ни ще тръгне на Windows Phone така както би го направила и на обикновен компютър.

Графичен дисплей

Windows Phone има дисплей с голяма резолюция, изграден от огромен набор от пиксели. Това осигурява добро качество на графиката и позволява да се изобрази голямо количество текст върху екрана. Колкото повече пиксели има на дисплея, толкова по-високо ще е качеството на изображението. Въпреки това се увеличава количеството памет, необходима за запаметяването на желаната картинка, и толкова повече работа ще трябва да свърши компютъра, за да я зареди на екрана. Това е изключително важно особено за мобилно устройство, където повечето работа на хардуера се превръща в по-голяма консумация на мощност и намалява живота на батерията. Резолюцията на дисплея е компромис между живота на батерията, цената за производство и яркостта на екрана (при по-малки пиксели се изразходва по-малко светлина).

Настоящите версии на of Windows Phone имат екранна резолюция поне 800 x 480 px. Тя може да намери приложение както в пейзажните (800 широчина и 480 височина) така и в портретните (480 широчина на 800 височина) форми. Телефонът съдържа акселометър, който долавя начина, по който се държи телефона. Операционната система на Windows Phone може в този случай да

настоя дисплея да съвпадне с посоката. Нашите програми могат да определят кога да заредят и според кои посоки да се настроят. Ако разработим програмите си да работят и за пейзажни, и за портретни състояния, тогава могат да им се изпращат съобщения, позволяващи им да приспособят дисплея си, когато потребителят смени насоката на устройството.

Един проблем, пред който са изправени разработчиците за мобилни устѳйста е множеството различни размери на екрана, които се предлагат. По принцип програмата е добре да се модифицира за всеки отделен по големина дисплей. Хардуерът на екрана на Windows Phone включва особеност, позволяваща да се измери големината на дисплея на устройството и да пасне на всеки размер. Дадена игра може да изисква определена размерност на екрана (напр. 320x240) и тогава дисплейният хардуер ще измери тази големина и ще я настрои според устѳйството, което се използва. Това е много полезно и позволява да се създават игри, които могат да се играят на апарати, даже таиѳа, които още не са изобретени.

Графичният процесор на Windows Phone

В първоначалните копютри цялата работа се извършѳвала единствено от процесора. Това включѳвало изобразяването на информация върху екран. Хардуерните инженери скоро откриват, че могат да забързат изображенията, акто изобретят ръчни устройства, които да управляват екрана. Основният процесор даѳа команди на графичния такѳв и отнема цялата работа за изрисѳване на екрана. По-напредналите графични процесори притежават 3D защита и са способни да извършѳват аритметика с палѳаща запетая и матрици, нужна за триизмерността. Те също съдържат засенчѳащи пиксели, които могат да бѳдат програмирани да преработѳват високоскоростно изображението на ѳсяка точка от екрана на момента на рисуването, добавяйки светлинни ефекти или мъгляѳини.

Доскоро само настолните компютерни системи и ѳдео игри конзолите са имал графични процесори, но сега те навлизат и в мобилните апарати. Платформата на Windows Phone съдърѳа графичен процесорен чип, използѳван да осигури 3D анимиращия ефект за телефонния екран и също може да бѳде използѳван в средата за разработѳване на XNA игри при създаването на движещи се 3D игри.

Touch input

В по-старите преносими устройства са използѳвани съпротивителното докосѳване на дисплея. Когато потребителят докосне такѳв екран, пластичната повърхност се огѳѳа и свързѳа със слой под нея. Прости схеми са използѳвани да измерят ел.съпротивление, предизѳвикано от докосѳването и определят къде точно се е състояло то. Съпротивителните тѳч скрийнове са евтини за изработѳка и работят отлично с писалѳка. ѳпреки това, начинът, по който работят затрудняѳа да се определят многобройни едновременно докосѳвания на екрана. Трудоемко е също така да се изработи подобен дисплей от тѳвърда материя, напр. стѳкло, тѳй като повърхността трябва да се огѳне, за да направи контакт, който засича мястото на ѳѳвеждане. Капацитивният сензорен екран работи различно. Набор от проводници под екранната повърхност засичат промяната в капацитета, причинена от присѳъствието на прѳст на плоскостта. Хардуерът за тѳч скрийна разбира къде из

повърхността се е състояло въвеждането на данни чрез докосване. Капацитивните тъч скрийнове са по-скъпи при производството, защото изискват допълнителен хардуер, който да преработи въведените сигнали, но сензорните мрежи могат да бъдат принтирани на опаковата част на стъкления екран, за да направят дисплея по-здрав. Този тип екран не е толкова прецизен както резисторния, използващ писалка, но е възможно да се изработи капацитивен екран, който да засича множество входни данни по различни части на дисплея.

Всички Windows Phone устройства имат тъч скрийнове, които имат възможността да засекат поне четири начални точки. Това означава, че ако създадете Windows Phone пиано програма, тя ще може да отчете най-малко четири ноти, натиснати по едно и също време.

Преходът към мулти-тъч вход е важна стъпка в еволюцията на мобилните апарати. Потребителят може да контролира софтуера чрез използването на мулти-тъч жестове като „ощипване“. Операционната система на Windows Phone осигурява вградена подкрепа за разпознаване на тези жестове. Вашите програми могат да приемат събития, когато юзъра извърши определен тип действие.

Локационни центрове

Windows Phone устройството е пространствено-ориентирано. Съдържа Глобална Позиционна Система (GPS), която приема сателитни сигнали и определя позицията на устройството до няколко метра. Тъй като GPS системата работи добре, когато телефонът има ясна визия към небето, мобилното устройство ще трябва да използва и други техники за определяне на положението включително локация на най-близката телефонна кула и/или WIFI връзка, която да използва. Този метод се нарича „подпомогнат“ GPS.

Windows Phone ОС разкрива методи, чрез които програмите могат да определят физическата позиция на устройството както и нивото на правдивост и резолюция на приакчения резултат. Програмите също така ще имат възможност да използват карти и средства за търсене, които могат да бъдат заредени според информацията за мястото. Но с това ще проучим малко по-късно. Средата за разработка също дава на разработчик наистина полезен GPS емулятор, за да можете да тествате как програмите Ви използват информацията относно разположението, без да се налага да ставате от бюрото си.

Акселометър

Представлява хардуерно устройство, което измерва ускорението – до тук нищо ново. Най-добрата му употреба е при установяването как телефонът бива държан. Това е така, тъй като акселометърът определя ускорението благодарение на земната гравитация. Това придава на телефона постоянна стойност на ускорение, която винаги сочи надолу. Програмите могат да разберат позицията на мобилния апарат чрез трите оси X, Y и Z.

Това означава, че могат да се пишат програми, които кореспондират, когато потребителя насочи телефона в определена посока, което би било много полезно при механизма на игрите. Много е лесно за програмите да се съдобиат с акселометърни доклади, както ще научим по-късно.

Компас

Телефонът също така съдържа електронен компас, за да може програмата да определи на коя посока е обърнат апарата. Това би било полезно в приложения, помагащи на потребителя да се ориентира и в „разширената реалност“, където телефона наслоява компютърно нарисувана информация на изображение на околостта.

Жироскоп

Механичният жироскоп е устройство, което винаги сочи в определена посока. Windows Phone съдържа електронна версия, която позволява да се определи кога телефонът е обърнат или завъртян. Програмите могат да използват акселометъра да определят как телефонът бива държан, но жироскопът дава много по-точна информация, дори за степента на завъртане.

Сензорна интеграция и симулация

Чудесно е да имаме много сензори в телефона, но това също така си има недостатък, който е, че обикновените програмисти (като теб и мен) трябва да изобретят софтуер, който да извлече най-доброто от всички различни сигнали, които се произвеждат. Възможни са трудности при тестването на това как програмите реагират на определен набор от движения и действия. За щастие, Windows Phone осигурява единствен софтуерен обект, който интегрира информацията от различните сензори и осигурява лесен начин, по който програмата може да разбере за посоката и вижението на телефона. Това ще даде най-добри резултати в зависимост от сензорите в реалното устройство. Също е възможно да се изобретят определени действия, които след това да бъдат повторени от Windows Phone емулятора, когато се тества дадена програма.

Камера

В днешни дни всички мобилни устройства имат камера и Windows Phone не е изключение. Камерата на телефона ще има поне 5 мега пиксела и повече. Това означава, че снимките ще съдържат 5млн. точки. Това се сравнява с прилична резолюционна дигитална камера (или най-доброто, което сме могли да си позволим преди пет години). Пет-мега пикселова картина може да бъде разпечатана на принтер 7”x5” с много добро качество.

Windows Phone приложението може да контролира камерата и да прави индивидуални снимки или потоци от видео. Възможно е също така да има достъп до самия поток, така че да добавя нови неща към дисплея, за да произведе приложения с „разширена реалност“ или да открие специфични обекти като баркодове или лица във фотографии. Снимките, които потребителят прави могат да бъдат запазени като част от медийното пространство на телефона. Нашите програми също могат да отворят тези изображения и да работят с тях.

Хардуерни бутони

Всички Windows Phone системи споделят подобен потребителски интерфейс. Като част от този дизайн всеки Windows Phone има набор от физически бутони, които са програмирани да изпълняват една и съща функция независимо то марката или модела на телефона.

Start: Стартовият бутон се натиска при започването на нова дейност. Това ще Ви отведе към стартовия екран на програмата, където бихте могли да изберете нова програма и да я заредите. Когато натискате този бутон, приложението, което в момента е пуснато бива моментно спряно (ще обсъдим това по-късно). Въпреки това Windows Phone ОС „запомня“ кое приложение е било спряно така че да можете да се върнете към него по-късно, натискайки Back-копчето.

Back: Бутонът служи за връщане към предишно меню на определено приложение. Също така се използва при спирането на стартираното в момента и връщане към предишно. Този бутон улеснява много работата с телефона. Може да се стартира ново приложение (напр. да се изпрати имейл докато се сърфира в мрежата) и след като то се изпрати, да се натисне Back бутона, който връща към браузъра. В мейл приложението Back бутона се използва при навигирането между различните екрани и менюта, но веднъж щом се изпрати съобщението, той ще Ви върне към стартовото меню, друга програма или сървър. Използва се и за активирането на преход между различните задачи на телефона. Ако бутонът бива натиснат продължително, телефонът ще товори екран с всички приложения, между които е възможно да се осъществи преход.

Lock: При всяко натискане на този бутон телефонът ще изключи дисплея си и по този начин ще се удължи живота на батерията. Когато потребителят избере стартиращия или заключващия бутон, на екрана ще се появи lock-екрана. По принцип телефонът може да се изключи автоматично след няколко минути бездействие. През това време е възможно някое приложение да продължи да работи. Тази особеност е полезна при такива приложения като навигация, където програмата трябва да остане активна, но това може да се отрази на живота на батерията.

Search: При натискането на този бутон се стартира ново търсене. Какво по-точно предстои да се случи зависи от това какво прави потребителят през това време. Ако той избере търсене докато е отворен сървър, ще види меню за търсене в уеб мрежата. Ако тази опция се посочи докато приложението “People” е активно, то ще започне търсене на контакти.

Camera: Когато потребителят избере приложението за камерата, това ще спре автоматично протичащата в момента програма, за да може да се направи снимка.

Начинът, по който тези бутони ще бъдат използвани има значение за програмите, които ще пишем. Една такава програма трябва да се справи с вероятността да бъде премахната от паметта по всяко време, напр. ако потребителят реши да фотографира докато играе игра, тогава играта ще бъде спряна и заетото пространство ще се изпразни. В момента, когато снимката се направи юзърът би трябвало да може да отвори обратно прозореца на играта и тя да бъде в същото състояние. Потребителят не трябва да забелязва, че играта е била спряна.

Предупредително съобщение се изпраща до програмите, на които предстои да бъдат спрени, а Windows Phone ОС осигурява няколко начина, по който приложението може да съхрани временна информация. По-късно в текста ще разгледаме този метод.

Не всички Windows Phone устройства ще имат физическа клавиатура за въвеждане на текст, но всички те могат да се уповават на тъч скрийна за това.

Памет и съхранение

Паметта е ендот от нещата, с които се хвалят притежателите на компютър. Както изглежда, колкото повече памет има компютърът, толкова „по-добре“. Тя всъщност идва с две основни качества. В самия компютър има място, където програмите протичат и също така място за допълнителна памет, използвано да съхранява данни и програми на устройството (хард диск и RAM памет). Един модерен настолен компютър вероятно ще има около 2 гигабайта (две хиляди мегабайта) памет за RAM и около 500 гигабайта от твърдия диск за съхранение, а 1MB е 1млн. байта. 1GB е хиляда милиона байта. Като груба ориентация, компресираният музикален трак използва около 6MB, високо качествена картина - около 3MB, а един час видео с добро качество - около 1GB.

Минималната спецификация на Windows Phone има поне 256 MB RAM и 8 GB памет. Това означава, че на база спецификацията Windows Phone ще има една осма от размера на паметта, и около 1/50 от размера на съхранение на настолната машина. Windows Phone ОС е оптимизирана за работа в малки количества памет. Въпреки това, някои от нещата, които прави, за да се увери, че устройството не е изчерпано относно ресурси оказват въздействие върху начина, по който се пишат програми, нещо които трябва да се знае.

Мрежова свързаност

А мобилния телефон всъщност е устройството с най- много връзки, което можете да получите. Едно такова устройство разполага с набор от мрежови съоръжения :

WiFi : Всички Windows телефони поддържат безжична мрежа, която осигурява висока скорост на връзката в мрежата, но работи само ако сте съвсем близо до точката за достъп до мрежата . За щастие, сега това се случва навсякъде, в много заведения за бързо хранене и дори в градските центрове се поддържа WiFi.

3G (Third Generation) : Това е следващото нещо, което се изпълнява в мобилната телефонна мрежа. Скоростта може да достигне WiFi , но е по- променлива. 3G достъпът може да бъде ограничен от гледна точка на обема данни, който мобилното устройство праща, а освен това са възможни допълнителни такси за използването на тази връзка.

GPRS : В много области на 3G мрежата не е налично покритие. Това ще доведе до ползването на GPRS мрежата, която е много по-бавна .

Мрежовите възможности са представени като TCP / IP връзки (ние ще проучи какво означава това по-късно в текста) . За съжаление, мрежовото покритие не е универсално затова софтуера на телефона трябва да може да се поддържа, дори когато няма връзка с база данни . В идеалния случай софтуерът трябва да се справя с различни скорости за свързване.

Windows Phone също така осигурява подкрепа за Bluetooth мрежа. Това се използва в контекста на свързване на външни устройства, като например слушалки и автомобилни аудио системи - нещо, до което нашите програми могат да си осигурят достъп в настоящата версия на Windows Phone.

Платформени предизвикателства

Windows Phone хардуерът е много мощен за мобилно устройство, но е все още ограничен от начина, по който трябва бъде преносим, а и мощността на батерията е ограничена. За жалост, потребителите, свикнали да работят с високо скоростни устройства с богати графични интерфейси очакват от телефона си подобно преживяване.

Като разработчици на софтуер нашата работа е да се осигури колкото се може повече от тези очаквания в рамките на средата, предоставена от телефона. Когато се пишат програми за такъв тип устройство трябва да се уверите, че те работят по най-ефикасния начин, възползвайки се максимално от наличната платформа.

Всеки път, когато се прави нещо за телефонна платформа, трябва да се мисли за последиците. Една от целите на тази книга е да ни направи много по-добри програмисти.

Добрите новини

В последните няколко секции се четат като списък на лоши неща, ограничения и компромиси, които обграждат мобилното развитие. Докато всички тези теми трябва да се помнят, също така не трябва да се забравя, че писането за мобилни устройства всъщност е много забавно. Наборът от функции, които устройството предлага и фактът, че е преносимо предоставя възможността да се създадат уникално нови приложения.

Също така, средствата, които ще бъдат използвани, за да се напише програма и вградените функции с предоставени инструменти, дават възможност за лесно създаване на наистина добре изглеждащи приложения.

1.3. Windows Phone екосистема

Windows Phone не е замислен като устройство, което да зависи само от себе си. Всъщност, то е част от екосистема, която съдържа редица други софтуерни системи, работещи покрай него, за да осигури работата на потребителя.

Zune Media софтуер за управление

Windows Phone устройството е свързано с компютър Windows чрез софтуера Zune. Това осигурява начин за управление на медиите и прехвърлянето им от и към телефонното устройство. Софтуерът Zune е средството, чрез което на фърмуера на телефона Windows могат да бъдат актуализирани и заредени нови версии. Софтуерът Zune също така осигурява връзка между средата за развитие Visual Studio и самия телефон.

Програмите, които се пишат могат да се възползват от средствата за масово осведомяване, прикачени към телефона чрез софтуера Zune. Много е лесно да се пишат програми, които зареждат снимки или възпроизвеждат музикални и видео файлове, съхранявани на телефона. Също така е възможно да се пишат програми за възпроизвеждане на музика, която да работи на фона на други програми, които потребителят директно използва.

Windows Live и Xbox Live

Притежателят на Windows Phone може да се регистрира от телефона си към Windows Live профила си. Ако Windows Live акаунтът е свързан с Xbox Live Gamertag, то профилът се пренася върху устройството и по този начин е възможно да се играят игри както на конзолата.

Windows Phone програмата може да използва Windows Live профила на собственика и XNA Gamertag информацията му.

Bing Maps

По подразбиране един телефон съдържа хардуер, който позволява да се разбере местоположението му. Услугата Microsoft Bing Maps предоставя карти, които покриват голяма част от Земята. Програмата Windows Phone може да използва своя мрежова връзка, за да се свърже с услугата Bing Maps, за да тегли карти и обширни гледки. Съществува и Silverlight контрол за карти, който улеснява програмата при добавяне на нови местоположения.

Windows услугата известяване

Въпреки че телефонът разполага с широка гама от мрежови технологии, все още ще има случаи, в които няма да може да се осигури връзка с мрежата. Windows Phone осигурява уведомителна услуга, която позволява програмите да получават известия от мрежата, дори когато програмата не е активна. Потребителят получава съобщение по телефона с информация, че приложението може да се стартира. Известията се съхраняват и управляват от услуга, която ще ги прехвърли в случай, че телефонът не е свързан към мрежата, когато съобщението се появи.

Например, може да имате система за продажби, която трябва да уведоми клиента, когато даден резервиран продукт е в наличност. Тази система може да се използва и при игрите, когато играчите отправят предизвикателства помежду си.

Windows Phone и Windows Azure

Windows Azure е набор от "облачни" услуги, които се предлагат от Microsoft. Вместо използването на личен компютър за предоставянето на данни за съхранение и обработка на потребителите може да се наеме място и обработчици в облака. Ако мрежово приложение стане много популярно, не е нужно да се купуват повече компютри, които да се свързват, а само да се увеличи потреблението на облачния ресурс.

Свървърни приложения

Облачните услуги също са много полезни, ако се налага голямо количество обработка веднъж или изключително рядко. Вместо голям брой компютри, които да са отговорни за най-големите потребности, работата може просто да се разпредели в облака. Могат да се пишат свървърни приложения за C# и да се тестват на облак-симулатор преди да намерят реалното си приложение. Клиентските приложения могат да се възползват от тези услуги чрез множество различни механизми на мрежата.

Съхранение на база данни

Докато един Windows телефон може да има база данни в себе си, често е по-полезно да се използват отдалечен достъп до базата данни, който да се ползва от мрежата. Това улеснява наличието на огромни количества памет, която може да се актуализира на централно ниво. Възможно е също така да се създадат база данни в облака, които да се използват от телефона чрез Windows приложението.

BLOB съхранение

Blob означава Binary Large Object. Blob може да бъде всичко - текста на книга, филм или изход от медицинска сканиране. Windows Azure осигурява съоръжение за съхранение, където приложението може да съхранява и обаботва тези обекти.

Authentication Services

В нередки случаи се появява проблем с удостоверяването, когато потребителят желае да получи достъп до ресурс. Доставчикът на ресурсите иска да се увери, че потребителят е реално лице. Това е особен проблем с преносими (и лесни за кражба) устройства като телефоните. Писането на високо авторизиран софтуер е много трудно и не е нещо, което може да се мине с лека ръка. Windows Azure осигурява начин за удостоверяване на тази услуга в рамките на облака.

В Windows Azure Toolkit

Телефон Windows Azure инструментариум предоставя проектни шаблони, набор от библиотеки и примерен код, които представляват първи стъпки в писането на облачни приложения, използвайки телефон (watoolkitwp7.codeplex.com).

Използването на екосистемите

Важно е да се знае, че телефонът не е просто телефон тези дни. Свързаният характер на приложението означава, че ще функционира във връзка с отдалечени услуги, които са на разположение през мрежовата връзка. Други части на телефона също се възползват от тази възможност. Телефонът разполага с вграден Facebook и камерата, а елементите за съхранение могат да качват снимките Windows Live SkyDrive или Facebook. Има методи, които позволяват лесното публикуване на новини в реално време и дори любими страници.

Мрежовите функции на телефона също могат да бъдат използвани, за създаването на персонални приложения между клиент и сървър. За целта са необходими две програми, една от които ще зарежда Windows Phone, а другата да работи на всеки компютър, свързан с интернет или дори на системата в "облака".

1.4. Изпълнение на програма в Windows Phone

Телефонът Windows предоставя платформа за стартиране на програми, някои от които могат да бъдат тези, които вече сме писали. Заслужава си да отдели известно време да се разбере начина, по който програмите са проектирани на телефона и по който ще се пишат.

Заявление включване на телефона Windows

Windows телефонът е проектиран на такава базата, че потребителят да бъде господар. Всички дизайнерски решения са направени по такъв начин, че независимо какво потребителят прави по това време, телефонът да бъде възможно най-ефективен.

Това означава, че когато потребителят стартира приложението си на телефона, то да бъде единственото, пуснато в момента. В случай, че има други активни приложения, активни на телефона, те остават на заден план докато не бъдат повторно повикани.

От изключително значение е потребителят да може да превключва от приложение на приложение с лекота и без излишни усложнения. Както вече стана ясно, бутонът „Назад“ предоставя на потребителя бърз начин да премине от една програма в друга; затова, когато създаваме нашите приложения, трябва да подсигурирм доброто настроение на потребителя през това време.

Едно от заключенията за всеки разработчик е, че трябва да свикне с мисълта, че всеки един момент програмата може да се "изрита от сцената", за да отстъпи на друга. Ако потребителят реши да използва отново приложението по някое време, трябва то да бъде устроено по такъв начин, че да продължи работата си на пълни обороти както преди.

Операционната система на Windows Phone осигурява "Бърза смяна на приложението" за случаите, когато приложението остава в паметта докато чака да бъде повторно повиквано. При подобна ситуация програмата разполага с цялата си памет и непокътнати ресурси, които само трябва да стартират наново. Въпреки това, не винаги има място за всяко едно приложение и затова някое може да бъде заличено от паметта или съхранено на друго място. Най-важното, обаче, което трябва да се запомни е, че от потребителска гледна точка преживяването трябва да бъде едно и също, когато се върне обратно към програмата. Точно както на театрален актьор ще му е необходимо известно време да се подготви за следващото изпълнение, приложението също ще се забави малко докато отново стартира, но след това трябва да се представи също толкова добре като преди. На програмата се оповестява, когато ѝ престои да бъде преместена на заден план, чрез което в паметта предварително се заделя място за съхранение. Този процес ще бъде разгледан по-обстойно в глава 9.

Background обработка

Ограниченията, наложени от процесора и трайността на батерията правят трудно за устройство с размер като на телефона да поддържа множество стартирани процеси. Потребителите на настолните компютри вече знаят какво е имаш десктоп, преплутан с много различни програми, стартирани едновременно, но на телефона това е непрактично и не на последно място дисплеят не е достатъчно голям, за да побере множество приложения.

Телефонът на Windows позволява приложенията да съдържат "фоновы задачи", които могат да поемат управлението, когато основното приложение не може да зареди. Такъв тип програми работят при стриктно управлявани обстоятелства, така че ако телефонът има няколко активни задачи, потребителят да не забележи, че телефонът работи по-бавно. Основно приложение на

такива приложения е поемането на задачи, предназначени за определени ситуации като възпроизвеждане на фонова музика, трансфер на файлове, редовни актуализации и редовна обработка на данни на определен интервал на време. Второстепенните приложения могат да уведоми потребителя за дадено събитие, изпращайки известие или обновяване на „Live Tile“ от екрана. След това потребителят може да стартира приложението, за да научи повече. Ще разгледаме как да създадем фоновы задачи в глава 9.

Windows Phone и код на управление

В началото компютърът се стартираше програма като файлът, съдържащ инструкции на програмата се зарежда в паметта и след това машината просто се подчинява от своя страна. Този подход работи добре, но не е и без своите недостатъци.

Първият проблем е, че ако видът на компютърна система е различен, то трябва да има друг файл от инструкции. Всеки компютърен производител произвежда хардуер, който разбира определен набор от двоични инструкции. Затова не е възможно да се вземе една програма, създадена за един вид хардуер и да се изпълни на друг.

Вторият недостатък е, че ако инструкциите са прости/опасни компютърът ще им се подчини така или иначе. Безмислените инструкции от файла, който се зарежда в паметта могат да предизвикат объркване в хардуера в компютъра. Опасните инструкции биха могли да накарали програмата да нанесе вреда на други данни.

The Microsoft Intermediate Language (MSIL)

Microsoft.NET се справя с тези проблеми с помощта на междинен език, който да опише това, което програма иска да направи. Когато се съставя C# програма компилаторът ще произведе файл, съдържащ инструкции в този междинен език. Когато програмата реално тече, тези инструкции се събират отново, но в инструкциите за ниско ниво, които се прочитат от използвания. По време на процеса на компилация инструкциите и програмата се проверяват да не правят нещо необмислено и опасно.

Програмите на Microsoft.NET са съставени от отделни компоненти, наречени асемблери. Това е съвкупност, която съдържа MSIL код заедно с всички ресурси от данни, които ще са необходими като например изображения, звуци и т.н. Устройството може да бъде самостоятелно или библиотека като по този начин осигурява ресурси за приложение. Windows Phone може да работи с всякакви C# модули и библиотеки, произведени от .NET съвместими компилатори. Това означава, че част от библиотечният код може да бъде написан на друг език като напр. Visual Basic, C++ или F#. Това автоматично означава, че ако вече имате библиотеки, написани на тези езици можете да ги използвате в телефонните приложения.

Идеята зад .NET е да се осигури единна рамка за изпълнение на код, който е независим от даден компютърен хардуер или език за програмиране. Стандартите за .NET уточняват как междинният език трябва да изглежда; формата на типовете данни, съхранявани в системата и също така включва дизайна на C# и Visual Basic.NET.

Компиляция Just In Time

Всъщност, когато една програма стартира, нещо трябва да конвертира MSIL (който е независим от конкретния компютърен хардуер) в инструкциите на машинния код, така че процесорът на компютъра да може действително да ги изпълнява. Този процес за компилиране се нарича Just In Time, защото реалният машинен код за набелязаното устройство е съставен на базата на междинните инструкции точно преди стартирането. Начинът, по който програмата се изпълнява в рамките на наблюдаваната среда се нарича код на управление. Компиляцията се изпълнява в момента преди програмата да проработи, т.е. потребителя избира програма от началната страница, MSIL се зарежда от паметта и след това се компилира.

Недостатъкът на този подход е, че вместо само да стартирате даден файл от инструкциите на компютъра, трябва да се свърши още много работа, за да проработят нещата както трябва. Необходимо е да се зареди междинен език, който да се компилира в машинен код, след което да наблюдава програмата докато тя е в действие. За щастие, съвременните процесори (включително този в Windows телефона) може да се постигне това, без да забавя нещата.

Плюсът е, че един и същ файл с междинен код може да бъде зареден и изпълнен на всяка машина, която има NET система. Можете да заредите точно същата компилирана програма на мобилно устройство, Windows PC и Xbox 360, въпреки че машините имат напълно различни операционни системи и базов хардуер.

Другото предимство е, че този процес на зареждане може да се разшири, за да гарантира, че програмите са основателни. .NET предвижда механизми, чрез които програмите могат да бъдат "подписани" с използването на криптографски техники, които затрудняват лошите хора да създадат "фалшиви" версии на кода Ви. В случай, че се превърнете разработчик на пазара на труда, ще имате назначен свой собствен криптографски ключ, който ще се използва за подписване на всички приложения, които сте създали.

В заключение, използването на междинен език позволява употребата на широк спектър от програмни езици. Въпреки че инструментите за програмиране за Windows Phone са фокусирани върху Visual Basic и C#, е възможно да се използва компилиран код от всеки език за програмиране, който има .NET компилатор. Ако имате съществуващи програми в C++ или дори F#, можете да използвате междинни библиотеки от тях в своите Windows Phone проекти. Не трябва да се забравя, обаче, че C# ще бъде нужен за Silverlight или XNA "front end" за тези програми.

Код на управление

Програми, създадени на Windows Phone се изпълняват в рамките на предвиденото пространство, заделено от операционната система. След като напишете Вашите C# програми, е добре да ги изпробвате в безопасна среда в телефона, предоставен от операционната система. На програмата няма да бъде позволен директен достъп до хардуера, което се оказва плюс, тъй като не допуска писането на програми, които спират на правилното функциониране на телефона. Що се отнася до разработчика, за него това не прави особена разлика. Когато една програма извиква метод, който да зареди снимка на екрана, системите, отговарящи за това ще заредят изображението.

Ролята на разработчика

Що се отнася до един разработчик, в случая добри и лоши новини, касаещи всичко това. От една страна, е необходимо да се научи само едни програмен език (C#), който да се прилага върху различни платформи. Програми, които ще се пишат, ще бъдат изолирани от потенциално увреждане от други програми на системата и можете да бъдете сигурни, че софтуерът, който продавате не може лесно да бъде подправен.

От друга страна, обаче, всичко това става за сметка на допълнителна работа от страна на системата, която зарежда приложението. Стартирането на програма не е само зареждането ѝ в паметта и спазването на инструкциите, които съдържа. Програмата трябва да бъде проверена, за да се гарантира, че не е била фалшифицирана и след това да се компилира по метода „Just In Time“ в инструкции, които да изпълняват задачата. Като резултатът от на потребителя ще му се наложи да изчака известно време докато първият екран се покаже, след като е избирал заявлението.

За щастие, най-силното качество на Windows Phone е, че тези закъснения обикновено не са проблем, но въпреки това при по-големи програми, те трябва да се разбият на по-малки части, така че не всичко да се зарежда наведнъж. Но като умни разработчици ние най-вероятно ще направим това така или иначе.

1.5. Разработка на приложения за Windows Phone

Писането на Windows приложения за телефон се пишат по абсолютно същия начин както и други приложения за настолен компютър с Windows ОС. За целта може да се използва Visual Studio IDE (Integrated Development Environment). Отстраняването на дефекти в Windows Phone устройство и на настолния компютър е еднакво лесно. Проектите, които се създават биха могли да споделят компоненти от платформи на десктопа, Windows Phone или дори Xbox.

Уменията за писане на Windows Desktop Silverlight и конзола умения в XNA могат да се съчетаят в писането за телефон. Ако се научите как да използвате Windows Phone, също се научавате как да пишете код за десктоп (Silverlight) или конзола (XNA). Това е страхотна новина, тъй като означава, че бихте могли да пишете (и дори да продавате) програми за Windows Phone, без да се налага да научавате много нови неща. Ако преди сте писали програми за настолни компютри, следващата стъпка към писането за Windows Phone ще Ви се стори много по-малко болезнена, отколкото сте очаквали.

Windows Phone емулатор

Средата за разработване на Windows Phone е снабдена с емулатор, предлагащ примерен Windows Phone, в който можете да се ровите от настолния Ви компютър. Ако притежавате PC система, поддържаща мулти-тъч вход, тя може да се използва с емулатора, за да се изпробват мултитъч жестовете от Windows Phone програми.

Докато емуляторът притежава характеристики, доближаващи се много до истинския телефон съобразно изискванията на софтуера, той не копира изцяло начина на изпълнение на хардуера. Програмите, работещи на емулятора използват мощността на персоналния компютър, която може да бъде в пъти по-голяма отколкото тази на процесора в телефона. Това означава, че въпреки, че можете да изпробвате функционалността на програмите си с помощта на емулятор, получавате само представа за това колко бързо програмата работи и какво потребителя усеща, когато я пусне на реалното устройство.

Емуляторът позволява да се тестват програми, контролирани чрез движение, позволявайки да се изпробва телефона на екрана на компютъра. Входните данни на програмата ще отразяват посоката, която се вижда на екрана. Определен набор от движения могат да бъдат презаписани, за да се проверят наново чрез тестове, използвайки особени движения или жестове.

Налице е също разпоредба за местоположението на емулятора. Чрез карта може да се избере къде да се разположи телефона. Друга възможност е да се създадат входни пътеки, които телефона да следва, след което те да бъдат заменени, така че да се създаде разстояние и да се тестват приложения, които взимат предвид пространството.

Достъп до Windows Phone съоръжения

Платформата Windows Phone осигурява библиотека от софтуерни ресурси, които позволяват на Вашите програми да се възползват от функциите, предоставени от самото устройство (камера, пускане на телефонни разговори и дори изпращане SMS съобщения). Също така те могат да се възползват от GPS ресурсите на телефона, за да Ви позволят да създавате приложения, осъзнати в мястото и пространството. Съоръженията са разделени в Launchers, които позволяват трансфера на програма към друго приложение и Choosers които използват системата, за да изберат елемент и след това да възвърнат контрола върху програмата. По-късно в този текст ще видим как да ги използваме.

Windows Phone свързаност

Както се подразбира, програмите на телефона Windows имат изключително добра свързаност. Заявленията на телефона могат да се възползват от TCP/IP протокола, за да се свържат към сървърите в интернет. Програмите могат да използват уеб услуги, а също и базирани на REST хоствани сесии. Ако не сте сигурни какво означава всяко нещо, не се притеснявайте, защото по-късно в текста ще изследваме използването на мрежовите услуги.

Silverlight и XNA развитие

Съществуват основно два варианта за развитие на Windows Phone. При създаване на заявление (напр. текстообработваща програма, електронна поща или калкулатор за сирене) може да се използва Silverlight. Това осигурява широк набор от функции само за създаване на такива програми. Ако желаете да създадете игра след това, можете да използвате XNA, предлагащ всички удобства за създаването на 2D и 3D игри с графики с висока производителност.

Но не сте принудени да работите по този начин. Можете да създадете свой калкулатор в XNA или да създавате успешно прости игри (напр. ребуси) в Silverlight.

Чрез нов проект във Visual Studio може да се избере вида на приложението, което се създава.

Комбиниране на Silverlight и XNA

Едно единствено заявление може да съдържа както Silverlight така и XNA основни характеристики. Можете да възприемете това като Silverlight програма, съдържаща страница с XNA игра. Това много улеснява създаването на менюта и потребителски интерфейс в играта на Silverlight и пускането ѝ в XNA. Дори могат да се добавят Silverlight контроли отгоре на пуснатата вече XNA игра. Ние ще проучи как да направите това в глава 8.

Съхранение на данни в Windows Phone

Полезните приложения трябва да могат съхраняват данни. Част от тези данни ще бъдат прости неща, например конфигурационни данни като размера на дисплея. Други елементи ще бъдат по-големи, вероятно достиганати резултати или мястото, до което играчът е достигнал. Тогава може да има нужда от още повече информация, която да бъде структурирана по определен начин, например клиентска или продуктова база данни. Windows Phone осигурява подходящо съхранение на всички тези нива.

Виртуална файлова система

Този начин на съхранение е наречен така, тъй като пространството за съхранение на едно заявление е изолирано от всички останали. Когато една програма се инсталира на Windows Phone, се дава достъп до пространството за съхранение, което е отделено от всички останали. Там има файлова система, която може да осигури взаимовръзка между име и стойност (например ColourScheme : Purple) за настройките на програмата. Заявленията могат да образуват завършени файлови йерархии в рамките на виртуалната файлова система съобразно техните нужди.

Програма, наречена Isolated Storage Explorer може да се използва, за да се видят папки и файлове в отделената памет при проектирането на приложение, така че да увери, че се съдържат правилните данни.

Локална база данни

Базата данни е колекция от данни, която се управлява от програма, наричаща се по същия начин. Използва се за създаване на програми, които съхраняват и обработват огромни количества данни и улесняват много приложенията, предоставяйки отговори на запитвания като "Каж ми кои от нашите клиенти от New York купи чифт обувки през ноември". Windows Phone притежава вградена база данни, което много улеснява създаването на приложение, съхраняващо структурирани данни.

Всяко Windows Phone приложение също има достъп до една SQL база данни, където могат да се съхраняват големи количества структурирна информация. Заявлението може да създава заявки с помощта на LINQ (Language INtegrated Query) библиотеки за извличане и променяне на данни.

Действителният файл на базата данни се държи в отделна памет на заявлението. В глава 6 ще разгледаме как се използват бази данни и LINQ.

Инструменти за разработка

Инструментите, които са Ви необходими, за да започнете са безплатни. Можете да изтеглите копие на Windows Phone SDK и да започна да пишете приложения за Windows телефон буквално в рамките на минути. Разработчиците, които имат по-напреднали и платени версии на Visual Studio 2010 могат да използват всички екстри на своите версии в развитието на мобилно устройство, чрез добавяне на Windows Phone SDK плъгин за системи си. Копие на Windows Phone SDK може да се изтегли от create.msdn.com

Профилирано изпълнение

Приложения за Windows PC обикновено не са ограничени от производителността на централния процесор. Абсолютната мощност на процесорите в настолните компютри и лаптопи води към използването на доста неефективен код. Въпреки това, е важно да направим нашите програми бързи и ефикасни за Windows телефона, за да се осигури както доброто ползване от страна на потребителите, така и да сме сигурни, че програмите не използват твърде много от енергията на батерията.

Windows Phone SDK включва използването на средство за показването на ползите от употребата на устройството, което може да се използва, за да се видят изискванията, които приложения отправят към процесора, а също така да се разбере къде програмите се записват. По този начин става възможно определянето на методи, които да оптимизират програмата, за да я направят по-производителна.

Windows Marketplace

Windows Marketplace е мястото, където се продават програми, които вече са създадени. Моделира се и се управлява от Microsoft, за да се гарантира, че заявленията отговарят на определени минимални стандарти на поведение. Ако желаете да продадете приложение, трябва да сте регистриран разработчик и да изпратите своя програма за одобрение.

Цената за регистриране като разработчик е \$ 99, но студентите могат да се регистрират безплатно чрез програмата DreamSpark. Разработчиците могат да регистрират своите телефонни устройства на Windows като "development" устройства. Visual Studio може да предостави на приложения към разработващо устройство и дори можете да преминете през програмите, които са пуснати от самото устройство.

Бихте могли да разпространявате както безплатни така и платени приложения. Регистрираният разработчик може да изпрати до 100 безплатни приложения за одобрение в дадена година. Ако искате да разпространявате повече от 100 безплатни приложения, трябва да заплатите \$20 за всяко допълнително такова. Имате възможността да създавате приложения с "демо" режим,

която може да се превърне в "цяла" версия, когато потребителят ги купува. Програмата Ви може да осведомява докато работи кога използва демо версията и кога не.

Когато изпратите приложение, което да се продава на пазара, то ще премине през процес на одобрение, за да се потвърди факта, че то не прави нищо забранено и че работи по начин, който потребителите очакват. Ако процесът на одобрение се провали, ще Ви бъде даден диагностичен доклад. Преди да изпратите приложение в системата, първо трябва да прочетете насоките за сертифициране на пазара.

Тестващ инструмент

В Windows Phone SDK се съдържа инструмент за тестване, който може да се използва за предварителното тестване на приложения преди те да бъдат изпратени на Marketplace. Този инструмент автоматично се синхронизира с процеса на одобрение от пазара, така че ако той се променя инструментът ще бъде актуализиран, за да съвпадат. Част от тестването е автоматично, но инструментът също описва всеки от употребяваните тестове, които ще направят екипа на одобрение. Това значително подобрява шансовете ни да изпратим приложение, което ще бъде одобрено още първият път.

Частни бета версии

Чудесен начин да се разбере какво е качеството на дадено приложение или игра е да се даде на група хора и да се види какво ще кажат те. Можем да създадем "invitation only" пропуск на даден продукт, така че 100 души да получат линк за сваляне на нашата нова програма. Това позволява на хората, които изпробват програмата да дадат обратна връзка в рамките на 90 дни.

Какво научихме

- 1 . Windows Phone е мощна изчислителна платформа.
- 2 . Всички телефонни устройства Windows разполагат с основна спецификация. Това включва определен размер на дисплей, капацитивен сензорен вход, който може да засече минимум четири точки, Global Positioning System Support, 3D графично ускорение, камера с висока резолюция и достатъчно памет за програма и съхранение на данни.
- 3 . Windows Phone устройството е свързано към Windows PC с помощта на Zune PC софтуер, който предвижда система за управление на медии за PC, а също позволява на медиите да бъдат синхронизирани с телефона.
- 4 . Windows Phone системите могат да се възползват от мрежови услуги, за да получават уведомления, да определят позиция им и да търсят из мрежата.
- 5 . При разработването на програми за Windows Phone на Zune софтуерът се използва за прехвърляне на програми в телефона за тестване. Софтуерът Zune също се използва за надграждане на фърмуера на телефона.

6 . Операционната система Windows Phone поддържа пълна многозадачност, но за да се представи в най-добрата си светлина само от един потребител трябва да бъде активен в дадения момент. Въпреки това, приложения могат да създават свои собствени "фонові агенти", които да могат да изпълняват специфични задачи, когато приложението не работи. Windows Phone също така предвижда "Fast Application Switching" , който съхранява приложенията, така че те да могат да се възобновят много бързо.

7 . В Windows Phone работят програми, които са събрани в Microsoft Intermediate Language (MSIL), който се компилира вътре в телефона точно преди стартиране на програмата. Самите програми се пускат в среда под наблюдение, която не им позволява да се намесват в работата на телефона.

8 . При разработването на софтуер за Windows Phone той може да създаде чрез Silverlight и XNA приложения. Те са написани на C# с помощта на Visual Studio 2010. Програмистите могат да използват Windows Phone емулатор, който работи на Windows PC и осигурява симулация на околната среда Windows Phone.

9 . Програмите да имат достъп до всички съоръжения на телефона като набиране, изпращане на SMS съобщения и т.н.

10 . Windows Phone SDK може да се използва за създаването на приложения за Windows Phone. Може безплатна да се изтегли от create.msdn.com Въпреки това, за да се разработват приложения на устройство за телефон е необходима регистрация като Windows Phone Developer, което струва \$ 99 на година, но е безплатна за студентите чрез инициативата на Microsoft DreamSpark . Регистрираният разработчик може да качи и продава своите приложения на Windows Phone Marketplace.

Глава 2. Въведение в Silverlight

Потребителският интерфейс е луксозно име за това, което хората всъщност виждат, когато използват вашата програма. Той се състои от бутони, текстови полета, етикети и снимки, с които потребителят работи, за да си свърши работата. Част от работата на програмиста е да се създаде този "преден край" и след това задава подходящо поведение зад дисплея, за да позволи на потребителя да управлява програмата и да получи това, което иска от нея. В този раздел ще разберете за Silverlight и как да го използвате, за да създадете потребителски интерфейс за нашите програми.

2.1. Програмен дизайн със Silverlight

Оказва се, че повечето програмисти не са толкова добри в проектирането на атрактивни потребителски интерфейси (въпреки че аз съм сигурен, че сте). В реалния живот, една компания ще наеме графични дизайнери, които ще създадат артистични изгледи. След това ролята на програмиста ще бъде да се сложи код зад тези екрани, за да се получи това, което се иска. Silverlight признава този процес чрез прилагане на много силно разделение между дизайна на екран и код, който се контролира от него. Това улеснява програмиста да се създаде първоначален

потребителски интерфейс, който впоследствие се променя от дизайнера в много по-привлекателен. Също така е възможно за един програмист да се вземе цялостно проектиране на потребителския интерфейс и след това се поберат поведенията, които стоят зад всеки един от компонентите на дисплея.

Инструменти за разработка

Проектантът на потребителски интерфейс използва инструмента за Expression Blend и програмистът използва Visual Studio инструмент за създаване на кода. Системата Silverlight осигурява наистина лесен начин да се съчетаят дизайна на потребителския интерфейс с код. Един добър начин да се работи, е за програмиста да използвате "контейнер" дизайн, за да се създаде поведението на програмата и след това да се включат окончателните дизайни по-късно в процеса на производство.

В Windows Phone SDK (Software Kit развитие) включва версии на Visual Studio и Expression Blend, които да се използват по този начин.

The Metro Design Style

От гледна точка на дизайна на хубаво нещо за Windows Phone е, че той носи със себе си цял набор от насоки за дизайна, които са посочени като "Метро". Това определя как контролите би трябвало да изглеждат и се установява набор от критерии, на които вашите приложения трябва да отговарят, ако искате те да бъдат "добре изглеждащи" приложения Windows Phone. Този режим на стил е пренесен във вградените компоненти, които са предоставени за да можете да използвате във вашите програми. Добрата новина от това е, че ако използвате бутоните, TextFields и други компоненти, които се доставят с Visual Studio ще бъде автоматично се придържа към насоките за стил. Това е от голяма полза на тези от нас, които не са много добри в дизайна, защото това означава, че ние сме само някога ще използвате неща, които изглеждат добре. Разбира се, че може напълно да замени свойствата на доставените компоненти, ако наистина искате лилаво текст на оранжев фон, но аз не бих наистина посъветва това.

Там всъщност е документ за Metro стил, който може да се чете, ако наистина искате да знаете как да направите правилното "търсещите" програми. Това си струва един поглед, ако искате да публикувате вашите програми в Marketplace, където хората ще имат определени очаквания за това как нещата трябва да изглеждат и работят. Можете да намерите ръководството на стила тук: msdn.microsoft.com/en-us/library/hh202915.aspx

За целите на този курс ние ще използваме Silverlight инструменти за проектиране в Visual Studio. Те не дават всичко графичното богатство, че можете да постигнете с Expression Blend, но и за програмисти, те са повече от достатъчни. Това трябва да сте сигурни, че нашите приложения се придържат към насоките на метрото и са чисти и лесни за употреба.

Silverlight елементи и обекти

От гледна точка на програмиране оглед на всеки от елементите на екрана на дисплея всъщност е софтуерен обект. Ние са се сблъскали с това преди. Един обект е една буца поведение и данни. Ако бяхме създаване на заявление да се грижат за банкови сметки, можем да създадем един клас да държи информацията за профила:

```
public class Account
{
    private decimal balance ;
    private string name ;

    public string GetName ()
    {
        return name;
    }

    public bool SetName (string newName){
    {
        // Final version will validate the name
        name = newName;
        return true;
    }

    // Other get and set methods here
}
```

Към този клас се държи на размера на парите на титуляра на сметката има (в членът на данни, наречена баланс), както и името на титуляра на сметката (в наречен на името на член на данни). Ако искам да направя нова инстанция Account може да използва новата ключова дума:

```
Account rob = new Account();
rob.SetName("Rob");
```

Това прави ново копие на профила, който е посочен от референтната ограби. След това тя поставя член-име на този профил, за да извърши операцията "Rob". Ние я използваме за създаване на обекти, които съответстват на неща, които искаме да съхраним. Когато правим игри, бихме могли да си измислим клас Sprite да държи снимката на спрайт на екрана, позицията на спрайт, и друга информация. Обектите са чудесен начин да се представят нещата, с които искаме те да работят. Оказва се, че обектите са като елементи на един дисплей. Всъщност, кутия за показване на текст върху екрана ще има свойства, като например позицията на екрана, цвета на текста, на самия текст и така нататък.

Помислете за следното:



Това е много проста програма за Windows Phone, наречена "Добавяне на Machine". Можете да използвате, за да се представят много прости суми. Просто въведете две числа в две текстови полета в горната част и след това натиснете бутона за равенство. След това ви награждава със сумата на тези две числа. В момента тя ни показва, че 0 плюс 0 е 0. Всеки отделен елемент на екрана се нарича UIElement или User Interface елемент. Отивам да се обадя тези неща елементи от сега нататък. Има седем от тях на екрана по-горе:

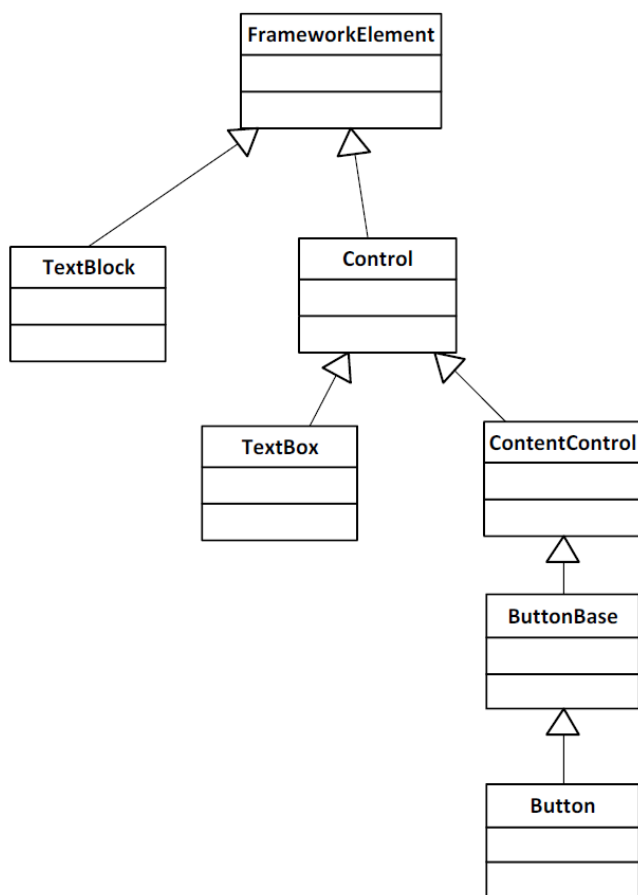
1. Малкият заглавието "Добавяне на Machine". Това е известно в насоките за Windows Phone стил като "Application дял".
2. По-голямата титла "Добави". Това е известно като "заглавната страница".
3. Горната текстовото поле, където мога да въведете число.
4. А текстова бележка държите знак +.
5. Долната текстовото поле, където мога да въведете друг номер.
6. Бутон, който натиснете, за да извършите сумата.
7. А резултат от изображението, която се променя, за да покаже резултат, когато бутонът е натиснат.

Всеки един от тези елементи, има особена позиция на екрана, конкретен размер на текста, както и много други свойства. Ние можем да променим цвета на текста в текстово поле, независимо дали тя се подравнява в ляво, дясно или център на полето за обгръщания и много други неща също.

Има три различни вида елемента на екрана:

1. TextBox - позволява на потребителя да въвежда текст в програмата.
2. TextBlock - блок от текст, който само предава информация.
3. Button - нещо, което може да натиснете, за да предизвика събития в нашата програма.

Всъщност, можете да пробие на свойствата на всеки един от тези елементи, определени в два вида, тези, че всички елементи трябва да имат, например позиция на екрана, и тези, които са специфични за този тип елемент. Например само TextBox трябва да запише позицията на курсора, където текст се вписва. От софтуерна гледна точка на дизайн, това е наистина добра употреба за йерархия клас.



Отгоре можете да видите част от йерархията, че Silverlight дизайнери построени. В горната клас се нарича FrameworkElement. Тя съдържа цялата информация, която е обща за всички контроли на екрана. Всеки от останалите класове е дете от този клас. деца вземете всички на поведението и свойствата на тяхната майка клас и след това се добавят някои от техните собствени. Въсъщност дизайнът е малко по-сложна, отколкото е показано по-горе, FrameworkElement клас е дете на клас, наречен UIElement, но тя показва принципи, които стоят зад контролите.

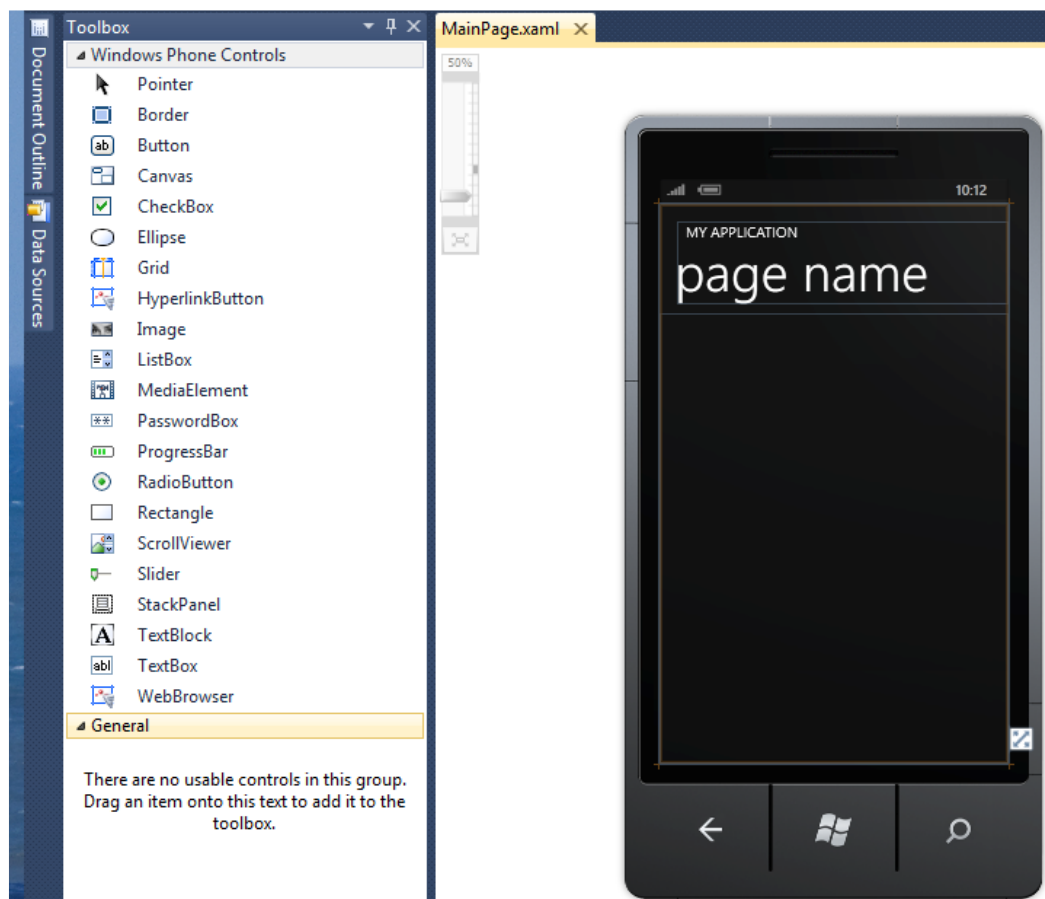
Създаване на клас йерархия като това има много предимства. Ако искаме персонализиран вид виждаш можем да се разшири клас TextBox и добавете свойствата и поведението, което имаме

нужда. Що се отнася до системата за Silverlight е зарежен, че може да лекува всички контроли на същото и след това да поиска всяка контрола да се изготвят по начин, съответстващ на този, компонент.

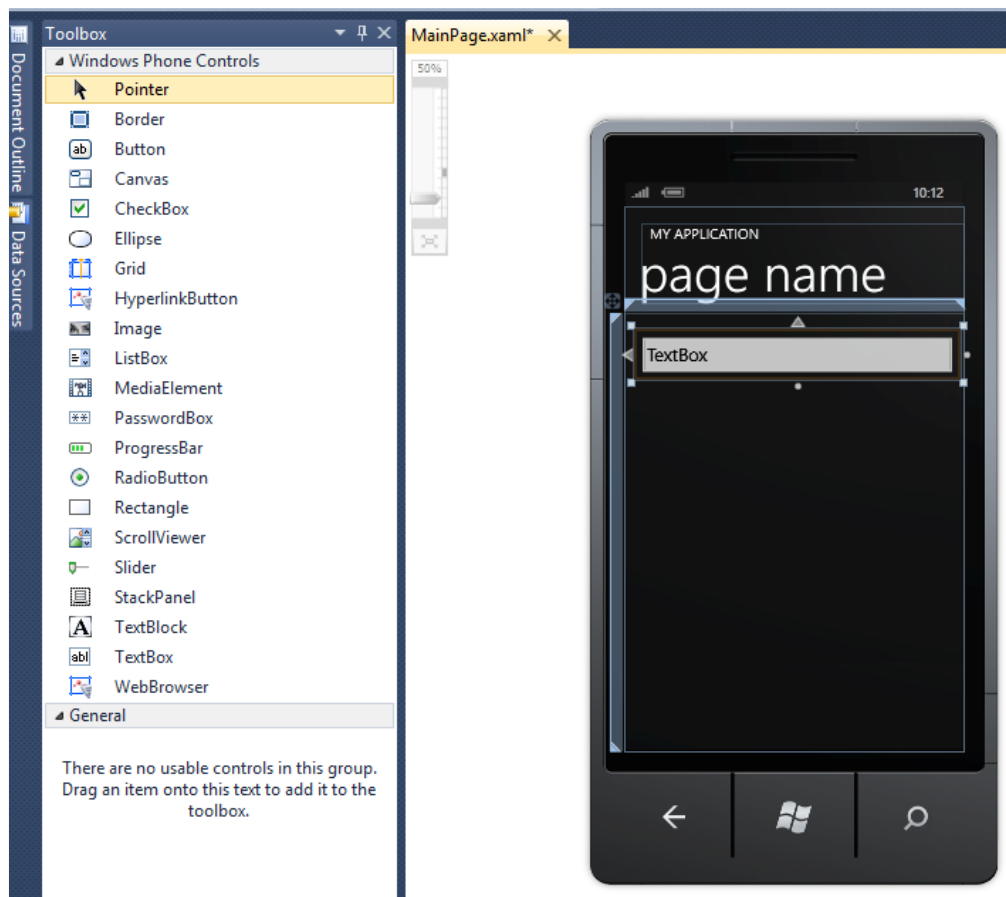
Така че, не забравяйте, че когато ние се приспособяват неща на страница дисплей и създаване и манипулиране на контроли ние сме наистина само с промяна на свойствата на обекти, както и ние ще се промени името и баланс стойности на обект на банкова сметка. Когато ние проектираме на Silverlight потребителски интерфейс тръгнахме данните вътре елементи на дисплея, за да ги позиционирате на дисплея. След това ще разберете как да направите това.

The Toolbox and Design Surface

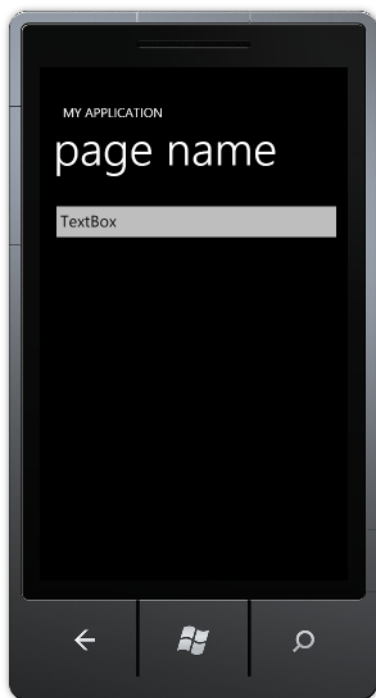
Можем да започнем от разследване как е бил създаден Добавянето машина горе. Оказва се, да бъде много лесно (ще отнеме наистина подробен преглед на Visual Studio в следващия раздел). Когато ние създаваме чисто нов проект Silverlight ние ще получиш празна страница и ние можем да отворим кутия с инструменти, която съдържа всички контроли, които бихме могли да искате да добавите към страницата:



Ние може да събере на потребителския интерфейс, като просто плъзгане контролите от кутията с инструменти върху повърхността на проектиране.



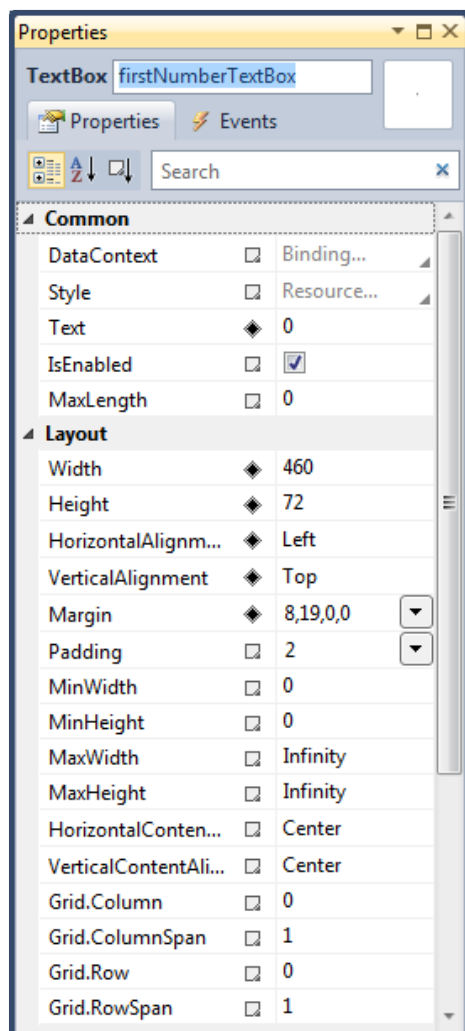
Над показва ефекта на плъзгане на TextBox извън зоната Toolbox и го пуснете върху Phone страницата Windows. Ако стартирате програмата, сега можем да видим нашия виждаш на екрана:



Това е Windows Phone емулятора работи нашата програма, която се показва в текстовото поле на екрана. Ако някога сте се занимавали с Windows Forms развитие (може би с помощта на по-ранни версии на .NET), тогава всичко ще бъде много запознат с теб. Ако не сте тогава той е бърз и лесен начин да се създаде потребителски интерфейс. Защото всички контроли са вече оформена в стила Metro, точно като Phone потребителския интерфейс на Windows, вие сте автоматично изграждане на приложение, което изглежда като "истинска" един. Дизайнерът има много хубави черти, които го правят много лесно да се приведе в съответствие своите контроли също, което е хубаво.

Managing Element Names in Visual Studio

След като ние сме вложили някои елементи на екрана, ние трябва да зададете имената на тях като на разумни стойности. Дизайнерът ще даде на всеки елемент безсмислена име като "TextBox1", което ми харесва да се промени нещо, което има смисъл в контекста на системата, която се създава. Ние може да променя свойствата на нещата на екрана, като щракнете върху елемента в дизайнер и след това търсенето на прозореца на Properties в Visual Studio за тази позиция.



Над можете да видите прозореца със свойства в горния текст полето на страницата. Името на елемент се дава в най-горната част на прозореца. Промених името на тази кутия да firstNumberTextBox. Никога няма да познаеш какво текстовото поле втората се нарича. Имайте предвид, че името на един имот в този контекст, всъщност ще определи името на променливата, декларирана в рамките на програмата за добавяне на машина. С други думи, като резултат от това, което съм направил по-горе сега ще бъде следното изявление в моята програма някъде:

TextBox firstNumberTextBox;

Visual Studio изглежда след обявяването на действителните C # променливи, които представляват елементи на дисплей, които ние създаваме и затова не е нужно да се притеснявате за това къде всъщност горното твърдение е. Ние просто трябва да се помни, че това е как работи програмата.

Properties in Silverlight Elements

След като сме дали нашето TextBox променлива подходящо име можем да продължим нататък, за да го даде на всички качества, които са необходими за това приложение. Ние също може да се промени много свойства за текстовото поле, включително ширината на текстовото поле, на

границата (което определя позицията) и така нататък. Стойностите, които виждате в прозорците горе имоти са тези, които отразяват текущото положение и размера на елемента на екрана. Ако плъзнете елемента около стойностите на полетата ще се променят. Ако променя стойностите в прозореца по-горе ще видите движение елемент в повърхността на проектиране. Отново, ако сте на Windows Forms вид на човек, който ще види нищо странно тук. Но не трябва да забравяте, че всичко, което правим е да промените съдържанието на даден обект. Съдържанието на прозореца със свойства ще се променят в зависимост от това каква позиция сте избрали

Silverlight properties and C# properties

Когато говорим за "свойствата" на Silverlight елементи на страницата (например текста показват в TextBox), ние всъщност говорим за стойността на имотите в класа, който въвежда текст полето. С други думи, когато дадена програма съдържа декларация, като например:

```
resultTextBlock.Text = "0";
```

това ще доведе до метод Set да тече вътре в resultTextBlock обект, който определя текста на TextBlock на съответната стойност. В този момент си струва освежаващо нашето разбиране за имоти в C #, така че да получите по-добро разбиране на това, което се случва тук.

C # класове и свойства

C # класове могат да съдържат данни на потребителя (на името на лицето, притежаващи дадена банкова сметка) и поведение (методи, наречени GetName и SetName че нека програма разберете това име и да я промените). Имоти осигуряват начин за представяне на данни в рамките на дадена класа, която е по-лесно да се работи с, отколкото използването Взemi и Set методи. Те се използват широко при управление на Silverlight елементи, и затова си струва опреснителен в този момент. А имот е член на клас, който притежава стойност. Видяхме, че можем да използваме променлива потребител да направи този вид на нещо, но ние трябва да се направи стойността потребител публично. Така например, банката може да ни попитате, за да следите на членовете на персонала. Един от елементите, които те могат да искат да притежават е възрастта на притежателя на банковата сметка. Можем да го направим по този начин:

```
public class Account
{
    public int Age;
}
```

Класът съдържа публична потребител. Мога да се сдобият с този потребител по обичайния начин:

```
Account s = new Account ();
s.Age = 21;
```

Имам достъп до обществена представител на клас директно; само чрез даване на името на члена. Проблемът е, че ние вече са решили, че това е лош начин да управляваме нашите обекти. Няма нищо, за да спре неща като:


```
s.Age = -100210232;
```

This is very naughty, but because the Age member is `public` we cannot stop it.

Създаване Get и Set методи

За да получите контрол и направи полезни неща, които могат да създадат и да получите методи, които са `public`. Те осигуряват достъп до съответния член в една управлявана начин. Ние след това направи членът Age `private` и никой не може да подправяте с него:

```
public class Account
{
    private int age;
    public int GetAge()
    {
        return this.age;
    }
    public void SetAge( int inAge )
    {
        if ( (inAge > 0) && (inAge < 120) )
        {
            this.age = inAge;
        }
    }
}
```

Сега имаме пълен контрол над нашата собственост, но ние трябваше да се напише много допълнително код. Програмистите, които искат да работят с възрастта стойността на предприятието трябва да се обаждат методи:

```
Account s = new Account ();
s.SetAge(21);
Console.WriteLine ( "Age is : " + s.GetAge() );
```

Използване на свойства

Имоти са начин за вземане на управлението на данни, като това малко по-лесно. Аг възраст имот за класа StaffMember ще бъдат създадени, както следва:

```

public class Account
{
    private int ageValue;

    public int Age
    {
        set
        {
            if ( (value > 0) && (value < 120) )
                ageValue = value;
        }
        get
        {
            return ageValue;
        }
    }
}

```

Стойността на възраст вече е създаден като собственост. Забележете как са там и да получите части към имота. Това се равнява директно към телата на получават и определени методи, които съм написал по-рано. Наистина Хубавото свойства е, че те се използват само като член на класа е:

```

Account s = new Account ();
s.Age = 21;
Console.WriteLine ( "Age is : " + s.Age );

```

Когато имотът Age, се дава стойност на `set` кодът се изпълнява. Ключовата дума `value` означава "нещо, което се определя". Когато имотът Age се чете кода на `get` се изпълнява. Това ни дава всички предимства на методите, но те са много по-лесно да се използва и да се създаде.

Валидиране на данни в свойства

Ако зададете възрастова стойност за невалиден едно (например ние се опитваме да се определи възрастта до 150) на набор поведението горе, ще изпълнява валидиране и отхвърли тази стойност (никой над 120 е позволено да има сметка в нашата банка), оставяйки възраст, както е било преди. А програма, която е с помощта на нашата възраст имот няма как да знае дали една задача до имота е отказано, без то действително проверява, че задачата е работил.

```

Account s = new Account ();
int newAge = 150;
s.Age = newAge;
if (s.Age != newAge )
    Console.WriteLine ( "Age was not set" );

```

Кодът по-горе определя възрастта на 150, което не е разрешено. Въпреки това, кодът ще се изискват изпитванията, за да видите, ако стойността е зададена. С други думи, той е до потребителите на вашите имоти да се уверите, че стойностите са зададени правилно. В случая на метод `Set` самият метод може да се върне невярно да покаже, че устройството е неуспешно, тук на ползвателя на имота трябва да направите малко повече работа.

Множество начини за четене на свойства

```
public int AgeInMonths
{
    get
    {
        return this.ageValue*12;
    }
}
```

Това е нов имот, наречен AgeInMonths. Тя може да се чете само, тъй като тя не осигурява набор поведение. Въпреки това, той се връща на възраст в месеца, въз основа на същия източник стойност като се използва от друга собственост. Това означава, че можете да предоставите няколко различни начина за възстановяване на една и съща стойност. Вие също може да осигури само за четене свойства, като оставя извън зададеното поведение. Напиши само имоти са възможни също и ако те оставя на `get`.

Свойства и уведомления

Може да се зададе въпроса "Защо ние използваме имоти в Silverlight елементи?" Логично е да ги използват в банкова сметка, където искам да бъде в състояние да се защитят данните вътре моите обекти, но в Silverlight програма, където мога да сложа всеки текст ми харесва вътре в TextBlock, изглежда, че няма смисъл да се налага проверка на входящата стойност. Всъщност използването на този код, ще забави процеса на настройка. Така че, като направи стойност Текст публично низ можем да направим програмата по-малки и по-бързо, което трябва да бъде нещо добро. Нали така? Е, нещо такова. Само че, когато ние се промени текста на TextBlock бихме искали текстът на Silverlight страница в предната част на потребителя да се промени, както добре.

Това е как нашата добавяне машина ще покаже резултата. Ако една програма само променя стойността на член на данни няма да има никакъв начин системата Silverlight би могъл да знае, че съобщението на екрана трябва да се актуализира. Въпреки това, ако членът Текстът е имот, когато дадена програма да го актуализира набор поведение в TextBlock ще получите да тече. Кодът в комплект поведение може да актуализира съхранената стойността на текстовото поле и то също може да доведе до актуализация на дисплея, за да направи новата стойност видими. Имоти осигуряват средствата, чрез които обектът може да получи контрол, когато стойността вътре в обекта се променя, и това е изключително важно. Простото твърдение:

```
resultTextBlock.Text = "0";
```

Page Design с Silverlight

Ние може да завърши на страницата на нашия Добавянето машина чрез плъзгане повече елементи върху екрана и определяне на техните свойства. Не забравяйте, че първото нещо, което правя след плъзгане на елемент в страницата е зададено името на този елемент. Един от първите симптоми на зле написани програма (за мен) е една страница, която съдържа много елементи, наречени "Button1" и "Button2".

От последователността горе Надявам се, че може да се види, че разработването на страница изглежда доста лесно. Вие просто трябва да плъзнете елементите върху страницата и след това да ги създаде с помощта на свойствата на всеки един. Ако ви се чете на езика на Silverlight ще откриете, че можете да дадете елементи графични свойства, които могат да ги направят прозрачни, да добавяте изображения към техния произход и дори да ги анимирате по екрана. В този момент ние са се преместили далеч отвъд програмирането и встъпват в областта на графичния дизайн. И аз ви пожелавам най-доброто от късмет.

2.2. Разбиране на XAML

горната част видяхме, че Silverlight дизайн може да се разбие на някакъв елемент манипулация с помощта на Visual Studio дизайнер и някои настройка собственост върху елементите , които сте създали . След като знаете как да направите тези неща , можете да създадете всеки потребителски интерфейс ви трябва.

Съвършено вярно е, че можете да работите по този начин, и вие може дори да мисля за вземане на всяка програма, която някога се наложи , без да докосвате XAML , макар че аз мисля, че вие ще се лишите от някои от най-силните характеристики на Silverlight .В момента всичко изглежда доста разумно и всички ние сме доста щастливи (поне така се надявам) . Така че това би било добър момент да обърка всички и да започне да се говори за XAML . Буквите стоят за " Extensible Application Markup Language" , който съм сигурен, че прави всичко ясно. Или пък не. XAML е езикът , използван от Silverlight , за да опише това, което една страница трябва да изглежда така .

Защо трябва XAML ?

Може би се чудите защо трябва XAML . Ако сте използвали Windows Forms преди може би си мислите , че сте успели много добре без този нов език, да пречи . XAML предоставя добре дефинирана връзка между външния вид на заявлението (свойствата на елементите на дисплея, които потребителят вижда) и поведението на заявлението (това, което се случва зад дисплея, за да направи работата за кандидатстване) .Разделението е добра идея, защото , както вече споменах, не съществува гаранция, че един добър програмист ще бъде един добър графичен дизайнер. Ако искате да направите добре изглеждащи приложения , което наистина трябва да има дизайнери , така и за програмисти, работещи на вашето решение. Това води до проблем с това, че в идеалния случай искате да се разделят работата на две , а вие не искате програмист не може да направи нищо, докато на графичния дизайнер е завършил създаване на всички екрани на менюто.

XAML решава тези два проблема. Веднага след като изискванията на потребителския интерфейс са изложени (колко текстови полета, колко бутони, което е там на всяка страница и т.н.), след това на дизайнера може да работи върху това как всеки трябва да изглежда, а програмистът може да се съсредоточи върху това, което те правя. Файловете XAML съдържат описание на компонентите на потребителския интерфейс и дизайнерът може да работи върху външния вид и разположението на тези елементи към съдържание сърцето им, докато програмистът получава една с вземане на кода, който върви зад тях. Няма нужда за програмиста да даде на дизайнера техните кодови файлове, тъй като те са отделни от дизайна, както и обратното.

Съдържание XAML файл

XAML файл за дадена страница на дисплея ще съдържа конструкции, които описват всички елементи по него. Всяко описание ще съдържа набор от именувани свойства. Линията на XAML, който описва първата ни TextBox е, както е дадено по-долу:

```
<TextBox Height="72" HorizontalAlignment="Left" Margin="8,19,0,0"
Name="firstNumberTextBox" Text="0" VerticalAlignment="Top"
Width="460" TextAlignment="Center" />
```

Ако сравните информацията в прозореца на Properties със стойностите в XAML горе ще откриете, че всички те се подредят точно . Името на текст полето е firstNumberTextBox , ширината е 460 и така нататък.

Ако се чудите как на страницата и прозореца на свойства държат сами синхронизирани то е, защото те и двете използвани за описание на файла XAML за страницата. Когато се движат нещата около страницата на редактора актуализира XAML с новите позиции, Прозорецът за свойства чете този файл и актуализира своите стойности съответно .

XAML е описан като декларативен език. Това означава, че той просто ни разказва за неща . Той е проектиран да бъде разбираема от хора , поради което всички имоти над имат смислени имена. Ако искате , можете да редактирате текста, съдържанието на файловете, XAML рамките на Visual Studio и промените както на външния вид на дизайнера , а също и стойностите на свойствите.

XAML Оказва се, да бъде много полезен. След като получите цаката на информацията, използвана за описание на компонентите се оказва, да бъде много по -бързо, за да добавите нещата към страница, и да ги движи само с редактирането на текста във файла XAML , отколкото плъзгане развоя на мача или движещи се между стойността на имотите . Намирам , че е особено полезно , когато искам голям брой подобни елементи на екрана. Visual Studio е наясно със синтаксис , използван , за да опише всеки тип елемент и ще осигури Intellisense помощ подкрепя като отидеш заедно,

Когато е построена заявление файла XAML се превръща в един файл , съдържащ инструкции ниско ниво , които създават реалните компоненти на дисплея , когато програмата работи . Това означава, че за обявяване на контрол вътре в XAML файл за дадена страница ще означава, че контролът ще съществува като обект нашата програма може да използва.

XAML много прилича на XML, на Extensible Markup Language може да сте чували за . Начинът, по който предмети и елементи се изразяват е базиран на XML синтаксис и е доста лесно да се разбере.

Extensible Markup Languages

В този момент може да се чудите какво всъщност е Extensible Markup Language. Е, това е език за маркиране, който е разтегателен. Аз съм сигурен, че помогна. Какво имаме предвид под това е, че можете да използвате правилата на езика, за да създадете конструкции, които описват всичко.

English е много прилича на това. Имаме писма и пунктуация, които са символите на английски текст. Ние също имаме правила (наречени граматика), които определят как да се направи на думи и изречения, и ние имаме различни видове думи. Имаме съществителни, които описват неща и глаголи, които описват действия. Когато нещо ново идва заедно можем да изобретят цял нов набор от думи, за да ги опиша. Някой трябваше да излезе с думата "компютър", когато компютърът е бил изобретен, заедно с фрази като "зареждане", "катастрофа" и "твърде бавно".

XML базирани езици са разтегателни в които можем да измислят нови думи и фрази, които се вписват в рамките на правилата на езика и да използват тези нови конструкции, за да се опише всичко, което сме искали. Те се наричат езици за маркиране, защото те често се използват за описване на подреждането на елементи на една страница. Думата коректурата първоначално е бил използван в печата, когато исках да кажа неща като "Печат името Роб Майлс в много голям шрифт". Най-известният език за маркиране е вероятно HTML, HyperText Markup Language, който се използва от World Wide Web, за да опише формата на уеб страници.

Програмистите често изобретяват свои собствени формати за съхранение на данни с помощта на XML. Като пример, един фрагмент от XML, който описва набор от най-високи резултати, може да изглежда по следния начин:

```
<?xml version="1.0" encoding="us-ascii" ?>
<HighScoreRecords count="2">
  <HighScore game="Breakout">
    <playername>Rob Miles</playername>
    <score>1500</score>
  </HighScore>
  <HighScore game="Space Invaders">
    <playername>Rob Miles</playername>
    <score>4500</score>
  </HighScore>
</HighScoreRecords>
```

Това е една малка XML файл, който описва някои високи записи оценка за система за видео игра. Елементът HighScoreRecords съдържа два рекорда елементи, по един за Breakout и един за Space Invaders. Двете High Score елементи се съдържат в статията HighScoreRecords. Всеки от елементите има собственост, която дава името на играта, а също така съдържа две допълнителни елементи, името на играча и получи резултата, който беше постигнат. Това е доста лесен за нас, за да се разбере. От текста по-горе, че не е трудно да работят на висока оценка за играта Space Invaders.

Линията на самия връх на файла казва каквото иска да чете файла версията на XML стандарта тя се основава на и кодирането на знаците в самия файл. XAML отнема правилата на разширяем език за маркиране и ги използва, за да се създаде език, който описва компоненти на една страница на един дисплей.

```
<TextBox Height="72" HorizontalAlignment="Left" Margin="8,19,0,0"
Name="firstNumberTextBox" Text="0" VerticalAlignment="Top"
Width="460" TextAlignment="Center" />
```

Ако ние сега да разгледаме описанието на TextBox Мисля, че можем да видим, че дизайнерите на XAML току-що създадени имена на полета, които съответстват на техните изисквания.

XML Schema

Стандартът за XML съдържа и описания как да се създаде схема , която описва определен формат документ. Например , в по-горе схемата за информацията за висок резултат ще каже, че класацията трябва да съдържа PlayerName и Резултат имот. Тя може също така да се каже неща, като рекорда могат да съдържат вальор (датата, на която е постигнато високата оценка), но че това не е необходимо за всеки Класация стойност.

Тази система на стандартен формат и схема означава, че е много лесно за разработчиците да създават формати за данни за конкретни цели. Това е подпомогнато и от факта , че има много инструменти за проектиране , за да помогне за създаване на документи и схеми. Рамката на NET . Дори предлага начин , чрез който дадена програма може да спести на обект като форматиран XML документ. Всъщност решение файл Visual Studio е действително складираното като XML документ.

Що се отнася до нас , то е да се припомни , че XML е полезно за този вид на нещо , но в момента искам да се запази акцентът върху себе XAML .

XAML и страници

Файл на XAML може да опише пълен страница на дисплея на телефона Windows . Когато създавате нов проект Phone Windows можете да получите една страница, която съдържа само няколко елемента . Като сложите повече върху страницата файла расте като се добавя всяко описание . Някои елементи работят като контейнери , което означава, че те могат да притежават други компоненти. Те са много полезни , когато искате да се неща , например има Grid елемент, който може да побере набор от други елементи в споразумението за решетка. Кодът на XAML може да съдържа описанията на анимации и преходи , които могат да бъдат прилагани към елементи на страницата, за да направи още по- впечатляващ потребителски интерфейси .

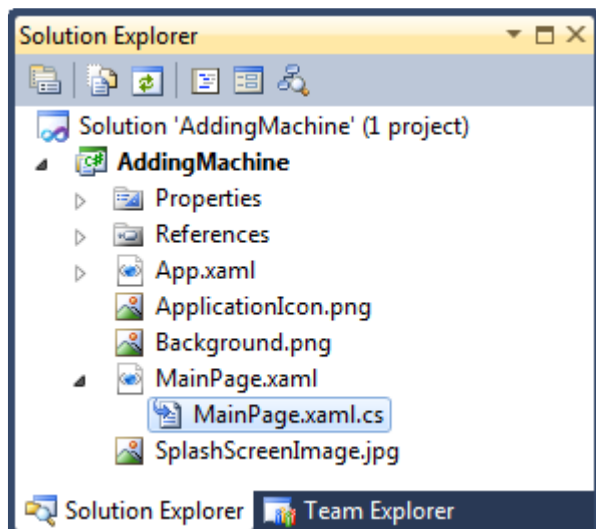
Едно нещо , което трябва да се има предвид, в този момент е, че XAML няма понятие от йерархия. Не е възможно да се създаде елемент XAML , който е получен от друг . Ние правим това, в областта на софтуера , така че ние можем да създадем нов обект, който е подобен на вече съществуваща такава . Ние знаем, че Silverlight елементи са представени в софтуерни гледна точка от обект, който е част от йерархия клас . Въпреки това , когато те са записани във файл XAML всички те са изразени като позиции, които са на едно и също ниво .

Ние няма да прекарват твърде много време в аспектите оформление на XAML , е достатъчно да се каже, че можете да създадете невероятно впечатляваща предна завършва за вашите програми , които използват този инструмент. Има и специален инструмент за дизайн , наречен " Expression Blend " за ползване от графични дизайнери. Въпреки това е важно да се помни , че в края на деня, в който програмата се работи по отношение на предмети , които излагат свойства и методи, които можем да използваме , за да работят с данните вътре в тях . В случай на нашите елементи на потребителския интерфейс , ако променим свойствата на елемент на дисплея потребителят вижда

ще се промени , за да отрази това. Обектите представляват елементи на екрана на нашата програма.

2.3. Създаване на Silverlight Application

Сега, когато знаем, че елементи на екрана са в действителност графичния реализацията на софтуерни обекти, следващото нещо, което трябва да знаете е как да се получи контрол върху тези обекти и да ги направи полезни неща за нас в нашето заявление. За да направите това, ние ще трябва да добавите някои C # код на програмата, която ще се извърши изчислението, че добавянето на нуждите на машините.



Всеки път, когато Visual Studio прави файл XAML, който описва една страница на дисплея на Windows Phone тя също прави файл програма, за да отида с него. Това е мястото, където можем да сложим код, който ще направи нашата молба работа. Ако действително ги накарайте да поглеждат в файлоvi MainPage.xaml.cs горните вие ще откриете, че тя всъщност не съдържа много код:


```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Net;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Animation;
using System.Windows.Shapes;
using Microsoft.Phone.Controls;

namespace AddingMachine
{
    public partial class MainPage : PhoneApplicationPage
    {
        // Constructor
        public MainPage()
        {
            InitializeComponent();
        }
    }
}

```

Повечето от файла е използването на твърдения, които позволяват на нашата програма да се възползват пряко от класове, без да се налага да се даде на напълно оформени името на всеки от тях. Например, вместо да каже `System.Windows.Controls.Button` можем да кажем, Бътн, защото файлът съдържа линията, използвайки `System.Windows.Controls`.

Единственият метод в програмата е на конструктора на класа на Главна страница. Както знаем, на конструктора на клас се нарича, когато се създава инстанция на класа.

All конструктора се е обадите на метод `InitializeComponent`. Ако отидете да погледнем вътре в този метод, вие ще откриете кода, който всъщност създава копия на елементите на дисплея. Този код се създава автоматично за вас от Visual Studio, базиран на XAML, който описва вашия сайт. Важно е, че можете да оставите тази покана, тъй като е и не променя съдържанието на самия метод, тъй като това най-вероятно ще си счупиш програма.

Имайте предвид, че в този момент ние сме дълбоко в " машинното отделение " на Silverlight. Аз съм само наистина ти разказвам за това, така че можете да разберете, че всъщност не е магия тук. Ако само на C# програми сте виждали досега започват с обявяване на основен метод, тогава е важно да се разбере, че няма нищо особено специална за Silverlight един. Има основен метод в основата на Silverlight приложение, то започва процеса на изграждане на компонентите и пускането им на екрана на потребителя да взаимодейства с.

Хубавото, доколкото се отнася до нас, е, че ние не трябва да се притеснявате за това как се създават тези обекти и се показва, ние може просто да използваме инструментите на високо равнище или на лесни за разбиране XAML да се проектира и изгради нашия дисплей.

Изграждане на Заявление

Сега можем да видим как да се създаде потребителски интерфейс за нашия номер добавяне на програма. Ако прибавим всички компоненти и след това да стартирате приложението дори изглежда, че тя може да направи нещо за нас. Ние можем да напишете в цифрите, които искаме да добавим, а дори и да натиснете бутона равнява ако сме искали:



Ако се докосваме вътре елемент TextBox клавиатурата се появява и ние да въведем номера в полето. Ако след това се докосваме никъде другаде на екрана, клавиатурата се движи от пътя. Ние изглежда да имам много поведение за много малко усилие, което е хубаво. Въпреки това, ние трябва да добавите някои бизнес логика на собствената ни сега, за да получите програмата, за да се получи отговор и ще го покаже.

Изчисляване на резултата

В момента програмата ни изглежда добре, но всъщност не направи нищо. Ние трябва да създадем някакъв код, който ще извърши необходимата изчисляване и показване на резултата. Нещо като това.

```
private void calculateResult()
{
    float v1 = float.Parse(firstNumberTextBox.Text);
    float v2 = float.Parse(secondNumberTextBox.Text);

    float result = v1 + v2;

    resultTextBlock.Text = result.ToString();
}
```

Обектите на TextBox излага свойство Text. Това може да се чете от или се записва. Създаване на стойност в свойството Текст ще се промени текста в текстовото поле. Четене на имота Текст позволява нашата програма да прочетете какво е въведена в текстовото поле .

Събития и програми

Ако сте направили всякакъв вид форма ориентираното програмиране, вие ще знаете всичко за събития. Ако не сте тогава не се притеснявайте , сега ние имаме хубав пример за ситуация, когато имаме нужда от тях , а те не са толкова плашещи или иначе. В по-стари времена , преди графични потребителски интерфейси и мишки , една програма, ще обикновено започват в началото , се кандидатира за известно време и след това завърши .

Но сега имаме сложни и богати потребителски интерфейси с много бутони и други елементи за потребителя да взаимодейства с . Думата процесор аз използвам в момента разкрива стотици различни функции чрез бутоните на елементите на екрана и менюто. В тази ситуация би било много трудно да се напише програма , която проверява всеки потребител елемент от своя страна , за да видите , ако потребителят се е опитал да използва това . Вместо това системата чака за тези елементи на дисплея , за да повиши дадено събитие , когато те искат внимание. Учителите правят това през цялото време. Те не отиват кръг класа иска всяко дете от своя страна , ако те знаят отговора. Вместо това те се питат децата да вдигнат ръка . The " ръка набирането " се разглежда като събитие, което учителят ще трябва да отвърнат.

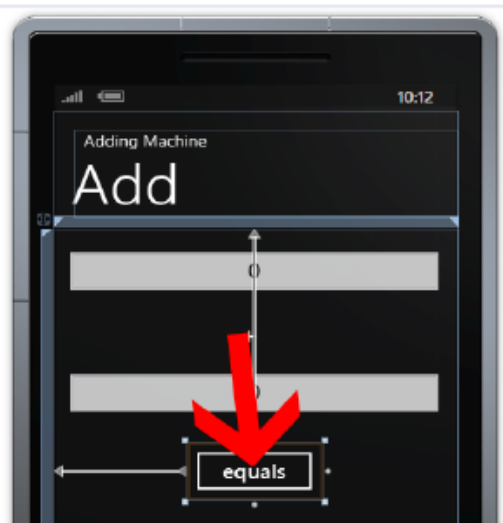
Използването на събития като това прави софтуер, дизайн много по-лесно . Нашата програма не трябва да се провери всеки елемент на екрана , за да видите , ако потребителят е направил нещо с него , вместо това просто се свързва към събитията, които той се интересува от.

За да накарате събитията да работят език програмиране се нуждае от начин за изразяване на позоваване на метод в даден обект. C # осигурява вида делегат , който прави точно това. Можете да създадете тип делегат , който може да се отнася до определен вид метод и след това да създадете копия на тази делегат , които се отнасят до метод в даден обект. Делегатите са много мощни , но могат да бъдат малко по- трудни за да получите главата си наоколо. За щастие ние не трябва да се притеснявате за това как делегати работят точно в момента , защото можем да получим Silverlight и Visual Studio да свърши цялата работа вместо нас.

Събития в Silverlight

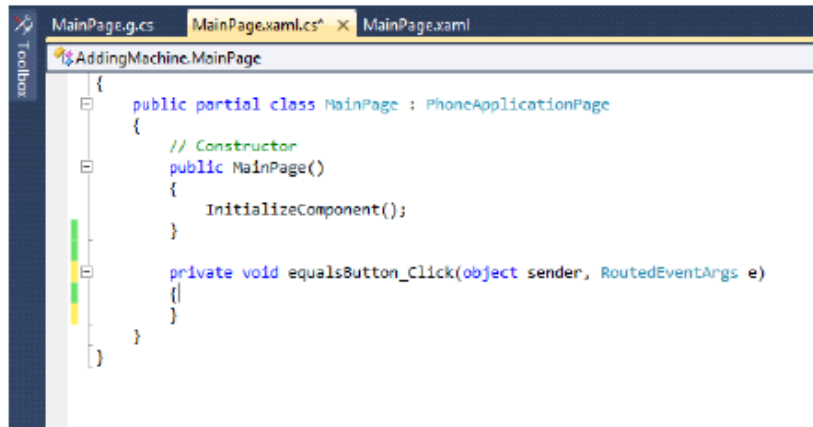
В C # събитие се доставя към обект чрез извикване на метод в този обект. в това отношение можете да разглеждаме дадено събитие и съобщение, тъй като едно и също нещо. От гледна точка на нашата ofview добавяне машина ние бихме искали да имат особен метод, наречен когато "е равно на" бутон е натиснат от потребителя. Това е единственият случай, че ние се интересуваме. Когато пламъците на събитието, които искаме да извикате метода calculateResult rope.

Ние можем да използваме редактор в Visual Studio да се свързва това събитие за нас. Това всъщност е толкова лесно, колкото да се появи магически. За да се свържете метод за кликуване случай на определен бутон ние просто трябва да кликнете два пъти върху този бутон върху повърхността на проектиране.



На снимката по-горе показва, където ние трябва да кликнете два пъти в Visual Studio. Когато кликнете два пъти върху бутона Visual Studio определя свойствата на бутона в XAML за свързване на бутон за метод в нашата страница. Той също така създава една празна версия на този метод и ни отвежда направо към този метод, така че можем да започнем добавяне на код, който трябва да работи, когато бутонът се натисне.

Ако ние просто едно щракване на мишката върху бутона за това има ефекта на избирането му, така че да можем да го движите на дисплея и се променя неговия размер. Ако искате да се свържете манипулатор на събитие трябва да бъде двойно кликуване. Ако направим това, ние ще видим на дисплея като този, показан по-долу.



Now, all we have to do is add the code to make this work.

```
private void calculateResult()
{
    float v1 = float.Parse(firstNumberTextBox.Text);
    float v2 = float.Parse(secondNumberTextBox.Text);

    float result = v1 + v2;

    resultTextBlock.Text = result.ToString();
}

private void equalsButton_Click(object sender, RoutedEventArgs e)
{
    calculateResult();
}
```

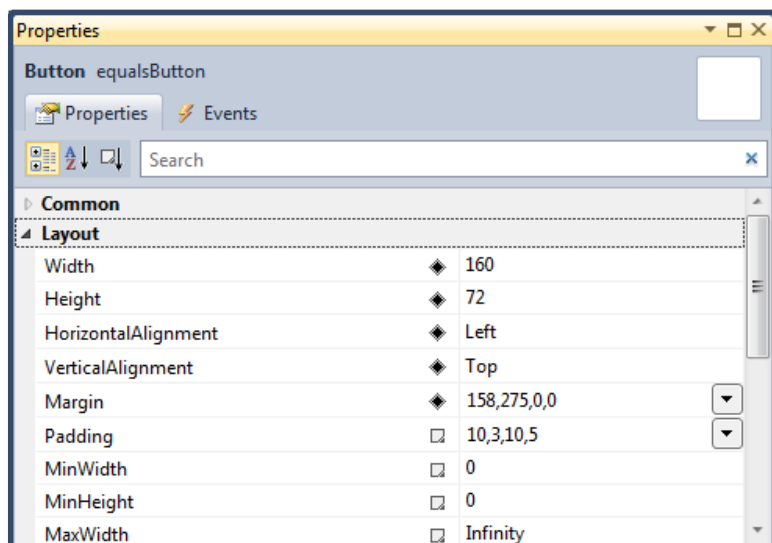
Манипулатора на събитие, което се създава автоматично се дава разумен име чрез Visual Studio. Отнема името на елемента и добавя текста "_CLICK" на края. Това подчертава значението на предоставянето на вашите елементи смислени имена, когато ги създаде, защото това име го прави много лесно за всеки да разбере какво се случва.



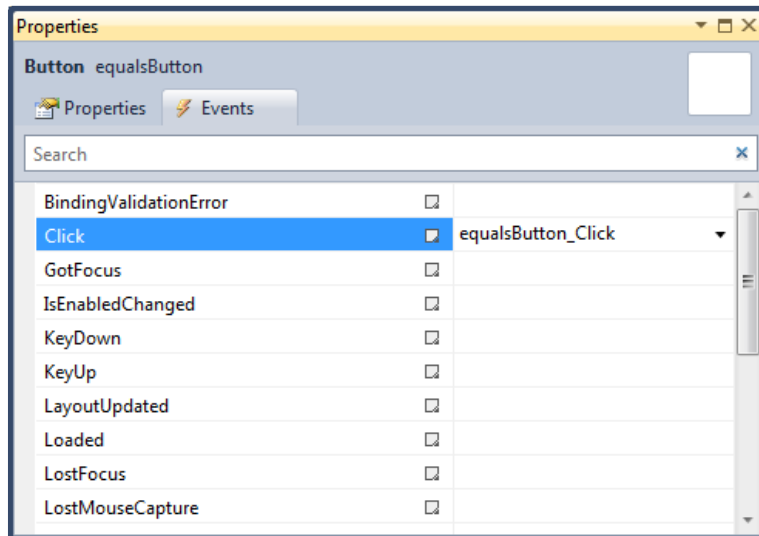
Програмата работи добре, както можете да видите по-горе. Разбира се, че не е свършен. Ако даден потребител в текстов, а не числова стойност на програмата ще се провали с изключение, но ние ще се справим с това по-късно.

Управление на имоти събития

В момента процесът на свързване на обработчици на събития за събития изглежда малко магия. И ако има едно нещо, аз съм сигурен, че е това, че компютрите не са магически. Ever. Така че сега ние трябва наистина да разберете как всъщност работи този процес събитие. Можем да започнем, като разгледаме свойствата на бутона равни в нашата молба.



Ако едно кликване върху бутона в редактора и след това да разгледаме информацията за имот в Visual Studio виждаме нещо много подобно на дисплея горе. Информацията тук излага позицията на бутона на екрана и куп други неща също. Прозорецът на имот има два раздела панели все пак. Ако погледнем в горната ние ще намерим един с името събития, и по-скоро страшно търсите мълния икона на болт. Ако кликнете върху раздела Събития на дисплея се променя, за да покаже на събитията, че този елемент може да генерира, и които са в момента, свързани с код:



Прозорецът Properties сега ни показва всички събития, които един бутон може да генерира. В момента само едно събитие има метод, свързан с него, и че е най-клик събитието. Това е свързано с метода equalsButton_Click. Ако искахме да изключите Натиснете събитието от метода, бихме могли да направим това, като просто изтриване на името на манипулатора от този прозорец. Ние може да доведе до проблеми за себе си, като се замени съдържанието на тази част от формуляра с нещо глупаво като "ILikeCheese".

Ако правим това, програмата няма да работи (или дори изграждане) повече, защото кликване Понастоящем събитието свързан с нещо, което не съществува.

Събития и XAML

До сега сме работили на принципа, че свойствата показване в Visual Studio се задвижва от файла XAML, който описва елементите на екрана. Ние можем да докажем, че това е така, като се погледнем в описанието на бутон във файла XAML за MainPage.

```
Button Content="equals" Height="72" HorizontalAlignment="Left"
Margin="158,275,0,0" Name="equalsButton" VerticalAlignment="Top"
Width="160" Click="equalsButton_Click" />
```

Както можете да видите, описанието на Button вече съдържа елемент, който свързва Click случай с името на метод.

Един важен момент е да се отбележи тук е, че ако файлът XAML казва, че един бутон е свързан с метод с конкретно име на програмата ще се провали за да работи коректно, ако този метод не е там.

Какво научихме

- 1 . Silverlight осигурява начин на проектиране на графичен потребителски интерфейс
- 2 . А Silverlight потребителски интерфейс се състои от елементи, които са неща, като например текстови полета и бутони .
- 3 . Интегрирана среда за разработка на Visual Studio предлага редактор, който може да се използва за добавяне на Silverlight елементи на потребителския интерфейс на страниците на заявление .
- 4 . От софтуерна гледна точка на всеки от елементите на потребителския интерфейс е представена от определен тип обект в йерархия клас, който е част от Silverlight .
- 5 . Дизайнерите могат да променят свойствата на елементите , използвайки средствата за проектиране в Visual Studio , или като ги променя по повърхността на дизайн или чрез модифициране на техните свойства .
- 6 . Действителните свойства на Silverlight елементи в един проект се провеждат в текстови файлове във формат XAML . Тези файлове са актуализирани от средствата за проектиране и използвани , когато програмата е построен, за да създадете софтуерни обекти , които се използват в разтвора.
- 7 . XAML (разтегателно Application Markup Language) е XML базиран език, който определя всички свойства на елементите на дизайна на една страница . Тя осигурява разделяне между външния вид и свойства на елементите на дисплея и кода на програма, която се намира зад тях .
- 8 . XML (Extensible Markup Language) е начин за създаване на потребителски езици, които могат да бъдат използвани , за да опише нещата .
- 9 . Елементите могат да генерират събития, които могат да се свържат с методи вътре в C # програма. Името на метода е дадено в описанието на XML за елемент .
- 10 . Методите , които са обвързани със събитията могат да осигурят бизнес логиката за потребителско приложение

Глава 3. Въведение във Visual Studio

Когато пишете програми за Windows Phone ще се използва Visual Studio. В този раздел ще разгледаме процеса на създаване и управление на Windows Phone проекти. Ние също така ще разберем как да тече и отстраняване на грешки в Windows Phone програми с помощта на програмата Windows Phone емулатор. Това позволява да тествате програмите си без да се налага да имате устройство.

Все пак, това не е просто един поглед към това как да използвате Visual Studio. Това би било твърде лесно. Ние също така ще надникнем под капака и да разберем как Visual Studio управлява съдържание, което прави нашите решения. Това ще ни стои в добро вместо него, когато дойде да се създаде Windows Phone приложения.

3.1. Проекти и решения

Първото нещо, което ще разгледаме е как Visual Studio управлява програмите, които сте създали.

Създаване на програми

Ние знаем, че когато ние създаваме програма, пишем набор от инструкции в един език от високо ниво (който, ако имаме късмет е C#). Компиляторът е програма, която взема най-високи нива инструкции, които програмистът създава и ги превръща в по-ниски нива инструкции за изпълнение на компютъра. Ние също така знаем, че в Microsoft .NET съставител произвежда продукция в междинен език (MSIL), който след "Just In Time" събира в хардуерни инструкции, когато програмата действително работи.

В абсолютния минимум, което трябва да се създаде работна програма следователно е C# файла източник (текстов файл с разширение на езика .CS) и компилатор. Компиляторът взема изходния файл и да го превръща в един изпълним файл (двоичен файл с разширение език .EXE), които след това могат да бъдат използвани от .NET система по време на работа, независимо от платформата, която да изпълнява програмата на. Използването на MSIL означава, че мога да взема точно същото .EXE файл и да го изпълнява на напълно различни хардуерни платформи, ако искам.

THE NET SDK

Ако просто искате да се създаде програми за Windows PC и не искате да свалите всички Visual Studio всъщност можете просто да получите съставител и автономна работа за NET от тук.: <http://msdn.microsoft.com/en-us/netframework/aa569263.aspx>

Когато инсталирате frameworrk получавате компилатор командния ред, което може да отнеме C# изходните файла и да ги конвертира в изпълними тези, които използват конзолни команди, като например:

CSC MyProg.cs

Тази команда ще съставят файлови MyProg.cs и да създадете файл, наречен

MyProg.exe.

В командния ред съставител е налична, ако сте инсталирали Visual Studio.

Възможно е да се използват тези прости инструменти, за се създават много големи програми, но също така е доста трудна работа. Трябва да се създаде големи, сложни команди за компилатора и вие трябва да се уверите, че всеки път, когато се направи нова версия на системата ви, че вие съставяте всеки файл източник, който има нужда. Освен това можете да управлявате и да включи

всички медии, която вашата система се нуждае и когато става въпрос за коригирането на грешките в програмата всичко, което ще получите са текстови съобщения, когато програмата не успее.

Visual Studio

Visual Studio разглежда всички проблеми отгоре. То осигурява една среда, в която можете да редактирате, изградите и изпълните вашите програми. Също така предоставя на проекти и решения, които ви позволяват да организирате своите системи и управление, което се случва в тях. И накрая, просто за да бъде още по-прекрасно, тя осигурява цялостно решение за отстраняване на грешки, която ви позволява да единствена стъпка чрез програмата си и дори изпълнява индивидуални изявления и промяна на стойностите в променливи като вашата програма работи. Skill с Visual Studio и разбиране за това как тя работи, е много продаваеми.

Първото нещо, което попадате с Visual Studio е огромният брой бутони и контроли, които трябва да се справите. Ако започнем да отваряте менюта ще стане още по-лошо. Все пак добрата новина е, че не трябва да натиснете всички бутони, за да започнете. Ние просто ще погледнем в основите на програмата. Можем да вземем по-разширени функции, като задълбочим.

Първият аспект на Visual Studio, който ще погледнем е на проекти и решения. Вие ще използвате тях, за да организирате всякакви системи, които създавате. Visual Studio не е в състояние да стартирате всеки програмен код, който не е в проект или решение. Добрата новина е, че, за да започнете, можете да използвате един от шаблоните, които осигуряват отправна точка за определени видове проекти. Ние ще използваме Windows Phone шаблони за сега. Има много други шаблони за различни видове програми.

Visual Studio прави разграничение между проект и решение, когато организирате работата си. Важно е, че сте разбрали разликата между тези две. Нека започнем, като погледнем в един проект.

Прост проект на Visual Studio

Един проект е контейнер за набор от програмни файлове и ресурси, които генерират особено монтаж файл. И така, какво означава това? Е, възможно простият проект ще бъде един съдържащ единна програма файл. Можете да създадете един от тях чрез задаване на Visual Studio за създаване на Console Application. Ние можем да направим това с помощта на File> New> Project command, за да отвори диалога New Project:

Диалогът New Project ни показва всички възможни шаблони на проекти, които са снабдени с Visual Studio. Шаблонът е предварително зададена колекция от файлове, папки и ресурси, които се използват за определен тип приложение. Можем да добавим към шаблона, след като тя е била създадена, и дори е възможно да се създават персонализирани шаблони на нашата собствена. Проектите, които видяхме по-рано са били създадени с помощта на шаблони за създаване на приложения за Windows Phone.

Ако изберем шаблон за Windows Console Application, както е показано по-горе Visual Studio ще създаде най-простия възможен проект за нас. Това е, което тя би изглеждала като в Visual Studio Solution Explorer.

Този проект съдържа един програмен файл, наречен Program.cs. Източникът на програмата в този файл е твърде просто:

Този проект съдържа един програмен файл, наречен Program.cs. Програмният код в този файл е много елементарен:

Ако заповядаме на Visual Studio, да стартирате програмата, тя ще компилира програмния клас и след това стартирате приложението, като се извика от Main метода. Това е начина, по който C# програми работят. Изходът на процеса на изграждане на тази програма ще бъде даден файл, наречен program.exe, която може да се управлява като програма за Windows PC.

Програмата работи в командния прозорец, който ще се появи за кратко на екрана, когато основният метод работи. Ние ще разгледаме текущите и дебъгване на програми, малко по-късно в този раздел.

Създаването на програма от няколко изходни файла

Разумно е често да се създава програма от няколко класови файла. Те могат да бъдат отделни части, написани от друг програмист или бихте могли да използвате библиотека на C# код, предоставени от някой друг. Изследовател на решенията в Visual Studio ще покаже най-новия файл от проекта:

Файлът Library.cs сега е част от проекта и когато Visual Studio изгражда проекта, той знае как да компилира изходния файл Librari.cs заедно с още един файл наречен Program.cs. Въпреки това, само един от тези изходни файлове може да има основен метод, в противен случай компилаторът ще станат объркани за това къде в програмата се очаква да започнат да се показват. The Library класа е момента празен.

Именно пространство и проекти.

Класът The Library и the Program клас са в именно пространство SimpleProject. От гледна точка на дизайна често е разумно да има различни части от системата в отделни пространства от имена. Това е особено важно за програмите, които се изграждат от различни хора. Ако програмистът пише код на библиотеката и решава, че името Save е подходящо за конкретен метод нямаш „Не искам, че сблъсък с метод Save, която искаме да използвате“

Не забравяйте, че именните пространства са логичен начин на групиране на обекти заедно. Те нямат никакво влияние върху, където се намира физически действителния код. Можем да сложим клас Library в отделно пространство от имена, просто чрез промяна на името:

The Library класа сега е в пространството от имена SimpleLibrary. Ако се опитаме да изградим проекта ние получим някои грешки:

Това е така, защото основният метод в класа на програма се опитва да създаде инстанция на класа Library. Този клас вече не е в пространството от имена SimpleProject, и така да стигнем грешките. За щастие полето за Описание на списъка за грешка се опитва да помогне, като ни казва, че ние може да липсва, като се използва директива. Такъв е случаят. Ние може да реши проблема чрез добавяне на използване на директорията да каже на компилатора за да гледа в пространството от имена SimpleLibrary ако тя се нуждае, за да намерите даден клас:

Друг начин за решаване на проблема е да се използва напълно квалифицирани имена за достъп до ресурсите в библиотеката:

Това, което трябва да запомните за всичко това е, че ние просто създаваме имена за предмети в нашия проект, ние не посочваме къде в системата те се съхраняват. Когато Visual Studio изгражда SimpleProject тя ще изглежда във всички файла източник в проекта, за да намерите елементи, които се нуждае.

Добавяне на ресурси с проект

Ако вашата програма трябва да използва ресурси (вероятно изображение за разпръскване на екрана картина), можете да ги добавите към даден проект също. Можем да добавим растерно изображение по същия начин, както добавя нов клас.

Когато сме добавяли ресурс на растерна графика то се появява в Solution Explorer за проекта.

Това решение вече съдържа файл с растерна графика, наречена Bitmap1.bmp. Сега можем да изберете как Visual Studio управлява това съдържание чрез модифициране на свойствата на Bitmap1 в решението.

Свойствено съдържание

Както всичко останало, което управлява Visual Studio, ресурсите, добавени към даден проект също имат свойства. Ние можем да работим със свойствата точно както ние работим с качествата на всеки друг елемент, който ние добавите към даден проект, ние използваме прозореца със свойства.

Настройките над питат Visual Studio, за да копирате растерна графика в същата директория, в програмата, когато това се основава на проекта. Програмата след това може да отвори този файл и да го използвате, когато той работи:

В горния код ще зареди растерна графика ресурс в растерна графика, наречена б.

Това е един много добър начин да се добавят точки към даден проект. Имайте предвид, че свойствата горе, не са тези, които по подразбиране, които са определени, когато се добави нов елемент на съдържание. По подразбиране (т.е. освен ако не сте задали друго) съдържанието не се копира в директория продукция.

Вграждане на съдържание в самата искова молба

Посочените по-горе настройки питат че растерна графика се съхранява в същата директория, като програмата. Това работи добре, но се приема, че когато инсталирате програмата файла с изображения се копират, както и двоичния код на програмата.

Ако искаме по-тясно обвързване между програмата и нейните ресурси, ние може да поиска този ресурс да бъде вграден във файла на програмата, когато програмата е изградена. Когато програмата е изградена програмата за изход изпълним ще съдържа растерна графика. Ние правим това чрез промяна на свойствата на съдържанието на елемента.

Свойството Build Action на файла Bitmap2.bmp сега подготвя Embedded Resource. Нашия образ сега се съхранява в самия програмния файл. Първи сдобият с това е малко по-ангажирани, но са гарантирани да работят където и нашата програма се разгръща, защото няма допълнителни файла:

Файлово събрание и изпълними файла

В този момент може да започне да се чуди за изходния файл. Тя изглеждаше достатъчно проста, когато бяхме само съставянето на прост C# програма, но сега тя съдържа картина, както добре. Това е така, защото на изхода на Visual Studio натрупването не е просто една програма, то е всъщност съвкупност. Съвкупност е контейнер, който може да побере цяла гама от неща, включително и програмен код. Когато една програма се изпълнява на системата .NET време на изпълнение. Ще открие събраниято и да отидете и да намерите класа в събраниято с основен метод в него. След това започва този метод работи. Файлът на проект разказва Visual Studio какво да сложи в събраниято.

Ако сте много стар, и не забравяйте, когато един файл, който свършва на. Търсейки просто съдържаше програмни инструкции, то това е малко объркващо. Добрата новина е, че не е нужно да се притеснявате; Visual Studio върши много добра работа за управление, което се случва в една програма.

Използването ildasm да погледнем вътре файла за сглобяване

Ако "някога съм искал да гледам вътре монтажен файл и да видим какво прави отбележете можете да използвате ildasm на програмата. Това е програма, можете да започнете от Visual Studio Command Line. За да отворите командния ред използвате следния път чрез вашите програмните файла:

Ildasm ви позволява да отворите всеки монтаж и да погледнем вътре. Това също ви позволява да видите MSIL (Microsoft Intermediate Language) инструкции, които компилаторът произведени. Можете да стартирате програмата само чрез издаване на команди ildasm:

След като програмата се изпълнява, можете да намерите вашия начин за монтаж и да го отворите:

Това е дисплей ildasm по простата проекта направих по-горе. Монтажът съдържа събраните класовете, които съставят съдържанието на програмата. Можете да видите, че класът

SimpleProject.Program съдържа два метода, метода на ctor (конструктора за класа) и основен метод. Можете да отворите тези и ги накарайте да поглеждат в кода вътре в тях.

Манифестът част на събранието съдържа "списък на съдържанието" за събранието.

Това е манифестът модул, който съдържа внедрен ресурс. Можете да видите името на ресурса, както и размерът на това.

Програмата ildasm може да ни покаже тази информация, защото компилиран клас съдържа метаданни, които се описват методите и членовете и дори може да включва информация за Intellisense да използва в Visual Studio. Ако някога сте се чудеха какво точно програма и как тя получава да стартирате можете да имате много забавно става чрез вижданията че ildasm предоставя. Вие, обаче, не може да има една и съща идея за това какво представлява забавно, както аз правя.

Библиотека монтаж

Там всъщност са два вида монтаж; такива, които съдържат основен метод някъде (изпълними такива) и библиотеки. Когато създавате нов проект можете да създадете проект библиотека, ако искате:

Това ще създаде чисто нов проект, който ще съдържа само библиотечните класове, т.е. нито един от класовете в събранието ще съдържат Main метод. Ако някоя от тях го правят, ще получите съобщение за грешка, когато се опитате да компилирате проекта. А монтажна библиотека събира във файл с разширение език .DLL. Това означава Dynamic Link Library. Думата библиотека звучи достатъчно разумно; на динамична връзка част означава, че класовете на библиотеката се зареждат динамично, когато програмата работи.

Когато програмата се отнася до метод в клас в библиотеката този клас ще бъде зареден в паметта и компилаторът Just In Time ще конвертира метода в машинен код, така че да може да работи. Ако никога не се използва този клас, тя никога не е зареден. Това е един добър начин за спестяване на памет, за сметка на малко повече усилия, когато програмата работи.

Ние може да даде други програмисти нашите DLL файла и те могат да използват класовете вътре в тях, без да има нужда от сорс кода на програмата. Дори по-добре, метаданните, които библиотеките съдържат го прави лесно за програмистите да намерят пътя си около тях.

Системни библиотеки и референции

А програма, работеща под .NET прави използването на системните ресурси, тъй като тя работи. Когато програмата прави иска да изпрати съобщение на конзолата ще се обади метод система да направите това:

Тези методи се провеждат в библиотеките на системата, които са всъщност .DLL файла, които са създадени от Microsoft и се разпространяват като част от .NET система. Монтажният файл съдържа

препратки към библиотечни файлове, които тя работи. Този списък от препратки се поддържа в Visual Studio през прозореца на Solution Explorer.

Секцията Позоваването на проект дава списък на всички ресурси, използвани от едно заявление . Когато Visual Studio създава нов проект на конкретен вид тя ще добави препратки към него ресурси нещата по този вид на нуждите на проекта. Понякога ще трябва да управлявате този списък себе си , например Phone проект Windows обикновено не се съдържа позоваване на компонент Microsoft.Devices.Sensors . Ако искате да използвате акселерометъра в приложение Windows Phone , което трябва да добавите този референтен себе си. Ние ще видим как да направим това малко по-късно в курса.

Имайте предвид, че позоваването е само, че една справка. Ако ние се включи позоваване на ресурс това не означава, че ресурсът става част от нашия проект , по-скоро , че събранието е в състояние да " достигнат " до този ресурс , когато тя работи. Ако ресурсът ISN " тон на разположение по време на изпълнение на програмата ще се провали. А даден ресурс има номер на версия , прикрепена към него , така , че ако една нова версия на библиотеката се добавя към система някакви стари програми все още ще бъде в състояние да използват предишната версия.

Процесът на настройване

Сега ние трябва да се радваме с идеята, че един проект съдържа цялото съдържание изисква да се създаде единен съборение. Когато една програма е изградена Visual Studio ще се съберат всички елементи, които са необходими, за да направи монтажа и след това да ги използват за създаване на файла. Сглобката на продукцията се съхранява в папка, създадена, когато проектът е създадена.

Тук можете да видите папката, която съдържа изхода от прост проект, който ние създадохме още от самото начало. В досието за заявлението на върха е изпълним съборение, който работи на компютъра. Можете да дадете на някого този файл, за да стартирате и вирус скенери, позволяващи, те ще бъдат в състояние да го изпълни.

На картинката по-горе е за отстраняване на грешки версия на програмата. Когато Visual Studio прави за отстраняване на грешки версия, ако също пише от база данни с информация за отстраняване на грешки, която дава информация за източника, когато ние отново отстраняване на грешки.

Visual Studio решения

Ако имате само някога искате да произвеждат един-единствен изпълним монтажен проект е всичко, което трябва. Все пак, ако искате да създадете няколко отделни файлове за сглобяване и използване на тази в едно заявление Visual Studio ви позволява да направите това също. А файлово решение е контейнер, който може да побере проектни файлове. Когато създавате нов проект, както направихме с SimpleProject по-рано, ние всъщност се получи разтвор, създаден за нас, която обвива проекта. Фактът, че ние сме били гледане на всичко това съдържание с Solution Explorer може да се счита представа тук.

Създаване решение мулти-проект

Можем да добавим нови проекти до решение, просто с десния бутон на решение и избор на нов проект от менюто, което се появява.

След това можем да вземем някой от шаблоните на проекта и направи нов проект, който след това се провежда като част от решението:

В посоченото по-горе направих нов проект, наречен библиотека DisplayLibrary. Това ще се съберат, за да се получи файл DLL които ще се съхраняват в двоичен папката за новия проект.

Тази библиотека може да се използва от всички проекти, които добавят препратка към файл .DLL.

Обвързването Проекти

В момента проектите в нашето решение, не са свързани по никакъв начин. Ако програми в SimpleProject искат да използват средства от ResourceLibrary те трябва да ги добавите като референтен ресурс:

В диалога по-горе аз съм добавяне на справка за DisplayLibrary към проекта в SimpleProject. За да си по-лесно програмиране мога да добавите с помощта на директива във файла източник програма.

Всичко това помага за да ни напомнят, че с помощта на директивата всъщност не свързваме нашата програма за всеки ресурс, който тя иска да използва. Ние само се нуждаем от помощта на директивата, за да се намали количеството на писане, че ние трябва да направите, когато пишете програми. Когато изграждаме SimpleProject ние откриваме, че двоичен директория за тази програма сега съдържа копия на DisplayLibrary DLL файла: Тази библиотека може да се използва от всички проекти, които добавят препратка към файл DLL.

Обвързването Проекти

В момента проектите в нашето решение, не са свързани по никакъв начин. Ако програми в SimpleProject искат да използват средства от ResourceLibrary те трябва да ги добавите като референтен ресурс:

Това е версия на програмата SimpleProject, която използва DisplayLibrary да се зареди и показва изображения. Външните ресурсите, използвани от монтаж са изброени в манифеста за този монтаж.

Тук можете да видите на манифеста за версия на програмата , която използва SimpleProject библиотека наречена DisplayLibrary . Имайте предвид, че версията на библиотеката се добавя към информацията, така че ако една нова версия на библиотеката е освободен тази асамблея ще продължи да използва старата.

Решения Мултипроекта

Можете да сложите много проекти на различни видове в едно единствено решение. Например бихте могли да създадете XNA игра за Windows Phone, Xbox 360 и Windows PC с помощта на един единствен файл решение. Всяко от устройствата може да има свой собствен проект файл, играта на двигателя може да се проведе в четвърти проект, който след това беше включена от останалите.

Ако пишете на Silverlight приложение, което се свързва към уеб сайт, който предоставя услуги, които може да се сложи и двата проекта заявление Silverlight и проекта за уеб сървър в рамките на едно и също решение.

Проектите в вашето решение , както и задачите, извършвани от всеки от тях трябва да бъдат разработени в началото на развитието си .

Ако вашето решение съдържа няколко проекта, които произвеждат изпълними резултати можете да зададете един като "Startup проект ", който ще бъде даден контрол , когато проектът се изпълнява.

Windows Phone решения

Съществуват основно два вида Windows Phone решения, които могат да бъдат създадени с помощта на Visual Studio. Това са Silverlight прилагане и XNA. Имайте предвид, че няма начин можете да направите само едно заявление Windows Phone, която използва и двете системи. Макар че би било хубаво да бъде в състояние да използват Silverlight за всички менюта на играта и след това XNA за геймплея това не е възможно в момента.

Silverlight Windows Phone проекти

Добавянето на машината, която сме създали в предишната глава е създаден като Silverlight проект. Ако погледнете решения за този проект можете да получите нещо като това:

Решение на Windows Phone Silverlight съдържа файла MainPage.xaml, който описва външния вид на главния екран. Той също така съдържа изображенията, които ще бъдат използвани за дисплея, когато правилата на програмата иконата на приложението екрана. Ако добавите нови страници във вашия Silverlight програма те ще бъдат добавени към този проект като .XAML файл, а също и кода на файл, който видяхме по-рано. Имайте предвид, че проектът включва и препратки към всички библиотеки на системата, които са използвани от телефона.

XNA Windows Phone Project

Рамката на XNA е разработен за писане на игри. Тя осигурява пълен игрален двигател, в която можете да поставите вашия актуализация на играта и изготвяте поведения. Той също така предоставя цялостно управление на съдържанието, така че снимки, текстури и звуци могат лесно да бъдат включени в една игра.

Когато правите XNA решения, чрез помощта на шаблон, предоставена от Visual Studio всъщност става създадени два проекта. Един от тях съдържа изпълним код. Другият ще проведе цялото

съдържание на играта. Идеята е, че ако ние искаме да направим една игра, която работи на множество платформи ние просто трябва да се добавят нови проекти за решение. Те могат да споделят на проекта за единен ресурс.

Над вас можете да видите на решение за частично завършен CheeseLander игра. Играта използва един елемент от съдържанието, изображенията на сиренето трябва да бъдат разтоварени. Ако добавим ресурси, като например изображения на проекта те ще бъдат съхранявани в проекта Content и може да се използва от всяка игра на проекта в това решение.

Работещи Windows Phone приложения

Ние знаем, че когато се съберат и да се изгради Windows PC апликация резултатът ще бъде файл с разширение на езика .exe. Търсейки, който съдържа метод наречен Main която ще се нарича, когато програмата започва да работи.

Ако създадете приложение за Windows Phone нещата са малко по-сложни. Програмата няма да се работи вътре в компютъра, вместо това трябва да се прехвърля в устройство за телефон с Windows, или емулатор, преди да започнете. Освен това, каквито и да било средства, че програмата се нуждае (например съдържание, което не е част от сглобяването на програмата) трябва да бъдат предоставени на разположение в целта устройство. Този проблем е решен чрез използването на файл контейнер, който притежава цялата заявлението и всички необходими средства. По-долу можете да видите изходния директория за програмата AddingMachine. Той съдържа всички ресурси, заедно с AddingMachine.dll файла, който съдържа кода на програмата.

Ако създадете приложение за Windows Phone нещата са малко по-сложни. Програмата няма да се работи вътре в компютъра, вместо това трябва да се прехвърля в устройство за телефон с Windows, или емулатор, преди да започнете. Освен това, каквито и да било средства, че програмата се нуждае (например съдържание, което не е част от сглобяването на програмата) трябва да бъдат предоставени на разположение в целта устройство. Този проблем е решен чрез използването на файл контейнер, който притежава цялата заявлението и всички необходими средства. По-долу можете да видите изходния директория за програмата AddingMachine. Той съдържа всички ресурси, заедно с AddingMachine.dll файла, който съдържа кода на програмата.

Има също файл, който има разширение език .XAP. Това е файлът XAP. Той е контейнер, който притежава цялата заявка.

Файлът XAP

Файлът .XAP е архивен файл (съхранява в същия формат, като Zip файлове), който съдържа всички файлове, които съставят заявлението. Той съдържа също така явен файл, който описва съдържанието на XAP.

Над вас можете да видите съдържанието на файла .XAP за прилагането на AddingMachine. Файлът .XAP е файл, който се прехвърля в Windows Phone устройство и се изпълнява. Когато натиснете бутона на изпълнение в Visual Studio този файл е създаден и след това се изпраща в устройството

или емулатор. Целевото устройство отваря файла, прочита манифеста и след това разархивира съдържанието. Накрая молбата е изпълнена. Ако вашата програма е съставена от множество библиотеки и съдържа изображения и звукови файлове, всичко това ще се поставя във файла XAP да се качват в целта.

Когато подава заявление за Windows Phone Marketplace всъщност изпращате файла .XAP, това е файл, който се зарежда в клиентския телефон, ако те купуват програмата.

3.2 Отстраняване на грешки в програми

Един от големите неща за околната програмната среда на Windows Phone, е лекотата, с която можете да проверявате програмите. Ние може да получим наистина добра идея за това какво се случва вътре в една програма с едно засилване през кода и като погледнете стойностите в променливи. Можете да проверявате програмите от двете истинско устройство или емулатор.

Използване на Windows Phone емулатор

Windows Phone емулаторът ви дава възможност да изпробвате своите програми и да откриете това, което изглежда, когато се изпълняват. Емулаторът работи по същия начин като истински телефон. Ефективно е Windows PC изпълнението на платформата Windows Phone, който върви точно по същия изходен код като действителния телефон. Можете да използвате емулатора, за да докаже, че вашата програма работи правилно.

Разполагането на емулатора

Можете да разположите програмите на емулатора, или реално устройство. За да изберете дестинация, когато стартирате програмата, която използвате комбо-поле вдясно от бутона Run, както е показано по-долу.

Възможно е да се избере Windows Phone устройство, дори когато човек не е свързан. В този случай ще получите съобщение за грешка, когато се опитате да стартирате програмата и Visual Studio открива, че устройството не е приложено.

Когато програмата започва да се изпълнява в точно същото е, както би направил на Windows Phone устройство. Можете да използвате PC мишка, за да кликнете върху предмети. Ако имате мулти-тъч екран на вашия Windows 7 PC, можете да използвате няколко докосвания на екрана на емулатор за достъп до мулти-тъч функции на потребителския интерфейс.

За да спрете програмата, можете да кликнете върху бутона за връщане назад (лявата стрелката), в основата на емулатора или натиснете бутона за спиране в Visual Studio.

Функции на емулатора

Имайте предвид, че емулаторът не е снабден с всички вградения в програмите, предоставяни с истинско устройство. Можете да сърфирате в интернет с помощта на Internet Explorer и можете да промените някои настройки на устройството, но много неща, например калкулатор, календар и

Zune отсъстват. Има някои емуляции предвидени да ви позволи да тествате вашите програми обаче, ако се напише програма, която работи с контакти от адресната книга (нещо, което ние ще направим по-късно), вие ще откриете, че някои контакти са създадени вътре в телефона, за да работите.

Изпълнение на емулятора.

Емуляторът не може да се движи с точно същата скорост, както на Windows Phone хардуера. Действителната скорост на дисплея на заявление е ограничен на емулятор, така че на дисплея ще отговаря на реалното устройство до известна степен, но е важно да пускате програми на недвижими устройство, за да получите точна представа за това, което ще бъде работата на потребителя.

Програми в емулятора

Когато Visual Studio разполага заявление до Windows Phone емулятор тя го инсталира на устройството пример за подражание. Приложението ще остане там, докато емуляторът е нулиран. Над вас можете да видите стартовото меню за емулятора след прилагането на AddingMachine са били изпълнявани по нея. Ние можем да стартираме тази заявка отново само с щракване върху иконата.

Отстраняване на грешки Visual Studio

Показателите за отстраняване на грешки са интегрирани в Visual Studio среда. Можете да стартирате програми от рамките на Visual Studio. Можете да добавите точки на прекъсване в програмата си. Когато една програма достига изявление номиниран като прекъсване Visual Studio ще направи пауза на програмата и ще ви позволи да погледнете съдържанието на променливи вътре в кода.

Можете да добавите точки на прекъсване на програмите, които работят вътре в Windows Phone устройство. Можете дори да настроите и премахване гранични стойности, тъй като самата програма се изпълнява.

Отстраняване на грешки е важна част от процеса на развитие. Ако ще се превърнете в успешен програмист вие ще трябва да свикнете с това чувство в стомаха си, че можете да получите, когато програмата не прави това, което сте очаквали. За щастие има някои още по-мощни инструменти и техники, които можете да използвате за коригиране на грешки в програмите.

Настройка и програмиране

Въпреки това, едно важно нещо тук е да запомните, че не смятам, че е приемливо за коригиране на грешки в програмите в живота. Всеки ред с код, който пишете трябва да се постави там въз основа на това, че знаете какво линията ще направи, и защо е там. Аз използвам за отстраняване на грешки, за да се определи изпълнението на решение, напълно съм обмислил. Моите програми са склонни да се обърка, защото имам пропуснато написало нещо, или защото не е нужно правилното разбиране на поведението на библиотека, която използвам. Ако не знаете как

да се реши проблема, че е много малко вероятно, че хвърлят няколко линии заедно може просто да реши проблема, който и да е повече от хвърляне на кофа на компоненти в една стена, ще получите нов Xbox.

Добавяне на точка на прекъсване в декларация

Гранични стойности се добавят по същия начин, както и за всяка програма, с който работите в Visual Studio. Кликнете в лявото поле на линията, ако искате програмата да се прекъсне при.

Линията е осветена, както е показано по-горе. Можете да изчистите точка на прекъсване, като процесът се повтаря. Когато програмата достига точката на прекъсване ще спрат да се пускат и вие ще бъдете в състояние да вземе един поглед на стойностите на променливите.

Стартиране на програма

Ние стартираме програмата, като щракнете върху бутона Run в менюто на Visual Studio. Бутонът изглежда така:. Когато натиснете бутона на програмата е разположена в целевото устройство и започва да работи. Можете да използвате клавиша F5 функция вместо да натискате бутона, ако предпочитате. Когато програмата работи, ако се стигне тази линия на изпълнението ще направи пауза. Разбира се, за да стигнем до това изявление в добавяне на машината ще трябва да въведете някои числа в текстовото поле и след това натиснете бутона за вход.

Променлив резултат не е бил настроен за резултата от изчислението все още, защото линията, която е осветена не е била изпълнена. За да получите най-линията извършва ние трябва да се засили чрез отчета.

Единично засилване

Ние може да стъпим само чрез една линия на програмата, като щракнете върху бутона Single Step в системата Visual Studio меню. Бутонът изглежда така:.

Всеки път, когато натиснете този бутон програмата ще се подчиняват на един отчет. Щракването върху бутона веднъж ще се премести на програмата върху следващия отчет. Ключовата функция F11 ще има същия ефект.

Там всъщност са три версии на бутона Единична стъпка.

Това означава, че се подчиняват на един израз. Ако единият израз е призив на метод тази версия на един етап ще стъпка в метода. Това се нарича "стъпка в" версия на една единствена стъпка. Използвайте F11, за да извърши това.

Това означава, че се подчиняват на един израз. Ако единият израз е призив на метод тази версия на един етап ще стъпи върху викането на метода. Това е полезно, ако не "ме интересува какво се случва вътре в метода. Това се нарича "Стъпка през" версия на една единствена стъпка. Използвайте F10, за да извърши това.

Това означава, тичам до края на текущия метод и изхода от нея. След това се счупят. Това е много полезно, ако се оттегли в един метод, по погрешка, или сте видели всичко от вас метод са засилване чрез която се нуждаете. Използвайте SHIFT + F11, за да извърши това.

3.3. Контролиране на изпълнението на програмата

Visual Studio поддържа контакт с работещата програма в целевото устройство. Може да издава команди, за да спре, пауза и възобновяване на изпълнение и ще окажат пряко влияние върху програмата.

Възобновяване на изпълнението

Ако искате програмата да възобнови изпълнението след прекъсване (т.е. сте направили всичко единния засилване имате нужда), тогава можете да натиснете бутона кандидатура отново, или натиснете клавиша функция F5.

Временно спиране на изпълнение на програмата

Поставянето на пауза при изпълнението обикновено не е толкова полезна, това, че програмата може да не е на място, където се прави нещо интересно, но понякога може да го използвате, за да разберете каква програма се прави, когато тя се появява, за да имам "заби". За да направите пауза на работеща програма щракнете върху бутона.... Програмата спира на твърдението, че той е действал, когато сте натиснали пауза.

Ако искате да направите пауза на работеща програма трябва да се помни, че можете да задавате точки на прекъсване в програмния код, дори когато самата програма се изпълнява. Бихме могли да изведе нашата точка на прекъсване в метода calculateResult докато кода е действал и след това натисна бутона за равни в емулятора, за да изпратите на програмата по този начин.

Спиране на програмата

Бутонът може да бъде използван, за да спрете програмата от изпълнение. Когато натиснете този бутон, Visual Studio ще спре програма след следващия отчет то се подчинява. Ние можем да го използвате, ако програмата се появява, за да имам остана. Ние трябва да бъдем малко по-внимателни с този бутон, все пак.

Когато една програма Windows Phone е спряно правилно го е изпратил поредица от съобщения, които го казват, края е близо и да се направи някаква поддръждане, което се изисква. Ние ще откриете как работи това по-късно в курса. Ако програмата е спряно с използване на бутона по-горе тези съобщения не се изпращат, и така програмата прекъсната по този начин може да загуби или повредени данни, че той е използвал, когато спирането на събитието ще бъде изтрит.

Управление Гранични стойности

Visual Studio също има прозорец за точка на прекъсване, който може да използвате, за да видите граничните стойности, които сте създали. Можете да покажете този прозорец, като отидете на менюто Debug > Windows в Visual Studio и избиране на опцията „Гранични стойности“.

Можете да използвате този прозорец, за да зададете свойства на точка на прекъсване. Можете да поискате от прекъсване да се изпълни само след определен брой пъти, или само един път, когато определено условие се изпълни. Това е ужасно полезно. Може да имате програма, която не успее след време петдесетия път. Много е хубаво да бъдете в състояние да добавите брояч или тест, така че програмата прескача покрай гранични стойности.

За да отворите свойствата на точка на прекъсване, в самия код или в прозореца на точки на прекъсване, щракнете с десния бутон точката на прекъсване и след това изберете прекъсване от контекстното меню, което се появява.

Използване на прозореца на непосредствените

Бързият прозорецът може да бъде показан като отидете на Debug > Windows меню и го изберете. Това е мястото, където можете да преглеждате и промените съдържанието на променливи. Можете да го използвате, за да оценявате изразите. Като пример на прозореца незабавно в действие ние може да зададете точка на прекъсване, точно преди метода `calculateResult` връща:

Когато стартирате програмата и натиснете бутона равен на програмата ще удари точката на прекъсване. Сега мога да премина към бързия прозорец и да изпълнявам променливи в този метод

Ако просто напиша името на променлива прозорецът показва стойността на тази променлива. Аз също мога да въведа изрази (например `v1 + v2` както е показано по-горе) и те са оценени и показани. Аз също мога да присвои стойност (например резултат = 100) и методи на обекти, за да видя получените резултати.

Непосредствен екран използва Intellisense, за да ви помогне да напишете на отчети, които искате да изпълните. Той дори работи, когато са настройка на програмата, съхранявани вътре в устройството за Windows Phone. Това е много мощен, но трябва да се използва с внимание.

Дизайн за отстраняване на грешки

Едно постъпково изпълнение на код е чудесен начин да разберете какво се прави. Когато пишете код, полезно да го проектираме така че можете лесно да преминете през нея и да разберете какво се прави. Кода в метода `calculateResult` лесно може да бъде като едно изявление:

Въпреки това тази версия е невъзможно да е стъпка през. Аз много вярвам, че в разпространението ми на код за определен брой на отчети и дори да се използват допълнителни временни променливи, когато се изпълнява. Това прави по-лесно за отстраняване на грешки и развивала дори струва вашата програма повече памет от в "малък" версия на кода по-горе Visual Studio ще трябва да създадете някои вътрешни временни променливи, за да извърши изчислението. Единствената разлика с "по-ефективна" версия по-горе е, че не може да се погледнете в тях.

Какво научихме

1. Visual Studio е среда за разработка, която се използва за създаване на приложения за Windows Phone. Можете да получите безплатна версия на този SDK.
2. Microsoft .NET приложения са организирани в модули. Комплект е изпълнима програма (.exe) или библиотека (.dll).
3. Visual Studio организира елементи, необходими за производството на едно събранане в проект. Един проект обединява изходни файлове и други ресурси за програмата като изображения и звуци, необходими за производството на събрание. Когато общото събрание се създава манифест се добавя към проект, който описва съдържанието.
4. Можете да използвате ildasm програма, която се доставя с Visual Studio за да видите съдържанието на модули.
5. Visual Studio решение обединява редица проекти, които може да се опише създаването на различни елементи на заявлението. Това може да включва производството на напълно различни компоненти, например на сървър и клиент части от голяма инсталация.
6. На Windows Phone приложение може да съдържа определен брой възли, включително библиотека ресурси. Комплект ще съдържа versioned препратки към всички ресурси, които той използва.
7. Когато приложение на Windows Phone е готов за разгръщане за целевото устройство се опаковат в .XAP файла, който съдържа файл-манифест, програмата възли и всички ресурси, които програмата използва.
8. Емулатор на Windows Phone работи на компютър с Windows и предоставя емуляция на функционалността на устройството, но не подражават изпълнение.
9. Visual Studio предоставя средства, с които програмист може да въведете точки на прекъсване в програма. Това са изявления, където изпълнението на програмата ще спре и програмистът може да видите съдържанието на променливите в програмата. Точки на прекъсване и единични засилване на код може да се използва в програми за изпълнение в Windows Phone устройството, както и емулатор.

Глава 4. Конструирание на програма с помощта на Silverlight

Във втора глава разгледахме някои от елементите, които Silverlight предоставя. Сега ще разширим нашите знания и ще се задълбочим в онова, което Silverlight може да направи на нас. До края на тази секция ще можете да сътворявате използваеми Silverlight приложения с много страници, които да работят на Windows Phone и да позволяват на потребителя да управлява своите данни. Главата е голяма с много демонстрации, но не се притеснявайте.

4.1 Подобряване на работата на потребителя

До момента сме способни да построим потребителски интерфейс чрез три Silverlight елемента:

- `TextBlock` за изобразяване на съобщения и надписи;
- `TextBox` за получаване на входни данни от потребителя;
- Бутон за получаване на събития.

Сега ще надградим тези три неща, за да създадем малко по-интересни места, където потребителят да работи. В същото време ще научим как Silverlight елементите работят заедно, за да построят потребителски интерфейси.

Манипулиране настройките на елемента

Виждали сме, че десктоп елементите на Silverlight всъщност са вградени чрез кодове, писани в класове и че можем да променим начина, по който изглеждат чрез промяна на настройките на входните им данни. Настройките на Silverlight елемента могат да се управляват от панела `Properties` във Visual Studio или чрез редактиране на XAML, който описва всеки елемент. Тези настройки, обаче, се правят докато се построява програмата. Можем също така да ги зададем, когато програмата е пусната. Вече сме правили това в нашето `AddingMachine` приложение, резултатът се изобразява, като се променят настройките на `ResultTextBlock`:

```
resultTextBlock.Text = result.ToString();
```

Програмите могат да манипулират всяка от настройките на изобразявания елемент. Това включва позицията на екрана, възможността да движим елфи в игрите.

Ще обърнем внимание и на други приспособления, които са на разположение. Ще използваме по-нататъшна манипулация на характеристиките, за да подобрим работата на потребителя с нашата `AddingMachine` програма. Въпреки това, само ще нахвърлим бегъл щрих върху онова, което реално е възможно. Силно Ви съветвам да разгледате добре онова, което отделните елементи могат да правят. Съществуват някои особено силни характеристики.

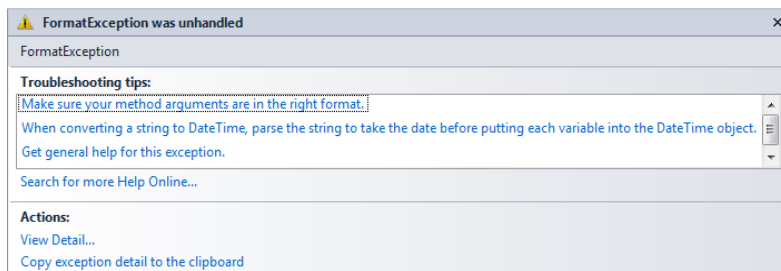
Търсене на грешки

Ще започнем с подобряване търсенето на грешки в нашата `AddingMachine`. В опита съм си прекарал почти еднакво време в това да пиша код, с който да поправам грешните входни данни на системата и да пиша програми за работещите части.

В момента редактора в нашата програма `AddingMachine` не съществува. Ако потребителят въведе текст в `TextBox`, програмата ще се опита да използва `Parse` метода, за да го преобразува в числа и ще се провали.



Отгоре виждате вид проблем при събиране в програмата. Потребителят ще твърди, че само е искал да разбере отговора на задачата „две плюс две“, но резултатът ще даде грешка.



Това е така, тъй като програмата използва много прост метод, за да преобразува текста в число в TextBox-a:

```
float v1 = float.Parse(firstNumberTextBox.Text);
```

Ако firstNumberTextBox съдържа текст, който не може да бъде конвертиран в число (напр. "две"), ще допусне изключение. Можем да подобрим това, използвайки различни версии на Parse метода:

```
float v1 = 0;

if (!int.TryParse(firstNumberTextBox.Text, out v1))
{
    // Invalid text in textbox
}
```

Кодът отгоре използва TryParse метода, който връща false в случая, че синтактичният подбор се провали. Ще се подчини на твърдението в условия оператор, ако текста в firstNumberTextBox съдържа невалиден текст. Би било хубаво, ако грешният текст се оцветява в червено в текстовото поле.

Промяна цвета на текста

Добър начин да се покаже статуса за грешка е да се промени цвета на текста. Това е особено хубав начин да се подчертае даден проблем. Silkverlight използва набор от четки, за да изобрази нещо на дисплея. Това може да бъде много мощен механизъм. Това означава, че можем да изобразим текст, използвайки изображения или материи и да се сдобием с някои доста интересни ефекти. В случай на грешка не бихме могли да сторим нещо кой знае какво освен да обозначим текста в

солиден червен цвят. Silverlight прирежава набор от подобни вградени цветове и за да маркираме текста в червено, трябва само да настроим четката за предния фон с желанния цвят:

```
float v1 = 0;
if (!float.TryParse(firstNumberTextBox.Text, out v1))
{
    firstNumberTextBox.Foreground = new SolidColorBrush(Colors.Red);
    return;
}
```

Тази визия на кода работи добре. Ако потребителят въведе невалиден номер (напр. някой, който съдържа текст), предния фон на това текстово поле се оцветява в наситено червен цвят и методът приключва, без да е изчислил резултата. Това, обаче, не е най-добрият подход. Програмата трябва да проверява повече от само първата стойност, когато се изпълнява. В случай, че потребителят е допуснал грешка в двете текстови полета, е добре да се издаде едно общо съобщение за това отколкото първо да обозначи грешката в първото поле, а след това и във второто.

Също така, съществува и по-сериозна грешка и тя е, ако потребителят въведе погрешен текст в полето, то ще се оцвети в червено, но ако се въведе правилно съдържание, полето няма да възвърне предишния си фон. Трябва да се добави else – случай, за да може полето да си върне правилния цвят. Необходимо е да се съхрани „правилната“ боя, за да се използва по предназначение. Пълната версия на calculateResult изглежда по следния начин:

```
private SolidColorBrush errorBrush = new SolidColorBrush(Colors.Red);
private Brush correctBrush = null;

private void calculateResult()
{
    bool errorFound = false;
    if (correctBrush == null)
        correctBrush = firstNumberTextBox.Foreground;

    float v1 = 0;

    if (!float.TryParse(firstNumberTextBox.Text, out v1)) {
        firstNumberTextBox.Foreground = errorBrush;
        errorFound = true;
    }
    else
        firstNumberTextBox.Foreground = correctBrush;

    // Repeat for v2

    if (errorFound)
        resultTextBlock.Text = "Invalid inputs";
    else
    {

```

```

        float result = v1 + v2;
        resultTextBlock.Text = result.ToString();
    }
}

```

Този код оцветява текста в червен цвят в случай на изникнала грешка. Освен това се запазва копие на правилния фон, така че ако стойността е вярна, да се запази точния цвят.

Кодът работи добре и предлага сравнителен потребителски опит. Също така се илюстрира, че пишейки програма за потребителски интерфейс, всъщност накрая се пише повече код отколкото само твърдения, които трябва да изчисляват резултата. Кодът, който се занимава с грешки от входни данни може доста да облекчи работата на кода, който ще пишете.

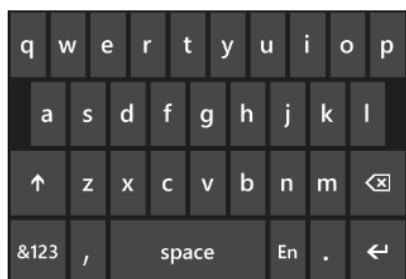
Редактиране на XAML за Silverlight елементи

XAML може да бъде описан като "декларативен" език. Това означава, че всъщност не се описва всяко поведение (т.е. не се казва на компютъра как да се прави нещо). Вместо това, само се указват горепосочените неща. По отношение на C# нещата, които искаме да използваме (напр. променливи) се указват на компилатора чрез деклариране. Езикът XAML се използва само, за да информира Silverlight за неща и по тази причина се нарича декларативен език. Ние ще трябва да свикнем с пускането на някои аспекти от програмата в декларацията на Silverlight елементите, които намират приложение. Един такъв случай е в конфигурацията на входния режим за TextBox елементите, които се използват за въвеждането на номера в AddingMachine програмата.

Конфигурация на TextBox и използване на клавиатура с цифри

В момента AddingMachine използва стандартното поведение на клавиатурата на телефона Windows, което да показва текстовата клавиатурата, когато потребителят избира този контрол. Би имало смисъл да се предостави на потребителя клавиатурен набор за въвеждане на цифри отколкото текстов такъв. Има няколко различни клавиатурни конфигурации, които Windows Phone може да изобрази. По-долу можете да видите стандартен "текстов" такъв.

За да се въвеждат цифри, потребителят трябва да избере '&123'.



Би било хубаво, ако всеки TextBox в изчислителната машина е настроен за цифров вход, когато текстовото поле се отвори. Това може да се направи чрез изменение на настройките на TextBox-а,

когато програмата се стартира, но най-добрият начин за това е XAML описанието на самия контролер. В момента XAML описва secondNumberTextBox както следва:

```
<TextBox Height="72" HorizontalAlignment="Left" Margin="8,175,0,0"
Name="secondNumberTextBox" Text="0" VerticalAlignment="Top" Width="460"
TextAlignment="Center" />
```

Това указва на Silverlight къде на екрана да сложи елемента, неговия размер и наименование. Също така има и информация за аранжирането на текста.

Всяка от стойностите в <TextBox .. /> е атрибут а текстовото поле. Атрибутите са прости средства, които не са структурирани и могат да бъдат изразени чрез име – двойка стойности. Отгоре може да се види, че height, name и други настройки на елемента са изразени по този начин.

За да се каже на текстовото поле какъв тип текст трябва да се въведе, може да се добави допълнителна информация. Това се нарича InputScope of the TextBox.

Тази информация се задава като property element, която е част от TextBox стойността. За да изразим това, трябва да се използва малко по-сложна версия на TextBox:

```
<TextBox Height="72" HorizontalAlignment="Left" Margin="8,19,0,0"
Name="firstNumberTextBox" Text="0" VerticalAlignment="Top" Width="460"
TextAlignment="Center">
    <TextBox.InputScope>
        <InputScope>
            <InputScopeName NameValue="Digits" />
        </InputScope>
    </TextBox.InputScope>
</TextBox>
```

Ако желаете XAML елемента да съдържа характеристики, е необходимо да се използва различна форма на описанието. Това е малко объркващо в контекста на Silverlight, така че е препоръчително да се огледа нещо, по-лесно за разбиране.

Елемент без екстри

XAML форматът може да се използва за създаването на език, който ще съдържа информация за хората. Може да се започне със съхраняването името на човека:

```
<Person name="Rob"/>
```

Елементът се нарича „Person” и има единствен атрибут, наречен “Name”, указан към “Rob”. XAML компилаторът забелязва /> накрая на дефинирането на елемента и знае, че е достигнал края на описанието на този елемент.

Елемент с екстри

Освен това, може да искаме да съхраним по-сложна информация за дадено лице като напр. къде живее. Адресът ще бъде съставен от много различни елементи и така адресът се превръща в свойство на лицето.

```
<Person name="Rob"> <Address addressType="HomeAddress"> <FirstLine text="18  
Pussycat Mews"/> <Town text="Seldom"/> <County text="Wilts"/> <PostCode  
text="NE1 410S"/> </Address> </Person>
```

Елементът съдържа едно качество, което е адреса, който сам по себе си е друг елемент с единствен атрибут (вида на адреса) и четирите качества. Това илюстрира важно качество на XML ориентираните езици. Характеристиката е XML елемент като всеки друг, което означава, че има атрибути и собствени свойства. Silverlight се възползва от това чрез Grid елемента, който може да съдържа колекция от Silverlight елементи (вкл. Grids), които да се видими на страницата.

XAML компилаторът може да разпознае дали елемент съдържа свойства, тъй като първият ред от описанието на този елемент завършва с > вместо с />. XAML компилаторът ще вземе предвид всичко, което забележи след първия ред на елемента докато не види

```
</ElementName>.
```

Ако в началото си мислите, че това е объркващо, добре дошли в клуба. Най-добрият начин това да се възприеме де да се вземе по внимание онова, което се съхранява. В случая на Person от по-горе името е единствен артикул, който може да се изрази в прост елемент. Въпреки това, адресът ще съдържа множество собствени свойства и така най-добре се превръща в характеристика. Съществува и възможността Person да има няколко адресни стойност, напр. HomeAddress и WorkAddress като по този начин горният дизайн улеснява добавянето на нови адресни типове.

Видът адрес е направен атрибут на адреса, което смятам, че е най-подходящо. Дали данните да се направят атрибут на състоянието или не, е нещо, което трябва да се има предвид, когато се използва XML (eXtensible Markup Language), за да се описват различни неща.

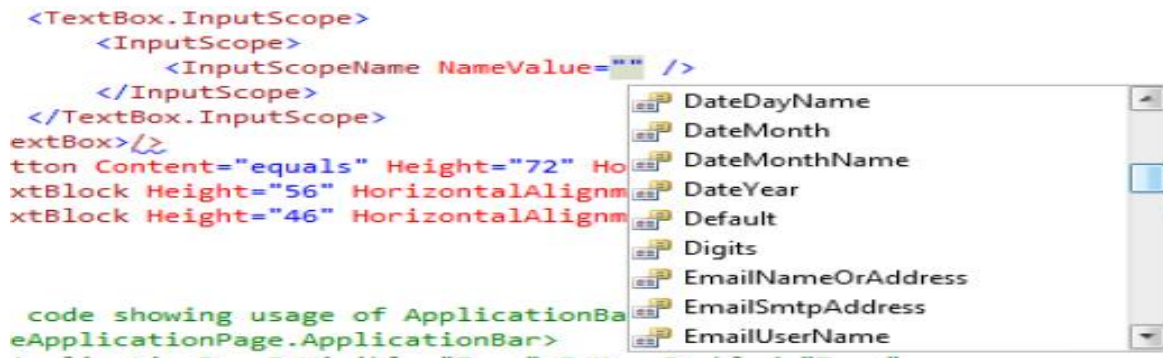
Ако се вънем към нашата TextBox , би трябвало да открием, че нещата ще бъдат малко по-смислени.

```
<TextBox Height="72" HorizontalAlignment="Left" Margin="8,19,0,0"  
Name="firstNumberTextBox" Text="0" VerticalAlignment="Top" Width="460"  
TextAlignment="Center"> <TextBox.InputScope> <InputScope> <InputScopeName  
NameValue="Digits" /> </InputScope> </TextBox.InputScope> </TextBox>
```

Семплата информация за текстовата кутия е дадена като атрибут докато InputScope стойността се задава като качество. InputScope сам по себе си съдържа набор от свойства, всяко от които описва опеределеа сфера. Това означава, че някой ден може да се нуждаем от входна характеристика, включваща чиса и текст, но не и пунктуация. Въпреки това, не искаме нещо толкова сложно. Само искаме да извлечем Digit входни данни и също така те да са част от InputScope, който сме задали.

Ако използваме горното описание за двете текстови полета в сметачната машина, получаваме широко подобрен потребителски интерфейс, където на клиента се предоставя клавиатура с числа, когато се достигне до входните елементи.

Това е пример на ситуация, където само добавянето на текст в XAML е най-бързият и лесен начин да се конфигурира държанието на входните данни. Можехме да използваме панела с характеристики на Visual Studio или дори сложно написан код, за да построим целостта от InputScopeName стойности и да ги прикачим към TextBox, но това решение е много по-чисто и също така ни позволява да усъвършенстваме познанията си по XAML.



Visual Studio предоставя характеристика, наречена "Intellisense", където редакторът Ви помага, като се опитва да разбере какво искате да напишете. Тъй като Visual Studio знае всички възможни стойности за InputScope, Ви ги предлага, когато започвате да въвеждате даден израз. Отгоре можете да видите какво се случва, когато Intellisense открие, че въвеждаме InputScope стойност.

XAML атрибути и екстри - истината

От гледна точка XAML се оказва, че атрибутите и характеристиките са взаимозаменяеми. Софтуерът, използващ XAML описанието получава информация от коя да е част на XML елемента. Вместо малко сложният вариант от по-горе, всъщност можем да напишем:

```
<TextBox InputScope="Number" Height="72" HorizontalAlignment="Left"
Margin="144,44,0,0" Name="startHourTextBox" Text="00" VerticalAlignment="Top"
Width="104" TextAlignment="Center" />
```

Това задава на входните параметри на TextBox параметри. Няма да работи, ако желаяем да управляваме няколко стойности по едно и също време, но въпреки това е полезно да се зададе една стойност. Обратното на това също е в сила.

```
<TextBox HorizontalAlignment="Left" Margin="8,175,0,0"
Name="secondNumberTextBox" Text="0" VerticalAlignment="Top" Width="460"
TextAlignment="Center"> <TextBox.Height> 72 </TextBox.Height> </TextBox>
```

В XAML височината на текстовото поле се изважда от атрибути и се превръща в стойност. Името на стойността трябва да съдържа името на типа елемент, от който е част. Не може просто да се укаже

Height, а е необходимо да запише по следния начин TextBox.Height. Това не е ограничение на XML (не беше нужно да се извърши тази стъпка в по-горния пример Person), но е необходимост за XAML.

От гледна точка на XAML писателят, не е от голямо значение коя по-точно форма ще се използва. За по-прости двойки име/стойност може да се използват атрибути, а за по-сложни структури – електри.

Стойностни характеристики на C#

В случай, че се питате как биха изглеждали същите функции в C#, ето как:

```
// Make a new input scope
InputScope digitScope = new InputScope();
// Make a new input scope name
InputScopeName digits = new InputScopeName();
// Set the new name to Digits
digits.NameValue = InputScopeNameValue.Digits;
// Add the name to the new scope
digitScope.Names.Add(digits);
// Set the scope of the textbox to the new scope
firstNumberTextBox.InputScope = digitScope;
```

Когато четекете този код внимателно, ще забележите, че той изгражда всеки елемент от XAML.

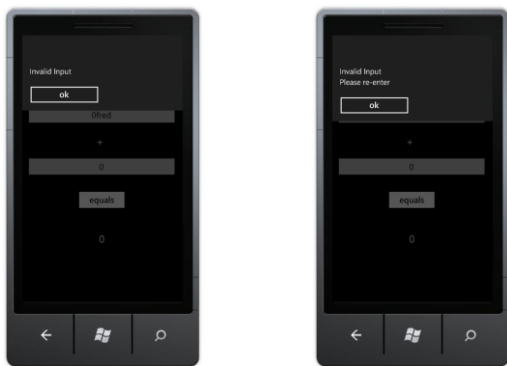
Проблемът в Демо01 Сметачна машина с Редактор внедрява версия на калкулатор, който извършва проверка и обозначава грешките в червено. Също така задава входните материали на TextBoxes да бъдат числени.

Изобразяване на MessageBox

Silverlight предоставя MessageBox обект, който може да бъде използван, за да се указват грешките на потребителите:

```
MessageBox.Show("Invalid Input");
```

По този начин в горния край на екрана на телефона ще излезе съобщение, което потребителят трябва да премахне преди да продължи нататък. Също така се пуска и алармен звук.



Ако искате да изпращате по-големи съобщения на мобилния потребител, бихте могли да сторите това, изобразявайки няколко реда в съобщение.

```
MessageBox.Show("Invalid Input" + System.Environment.NewLine + "Please re-enter");
```

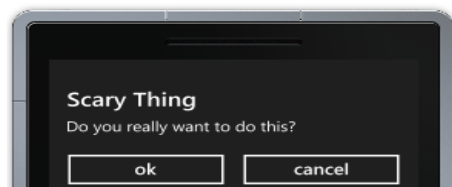
Елементът `System.Environment.NewLine` осигурява символ за нов ред, подходящ за дадената платформа. `MessageBox` е полезен в случай, че искате да бъдете сигурни, че потребителят е обърнал внимание на съобщението преди да продължи. Освен това, необходимо е да се зпомни, че когато то се изобразява, програмата ще спре в този момент.

Проблемът в `Demo02 AddingMachine with MessageBox` изобразява поле за съобщение, когато се въведе грешна стойност от потребителя.

Поле за съобщения със селектиране

Може да пожелаем потребителят да направи избор в `MessageBox`:

```
if (MessageBox.Show("Do you really want to do  
this?", "Scary Thing",  
MessageBoxButton.OKCancel) ==  
MessageBoxResult.OK) { // do scary thing here }  
else { // do something else }
```



Тази версия показва съобщение с опция да бъде отменено, както е показано по-долу. Условният оператор `if` се изчислява спрямо реакцията на потребителя.

Добавяне и използване на придобивки

В предходната част видяхме как `Visual Studio` управлява екстри като изображения и звук и как ги добавя към `XAP` файл за трансфер към посоченото устройство. Сега ще видим как приложението ни може да добавя наши екстри и да ги използва.

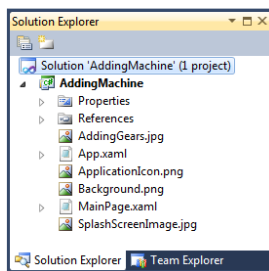
Заден екран на машина за събиране

Помолени сме от сина на шефа на наш клиент (който се смята за графичен дизайнер) да добавим изображение като заден фон на приложението за смятане. Това всъщност е фотография на механизма на „Различния мотор“ на Чарлс Бабидж. Клиентът мисли, че това изображение ще направи програмата много по-добра. Ние не сме убедени, но тъй като той е синът на шефа, трябва да се примирим с това.

Принадлежности за Windows Phone

Когато се добавя атрибут като този, важно е да се вземе под внимание, че таргет устройството няма много памет или изключително голяма резолуция на екрана. Ако горната картинка беше част от високо резолуционна камера, можеше да съдържа много повече детайли отколкото е необходимо за телефонно приложение. Запомнете, че най-високата резолуция за Windows Phone е 800x480 пиксела. Няма смисъл в използването на фоново изображение с по-висока резолуция от тази.

Добавяне на изображения като част от съдържанието



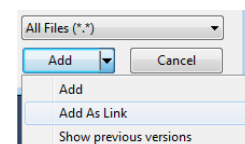
Изображение може да се добави като част от съдържанието на проект AddingMachine. Синът на шефа ни е дал jpeg изображение AddingGears.jpeg, което да използваме. Оказва се, че най-бързият начин, по който да се добави съдържанието във Visual Studio е като само извлечем обекта от папката и да го поставим в проекта.

По този начин се прави копие на файловия ресурс в директорията на проекта в подходящия момент.

Линкове към екстрите

Ако искаме да разпространим дадена принадлежност към няколко различни проекта (вероятно имате лого на компанията, което може да се използва от много различни програми), тогава ожем да добавим линк към ресурса, но за да направим това, е необходимо да добавим файла чрез диалога Add Existing Item, след което се избира “Add as Link”.

Ако променим логото на файла, новата версия ще бъде избрана от проекти, които ще я използват следващия път, когато се обновяват.

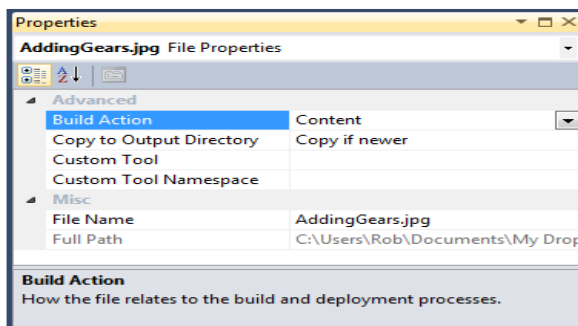
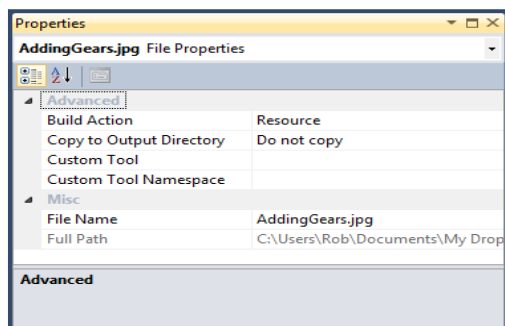


“Build Action” на придобивката

Когато Visual Studio построява проект, има няколко начина, по които екстрите могат да бъдат добавени в завършената програма. Под внимание ще вземем две; добавяне на екстра като файл от съдържанието или като ресурс вътре в асемблера на програмта. Решението, кое да използваме, може се обмисли докато програмата зареди и стартира, така че е полезно да се знае малко за това как механизма действа.

Добавяне на придобивка като съдържание

Когато се добавя екстра към проект, тя по начало се добавя с вградено действие „Ресурс“. В случай, че желаем да използваме придобивката като съдържание, е нужно да се променят свойствата ѝ, така че да се превърне в част от него. Процесът се извършва, като се променя входа на Build Action за дадения файл.



Придобивките на съдържанието се копират от Visual Studio в директорията на приложението докато програмата е в процес на посрояване. Настройката “Copy if newer” означава, че Visual Studio ще копира нова версия на изображението в двойната папка, ако сегашното изображение в проекта е заместено от по-нов файл.

Изпозване на Image Content в програма

Предимството на даденото съдържание е достъпно за приложението също както файлът от папката на самата програма. За да се изобрази желаната картинката за съдържанието на нашата програма, нужно е да се добави ред от XAML, който да описва файлът – изображение, който искаме да използваме:

```
<Image Height="611" HorizontalAlignment="Left" Margin="8,0,0,0"
Name="gearBackgroundImage" Stretch="Fill" VerticalAlignment="Top" Width="472"
Source="AddingGears.jpg" />
```

Елементът Image показва картинката на страницата. Той има Source атрибут, който идентифицира файлът, от където е заредено изображението. Ако се въведе файлово име като източника, изображението ще се зареди от файла. Когато програмата е стартирана, файлът ще бъде зареден от папката и тогава показан на екран. Silverlight рисува елементите на екран, като по този начин те са дефинирани в XAML. Ако този ред е на върха на Grid-a, съдържащ елементите, изображението ще се покаже на заден фон:

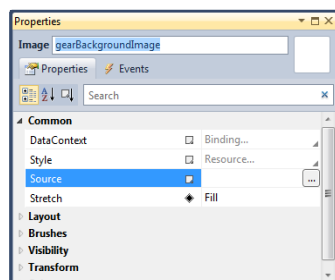


Това е резултатът от програмата. От артистична гледна точка не сме сигурни, дали картинката е подходяща, но механизмът ще работи.

Проблемът в Demo03 AddingMachine with Background Content показва фоново изображение за приложение, което е заредено като част от съдържанието.

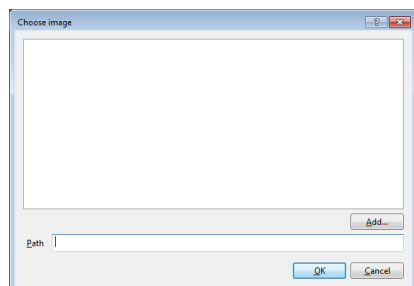
Добавяне на изображения като ресурс

Съществува и друг начин да се добави изображение към проект. Това включва създаването на нов image атрибут от Toolbox във Visual Studio редактора и тогава да се избере ресурс за изображение, който да се използва за този атрибут чрез панелът Property на Visual Studio.

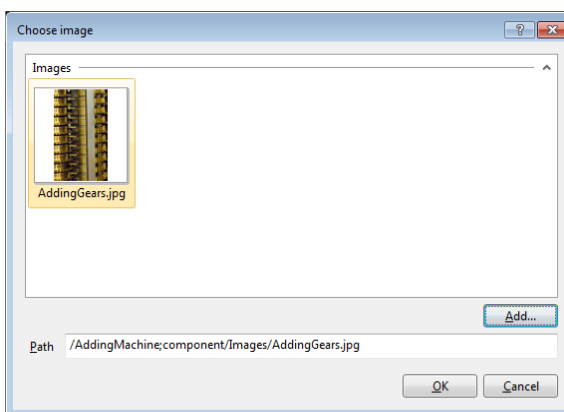
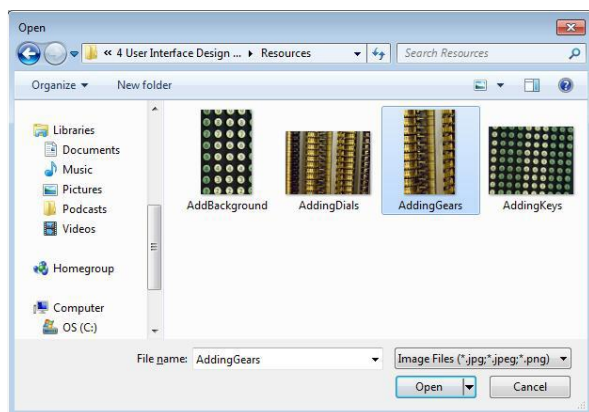


Първата стъпка в този процес е да се извлече изображението от Visual Studio Toolbox върху страницата. Тогава се отваря Property панела за този ресурс:

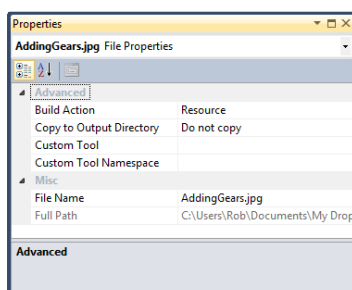
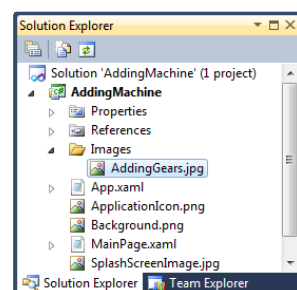
Името е променено на gearBackgroundImage и следващото нещо, което трябва да се направи е да се определи източник за изображението, кликайки върху „....“ бутона отдясно на източника. Това отваря диалогът Choose Image.



Диалогът показва всички изображения, които са запазени до момента в приложението. Тъй като не е добавено нито едно, диалогът стои празен. Ако кликнем върху Add, можем да отворим файлов браузър, за да търсим подходящи картинки от компютъра.



Това добавя специфичното изображение на онези, които са достъпни в проекта.



Също така се създава ресурсен файл в проекта за този обект. Всякакви други изображения, които се добавят в проекта, ще бъдат добавени в папката, която е създадена чрез VisualStudio. Свойствата на тези картинки ще бъдат зададени акто ресурси, а не съдържание.

XAML, който описва изображението съдържа неговият източник, който се отнася до ресурса на приложението:

```
<Image Height="611" HorizontalAlignment="Left" Name="gearBackgrounds"  
Stretch="Fill" VerticalAlignment="Top" Width="472"  
Source="/AddingMachine;component/Images/AddingGears.jpg" />
```

Когато програмата е пусната, дисплеят остава абсолютно същият, но има съществена разлика в това, как ресурсът се използва и къде се намира.

Проблемът в Demo04 AddingMachine with Background resource изобразява фоново изображение за приложението, заредено като ресурс.

Съдържание – Ресурси

Когато се създава приложения за Windows Phone, от значение е да се разбере разликата между съдържание и ресурс в приложението.

Една част от съдържанието е само файл, запазен в директорията на приложението. Програмите ни могат да отворят и използват файловете на съдържанието, когато имат нужда от тях. Добавянето на съдържание не се отразява върху размера на самата програма, макар че когато програмата е стартирана и зарежда съдържание, ще се нуждае от повече памет.

Артикул от ресурса се съхранява в програмния асемблер. Добавянето на фоново изображение на механизъм към асемблера от по-горе равни програмата по-голяма с около 317Кб. Когато Windows Phone устройството зареди програмата, се преглежда целият асемблер и се прави проверка, за да се увери, че кодът е легален. Тези проверки отнемат повече време, ако програмата е по-голяма. Но ако на нея й отнеема много време, за да стартира докато се зарежда, то тя ще бъде унищожена от операционната система на Windows Phone. Това означава, че трябва да се внимава, когато асемблерът се натоварва с много неща.

Добавяне на изображение като част от съдържанието прави самата програма по-малка, което означава, че тя ще се зареди по-бързо. Освен това, файловете от съдържанието се нуждаят от малко повече време, да се заредят, тъй като се взимат от съдържащата ги папка, когато биват използвани. Добавянето на нашето изображение като ресурс ускорява процеса на зареждане и взимане на изображението, но уголемява самата програма и по този начин я прави по-бавна.

Ако това Ви се струва объркващо, ето някои опорни точки:

- Ако имате много големи ресурси и нямате нужда да зареждате всички тях наведнъж в програмата, използвайте съдържание. Пример би могъл да бъде, там, откъдето се позволява на потребителя да избере различни фонове изображения за екрана си.
- В случай, че имате мали ресурси, които се използват през цялото време на програмата, се използва ресурс. Например, ако имате икони, означаващи определена среда или опции.

- Ако откриете, че на програмата Ви й отнема много време, за да стартира, би трябвало да преместите някои от ресурсите извън програмният асемблер и сами да добавите зареждащ екран, който да показва къде да бъдат в паметта, когато програмата стартира. Също така, можете да ускорите стартовия процес на програмата, като я разделите на по-малки асемблери, които са свързани помежду си.

4.2. Манипулация на данни и изобразяването им

Нашата сметачна машина сега е използвана, но клиентът все още не е напълно доволен. Той не харесва начина, по който потребителят трябва да натисне Calculate бутона всеки път, когато желае нов отговор. Това, което иска той от програмата е тя да може да разбира, кога текста в полето с цифри се променя и да го обнови автоматично. За щастие, това е лесно изпълняема за нас задача. Необходимо е да посетим събитията наново, за да открием повече.

Събитието TextChanged

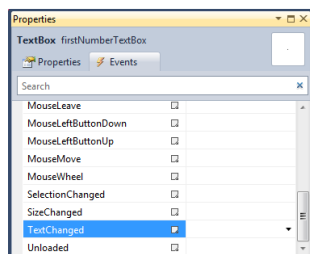
Първо видяхме събития, когато имаме предвид елемента Button. Той генерира събитие, когато е активиран. От гледна точка на програмата събитието се нарича метод. Visual Studio може да се използва, за да “закачи” събитие към бутон само чрез двоен клик върху бутона в конструктора. До момента Calculate бутона притежава клик-държание, което изчислява и изобразява резултата. Методът, който ще се извика се индентифицира в описанието на Button:

```
<Button Content="equals" Height="72" HorizontalAlignment="Left"
Margin="158,275,0,0" Name="equalsButton" VerticalAlignment="Top" Width="160"
Click="equalsButton_Click" />.
```

Когато се кликне върху бутона, това предизвиква избраното поддържащо устройство на събитие да се зареди вътре в класа на страницата. В случая на AddingMachine това ще извика calculateResult метода:

```
private void equalsButton_Click(object sender, RoutedEventArgs e) {
    calculateResult(); }
```

Може да се разбере какво могат да сътворят събитията от TextBox, като се избере TextBox в редактора на Visual Studio, а след това – Events в панела с характеристики:



TextBox може да създаде много събития, включително и някои изглеждат на пръв поглед безполезни за Windows Phone. Visual Studio дава преднина на събитието, което е най-полезно от TextBox, т.е. TextChanged. Това се зарежда всеки път, когато текста в TextBox се променя. Ако кликнем два пъти върху TextChanged в Properties панела, Visual Studio ще създаде метод и ще го прикачи към това събитие:

```
private void firstNumberTextBox_TextChanged(object sender, TextChangedEventArgs e)
{ }
```

```
}
```

Необходимо е само да се добави `calculateResult` в този метод:

```
private void firstNumberTextBox_TextChanged(object sender, TextChangedEventArgs e)
{
    calculateResult();
}
```

Същото може да се тори за второто текстово поле, така че ако някое от тях бъде променено, то сумата да се обнови. Сега това означава, че `Calculate` бутона може да се премахне.

Двойно променени събития



Ако заредим тази програма, ще открием, че тя работи сравнително добре. Въпреки това, се забелязва проблем с валидацията на данни. В случай, че потребителят въведе текст вместо номер, програмата ни ще обърне внимание на това и ще изведе съобщение:

Освен това, съобщението се изписва два пъти за всеки невалиден символ, който е въведен. Това е сякаш `TextChanged` се зарежда два пъти за всеки символ. Оказва се, че такъв е и случаят, който е вече позната тема на Windows Phone приложенията. Можем да се разминем с това, като проверяваме за събития, които успешно са се изменили, но имат едни и същи данни:

```
string oldFirstNumber = ""; private void firstNumberTextBox_TextChanged(object sender, TextChangedEventArgs e) { if (firstNumberTextBox.Text == oldFirstNumber) return; oldFirstNumber = firstNumberTextBox.Text; calculateResult(); }
```

Този метод извиква единствено `calculateResult`, ако текста в `TextBox` реално се е изменил.

Проблемът в `Demo05 AddingMachine with no button` реализира сметачна машина, която използва `TextChanged` събития и не притежава бутон за изчисления.

Свързване на данни

Свързването на данни е прекрасно нещо, което играе ролята на „лепило“ в кода, който пишем и позволява свързването на елементи за дисплей и методи, които работят върху тях. Способни сме да вземем свойство на елемент от екрана и да го свържем директно към обект. Всичко, което трябва да направим, е да създадем обект, който се държи по особен начин и така ще можем да получаваме стойности в обекта, пренесен в Silverlight елемент без уилие от наша страна.

Връзката между данните работи в коя да е директория. Може да свържем `TextBox` с нашето приложение, така че когато потребителят променя текста в текстовото поле, нашето приложение да бъде информирано за това. Също така, можем да свържем `TextBlock` към обект, така че когато свойството в обекта е променено, съдържанието на `TextBlock` да се обнови, за да пасне. Това е „двустранно“ свързване, където програмата може както да променя елемент, така и да отваря на настъпили промени. Най-накрая, можем да свързваме всяко свойство на обекта, така че

програмата да бъде способна да мести Silverlight елемент през екрана, като само променя X и Y характеристиките на желания обект.

Създаване на обект, който да бъде свързан

Ще започнем със създаването на версия на сметачната машина, която използва свързани данни. Началната точка е клас, който ще свърши цялата работа. Това ще изобличи свойствата на данните, които ще се свържат с тези на елемента в обекта, който ще бъде изобразен. Обектът трябва да съдържа следните три неща:

- Текстът в началото на TextBox
- Текстът в края на TextBox
- Текстът в получения TextBlock

Можем да създадем AdderClass, който да изрази всичко това:

```
public class AdderClass
{
    private int topValue;
    public int TopValue
    {
        get
        {
            return topValue;
        }
        set
        {
            topValue = value;
        }
    }

    private int bottomValue;
    public int BottomValue
    {
        Get
        {
            return bottomValue;
        } set
        { bottomValue = value; } }
    public int AnswerValue
    {
        get
        {
            return topValue + bottomValue;
        }
    }
}
```


Този клас притежава три свойства. Две от тях имат `have` и `get` държание, така че потребителят на този клас да може да открие стойностите на най-горните и долните данни (нещата, които се добавят заедно) и да им зададе нови стойности. Третата характеристика е само получената стойност (резултатът от сумата). Това никога не се задава като стойност; юзърът на `AdderClass` само ще прочете това свойство и ще получи отговора по-късно.

Класът извършва цялото „мислене“, нужно, за да работи машината. Програмата може да се възползва от това, да зададе горни и долни стойности и тогава да се върне и прочете отговора. Ако сме искали да я променим в „изваждаща“, само трябва да сменим класа с друг, който ще работи различно. Това би било добро дизайнерско решение независимо дали използваме Silverlight или не.

Добавяне на свойство за изместване

За да може класът да свързва Silverlight обектите, в него трябва да е вграден `INotifyPropertyChanged` интерфейс:

```
public interface INotifyPropertyChanged { // Summary: // Occurs when a property value changes. event
    PropertyChangedEventHandler PropertyChanged; }
```

За да се осъществи този интерфейс, класът трябва да съдържа заявление за събитие, който да се използва от `AdderClass`, за да съобщава всичко интересно, когато свойството се изменя.

```
public event PropertyChangedEventHandler PropertyChanged;
```

Типът `PropertyChangedEventHandler` се използва от Silverlight елементите, за да управлява съобщенията. Описва се в стандарта `System.ComponentModel`. Ако Silverlight компонента желае да се свърже с някой от свойствата в класа, може да добави заявления в събитието.

`AdderClass` използва заявлението, за да съобщава на останалите компоненти кога някое свойство в класа се е променило. Това ще предизвика обновяване на екрана. Silverlight обектите, които са свързани с нашите свойства се свързват с тази заявка, така че да бъдат уведомени, когато се налага. Крайната версия на класа изглежда по следния начин:

```
namespace AddingMachine
{
    public class AdderClass : INotifyPropertyChanged
    {
        public event PropertyChangedEventHandler PropertyChanged;
        private int topValue;
        public int TopValue
        {
            get
            {
                return topValue;
            }
            set
            {
                topValue = value;
            }
        }
    }
}
```

```

if (PropertyChanged != null)
{
    PropertyChanged(this, new PropertyChangedEventArgs("AnswerValue"));
} } }

// repeat for bottom value
public int AnswerValue
{
    get
    {
        return topValue + bottomValue;
    } } }

```

AdderClass сега съдържа член, наречен PropertyChanged. Когато класът желае да промени свойство на стойност, трябва да изпробва PropertyChanged на стойността. В случай, че не е нула, това означава, че нещо е свързано към заявката. PropertyChanged може да се счита като списък по имейл. Процесът може да се запише към списъка и да бъде информиран за интересни събития, които се случват в AdderClass. Ако никой не се е записал към този списък (т.е. PropertyChanged стойността да е нула), тогава няма смисъл в извикването на метод, който да описва промяната. Освен това, ако не е нула, това означава, че има нещо, което иска да бъде уведомявано за промените в AdderClass.

Форматът на извикването на метода ни предоставя събитието по следния начин:

```

PropertyChanged(this,new PropertyChangedEventArgs("AnswerValue"));

```

Първият параметър е обратна връзка към обекта, който генерира събитието. Вторият е стойност на аргумента, заредена с името на свойството, което се е променило. Silverlight елементът ще използва обратна връзка, за да знае кои характеристики са достъпни (т.е. ще се връща към общодостъпните свойства, предоставени от AdderClass). Ако получи съобщение, че AnswerValue се е изменил, тогава ще обнови всяка характеристика на екранния елемент, свързана към свойство на този клас.

Забележете, че това е само текстов низ, ако напишем "Answervalue" по грешка, това ще предотврати правилното функциониране на въпдейта на съобщения.

Когато програмата изключи промененото събитие, това ще предизвика извикването на обновена стойност на AnswerValue от екрана на резултата. Когато get държанието на AnswerValue е извикано, то изчислява и връща общия резултат.

```

public int AnswerValue { get { return topValue + bottomValue; } }

```

Не се притеснявайте, ако намирате това сложно. Просто запомнете защо се прави. Този обект ще бъде свързан с потребителския интерфейс, който ще каже на нашия

AdderClass кога стойностите на двата входа са променени от потребителя. Когато входните данни са изменени AdderClass трябва да има незабавен начин, по който да съобщи на потребителския интерфейс, че в момента има нов краен резултат, който да бъде показан.

Добавяне на стандарт, съдържащ нашия клас в главната страница

Сега имаме обекта, който искаме да прикачим към данните. Следващото нещо, което е необходимо е да се свърже кода към XAML, който описва потребителския интерфейс. По тази причина трябва да се добави стандарт, съдържащ AdderClass в XAML.

```
xmlns:local="clr-namespace:AddingMachine"
```

Ако видите последната версия на AdderClass от по-горе, ще откриете, че това е стандарта, използван за сметачната машина. Добавяйки на ред н-а-отгоре в XAML файла при MainPage.xaml, ние указваме на системата, че всякакви класове в AddingMachine стандарта би трябвало да са достъпни в приложението ни в XAML.

По същия начин, по който C# програмата трябва да има всички употребяващи се директиви най-отгоре на файла-източник, XAML файлът също трябва да има всички използвани стандарти в началото си. Ако се погледнат първите редове на файла, ще се забележи множество от добавени стандарти. Това са нашите класове за Silverlight елементите, които изграждат потребителския интерфейс.

Веднъж, след като стандартът е добавен, може да се започне създаването на име за ресурсите, които могат да бъдат открити там.

```
<phone:PhoneApplicationPage.Resources> <local:AdderClass x:Key="AdderClass" />
</phone:PhoneApplicationPage.Resources>
```

Ако това Ви се вижда като много работа, тогава помнете, че се прави само веднъж и че класът, който добавяме може да съдържа много свойства, с които потребителския интерфейс да комуникира.

Добавяне на класове към елемент от страницата

Сега след като стандартите са на разположение, определени класове могат да бъдат свързани към елементи от страницата ни. Елементът, към който ще добавим класа ще бъде Grid, който съдържа всички наши екранни елементи. Добавянето на клас към елемент автоматично прави всички съдържащи елементи достъпни, като по този начин TextBox и TextBlock елементите могат да използват този клас:

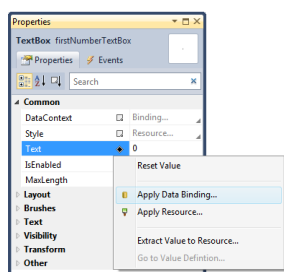
```
<phone:PhoneApplicationPage.Resources> <local:AdderClass x:Key="AdderClass" />
</phone:PhoneApplicationPage.Resources>
```

Получаваме елемент, с който да работим чрез задаването на стойност на DataContext за този компонент. За нашето приложение това ще бъде статичен ресурс, който ще бъде част от програмата. Това може да изглежда малко усложнено, но позволява създаването на тестов даннов контекст, който може да се използва вместо „истински“ такъв.

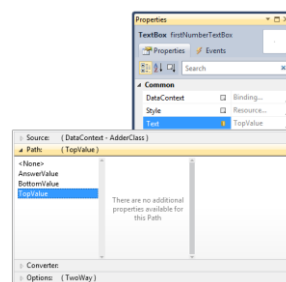
Свързване на Silverlight елемент към дадено свойство

Сера имаме AdderClass на разположение като даннов контекст на елементите от страницата. Това задава връзка между страницата на Silverlight и обекта, който ще предостави формални отношения. Следващото необходимо нещо е да се свържат свойствата от всеки елемент с тези, които бизнес обекта ще изобразява.

Това може да се направи от Property панела за всеки един от елементите. Нека започнем със свързването на текста от firstNumberTextBox с TextTopValue свойството в нашия AdderClass. Ако кликнем върху TextBox във Visual Studio редактора, можем да отворим Property панела. Тогава можем да намерим Text свойството. До този момент то се задава само като низ, а ние искаме да добавим Data Binding към обекта на машината за събиране:



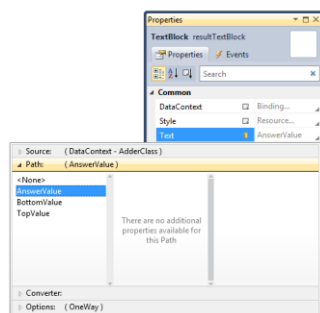
Ако кликнем върху „Adding Data Binding” следващото нещо, което трябва да се направи е да се намери характеристика, която да бъде свързана. Зад кулисите Visual Studio спира за момент и отваря AdderClass, ако използва свойствата, които по подразбиране ще бъдат използвани. След това може да изобрази меню



на тези, които ще се предоставят на достъп:

Тук се случва магията. Просто чрез избиране на искания елемент от списъка можем да свържем свойството на Silverlight елемента с обекта. Обърнете внимание, че опцията TwoWay е избрана, така че когато потребителят променя стойността на TextBox, обелът да бъде информиран.

Когато свързваме textBlock, имаме единствено OneWay опцията като начин на свързка, тъй като не е възможно да се изпращат данни от textBlock към програмата.



Веднъж след като сме осъществили тези линкове, преограмата ще проработи. Интересното нещо тук е, че ако се погледне в MainPage.xaml.cs файла за този програмния код, който кара програмата да работи, ще откриете, че той е напълно празен. Цялата работа сегасе извършва от миниатюен клас, свързан към елементите от дисплея. Промени в съдържанието на textbox елементите имаше за цел да държи входните събития в Adder клас. Този клас променя съдържанието на AnswerValue, свързан към ResultTextBlock на екрана.

Проблемът от Demo06 AddingMachine with data binding представя машина за събиране, която използва свързване на данни и входни данни от AdderClass.

Свързване на данни чрез използване на даннов контекст

Можете да търсите начин, по който да свързвате данни и да си мислите, че изглежда добре и знаете как действа, но да имате много ресурси и други неща за подвключване. Хубавите вести тук

са, че има пряк път, по който просецът ще улесни значително всичко. Това означава добавяне на твърдение към кода.

Задаване на Data Binding в XAML

Вече видяхме как управляващите елементите могат да бъдат свързани към свойствата на обекта. Сега можем да свържем свойство на обекта към свойство на елемента и Silverlight ще бъде моста между тях.

```
<TextBox Height="72" HorizontalAlignment="Left" Margin="8,19,0,0"
Name="firstNumberTextBox" Text="{Binding TopValue, Mode=TwoWay}"
VerticalAlignment="Top" Width="460" TextAlignment="Center" >
```

Показано е как механизма действа чрез XAML. Подчертан е начинът, по който данните в полето за номера се свързват. Така се задава името на свойството, което текстът ще свърже и начинът на връзка. Връзката е зададена като TwoWay, защото искаме промените от програмата (когато тя е изчислила резултата) със сигурност да бъдат изобразени. Понякога само искаме да използваме екранните елементи, за да покажем стойността (напр. резултат от изчисление). В този случай OneWay намира приложение.

Настойване на DataContext

Сега всичко, което трябва да се стори е да се създаде отделен случай на AdderClass и да се зададе DataContext на елемента, който съдържа firstNumberTextBox за този случай. Това се извършва в конструктора на главната страница.

```
// Constructor public MainPage() { InitializeComponent(); AdderClass adder = new
AdderClass(); ContentGrid.DataContext = adder; }
```

DataContext на Silverlight елемента идентифицира обекта, който ще съдържа всички свойства на контролера както и онези, които той използва.

ContentGrid на страницата съдържа textbox и TextBlock елементите. Всяка връзка, която те съдържат ще бъде добавена към новия елемент AdderClass, който създадохме.

С други думи, заменихме всичката сложна работа с property панели и ресурси с по-малко количество XAML и единични твърдения.

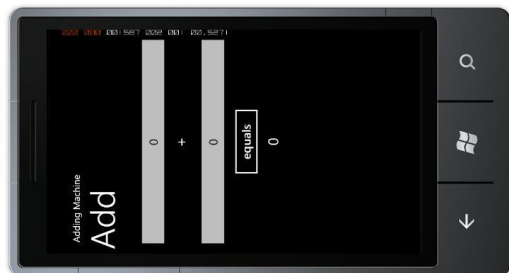
Казусът Demo06 AddingMachine with data binding показва изчислителна машина, която използва линкове между данни, за да свърже входните данни към AdderClass.

4.3. Управление на външния вид на страницата на приложението

Сега ще си починем малко от свързването на данните и ще обърнем внимание на други теми, засягащи външния вид на приложението, най-вече имайки Windows Phone на ум. Ще започнем с ориентирането на телефона и ще продължим с интересни и полезни начини, по които дизайнът може да стане видим.

Програми за пейзаж и портрет

Може да сте забелязали, че има два начина, по които потребителите държат устройството – пейзаж (по ширина) и портрет (по височина). Телефонът, сам по себе си, е способен да разбере по кой начин бива държан и някои вградени приложения реагират коректно, когато върху него се указва натиск от пръст. За жалост, нашата сметачна машина не се справя много добре с това:



Няма нищо лошо да се работи по този начин, стартовото меню на Windows Phone работи само когато телефонът е изправен. Но запомнете, че ако решите да поддържате и двата варианта, ще трябва да прекарате два пъти повече време да конструирате потребителския интерфейс и тогава да доабите код, който да реагира, когато потребителят наклони телефона. Финалната

права (поне за мен) е, когато клиентът каже: „Знаете ли, би било страхотно, когато програмата работи в случай, че телефонът се държи вертикално“, да не реагирате веднага, казвайки „Мога да наоравя това“. Вместо това би трябвало да се каже „Това ще струва допълнително“. Защото ще е така. Освен това, ако са Ви предложили добри пари, за да добавите тази особеност, добре е да се знае най-добрия начин, по който да се изпълни задачата.

Избор между пейзаж и портрет в MainPage.XAML

Страницата може да подсказва на Silverlight посоката, която поддържа. Това се изразява като свойство на PhoneApplicationPage:

```
SupportedOrientations="Portrait" Orientation="Portrait"
```

Настойките, които виждате отгоре са онези, които Visual Studio залага в страницата, когато я създава. Това означава, че тя ще работи в режим Портрет с началните си настройки за Портрет, което си е разбираемо. Можем да променим това, ако искаме:

```
SupportedOrientations="PortraitOrLandscape" Orientation="Portrait"
```



Това е малко подобрене, при което докато програмата е в дестиве ние можем да докоснем екрана и да видим следното:

Сега програмата възпроизвежда екрана, когато посоката се сменя, но това е защото компонентите са позиционирани напълно в решетка. Някои от тях са на погрешно място, а бутните за изчисление и отговор вече не се

побират на екран.

```
<TextBox Height="72" HorizontalAlignment="Left" Margin="8,19,0,0"
Name="firstNumberTextBox" Text="0" VerticalAlignment="Top" Width="460"
```

```

TextAlignment="Center" />
<Button Content="equals" Height="72" HorizontalAlignment="Left"
Margin="158,275,0,0" Name="equalsButton" VerticalAlignment="Top" Width="160"
Click="equalsButton_Click" />

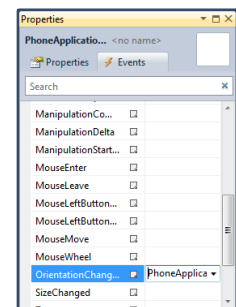
```

Двата компонента имат стойности Margin, използвани за точното им позициониране на екран. За жалост, някои от позициите не се видими на дисплея. Silverlight няма да създаде проблем. Компонентите няма да бъдат изобразени.

Има два начина, по които това може да се поправи. Единият е да се предостави напълно различена обложка за пейзажа и превключване между двата вида изобразяване, когато ориентацията се смени.

Събитие при промяна на посоката

Вече видяхме как Silverlight елементите могат да генерират събития. Прикрепихме събитие към събитието Click, породено от Button, който потребителят натиска, когато желае резултат от изчислението. Също така, можем да прикрепим устройства, които да следят събитията и към други събития. PhoneApplicationPage може да генерира цял набор от събития, включително онези, които се премахват, когато посоката се сменя.



Отгоре виждате как методът може да се добави към събитието. Когато посоката на телефона се смени, този метод ще бъде извикан и прикрепен на дисплея. Само трябва да добавим малко код, за да се случи това:

```

private void PhoneApplicationPage_OrientationChanged( object sender,
OrientationChangedEventArgs e) { if (e.Orientation == PageOrientation.PortraitUp) {
setPortrait(); } else { setLandscape(); } }

```

Този код използва параметъра на метода, за да разбере новата посока и да извика подходящия метод. Методите могат да видоизменят рамките на контролерите, като ги излагат по подходящ начин:

```

private void setLandscape(){ firstNumberTextBox.Margin = new Thickness(8, 19, 0,
0); firstNumberTextBox.Width = 207; secondNumberTextBox.Margin = new Thickness(252,
19, 0, 0); secondNumberTextBox.Width = 207; plusTextBlock.Margin = new
Thickness(221, 35, 0, 0); resultTextBlock.Margin = new Thickness(538, 35, 0, 0); }

```



Това е методът setLandscape. setPortrait метода е много подобен.



Контролерите са позиционирани чрез указване на рамка около тях, която е зададена от Thickness стойността, притежаваща margin и border размер. Всъщност не искаме да чертаем никакви рамки за тези контролери, така че размерът на рамката се задава

като 0. Ако тези методи се добавят към приложението, то ще работи еднакво добре в която и да е позиция:

Съществува малка разлика: когато „правилното“ Windows Phone приложение е завъртяно, често показва хубава анимация при завъртането. Това се прави по желание, но при нужда е необходимо устройство като Expression Blend, което да създаде анимацията.

Проблемът в Demo07 AddingMachine with auto orientation разисква добавянето на машина, която автоматично да управлява промяната на посоката.

Използване на контейнери за изобразяване на обложка

Вече знаем как да накараме приложението да приспособи свойствата на контролерите под напътствието на програма. Въпреки това, все още трябва да измислим стойности, които да накарат обложките да работят. За щастие, Silverlight може да ни помогне и с това; всъщност може почти автоматично да извади целия дисплей. Това може да спести много труд. Ключът в разбирането на механизма е да се види как Silverlight контейнерите биха могли да поддържат други обекти. Има няколко различни типа контейнери в Silverlight. Ще използваме StackPanel.

StackPanel съдържа стек от Silverlight елементи. Изрежда ги по реда, по който са зададени. Това се случва вертикално (надолу по екрана) или хоризонтално (напречно на дисплея). Вместо да предоставим на всеки елемент от екрана с рамка, която да го позиционира, можем да използваме StackPanel за целта:

```
<StackPanel>
  <TextBox InputScope="Digits" Height="72" HorizontalAlignment="Center"
    Name="firstNumberTextBox" VerticalAlignment="Top" Width="460"
    TextAlignment="Center" />
  <TextBlock Height="56" HorizontalAlignment="Center" Name="plusTextBlock" Text="+"
    VerticalAlignment="Top" FontSize="32" Width="25" />
  <TextBox InputScope="Digits" Height="72" HorizontalAlignment="Center"
    Name="secondNumberTextBox" VerticalAlignment="Top" Width="460"
    TextAlignment="Center"/>
  <TextBlock Height="46" HorizontalAlignment="Center" Name="resultTextBlock"
    VerticalAlignment="Top" FontSize="30" Width="160" TextAlignment="Center" />
</StackPanel>
```



Страхотното нещо тук е, че ние не трябва да решаваме къде да бъде всеки елемент от екрана – те просто се натрупват един върху друг. Местоположението на елементите е променено на “Center”, така че StackPanel да дава на обекта определено място от екрана, където да се центрира. Отдолу може да се

види кога зареждаме програмата с обложката:

Всеки обект се изобразява в стек. Един много полезен страничен ефект на този подход е, че StackPanel ще пренареди елементите, ако размерността на екрана се измени. Това означава, че ако ориентацията на телефона се промени, то автоматично се намества:

Забележете, че тъй като обектите са центрирани, сега те ще могат да се изобразят по средата на пейзажния екран.

Можем да добавим StackPanel да изобразява дизайн чрез извличането му от Visual Studio Toolbox на дизайнерската повърхност или чрез въвеждане ръчно на XAML текст. В случай, че желаем да направим хоризонтален StackPanel вътре в някой друг, трябва само да добавим допълнителен атрибут:

```
<StackPanel Orientation="Horizontal">
```

Можем да сложим StackPanel в друг, за да усложним показаното, както ще видим по-късно.

Има и други видове контейнери на разположение. Един от тях, който използва Windows Phone страници е Grid. Той позволява да се създава правоъгълна решетка от елементи, всеки от които съдържа един или повече обекти за изобразяване, позиционирани вътре. Всъщност, Windows Phone програмта показва, че сме използвали един елемент от решетката, който изпълва цялото пространство.

Проблемът, разгледан в Demo08 AddingMachine with StackPanel предлага сметачна машина, която използва StackPanel, който автоматично да се справя с промените на посоката.

4.4. Изобразяване на списъци с данни

Изобразяването на отделни елементи работи много добре, но бихме искали да можем да правим същото и за списъци от данни. Кое то ни води до друг проблем; трябва да имаме неща в този списък.

Създаване на потребителски списък

Ще направим малка програма, управлявана от потребител. Ще бъде много просто, просто въвеждане на клиентското име, адрес и ID номер, но ще я разширим по-късно, дори ще стигнем до там, че да прикачим цяла SQL база данни. Първата ни версия на клиентски клас ще изглежда така:

```
public class Customer
{
    public string Name { get; set; }
    public string Address { get; set; }
    public int ID { get; set; }

    public Customer(string inName, string inAddress, int inID)
    {
        Name = inName;
        Address = inAddress;
        ID = inID;
    }
}
```

```
}
```

Въвели сме даннови полета като войства, а защо те да добра идея ще видим по-късно. Клиентският клас само съдържа прост метод, конструктор, който създава примерен вариант на класа от три входни данни.

Сега се нуждаем от списък с клиенти. Най-добрият начин да се съхранят нещата е да се използва List collection class, предоставен от C#. Можем да създадем Customers клас, който съдържа списъка с името на клиента като низ и списък с клиенти:

```
public class Customers
{
    public string Name { get; set; }
    public List<Customer> CustomerList;
    public Customers(string inName)
    {
        Name = inName;
        CustomerList = new List<Customer>();
    }
}
```

Customers класа съдържа само елементарен метод, който е конструктор. Това задава името на класа и тогава създава празен списък, където да го съхрани. Ако желаем да създадем клиентски списък, за да подредим имейл клиентите си, бихме написали следното:

```
Customers mailCustomers = new Customers("Mail Order Customers");
```

Сега имаме два класа, единият от които съдържа единствен клиент и друг, в който се намира голям набор от такива.

Създаване на примерни данни

Следващото нещо, което искаме е купища клиенти. Можем да се опитаме да ги въведем ръчно и да прекараме дълги хубави часове в създаването на интересни имена, но това би било глупаво. По-добре е да се създаде метод, който да свърши трудната работа вместо нас. Можем да сложим този метод в Customers класа, така че класът да може да ни дава тестови данни:

```
public static Customers MakeTestCustomers()
{
    string [] firstNames = new string [] { "Rob", "Jim", "Joe", "Nigel",
    "Sally", "Tim"} ;
    string[] lastsNames = new string[] { "Smith", "Jones", "Bloggs", "Miles",
    "Wilkinson", "Brown"
    };
    Customers result = new Customers("Test Customers");
    int id = 0;

    foreach (string lastName in lastsNames)
    {
        foreach (string firstname in firstNames)
        {
            //Construct a customer name
            string name = firstname + " " + lastName;
        }
    }
}
```

```

        //Add the new customer to the list
        result.CustomerList.Add(new Customer(name, name + "'s House",
        id));
        // Increase the ID for the next customer
        id++;
    }
}
return result;
}

```

Кодът ще създаде 36 клиенти само чрез комбинирането на първите и последни имена, които са дадени. Ако искаме няколко стотин клиента, смо трябва да добавим повече низове с имена. Всеки клиент има уникален ID номер. Методът `MakeTestCustomers` е направен статичен, така че да не се налага да имаме списък с клиенти преди да създадем пробни данни. Създаденият списъкът с клиенти се нарича “Test Customers”. Можем да го извикаме доста лесно:

```
customerList = Customers.MakeTestCustomers();
```

Когато се прави нещо, което би трябвало да съдържа голям набор от елементи, следващата стъпка е да се създаде бърз начин, по който да се прави това. Ако се притеснявате, че това ще се отрази върху размера на програмата, бихте могли да изпозвате компилация от условия, така че `MakeTestCustomers` метода да бъде единствено включен в класа, ако определен дебъгиращ символ е открит докато се създава програмата.

Надявам се, че това илюстрира много важен принцип. Щом направит нещо, необходим е начин, по който това нещо да се тества, че ще работи добре. Ако продавахме системата за управление на клиенти на собственик на магазин, тя не би била много впечатлена, ако ѝ покажем, че системата действа само с няколко клиента, които сме въвели ръчно. Тя би искала програмата да управлява хиляди входни данни. Не е толкова трудно да се създаде код, който да прави това, както видяхте по-горе, и това прави изпробването на програмата доста лесно и убедително.

Използването на `StackPanel` за изобразяване на списък

Първото нещо, което ще направим с нашия списък с клиенти е да изобразим всичките имена на екрана на телефона. Можем да сожим всяко име в `TextBlock`. Вече сме създавали `TextBlock` елементи преди; те изобразяваха как нашата сметачна машина ще изобрази отговора на потребителя. Сега искаме нещо, което ни предостави тексовите полета автоматично, дори и да не знаем колко ще бъдат те, когато програмата стартира. Това също знаем как да сторим. Може да се използва `StackPanel`, за да се изобрази автоматично огромния набор от елементи. Ето как XAML ще покаже това:

```

<StackPanel HorizontalAlignment="Left"
    Margin="0,0,0,0" Name="customersStackPanel"
    VerticalAlignment="Top"/>

```

Може да се питате къде са всичките `TextBlock` стойности. Отговорът е, че те могат да се създадат в програмата:

```

customers = Customers.MakeTestCustomers();
foreach (Customer c in customers.CustomerList)
{
    TextBlock customerBlock = new TextBlock();
    customerBlock.Text = c.Name;
    customersStackPanel.Children.Add(customerBlock);
}

```

Този код създава редица от пробни клиенти, а след това работи с тях. За всеки от тях се създава нов TextBlock случай, който извиква customerBlock. Тогава се задава text свойството на customerBlock за името на клиента. Най-накрая се добавя customerBlock към наследниците на customersStackPanel. Това илюстрира нещо от изключително значение. Вече видяхме, че елементите могат да се създават елементи, които да бъдат изобразени на страницата чрез XAML. Освен това, също могат да се изобразят елементи, като ги конструираме като обекти на програмата. Програмата от по-горе ще работи добре само с набор от клиенти.



Отгоре виждате как StackPanel изобразява стек с имена на клиенти. Но въпреки това съществува проблем. Има някои клиенти под Nigel Miles, но не можем да ги видим, тъй като са „изпаднали“ от екрана. Височината на StackPanel ще нарастне, за да побере всички елементи, но има вероятност те да не се видят на екран.

Silverlight няма да изведе грешка, ако желаем да изведем нещо, което няма да се събере на екрана, но дисплеят няма да изглежда както трябва. Затова е необходимо да се намери начин, който да позволи на потребителя да скролира надолу по екрана и да види останалия списък. Това може да се осъществи чрез ScrollViewer. Този метод позволява на прозореца да бъде

скролиран от потребителя:

```

<ScrollViewer>
    <StackPanel HorizontalAlignment="Left" Margin="9,6,0,0"
        Name="customersStackPanel" VerticalAlignment="Top"/>
</ScrollViewer>

```



Отгоре виждате версията, която позволява да се скролира.

Проблемът в Demo09 CustomerManager List показва изобразяването на списък с клиенти, както видяхме по-горе. Ако се зареди тази програма, ще се видят полетата за скролиране, които се показват автоматично, и че екранът ще се свива и разширява по най-благоприятния начин. Цялото това държане се предоставя чрез ScrollViewer контролера.

Използване на Listbox за ообразяване на списък

Вече видяхме, че е напълно OK за приложение да създаде Silverlight елементи, когато то стартира. Изобразените елементи могат да бъдат описани вътре в XAML файла, като в този случай те се пускат автоматично, когато програмата стартира; или програмата може да създаде нови примерни

елементи и да ги добави към контролерите на екрана в процес на работа. Току що използвахме тази възможност, за да изобразим списък с клиенти, но програмата ни трябваше да свърши цялата работа. Трябваше да се създадат всичките обекти, които да бъдат изобразени и тогава те да се добавят към StackPanel, където да бъдат показани.

Последният път, когато се сблъскахме с този проблем, открихме, че Silverlight позволи да се използва Data Binding, който беше голямо улеснение. Data Binding кара Silverlight да извърши трудната работа за нас, а ние трябваше само да свържем съдържанието на Silverlight обекта с данните, които искахме да бъдат изобразени и това работеше. До сега сме се научили да свързваме индивидуални елементи към отделни стойности (напр. резултатът от програмата AddingMachine). Сега ще разширим това познание и ще свържем клиентския обект към списък.

Въпреки това, процесът ще е по-сложен отколкото свързването, което направихме последния път. Хубавите новини тук са, че това е много лесно и настина много мощно веднъж щом сме го усвоили. Така че, хайде да го направим

Свързване на данни на единични обекти

В прогмата за сметачна машина използвахме свързваме между данни, за да четем номера от потребителя (съдържанието на TextBox елементите бяха свързани към стойности, които трябваше да бъдат събрани) и също така използвахме свързването на данни, за да покажем резултата от изчислението (AnswerValue беше прикачен към resultTextBlock). Всичко това беше направено в XAML, който описваше изобразяването на екрана.

```
<TextBlock Height="46" HorizontalAlignment="Left" Margin="158,208,0,0"
Name="resultTextBlock" Text="{Binding Path=AnswerValue}" VerticalAlignment="Top"
FontSize="30" Width="160" TextAlignment="Center" />
```

Отгоре виждате свързването, което използвахме при сметачната машина, за да изобразим резултата от изчислението. Тесктът от resultTextBlock изобразява елемента, свързан към AnswerValue свойството в класа Adder. Когато класът изработваше новия отговор, той се изобразяваше автоматично чрез data binding.

Създаване на Data Template

Бихме желали да свържем Customer, за да изобразим елемент, но можем да сторим това директно, тъй като стойността на клиента включва куп различни компоненти. Клиентската стойност съдържа име, адрес и клиентски номер. Това, което искаме е да проектираме шаблон, който задава как клиентът да бъде изобразен. Това ще ни отведе до избора, кои части искаме да покажем на екран (вероятно искаме да пропуснем номера му) и как те да бъдат форматирани. Вероятно ще се получи нещо от сорта:

```
<DataTemplate>
    <StackPanel>
        <TextBlock Text="{Binding Name}"/>
        <TextBlock Text="{Binding Address}"/>
    </StackPanel>
```

</DataTemplate>

Това е данновият шаблон, който е куп от Silverlight, описващ как данните да се изобразят. Казваме на Silverlight, че желаем да изобразим данните във формата на StackPanel, съдържащ два обекта, името и адреса. Като резултат трябва да се изобрази нещо такова с два текстови низа, един върху друг:

Rob Miles
Rob Miles's House

Забележете, че изобразяваме само името и адреса на Customer. Не съществува свързване на данни за Customer ID стойността и по тази причина не се показва.

Употреба на DataTemplate в списък

Използваме даннов шаблон, когато казваме на Silverlight как да покаже характеристиките на обекта. Ще използваме шаблон, за да изобразим всеки един от клиентите в списъка. Това се осъществява чрез поставянето на `DataTemplate` в `ItemTemplate` за `ListBox`, който ще покаже нашите клиенти:

```
<ListBox Name="customerList">
    <ListBox.ItemTemplate>
        <DataTemplate>
            <StackPanel>
                <TextBlock Text="{Binding Name}"/>
                <TextBlock Text="{Binding Address}"/>
            </StackPanel>
        </DataTemplate>
    </ListBox.ItemTemplate>
</ListBox>
```

Добре, време е да се върнем малк назад. Списъчната кутия на Silverlight ListBox изобразява елемент, наречен `customerList`, който ще изобрази списък с нашите клиенти. За да се направи това, е необходимо да се знае как да се покаже екрана на един Customer. ListBox използва `ItemTemplate` елемент, за да проектира дизайна на обекта в списъка, като в случая ще използва `DataTemplate` за да извърши това.

Така че, веднъж след като сме настроили Silverlight, на ListBox трябва само да се укаже колекцията от данни, която да изобрази:

```
customers = Customers.MakeTestCustomers();  
customerList.ItemsSource = customers.CustomerList;
```



Този код създава списъкот пробни клиенти и тогава задава на `ItemsSource` `CustomerList` към списъка с клиенти. `ListBox` ще изобрази всеки обект като елемент на списъка.

Програмата не прави нищо друго освен да задава свойството `ItemsSource` на `ListSource`. Всичко друго се прави автоматично. Списъчната кутия дори има ленти

за превъртане, които са автоматично там, така че ако списъка е прекалено голям за екрана, потребителят да може да превърти нагоре или надолу.

От този пример се вижда колко е лесно да се изобрази набор от данни. Единствено трябва да се проектира шаблона, а след това да се използва свързване между данните, за да могат те да се изобразят.

Добавяне на стил към екрана

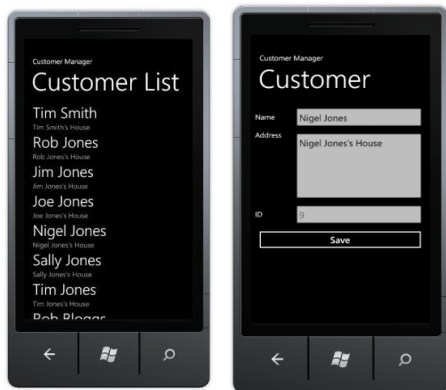
Дисплеят от по-горе работи добре, но не е много интересен. Също така е трудно да се види кое е името на клиента и кое – неговият адрес. Можем да изберем между различни шрифтове и стилове за името и адреса, но биме могли и да се възползваме от някои вградени стилове, предоставени като ресурси за Windows Phone приложенията.

Предимството на тези стилове е, че работят с различни цветови схеми, както и със светли и тъмни фонове. Добавя се информацията за стил към обектите на текстовото поле в DataTemplate:

```
<DataTemplate>
    <StackPanel>
        <TextBlock Text="{Binding Name}"
            Style="{StaticResource PhoneTextExtraLargeStyle}"/>
        <TextBlock Text="{Binding Address}"
            Style="{StaticResource PhoneTextSubtleStyle}"/>
    </StackPanel>
</DataTemplate>
```

ExtraLargeStyle намира приложение за името на клиента, а SubtleStyle – за адреса. Като резултат дисплеят изглежда много по-добре.

Избиране на обекти от ListBox



Сега можем да изобразим списък с обекти много лесно. Следващата стъпка е да направим приложение, което да редактира точно данните. Вече сте виждали подобен тип приложение преди. Когато потребителят избира един от обектите в списъка, трябва да се отвори екран, който позволява редакция на обекта:

потребителят избира Nigel Jones

Екраните отгоре показват как това се изпълнява. Потребителят превърта нагоре-надолу, за да открие клиента, с когото желая да работи, след което го избира. Приложението изобразява екран за редакция за този клиент. Можем да използваме този пример по всяко време, когато имаме списък с неща, от които потребителят може да избира.

Много е лесно да се накара ListBox да предизвика събитие, когато даден обект е избран:

```
<ListBox Name="customerList"

SelectionChanged="customerList_SelectionChanged">
```

Тук customerList се декларира в XAML за главната страница. Устройството, управляващо събитията трябва да се създаде във файла-източник MainPage.xaml.cs.

```
private void customerList_SelectionChanged(object sender, SelectionChangedEventArgs)
{
    // when we get here the user has selected a customer
    Customer selectedCustomer = customerList.SelectedItem as Customer;
    MessageBox.Show(selectedCustomer.Name + " is selected");
}
```

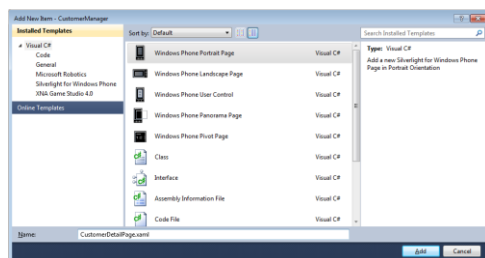
customerList осигурява качество, наречено SelectedItem, което е обратна връзка към избрания обект. Методът трябва да преобразува това във връзка към клиента и тогава програмата ще може да изобрази текстово поле с името му.

Проблемът в Demo10 CustomerManager Binding List разглежда изобразяването на списък за клиенти, както видяхме по-горе. Silverlight за MainPage.xaml съдържа ListDisplay елемент и шаблон, а програмата използва свързване на данни, за да изобрази резултата. Ако се избере обект от списъка, програмата показва MessageBox с избраното име.

4.5. Страници и навигация

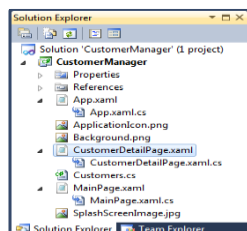
До момента всички прогрмаи, които сме написали работят в рамките на една Silverlight страница. Освен това, много приложения са действащи върху няколко страници. Приложението Customer List, което създадохме ще се нуждае от различна страница, която ще изобразява информация за клиента, с когото работим.

Добавяне на нова страница



Лесно е да се добави нова страница. Ако изберем Project>Add New Item в рамките на Visual Studio, ни се отваря меню с възможности, от които можем да избираме.

В случай, че изберем Windows Phone Portrait Page, се добавя нова страница. Преди това да се случи е разумно страницата да се наименова с нещо по-смислено от



„Страница1“. Името на обекта може да се промени по-късно, ако желаем това. Приложението сега съдържа толкова страници от колкото се нуждаем.

Тук можем да видим, че проекта съдържа CustomerDetailPage. Новите страници се редактират по същия начин като MainPage. Когато страницата се отвори, ние можем да преместим елементи върху Visual Studio ToolBox, да манипулираме техните свойства и дори да редактираме директорията на

XAML, ако желаем.

Навигация между страниците

Silverlight модела за навигация между страниците има повече общо с интернетата отколкото с Windows формите. Ако използвате приложения на Windows Forms, ще сте запознати с методи като “Show” и “ShowDialog” , които се използват, за да може дадена форма да позволява изобразяването на друга.

Silverlight не работи по този начин. Когато искате да се предвижвате из Silverlight приложението, трябва да помислите за навигация между страниците, всяка от които има адрес, изразен като uri (Uniform Resource Indicator). Обектът NavigationService предоставя методи, които изпълняват навигацията за нас:

```
// Navigate to the detail page NavigationService.Navigate(new
Uri("/CustomerDetailPage.xaml", UriKind.RelativeOrAbsolute));
```

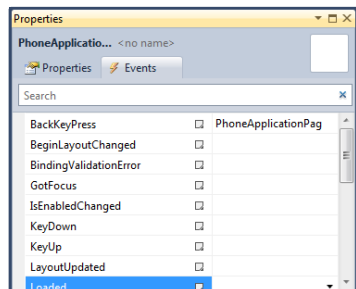
Този код ще накара приложението да отиде на страницата, наречена CustomerDetailPage. Забележете, че на Navigate метода се дава uri стойност, съдържаща името на страницата както и вида Uri, който се използва.

Когато създаваме uri, също трябва да изберем неговия вид, задаваме го да бъде relativeOrAbsolute, макар че този тип е относителен спрямо съществуващия. Правим това, за да осигурим максимална подживност, когато зареждаме ресурси.

Съществува и друго нещо, за което трябва да сме нащрек. Стойността на Uri е зададена като низ в нашата програма. Ако по грешка се въведе низ (напр. CustomerDetailsPagerino.whamlxaml), прогмата ще го изкомпилира както трябва, но когато зарежда ще даде грешка при навигацията.

Използване на бутона Back

Потребителите ще очакват бутон Back от Windows Phone, който да връща към предишната страница. Точно това и прави той. Ако потребителят навигира към CustomerDetailPage, както е показано по-горе, и тогава натисне бутона, ще се върне към предходната страница. В случай, че бутонът Back се напише при главната страница, приложението се затваря.



Понякога просто искаме да прескочим това действие. Може да не искаме потребителят да напуска страницата в случай, че не са запаметили някои данни. Можем да направим това чрез добавянето на устройство за управление на събитията към BackKeyPress събитието:

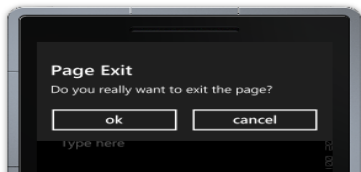
Методът за управление на събитието може да прекъсне държанието, което връща назад при необходимост.

```
private void PhoneApplicationPage_BackKeyPress(object sender,
System.ComponentModel.CancelEventArgs e)
{
    e.Cancel = true;
}
```

Ако методът отгоре е свързан към BackKeyPress събитието, потребителят няма да може да използва бутона Back, за да излезе от страницата.

Понякога може да искаме потребителят да потвърди, че желае да напусне страницата. За целта се използва MessageBox клас:

```
private void PhoneApplicationPage_BackKeyPress(object sender,
System.ComponentModel.CancelEventArgs e)
{
    if (MessageBox.Show("Do you really want to exit the page?", "Page Exit",
        MessageBoxButton.OKCancel)
        != MessageBoxResult.OK)
    {
        e.Cancel = true;
    }
}
```



Кодът изобразява съобщение и пита потребителя, дали наистина иска да напусне страницата. В случай на отказ, бутонът „назад“ се отменя. Екранът изглежда по следния начин:

Тази форма на може да се използва по всяко време, когато се задават кратки въпроси.

Предаване на данни между страниците

Всяка Silverlight страница има отличителни белези и се различава от останалите. Те не споделят данни. Понякога може да искате да споделите данни между страниците. Ако данните са във формата на прост текстов низ, най-лесният начин това да се направи е чрез добавяне на uri, което се предава към страницата. Тази техника се базира на начина, по който уеб страниците изпращат заявки към сървъра.

За пример можем да вземем CustomerManager. Когато потребителят избере клиент от списъка, бихме искали да предадем името и адреса, избрани в страница за редактиране.

```
private void customerList_SelectionChanged(object sender, SelectionChangedEventArgs e)
{
    // Get the selected customer from the list
    Customer selectedCustomer = customerList.SelectedItem as Customer;
    // Build a navigation string with the customer information in it
    NavigationService.Navigate(new Uri("/CustomerDetailPage.xaml?" + "name=" +
        selectedCustomer.Name + "&" + "address=" + selectedCustomer.Address,
        UriKind.Relative));
}
```

Този метод се изпълнява всеки път, когато потребителят избере клиент. Първото нещо е да се извлече избрания клиент. Тогава се създава Uri, което съдържа името и адреса на избрания обект, което след това се използва за навигация към страница с детайли за клиента.

"/CustomerDetailPage.xaml?name=Sally Smith&address=Sally Smith's House"

Този формат на Uri би се изпълни за Сали Смит. Може да се види, че заявката е част от ? знака на Uri. След това на ред идват две наредени двойки „име-стойност“, разделени с „&“. Първият обект е името на заявката, а вторият – адреса.

Сега трябва да разберем как страницата-получател може да ползва тази информация и да я изобрази. За целта е нужно да се имат предвид събитията, които се появяват при навигирането между Silverlight страниците.

Използване на събития за навигация между страниците

Има две наистина важни събития за една Silverlight страница. Едното от тях се пуска, когато се навигира към страницата (OnNavigatedTo), а другото – когато страницата се остави (OnNavigatedFrom). За да се сдобием с достъп до тях, трябва да прескочим тези методи в нашата страница.

При създаването на нова страница класът, който я контролира се извлича от родителски клас, наречен `ParentApplicationPage`:

```
public partial class CustomerDetailPage: PhoneApplicationPage
{
    // CustomerDetailPage methods go here
    // we override the ones that we want to provide
    // our own behaviours for
}
```

Това е декларация на `CustomerDetailPage`. Добавят се нови методи към страницата и по този начин се отменят методите, добавяйки нов начин на държане. Искаме приложението да поеме контрол, когато потребителят се насочва към страницата. Това е мястото, където искаме да заредим информация от Uri и да я изобразим на страницата. Методът `OnNavigatedTo` изглежда така:

```
protected override void OnNavigatedTo(System.Windows.Navigation.NavigationEventArgs e)
{
    string name, address;
    if (NavigationContext.QueryString.TryGetValue("name", out name))
        nameTextBlock.Text = name; if
        (NavigationContext.QueryString.TryGetValue("address", out address))
            addressTextBlock.Text = address;
}
```

Методът използва `QueryString` в `NavigationContext` обекта и се опитва да извлече данни стойности извън низа. Запитва всяка стойност по име и в случай, че стойността е възстановена, я изобразява в текстовото поле. Всеки път потребителят се връща на страницата, към която е прикачен метода и слага най-новото съдържание там. Можем да използваме събитието `OnNavigatedTo` колкото пъти искаме в рамките на предназначенията страница.

Нещо, което трябва да се запомни е, че имената трябва да съвпадат и на двете страници. Компиляторът няма да засече грешки като напр. спелуване на думата “address” като “addresss” в

горния код. Ако сторим това, програмата ще се изпълни правилно, но информацията за адреса няма да излезе правилно.

Кодът може да се използва, за да създаде две приложения за страници, които позволяват на потребителя да избере клиент и види страницата с детайли за този клиент. За жалост, има мъничък проблем с навигацията между страниците, ако направим това. Съществува едно поведение, което ще бъде обръквашо:

1. Избор на обект от списъка.
2. Навигация към страницата с детайли относно клиента.
3. Натискане на бутона "назад", за да се върнем към списъка.
4. Избор на същия обект от списъка.

Ако потребителят направи това, ще забележи, че не се връщат към същия клиент. Трябва да кликнат върху друг клиент и тогава да се върнат към първоначалния, за да го изберат. Причината за това е, че програмата е устойчива на SelectionChanged събитие. Доколкото се касае изборът, той не се изменя, когато потребителят преизбере едно и също нещо. Това може да се поправи чрез изчистване на селекцията независимо кога се навигира към страницата:

```
protected override void OnNavigatedTo(System.Windows.Navigation.NavigationEventArgs)
{
    customerList.SelectedItem = null;
}
```

Сега, ако потребителят избере един и същ предмет, това ще се брои за валидно събитие при избор. Разбира се, настройването на избора на нула всъщност е селектиращо събитие и така, ако програмата нисе опита да използва това качество, че се провали с изключение на нулата. Това лесно ще се поправи; устройството, управляващо събитията трябва да се увери, че е избран елемент от списъка преди да се опита да го използва:

```
private void customerList_SelectionChanged(object sender, SelectionChangedEventArgs)
{
    // Abandon if nothing selected
    if (customerList.SelectedItem == null) return;
    // Rest of method here
}
```

Горната версия на метода се връща, ако няма избран елемент.

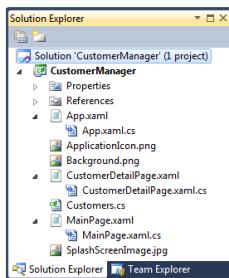
Проблемът, разгледан в Demo11 CustomerManager Shared Object е приложение на Silverlight с две страници. Може да се избере клиент и да се види как той е изобразен в страницата за детайли.

Споделяне на обекти между страниците

Предаването на мсиви между страниците е полезно, но често се налага да се споделят по-големи обекти, съдържащи структурирани данни. Често потребителят ще работи с „нещо“ от някакъв вид – игра или документ и рязличните страници ще предоставят различни гледки на този обект. В нашето приложение Customer Manager наистина ще трябва да прехвърлим Customer примера в страницата, която ще се изобрази, така че тя да работи със съдържанието на клиента. В момента на тази страница са й дадени стойностите на обектите за избрания клиент, подадени като низове.

Това, което реално искаме е обект, който е познат на всички страници и им е достъпен. Изглежда много лесно, защото всичките програми на Windows Phone Silverlight имат нещо такова, вградено в начина, по който работят. Нарича се App.xaml страница.

App.xaml страница



Когато създаваме Silverlight приложение, се появява MainPage.xaml и App.xaml страница.

App.xaml може да се счита като контейнер за телефонни приложения. Той осигурява контекста, където се изобразяват страниците. Дори не се показва на екран, а само представлява началната точка за приложенията, когато се зареждат. Файлът App.xaml.cs съдържа методите, които стартират приложението. Също така притежава методи за управление на „живота“ на приложението, които ще се използват по-късно.

Файлът App.xaml.cs може да се редактира и да се добавят собствен код и член-данни към него. Ако искаме да съхраним общодостъпни данни, които да се ползват от всички страници на приложението, можем да декларираме това тук.

```
public partial class App : Application
{
    // To be used from all pages in the application
    public Customer ActiveCustomer;
}
```

Тук сме добавили стойност ActiveCustomer, която да може да се използва от всички страници. Тя ще държи обратна връзка към клиента, който в момента се обработва от потребителя.

Всяка Silverlight страница може да се сдобие със справка на приложението, което съставлява:

```
protected override void OnNavigatedTo(System.Windows.Navigation.NavigationEventArgs e)
{
    base.OnNavigatedTo(e);
    // Get the parent application that contains the active customer
    App thisApp = Application.Current as App;
    // Set the data context for the display grid to the active
    // customer customerDisplayGrid.DataContext = thisApp.ActiveCustomer;
}
```

Членът `Current` на класа `Application` е връзка, която може да се отнася до обекти от вида `Application`. Когато приложението се стартира, той е настроен да се обърне към примерното приложение, което в момента се използва.

Класът `App.xaml.cs` дефинира клас-наследник `App`, който разширява родителския такъв, `Application`. За да си осигурим достъп до обекти, дефинирани от класа `App`, ние трябва да се обърнем към типа `App` и чак тогава да осъществим връзка към настоящото приложение. Отгоре се вижда как операторът се използва, за да преобразува справката в такава с правилен тип. Това е кодът, който се изпълнява вътре в `customerDetailsPage`, когато се навигира към страницата. Това изважда `ActiveCustomer` от приложението и задава даннов контекст на `customerDisplayGrid`, който да се обърне към избрания клиент. Вече сме използвали свързване на данни, за да позволим директното свързване на всички клиенти към страницата за избор на клиенти; тук го използваме, за да изобразим стойностите на екран.

Дизайн на данните

Тъй като често може да има страници в приложението, които искат да използват списъка с клиенти, от полза би било този списък също да се сложи в класа

`App.xaml.cs`. Пробните данни в този случай се задават, когато приложението стартира. В завършеното приложение това е мястото, където данните се зареждат от мястото им за съхранение.

Проблемът в `Demo11 CustomerManager Shared Object` е приложение от две страници на Silverlight, което използва общи променливи и позволява на потребителя да избере един обект от списъка, като го разглежда в страницата с детайли. Също така се използва и Silverlight Grid контролер, който да оформи дисплея на клиента. Това е много полезно, ако желаете да прикачавате обекти на страницата.

4.6. Употреба на *ViewModel* класове

До момента нашата програма е много добра в изобразяването на клиент, но реално не ни позволява да променяме детайлите му. Потребителят може да променя съдържанието на текстовите полета на страницата, показваща клиентите, но промените няма да се отразят върху банните на програмата.



Това, което искаме да се случва е, когато потребителят обнови полето Адрес, напр. "Nigel Jones's New House", това да бъде автоматично записано при избрания клиент. Това може да се направи чрез `DataBinding`; по този начин извлякохме числата от текстовите полета при сметачната машина, която създадохме по-рано.

За начало указваме на Silverlight, че текстовото свързване в `TextBox` е двупосочно.

Това означава, че промени в данните ще бъдат изобразени в класа на екран и че тези промени в `TextBox` ще обновят свойството на класа, към който са прикачени.

```
<TextBox Name="nameTextBox" Text="{Binding Name,Mode=TwoWay}"/>
```

След това можем да добавим клас `Customer`, който да управлява събитията при свързване. Този код ще се увери, че когато името на клиента се променя и текстът на екрана също ще се промени, и ако съдържанието на `TextBox` се промени, то свойството име на клиента ще се измени автоматично. Направихме това в `Adding Machine` с помощта на класа `Adder.cs` и това изглеждаше, че работи добре.

Но, първо, най-добре би било е да се спрем и да обмислим онова, което правим.

Дизайн с клас `ViewModel`

Бихме могли да добавим поведение за свързване на данни към класа `Customer` и от програмистка гледна точка това би работило добре. Но съществуват някои причини, поради които това не би било добра идея от дизайнерска гледна точка.

Разделяне на целите

Има едно правило в софтуерния дизайн то е, че даден клас трябва да изпълнява едно единствено нещо. Работата на класа `Customer` се състои в това да поддържа клиентските данни, при което да не участва в процеса на редакция. Когато запамятаваме или прехвърляме клиентски данни, не би трябвало да се тревожим от методите и поведенията, засягащи процеса на редактиране. Ако прехвърлим наш клиент върху друго устройство за съхранение (ще погледнем това по-късно) всяко поведение, касаещо свързването на свойствата би било загуба на пространство, тъй като няма да се използва в този контекст.

Осигуряване на валидация и преобразуване на данни

При съвършени условия, нещо трябва да се уверява, че онова, което е въвежда във формата, отговаря на нашите бизнес правила, т.е. „името трябва да съдържа само главни букви и интервали“. Тези правила могат да се сложат в `Customer` класа, но би било много по-лесно да можем директно да ги свържем към процеса за входни данни, тъй като това ще е момента, когато ще намерят приложение. Също така, би било полезно да се осъществи преход на някои входни стойности, напр. време и дата, от формат, по който са въведени и представени на екрана, използвани в мястото за съхранение.

Предоставяне на поведение `Undo`

Това е въпрос, касаещ потребителския интерфейс, но е също от изключително значение. Ако видите екрана за редакция, ще забележите, че там има голям `Save` бутон. Идеята е, че потребителят ще го натисне, когато е приключил с редактирането. Това е и мястото, където промените ще се отразят на данните. Освен това, съществува и огромен `Cancel` бутон, който ще затвори изцяло екрана и ще прекъсне всякакви промени.

Ако използваме директно свързване на данни, то би било пробелм, тъй като обектът на данните вече ще е променен, за да отразява онова, което въвежда потребителят. Това означава, че ще се наложи да се направи копие на данните при входа на формата и тогава да ги преместим в обект. Тъй като причината, поради която използваме свързване на данни е да предотврати писането на

такъв вид код, би било жалко да започнем да го пишем наново, само защото на потребителя може да му хрумне нещо ново.

Създаване на клас ViewModel

Тези проблеми могат да се адресират чрез класа ViewModel, който има задължението да представи част от данните, които искаме да представим. Той се грижи за представянето на данните и тяхното свързване към компонентите от формата; също така може да изпълнява всякаква проверка на данни при необходимост. Класът ViewModel се създава, за да представи дейностите, които ще се изпълнят в определен потребителски интерфейс. Той може да се възприеме като „лепило“, което свързва потребителския интерфейс с настоящите обекти на данните. Съдържа свойствата, с които желаем да работим в интерфейса. Отдолу е написан такъв, наречен CustomerView.

```
public class CustomerView : INotifyPropertyChanged
{
    private string name;
    public string Name
    {
        get
        {
            return name;
        }
        set {
            name = value;
            if (PropertyChanged != null)
            {
                PropertyChanged(this, new
                PropertyChangedEventArgs("name"));
            }
        }
    }

    // Address information here
    private int id;

    public int ID
    {
        get
        {
            return id;
        }
    }
    public event PropertyChangedEventHandler PropertyChanged;
    public void Load(Customer cust)
    {
        Name = cust.Name;
        Address = cust.Address;
        id = cust.ID;
    }
}
```



```

        public void Save(Customer cust)
        {
            cust.Name = Name;
            cust.Address = Address;
        }
    }

```

Местене на данни в и извън Viewmodel

Класът CustomerView също съдържа методи Load и Save, които задават как програмата да зададе компонентите на преглеждания в момента клас към данните, с които се работи. Като пример, обмислете какво се случва, когато започне да се редактира клиент:

```

// Viewmodel for the details page
CustomerView view = new CustomerView();

protected override void OnNavigatedTo(System.Windows.Navigation.NavigationEventArgs e)
{
    // Get the parent application that contains the active customer
    App thisApp = Application.Current as App;
    // Load the active customer into the viewmodel
    view.Load(thisApp.ActiveCustomer);
    // Set the data context for the display to the viewmodel
    customerDisplayGrid.DataContext = view;
}

```

Когато програмата се прехвърли върху страницата с детайли, методът OnNavigatedTo се извиква посредством Silverlight, за да основе страницата. Този метод намира активния в момента клиент (този, който е избран) и тогава го зарежда в CustomerView, който се използва на тази страница.

Когато потребителят натисне Save бутона, програмата трябва да вземе съдържанието от настоящия модел и да го върне към клиента:

```

private void saveButton_Click(object sender, RoutedEventArgs e)
{
    // Get the parent application that contains the active customer
    App thisApp = Application.Current as App;

    // Copy the data from the viewmodel into the active customer
    view.Save(thisApp.ActiveCustomer);

    // Go back to the previous page
    NavigationService.GoBack();
}

```

ViewModel и тестване

Класът ViewModel изглежда добре, но може да се чудите защо го ползваме. Изглежда, че можем да получим подобно поведение само чрез поставяне на членовете на класа Customer директно в TextBox елементите на страницата и да ги препрочете, ако потребителят натисне бутона Save.

Докато това работи, се пропуска един важен аспект на програмното развитие, а именно тестването. Ако друг редактор на клиентски данни взаимодейства пряко с елементите, е много

трудно автоматично да изпробваме дали работи или не. Тестовият код би въвел нещата в текстовите полета и проверил съдържанието на другите полета, за да се увери, че приложението работи както трябва. Това се усложнява още повече, ако потребителският интерфейс няма валидация за изпълнение и транслиране на данни.

Въпреки това, при положение, че класът `ViewModel` извършва тази работа, би било много по-лесно да се тества. Тестова обстановка може просто да свърже същите елементи във `ViewModel` като потребителския интерфейс и да имплементира текстови съобщения. Няма нужда от тестване чрез въвеждане на текст на екрана, защото тестовата обстановка може да симулира свързването на данни.

Навигация между страниците, използваща метода `GoBack`

Методът `GoBack` се осигурява посредством `NavigationService`, който отвежда потребителя към предходната страница. Това е по-ползотворно от използването на `URI` за навигация, защото не пресъздава страницата. Ако страницата се навигира чрез метода `Navigate`, осигурен от класа `NavigationService`, това ще създаде нова страница всеки път, когато е посетен. Това може да забави приложението и да доведе до създаването и унищожаването на много `Silverlight` елементи, които ще трябва да бъдат възстановени от `Garbage Collector`.

Още повече, че потребителят ще бъде леко раздразнен, ако след като е скролирал надолу списъка, за да избере обект, с който иска да работи, бива отведен на върха на списъка след редкация.

Както и да е, използването на метода `GoBack`, за да ни отведе към оригиналната страница ни създава проблеми. Трудността се явява изобразяването на оригиналната страница няма да отрази промените, които потребителят е направил с данните. Дадената последователност трябва да се вземе под внимание:

1. Потребителят избира Роб Майлс.
2. Потребителят променя името на „Великият Роб Майлс“ (и доста правилно).
3. Потребителят натиска бутона `Save`, за да извърши промените.
4. До връщането към екрана с избор името все още е „Роб Майлс“.
5. Объркващи резултати.

Проблемът е, че `ListBox` е в неведение, че съдържанието на клиентския архив се е променил. Когато списъкът с клиенти се изобрази, той направи копие на данните върху екрана и не знае, че те трябва да се обновят. Реалните данни са се актуализирали, но това, което се вижда, не отразява промените. За да разрешим този проблем, трябва да се задълбаем повече в `Silverlight` и да открием как да използваме `ObservableCollection`.

Видими колекции

Видяхме, че най-добрият начин да се свърже даннов обект към Silverlight страница е да се сложи ViewModel помежду им. ViewModel може да бъде пригоден за определения изглед, който се изисква, а данновият обект може да се съсредоточи само върху съхранението на данни.

Това също е в сила, когато се има предвид събиране на данни. В момента свързваме списъка с клиенти директно към ListBox. Това е добре само за преглед на съдържанието от списъка, но, както вече видяхме, ако някой от елементите на списъка се промени, ние нямаме имаме метод за известие, който да уведоми Silverlight, че онова, което е показано на екран трябва да се обнови. Онова, от което се нуждаем е друг клас ViewModel, който да представлява списъка с клиенти и да осигурява начин, по който ListBox мже да бъде уведомен за промени за съдържащите го елементи.

За целта е създаден класът ObservableCollection. Той може да бъде използван, за да поддържа колекция от обекти и да осигурява поддръжка при уведомления, така че ако събржанието на колекцията се промени, то ListBox да промени събитията. Можем да създадем ObservableCollection много лесно:

```
ObservableCollection<Customer> observableCustomers;

private void PhoneApplicationPage_Loaded(object sender, RoutedEventArgs e)
{
    // Get the parent application that contains the customers list
    App thisApp = Application.Current as App;
    // Make an observable collection of these
    observableCustomers = new ObservableCollection<Customer>(
        thisApp.ActiveCustomerList.CustomerList);
    // Set the list to display the observable collection
    customerList.ItemsSource = observableCustomers;
}
```

Това е кодът, който се зарежда, когато главната страницана стартира Customer Manager. От клиентския списък тя създава нов такъв, наречен observableCustomers. Тогава задава източника на customerList към новосъздадения списък. Сега, ако програмата прави промени на списъка, дисплеят ще отрази това.

За жалост, програмата все още не отразява желаните промени. Ако потребителят обнови името на клиента, това няма да се отрази на текста върху екрана. Това е така, тъй като ObservableCollection реагира на промените в съдържанието на списъка, а не на онези, засягащи данните в елемент от този списък.

Всъщност, това, което се получава е по-лошо от това дисплеят да не се обнови правилно. Онова, което ще се случи е, че при необходимост да се възстанови елемент ListBox ще се върне към първоначалните данни и ще го изобрази. Това означава, че ако потребителят превърти нагоре-надолу в списъка за известно време, има някакъв шанс вярните данни да се появят случайно, тъй като ListBox само ще създаде дисплейни обекти, които са видими и може да пренебрегне онези, които са премахнати от екрана. Това може да доведе до някои много объркващи ефекти, където програмата ще изглежда, че трябва да наваксва.

За да се появи екрана с обновяването, трябва да направим нещо на съдържанието на списъка, за да накараме процеса да се случи. Един начин за това е да се махне нещо от списъка и след това да се върне обратно. Промените на съдържанието на видимата колекция генерират събития, които се избират от ListBox, изобразяващ списъка.

```
protected override void OnNavigatedTo(System.Windows.Navigation.NavigationEventArgs e) {  
    // Get the parent application that contains the customer list  
    App thisApp = Application.Current as App;  
  
    if (thisApp.ActiveCustomer != null)  
    {  
        // Find the customer in the list  
        int pos = observableCustomers.IndexOf( thisApp.ActiveCustomer);  
        // Remove it  
        observableCustomers.RemoveAt(pos);  
        // Put it back again - to force a change  
        observableCustomers.Insert(pos, thisApp.ActiveCustomer);  
    }  
}
```

Това е кодът, който се зарежда, когато потребителят се връща обратно към началната страница. Ако им активен клиент (напр. такъв, избран за редакция), методът премахва клиента от списъка, след което го връща на същото място. Това принуждава отстъплението на обекта и сега потребителят ще види, че промените се отразяват бързо. Това е от изключително значение в случай, че имаме списък, съдържащ няколко хиляди клиенти.

Съхранение на данни

Когато използвахме класът ViewModel, за да редактираме клиент, имаме методи Load и Store, които отвеждат клиента към класа CustomerView, след което възстановяват подновените стойности. При ObservableCollection трябваше да сторим нещо подобно. За щастие, съществуват няколко начина, по които ObservableCollection връща данните, които изобразява.

```
thisApp.ActiveCustomerList.CustomerList = observableCustomers.ToList<Customer>();
```

Методът ToList method извежда списъка на клиентите и го връща.

Видими колекции и ефикасност

Може да си мислите, че добавянето на ObservableCollection забавя програмата и използва повече памет. Но всъщност не е така. Колекцията е друг набор от връзки към същите обекти на клиента, които се съхраняват. Това означава, че единствената допълнителна памет, която е необходима е за видимата колекция, която не е голяма.

Ако програмата се нуждае от това само да види съдържанието на списъка, тогава не е нужно да се използва ObservableCollection. Това се случва, само когато съдържанието на списъка трябва да се промени поради обновяване на данните в списъка, към който се добавя.

Проблемът в Demo12 Complete CustomerManager е приложение на Silverlight от две страници, което позволява на потребителя да избере и редактира обекти в списъка с клиенти. Промените на обекта се обновяват в гледката на списъка, а потребителят се връща към първоначалната позиция на края на всяко редактиране.

Какво научихме

1. Програмите могат да манипулират свойствата на Silverlight елементите, за да изобразят информация на потребителя. Това включва позицията и цвета на обектите на дисплея.
2. Някои свойства на елементите най-добре се редактират директно със XAML. XAML информацията за даден елемент се структурира предвид свойствата, които могат да бъдат поместени в тях самите. Textbox има набор от свойства, които задават началните настройки на клавиатурата за въвеждането на данни. Те може да предизвика показването на клавиатура с числа вместо с букви.
3. Windos Phone може да изобрази полета за съобщения на потребителя, които да показват съобщения с множество редове; както да се потвърдят или отменят действията на потребителя.
4. Добавки като изображения могат да бъдат добавени към Windos Phone приложенията като съдържание или ресурс. Обект на съдържанието е копиран в директорията на приложението като отделен файл и може да се използва от там в програмата. Обектите на съдържанието не забавят зареждането на асемблерите, но самите те могат да се заредят по-бавно, когато приложението стартира.
5. Silverlight елементите могат да генерират събития в отговор на потребителските действия. Едно такова действие е събитието TextChanged, произведено от TextBox.
6. Silverlight осигурява поддръжка на свързването на данни, където свойствата на обекта в програмата могат да бъдат свързани с тези на изображения Silverlight елемент. Свързването може да бъде едностранно, където показваният елемент е използван, за да отпечата стойността на програмния обект или двустранни, където промените на обекта от страницата дават като резултат обновяване на свойството за свързване в класа.
7. Възможно е да се осъществи връзка между колекция от обекти и ListBox Silverlight елемент, за да се покаже списък от елементи. Данновият шаблон се използва, за да изрази как индивидуалните даннови стойности на всеки елемент ще бъдат изобразени.
8. Silverlight приложенията могат да бъдат съставени от няколко страници. Навигацията между тях се осъществява от клас-помощник, на който се дава uri на страницата, към която да се отправи. Простите обекти от текста могат да бъдат предадени през страниците, като се сложат в запитващ низ, прикрепен към uri.

9. Страниците могат да получават събития, когато се навигира към/от тях. Събитието `OnNavigatedForm` осигурява опция за отмяна, така че потребителят да може попитан, дали иска да потвърди навигацията.
10. По-големи даннови обекти могат да бъдат споделени между страниците в приложението чрез класа `App`, ойто е част от `Windows Phone` приложението. Всяка `Silverlight` страница в приложението може да получи справка за обекта на приложението, от който е част.
11. Програмистите могат да създадат `ViewModel` класове, зададени към данните, които биват редактирани и свързани към `Silverlight` контролерите на страницата. Те действат като „лепило“ между актуалните данни в програмата и индивидуалните презентационни и редакционни нужди на потребителския интерфейс, който е в процес на развитие.
12. Механизмът `ObservableCollection` позволява промените на съдържанието на колекцията да се отразяват върху гледката на списъка, който ги изобразява.

Глава 5. Isolated Storage на Windows Phone

Всяко приложение или игра има своите собствени специфични нужди за съхранение на данни. Съхранението може да бъде толкова просто като име - чифт стойност (например Класация = 100). От друга страна, той може да се съхранява цялата база данни на продукта, заедно с всички свои клиенти. Или може да бъде съхраняване на текстови документи или снимки, направени с камерата на телефона. В този раздел ние ще проучи начините, по които C # приложения на устройството могат да се свързват и използват услуги за данни, предоставени от телефона.

5.1 Съхраняване на данни на Windows Phone

А правилното прилагане ще трябва да се съхраняват данните , които той може да използва всеки път, когато тя работи . А `Silverlight` приложения ще трябва да държи настройки и информация на потребителя, по които работи и XNA игра ще искате да съхранявате информация за хода на играч в играта заедно с висок рейтинг и настройки играч . Програми за `Windows Phone` може да се използва "изолиран съхранение" , за да държат този вид информация . Съхранението се нарича "изолиран" , тъй като не е възможно за една заявка за достъп до данните, съхранявани от друг . Това е различно от компютър `Windows` , където всяка програма може да получите достъп до всеки файл на системата.Изолирана зоната за съхранение може да побере много големи обеми от данни , до лимита на разположение в самия телефон съхранение . А `Windows Phone` ще имат най-малко 8G от вградени в съхранение, което се поделя между медиите (съхранява музика , снимки и видео клипове) и всички приложения на устройството.При подаване на заявление е извадена от телефона всички изолирани съхранение , предвидени за него се заличава. Когато заявление бъде модернизирана чрез `Marketplace` (т.е. ние пускаме нова версия на нашата програма или игра) изолираната съхранение запазва оригиналното съдържание . Ако данните в нея трябва да бъдат актуализирани, за новата версия нашето заявление трябва да направите това .

Използване на Isolated Storage File System

Можете да използвате изолиран съхранение по същия начин, както използвате файлова система. Единствената разлика е начинът, по който програма прави връзка със самата съхранение. Въпреки това, след като една програма има връзка с файловата система тя може да го използвате, както би направил всеки друг, създавайки потоци за прехвърляне на данни във файлове и дори при използването на папки, за да се създаде структуриран filestore. Една програма може да съхранява много големи обеми от данни по този начин. Като пример можем да създадем една проста Silverlight приложение, което дава на потребителя прост бележник, където те съхраняват прости съобщения. Тази версия на програмата използва един-единствен файл, но може лесно да бъде разширен, за да се осигури система за съхранение бележка.



Потребителят може да напишете в дневници, които се записват, когато бутона Save е натиснат. Ако бутонът Load е натиснат на записки са натоварени от склада.

```
private void saveButton_Click(object sender, RoutedEventArgs e)
{
    saveText("jot.txt", jotTextBox.Text);
}
```

Това е кодът за бутона Save. Това отнема текста от текстовото поле и той преминава в един разговор на метода saveText заедно с името на файла, за да го съхраните.

```

private void saveText(string filename, string text)
{
    using (IsolatedStorageFile isf =
        IsolatedStorageFile.
            GetUserStoreForApplication())
    {
        using (IsolatedStorageFileStream rawStream =
            isf.CreateFile(filename))
        {
            StreamWriter writer = new StreamWriter(rawStream);
            writer.Write(text);
            writer.Close();
        }
    }
}

```

Методът saveText създава поток, свързан с определен файл в изолирана съхранение и след това пише текста до него. Ако сте използвали потоци в C # програми вече ще бъде удоволствие да откриете, че те работят по абсолютно същия начин. В този случай методът създава StreamWriter и след това просто изпраща текста до този поток.

Бутонът за четене използва метод loadText да направя обратното на тази операция.

```

private void loadButton_Click(object sender, RoutedEventArgs e)
{
    string text;

    if ( loadText("jot.txt", out text ) )
    {
        jotTextBox.Text = text;
    }
    else
    {
        jotTextBox.Text = "Type your jottings here....";
    }
}

```

Методът на loadText опитва да отворите файл с името е дадено. След това той се опитва да чете низ от този файл. Ако някоя част от операцията по четене не успее метод loadText връща false и jotTextBox е настроен на изходно съобщение.

Ако файлът може да се прочете правилно методът loadText връща true и определя параметрите на изхода на съдържанието на файла. След това тя се използва, за да настроите текста в jotTextBox.

Имайте предвид, че ние сме с помощта на out параметър, за да се даде възможност на метода loadText да върне низ, който бе прочетен, както и дали тя работи или не.


```

private bool loadText(string filename, out string result)
{
    result = "";
    using (IsolatedStorageFile isf =
        IsolatedStorageFile.GetUserStoreForApplication())
    {
        if (isf.FileExists(filename))
        {
            try
            {
                using (IsolatedStorageFileStream rawStream =
                    isf.OpenFile(filename,
                        System.IO.FileMode.Open))
                {
                    StreamReader reader =
                        new StreamReader(rawStream);
                    result = reader.ReadToEnd();
                    reader.Close();
                }
            }
            catch
            {
                return false;
            }
        }
        else
        {
            return false;
        }
    }
    return true;
}

```

Методът на loadText ще връщане фалшиви ако входния файл не може да бъде намерен или процеса на четене се провали. Имайте предвид, че тя използва метода на ReadToEnd да се чете от файла, така че програмата може да прочетете няколко реда дневници.

Разтворът в Demo 1 Jotpad съдържа Windows Phone Silverlight бележник тампон, който съхранява текст във файл и го зарежда обратно. Ако спрете програмата и да я стартирате отново от списъка с приложения в емулятора, ще забележите, че запазените данни се възстановява, когато натиснете Load. Имайте предвид, че можете да разширите спести и товарните методи за съхраняване на още повече данни просто чрез добавяне на допълнително пишат и четат отчети и други параметри. Алтернативно бихте могли да направите на товара и спаси методи работят по даден предмет, а не на брой отделни елементи.

Използване на Isolated Storage съхранение настройки

Доста често единственото нещо, което една програма трябва да магазин е проста стойност настройки. В този случай тя изглежда много усилия, за да създавате файлове и потоци, само за да се съхранява една проста стойност. За да се направи лесно да настройките съхранявате Phone програма Windows може да използва Isolated магазина настройки за съхранение. Това работи като

речник, където можете да съхранявате произволен брой двойки име / стойност в изолирана зона за съхранение.

Въведение в света на речници

Речници са много полезен механизъм за събиране, предоставени като част от NET рамката.. А речник е направен с помощта на два вида. Един от тях е типа на ключа, а другият е от типа на елемента се съхраняват. Например, ние може да създаде клас, наречен човек, който държи на данни за, изненада изненада, едно лице:

```
class Person
{
    public string Name;
    public string Address;
    public string Phone;
}
```

Този Person class над съдържа само три полета, но тя можеше да побере много повече. Бихме искали да бъде в състояние да се създаде система, в която можем да дадем името на лице, а след това системата ще намерите стойността на лицето с това име. А Dictionary ще направи това за нас много лесно:

```
Dictionary<string, Person> Personnel =
    new Dictionary<string, Person>();
```

Когато ние създаваме нов тип речник ние го дам типа на ключа (в този случай низ защото ние влизаме в името на лицето) и типа на елемента, се съхраняват (в този случай, човек). Сега можем да добавим неща към речника:

```
Person p1 = new Person { Name = "Rob", Address = "His House",
    Phone = "1234"
};
```

```
Personnel.Add(p1.Name, p1);
```

Този код прави нов човек инстанция и след това го съхранява в речника на персонал. Имайте предвид, че кодът е на името от лицето, което е било създадено и използва това като ключ. Програмата вече може да използвате низ като показалец за намиране на елементи в речника:

```
Person findPerson = Personnel["Rob"];
```

Позоваването findPerson е настроен да се позове на инстанция Person с името Роб. Речникът върши цялата тежка работа за намиране на този конкретен запис. Това го прави много лесен за съхранение на голям брой елементи и да ги намерите по-късно.

Речникът ще откаже да съхраняват елемент с помощта на същия клавиш като една вече е налице. С други думи, ако се опитам да се добави втори човек с името "Rob" операция Add ще хвърли изключение. А програма също така ще получи изключение, ако го пита речника за запис, който не съществува:

```
Person findPerson = Personnel["Jim"];
```

За щастие има и механизъм за определянето на това дали или не даден бутон се намира в речника:

```
if (Personnel.ContainsKey("Jim"))
{
    // If we get here the dictionary contains Jim
}
```

Речници са много полезни за съхраняване на двойки име-стойност. Те премахват необходимостта да се напише код, за да търсите из колекции, за да намерите неща. Ние можем да използваме множество речници също, например бихме могли да добавим втори речник на нашата система за персонала, който ще ни позволи да се намери човек от неговия телефонен номер.

Речници и Isolated Storage

Класът IsolatedStorageSettings ни предоставя система за съхранение речник основава за настройки, които можем да използваме, за да съхранява стойностите на настройките. Речникът за настройки съхранява колекция от предмети, с помощта на низ като ключ.

Спасяването на текст в настройките на изолирани съхранение

Бихме могли да я превърне в нашия бележник да използвате класа на настройки, както следва:

```
private void saveText(string filename, string text)
{
    IsolatedStorageSettings isolatedStore =
        IsolatedStorageSettings.ApplicationSettings;
    isolatedStore.Remove(filename);
    isolatedStore.Add(filename, text);
    isolatedStore.Save();
}
```

Тази версия на метода saveText използва името на файла като ключ. Тя премахва съществуващ ключ с даденото име на файла и след това добавя предоставения текст като нов запис. Методът на Remove може да бъде призован да премахнете елемент от речника. Той е даден ключът към елемента, които трябва да бъдат отстранени. Ако ключът не присъства в речника метода Remove връща фалшиви, но ние можем да пренебрегнем това. След като сме направили нашите промени до магазина, че трябва да се обади на метода Save да се задържат тези промени.

Зареждане на текст от настройките на изолирани съхранение

Методът на loadText ще донесе стойност от речника на настройки:

```
private bool loadText(string filename, out string result)
{
    IsolatedStorageSettings isolatedStore =
        IsolatedStorageSettings.ApplicationSettings;
    result = "";
    try
    {
        result = (string)isolatedStore[filename];
    }
    catch
    {
        return false;
    }
    return true;
}
```

Методът на loadText извлича желания елемент от съхранението на настройки. Той е направен малко по-сложно от факта, че за разлика от класа Dictionary, класа на IsolatedStorageSettings не предвижда метод ContainsKey, че можем да използваме, за да се види дали даден елемент е налице. Горният метод просто хваща изключение, че се хвърля, когато даден елемент не може да се намери и връща фалшиви, за да покаже, че тази новина не е налице. Имайте предвид, че трябва да хвърли низ стойността, която се зарежда от речника на настройки. Това е така, защото речникът държи предмети (така че да можем да сложим всякакъв вид в нея).

Сега имаме два начина да се задържат на данни на Windows Phone устройство. Ние може да съхранява големи количества данни в filestore, че ние можем да създадем структура и как ние обичаме. Като алтернатива може да се съхранява отделни позиции от данните по име в речник настройки. Механизмът за изолиран съхранение може да се използва от Silverlight и XNA програми игри, така.

Разтворът в Demo 02 Settings JotPad съдържа Windows Phone Silverlight бележник тампон, който съхранява текст в речника на IsolatedSettings.

The Isolated Storage Explorer

Когато ние се развиват заявление, че е полезно да бъде в състояние да видите файловете, които са били съхранявани там. В Windows Phone Developer Toolkit съдържа инструмент, който ви позволява да направите точно това. Това е програма, която се инсталира с Phone разработчик инструментариума Windows. На моята система аз мога да го намеря по пътя по-долу:

```
C:\Program Files (x86)\Microsoft SDKs\Windows Phone\v7.1\Tools\IsolatedStorageExplorerTool
```

The Isolated Storage Explorer е дадена GUID Product (глобално уникален идентификатор) на заявлението. Това коя област на Isolated Storage да прочетете разказва. Ние можем да намерим GUID на продукта в WMAAppManifest.xml файла за приложението:

```
<App xmlns="" ProductID="{3363ac33-4f45-4b21-b932-fa2084b6deb0}"  
Title="JotPad" RuntimeType="Silverlight" Version="1.0.0.0"  
Genre="apps.normal" Author="JotPad author"  
Description="Sample description" Publisher="JotPad">
```

За да прочетете съдържанието на изолиран магазин ние издаваме командата ISETool с подходящи параметри.

```
ISETool ts xd 3363ac33-4f45-4b21-b932-fa2084b6deb0 c:\isoStore
```

Тази команда приема моментна снимка (tc) на емулятора (xd) изолиран съхранението за прилагането Jotpad горе и го поставя в папка C: \ isoStore.

Има много други неща, които можете да направите с изследовател. Пълното описание на командата е както следва:

```
Command Parameters: <ts|rs|dir[:device-folder]> <xd|de> <Product GUID> [<desktop-path>]
```

Korato: ts - (take snapshot) за да изтеглите съдържанието на изолиран магазин от устройство на работния плот.

rs - (restore snapshot) за да качите съдържанието на изолиран магазин от десктоп към устройство.

dir- (list directory) за списък на съдържанието на папката на устройството или корена ако не е споменато

xd - Emulator.

de - Device. P

Product GUID - (product ID) разположен в WMAAppManifest.xml файла на проекта.

desktop-path - десктоп път за изтегляне и качване.

Какво научихме

1. A Windows Phone осигурява система от изолирани съхранение, с което всяка заявка по телефона е в състояние да съхранява собствените си данни върху устройството.
2. Програмите могат да създават папки и файлове в тяхната изолирана зона за съхранение и използване на стандартните входно / изходни съоръжения, основани на потока, за да общуват с тях.

3. Алтернативен метод за съхранение на данни е да се възползват от механизъм, базиран речник за съхраняване на двойки име-стойност, като например настройки стойности (Background colour = blue).
4. The Isolated Storage Explorer инструмент позволява на програмистите да видят съдържанието на изолирания съхранение за заявление в процес на разработка

Глава 6. Използване на бази данни на Windows Phone

6.1 Преглед на База данни за съхранение

Базата данни е колекция от , ами, данни , която се организира и управлява от компютърна програма, която често е и доста объркващо , наречен база данни.

Друг компютърна програма може да задам въпроса на базата данни и базата данни ще се върна с резултатите . Нали не мислиш, на база данни, като част от програмата си като такъв, той е компонент в разтвор, който се занимава със съхраняването на данни.

Бихме могли да откриете как работи с база данни , като един бърз поглед на един много прост пример . Помислете как ще се създаде база данни за провеждане информация за интернет магазин. В момента има голям брой клиенти , които ще правят поръчки за определени продукти .

Ако наемете дизайнер база данни, те ще ни зареди с огромна сума пари и след това те ще дойдат с някои проекти за таблиците в базата данни , която ще се проведе на информацията в нашата система. Всяка една от масите ще имат набор от колони, които притежават едно специфично свойство на дадена позиция , и един ред по масата ще опише един-единствен елемент. Това е дизайн за клиентската маса :

Customer ID	Name	Address	Bank Details
123456	Rob	18 Pussycat Mews	Nut East Bank
654322	Jim	10 Motor Drive	Big Fall Bank
111111	Ethel	4 Funny Address	Strange bank

Това е таблица **Клиентите**, че нашата система за продажби ще използвате. Той съдържа същата информация, която се съхранява в мениджъра на клиенти, с добавянето на един стринг, който дава банкови данни за този клиент. Всеки ред на таблицата, ще разполагат с информация за един клиент. В таблицата може да бъде много по-голям от този, то може също така да провеждат имейл адреса на клиента, си рожден ден и така нататък.

Product ID	Product Name	Supplier	Price
1001	Windows Phone 7	Microsoft	200
1002	Cheese grater	Cheese Industries	2
1003	Boat hook	John's Dockyard	20

Това е таблицата с **Продукти**. Всеки ред от тази таблица описва един-единствен продукт, че магазинът е на склад. Тази идея може да бъде удължен, така че по-скоро от името на доставчика на системата използва ID доставчик, който идентифицира дадено ред в таблицата с доставчици.

Order ID	Customer ID	Product ID	Quantity	Order Date	Status
1	123456	1001	1	21/10/2010	Shipped
2	111111	1002	5	10/10/2010	Shipped
3	654322	1003	4	1/09/2010	On order

Това е таблицата за **Поръчки**. Всеки ред от таблицата описва конкретна цел, която е била поставена. Тя дава ID Продукт на продукт, който е купен, и идентификационен номер на клиент на клиента, че го е купил.

Чрез комбиниране на таблици можете да работите, че Роб е купил телефон Windows, Етел е купил пет сирене рендета и четирите лодки куките за Джим са по поръчка.

Това е малко като детективска работа или решаване на пъзели. Ако погледнете в таблицата с клиентите можете да откриете, че идентификацията на клиентите за Rob е 123456. След това можете да погледнете през масата за цел и да намерят този клиент 123456 поръча нещо с ID на 1001. След това можете да погледнете през масата на продукта и да намерите този продукт 1001 е Windows Phone (който всъщност не е толкова изненадващо).

Бази данни и заявки

Караш на база данни, като го задават въпроси или запитвания. Бихме могли да се изгради заявка, че ще намерите всички поръчки, които Rob е поставил. Системата на базата данни ще се търси чрез таблицата за поръчки и да се върнете всички редове, които имат идентификационен номер на клиент на 123 456. След това може да използвате тази таблица, за да разберете точно какви продукти са били закупени от търсене чрез масата на продукти за всяка от идентификациите продукт в поръчките, които са били намерени. Това ще се върне още един куп резултати, които се идентифицират всички неща, които са закупили. Ако аз искам да попитам на базата данни, за да ми даде всички поръчки от Rob I биха могли да създадат заявка като тази:

Това ще се върне на "мини-маса", която току-що, съдържаща редове с идентификацията на клиентите на "123456", т.е. всички поръчки, направени от Роб. Командите Аз съм с по-горе са на

език, наречена SQL или Structured Query Language. Това е специално измислена за задаване на въпроси на бази данни.

```
SELECT * FROM Orders WHERE CustomerID = "123456"
```

Компании като Amazon правят това през цялото време. Това е начина, по който управлява техните огромни запаси и огромен брой клиенти. Тя е също и как те създават свои "Amazon има препоръки за вас" част от сайта, но ние ще ви позволи този слайд за сега.

Свързване към база данни

За съжаление обектно ориентирани програми не работят по отношение на таблици, редове и заявки. Те работят по отношение на обекти, които съдържат свойства и методи на данни. Когато една програма се свързва с база данни имаме нужда от начин на управление на този преход от таблици за обекти.

Бази данни и класове

Ние вече видяхме, че можем да представим данните в класове. Класът на клиентите в приложението Мениджър Клиент ние работим върху държи информация за всеки клиент:

```
public class Customer
{
    public int CustomerID { get; set; }
    public string Name { get; set; }
    public string Address { get; set; }
    public string BankDetails { get; set; }

    public Customer(int inID, string inName, string inBank,
                    string inAddress,)
    {
        Name = inName;
        Address = inAddress;
        BankDetails = inBank;
        ID = inID;
    }
}
```

За да задържите голям брой клиенти, ние създадохме един клас, който притежава списък с тях.

```
List<Customer> Customers = new List<Customer>();
```

Всеки път, когато добавите нов клиент го добавите към списъка с клиентите си и да намерят клиенти използвам цикъла foreach да работят чрез клиентите и намерете едно искам. Като пример, да намерите всички поръчки, направени от клиентите мога да направя нещо като това:


```

public List<Order> FindCustomerOrders(int CustomerID)
{
    List<Order> result = new List<Order>();

    foreach ( Order order in Orders )
    {
        if (order.CustomerID == CustomerID)
        {
            result.Add(order);
        }
    }
    return result;
}

```

Методът претърсва всички поръчки и изгражда нов списък, съдържащ само тези, които имат необходимата идентификация на клиентите. Това е малко повече работа, отколкото заявката за SQL, но тя има същия ефект.

Използването LINQ за да се свържете с Данни на обекти

Хората , като използване на бази данни, защото е по-лесно да създадете заявка , отколкото се напише код , за да извършите търсене. Въпреки това, те също така с помощта на предмети , защото те са един чудесен начин да се структурира програми. За да използвате бази данни и обектно-ориентирани програми заедно програмистът обикновено ще трябва да пише много " лепило" код, който прехвърля данни от SQL таблици в обекти и след това отново , когато те се съхраняват . Това е голям проблем в големи строежи, при които те могат да имат много таблици с данни и класове въз основа на информация в таблиците.

Language Integrated Query, или LINQ , е резултат от усилията да се премахне необходимостта от всичко това лепило. Тя се нарича "Language Integrated ", защото те всъщност трябваше да се промени дизайна на езика C # , за да бъде в състояние да направи работата функция.

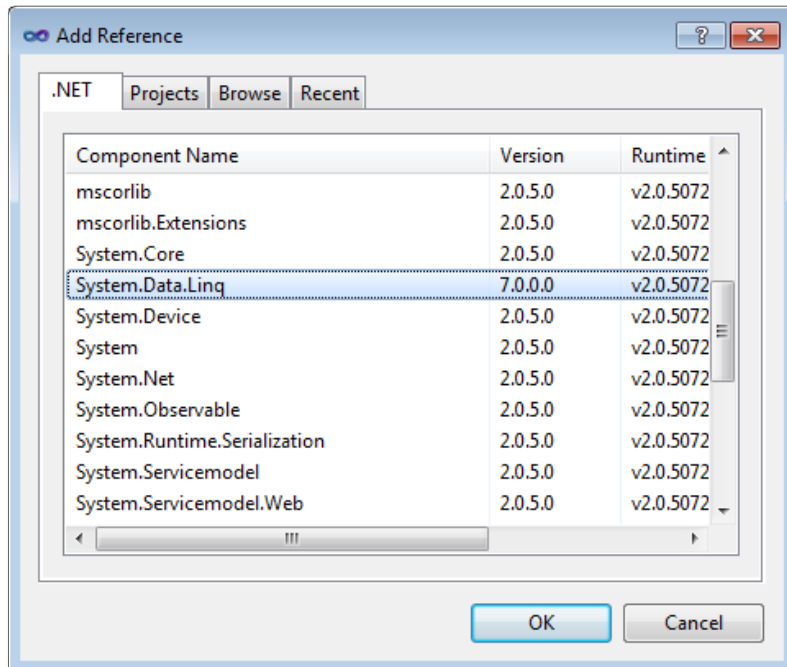
Windows Phone подкрепя създаването на бази данни за нашите програми, за да се използват и всички взаимодействия на базата данни се извършва с помощта на LINQ . Така че сега ние ще видим как можем да вземем нашия клас дизайн и да го използваме, за да се създаде база данни . Ние ще използваме SQL за съхранение на клиентите в нашата Мениджър Клиент . След това ние ще добави поддръжка за поръчките за продукти , които те пускат . Това ще позволи на потребителите ни държат голям брой клиенти , поръчки и продукти в база данни по телефона.

Следващото нещо, което ще направим, е да започнем да проектираме таблиците в базата данни, но преди да можем да направим това, което трябва да направите LINQ работи с този проект.

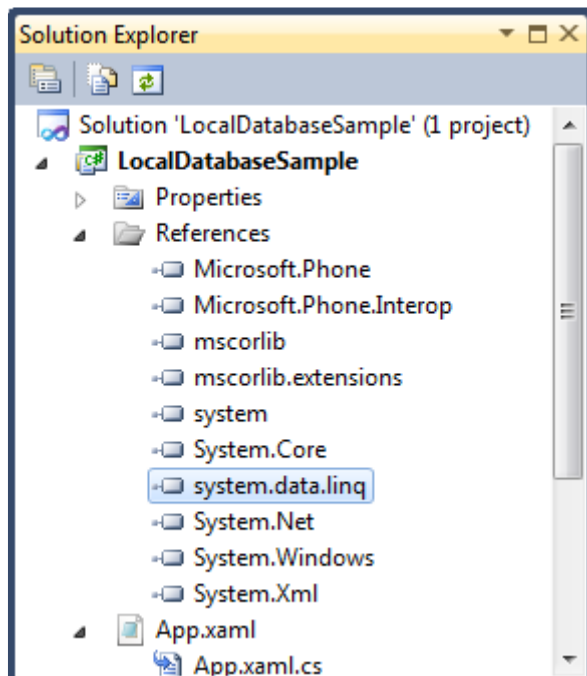
Библиотеките LINQ

Преди една програма може да използвате кода LINQ ние трябва да се добави позоваване на библиотеките System.Data.Linq към проекта. Тъй като не всички програми се нуждаят от подкрепа на база данни, тази справка не е нормално ляво, когато се създава нов проект. Ние можем да

отворите диалоговия прозорец Add Reference, като изберете Add Reference от менюто Project във Visual Studio. Това ни дава списък на библиотеки от които можем да изберете тази, която искате.



Когато се натисне OK се добавя препратка към списъка на ресурси, които програмата изисква.



Следващото нещо, което трябва да направите е да добавите малко `<using>`, използвайки директиви, които да направят по-лесно за достъп до класовете в тази библиотека.

```
using System.Linq;
using System.Data.Linq;
using System.Data.Linq.Mapping;
using System.ComponentModel;
using System.Collections.ObjectModel;
```

Сега можем да използваме всички класове от тези библиотеки, без да се налага да използвате напълно квалифицирана формата на всяко име.

Създаване на LINQ Таблица

Следваща искаме да създадем някои трапезни проекти, които нашата база данни ще управлява. Ще има три маси в нашата база данни:

- Клиенти
- Поръчки
- Продукти

Можем да започнем от обмисля как ние ще се съхранява информация за даден клиент. От първоначалната таблица, която сме създали клас Customer който описва данните на клиента трябва да съдържат. Сега искаме да използваме този клас да се създаде дизайн за таблица в база данни, която ще се проведе на всички наши клиенти. Ние можем да направим това, като първоначалното ни класа на клиента и да я промените така, че LINQ да използвате дизайна на клас за създаване на таблица в базата данни от него. Това е нашият клас Customer дизайн.

```
public class Customer
{
    public string Name {get; set;}
    public string Address { get; set; }
    string BankDetails { get; set; }
    public int CustomerID { get; set; }
}
```

Този клас има три струни и цяло число, които са всички членове на класа. Те са реализирани като свойства, с GET и определени поведения. В началото на нашия клас Customer за LINQ ще изглежда доста сходни:

```
[Table]
public class Customer : INotifyPropertyChanged,
                       INotifyPropertyChanging
{
    // Customer table design goes here
}
```

[Table] елемент е атрибут . Атрибутите се използват за класове знаме с информация, която може да се качват от програми, които гледат на метаданните в компилиран код. Метаданни , както аз съм сигурен, че всички са наясно , е " данни за данните " . В този случай данните се добавят е атрибут , и данните, които тя е около е класа на клиентите .

Когато се съставя C # клас компилаторът добавя много метаданни към изхода , който се произвежда , включително точната версия на кода , подробностите за всички методи , които съдържа, и така нататък. Други програми могат да четат този метаданни , заедно с информацията за класа . Това как ildasm произвела продукцията видяхме в глава 3 , когато ние инспектира съдържанието на файловете , произведени от Visual Studio.

Кодът по-горе добавя [Таблица] атрибут към класа на клиенти, които LINQ тълкува в смисъл, " Този клас може да се използва като основа на таблица с данни " . Налице е всъщност нищо особено вътре в [Таблица] самия атрибут , той просто служи като маркер.

Кодът по-горе също така се посочва, че клас Customer реализира интерфейсите INotifyPropertyChanged и INotifyPropertyChanging . А клас изпълнява интерфейс , когато тя съдържа всички методи , които са описани в интерфейса. Двете интерфейси съдържат методи, които ще бъдат използвани да се каже LINQ когато съдържанието на данните в класа се променя .

Видяхме тази ситуация и преди, когато данните , свързани с определен Silverlight дисплей елемент , необходими за комуникацията промени , така че дисплеят да актуализира автоматично , когато данните се променя. Тази ситуация е много подобна , с изключение на промяна на данните ще предизвика актуализации в базата данни .

Всеки интерфейс съдържа едно събитие делегат . Това е делегат за INotifyPropertyChanged

```
public event PropertyChangedEventHandler PropertyChanged;
```

Инфраструктурата LINQ ще се свърже със събитието PropertyChanged, така че може да се каже, когато стойността на имота се е променило. Налице е също така един делегат трябва да се използва, за да каже LINQ, когато стойността се променя. Това събитие е уволнен, преди да бъде езекутиран на промяната:

```
public event PropertyChangingEventHandler PropertyChanging;
```

Нашата работа е да се уверите, че класът на Customer изстрелва тези събития, когато измененията в данните. С други думи, ако един програмист прави нещо като това:

```
activeCustomer.Name = "Trevor";
```

- Кодът, който се актуализира на имота Наименование Трябва също да уведомите LINQ, че данните се е променила. Това означава, че имотът Наименование може да изглежда така:

```

private string nameValue;

public string Name
{
    get
    {
        return nameValue;
    }
    set
    {
        if (PropertyChanging != null)
        {
            PropertyChanging(this,
                new PropertyChangingEventArgs("Name"));
        }
        nameValue = value;
        if (PropertyChanged != null)
        {
            PropertyChanged(this,
                new PropertyChangedEventArgs("Name"));
        }
    }
}

```

Поведението на Get за имота просто връща стойността на елемента от данни, която държи на името .

Тестовите за Set поведение , за да видите , ако нещо е приложен към манипулаторите на събития за събитията и призовава един преди Имотът се променили, а другият след това. Делегатски призовава са снабдени с два параметъра . Първият е този , който е препратка към активния в момента инстанция Customer (на един , чието име се променя) . Вторият аргумент е едно събитие, което съдържа името на имуществото, което се променя . LINQ може да ги използва за да се получи това, което се е променило и извършва подходяща актуализация на базата данни. Както видяхме , когато се използва Silverlight , ако името на имота е посочен неправилно (да кажем " име" или " пауме ") актуализацията няма да работи коректно .

Промяната генерира " преди" и "след" събития , така че LINQ може да управлява данните, се променя по-ефективно. Ако това изглежда като много допълнителна работа , не забравяйте, че в резултат на нашите усилия ще бъде, че когато ние актуализираме обекти от нашата база данни, ние не трябва да се притеснявате за съхранение на последиците от нашите промени. Всичко това ще се случи автоматично.

Ако нашите обекти съдържат много полета с данни , че има смисъл да се опрости управлението на променената събитие от написването някои методи , които правят управлението на уведомление за нас :

```

private void NotifyPropertyChanged(string propertyName)
{
    if (PropertyChanged != null)
    {
        PropertyChanged(this,
            new PropertyChangedEventArgs(propertyName));
    }
}

private void NotifyPropertyChanging(string propertyName)
{
    if (PropertyChanging != null)
    {
        PropertyChanging(this,
            new PropertyChangingEventArgs(propertyName));
    }
}

```

След това можем да опрости имота име, за да това поведение:

```

private string nameValue;

public string Name
{
    get
    {
        return nameValue;
    }
    set
    {
        NotifyPropertyChanging("Name");
        nameValue = value;
        NotifyPropertyChanged("Name");
    }
}

```

Ако това NotifyPropertyChanged неща изглежда доста изморителна, не забравяйте, защо го правим. Ние го правим, така че да можем да пишем код, като например:

```

activeCustomer.Name = "Fred Blogs";

```

- И са вписването в базата данни за активния в момента клиентът автоматично обновяване. Това е наистина полезно. Ако направим използване на данни Двупосочен задължителен резултатът ще бъде, че след като сме се свързали нашите обекти на данни за елементите на дисплея ще се актуализират автоматично, което е много полезно, наистина.

Има едно последно нещо, което трябва да добавите към имота Име, за да може тя да се използва от LINQ. Трябва да се отбележи имота с [Колоната] атрибут, така че LINQ знае, че тя трябва да се използва в базата данни:

```
[Column]
public string Name
{
    // Name property behaviour goes here.
}
```

Адрес и банкови данни елементите могат да се добавят към класа на [Customer](#) по абсолютно същия начин. Всеки от тях ще очертае върху друга колона в базата данни.

Създаване на Primary Key

Елементът на клиент е малко по-различен обаче. В нашия оригинален Мениджър Клиент е целочислена стойност, която идентифицира всеки клиент. Ако клиентът се оженва и променя името си ID на клиента е там, за да сме сигурни, че все още може да ги намерите в заявлението. По отношение на бази данни ние използваме ID на клиентите като първичен ключ за тази таблица. Първичният ключ се използва за еднозначно идентифициране на определени клиенти в тази таблица. Ние може да има двама клиенти с името "John Smith", но ние никога няма да имаме двама клиенти със същия идентификатор. The ID действа точно по същия начин, както и номера на социалната сигурност или номер на банкова сметка.

Когато ние създаваме колоната ID ние трябва да кажем на LINQ, че това е първичен ключ за таблицата. Можем също така да попитам LINQ да се уверите, че всички стойности в тази колона са уникални, а ние дори може да питам, че стойностите на този елемент са автоматично генерирани. Ние правим това чрез добавяне на информация към колоната атрибут за CustomerID свойства:

```
[Column(IsPrimaryKey = true, IsDbGenerated = true)]
public int CustomerID { get; set; }
```

Когато добавите нов запис клиент на базата данни не е нужно да се притеснявате за идентификацията, тъй като той ще бъде генериран за нас. Ние можем да разбира видите стойността на ID, така че ние можем да кажем следващия "Джон Смит" какво им е уникален идентификатор.

Имайте предвид, че ние никога не е позволено да променят ID за клиент. Това е един уникален ключ, който ще бъде създаден автоматично от базата данни за всеки клиент на свой ред. По отношение на бази данни това свойство се нарича първичен ключ за запис на клиента. Първичния ключ на клиента ще бъде използвана в нашата база данни за изпълнение на връзка между масата на клиентите и таблицата Orders. Всеки ред на базата данни за поръчки ще съдържа стойността на ID клиент, който идентифицира клиента, който поставя този ред. Когато LINQ прочитания метаданните за този имот, който ще използва тези настройки, за да реши какъв вид на колона, за да създадете база данни.

Създаване на Context LINQ Data

Сега, когато имаме дизайн, за ред в таблицата, можем да създадем самата таблица. В предишното ни приложение мениджър на клиентите ние използвахме Списък за провеждане на клиентите. В LINQ ние създаваме DataContext която ще управлява връзката с базата данни ние сме на път да се създаде. Това ще съдържа всички таблици в заявлението, които в момента просто означава, клиентите:

```
public class SalesDB : DataContext
{
    public Table<Customer> Customers;

    public Customers(string connection) : base(connection)
    {
    }
}
```

Класът SalesDB разширява класа DataContext , която се предоставя от LINQ като основа на дизайна на базата данни . В конструктора на този клас просто извиква конструктора на родителския клас . Низ връзка дава място на базата данни ние ще използваме . Един от големите неща за струни връзка с база данни е, че те може да се използва за свързване към бази данни по мрежа, така че една програма може да работи с база данни на далечна сървър. Ние няма да направим това , ние ще се свърже базата данни във файл проведе в изолирано съхранение на телефона.

DataContext служи за връзка с база данни. Можете да мислите за него като за малко като поток, който свързва програма във файл. Той излага методи, които можем да използваме , за да управлявате съдържанието на базата данни .

Низ на свързване описва как програмата ще се свърже с базата данни. В някои приложения тази връзка може да бъде сървър за база данни на далечен машина . В този случай исканията на базата данни се изпращат на отдалечената система и програмата работи с отговорите . В случай на базата данни на Windows Phone това е път към файл проведе в Isolated Storage .

Създаване Sample Data

Когато ние създадохме нашата оригинална програма Мениджър Клиент сме създали метод за създаване на примерни данни . Ние ще направим точно едно и също нещо с нашата версия база данни. Това ще се пренесат на базата данни с известни данни, които след това ние можем да работим с . Имайте предвид, че методът се нарича MakeTestDB , защото ние ще се добави допълнителна настройка на тест за другите маси , когато ги добавите по-късно . Можем да започнем с точно същата проба информация:


```

public static void MakeTestDB(string connection)
{
    string[] firstNames = new string[] { "Rob", "Jim", "Joe",
                                         "Nigel", "Sally", "Tim" };
    string[] lastsNames = new string[] { "Smith", "Jones",
                                         "Bloggs", "Miles", "Wilkinson", "Brown" };

    SalesDB newDB = new SalesDB(connection);

    if (newDB.DatabaseExists())
    {
        newDB.DeleteDatabase();
    }

    newDB.CreateDatabase();

    foreach (string lastName in lastsNames)
    {
        foreach (string firstname in firstNames)
        {
            //Construct some customer details
            string name = firstname + " " + lastName;
            string address = name + "'s address";
            string bank = name + "'s bank";
            Customer newCustomer = new Customer();
            newCustomer.Name = name;
            newCustomer.Address = address;
            newCustomer.BankDetails = bank;
            newDB.CustomerTable.InsertOnSubmit(newCustomer);
        }
    }

    newDB.SubmitChanges();
}

```

Първото нещо, методът прави е да направи връзка с базата данни:

```
Customers newDB = new Customers(connection);
```

Променливата newDB сега представлява една връзка към базата данни с даден низ за връзка. Методът може да се използва тази връзка за издаване на команди за управление на базата данни.

Следващото нещо, че методът прави, е да проверите дали вече съществува в базата данни. Ако това е така, базата данни ще бъде изтрита.

```

if (newDB.DatabaseExists())
{
    newDB.DeleteDatabase();
}

```

Важен принцип на изпитване е, че тест винаги трябва да започне точно от едно и също място всеки път. Ако вече има база данни във файл в изолирано съхранение той ще трябва да бъдат изчистени, преди да се добавят данните от изпитването.

```
newDB.CreateDatabase();
```

Следващото нещо, методът прави, е да създадете нова база данни, която съдържа набор от празни маси.

След като вече имаме нова наша база данни можем да се създаде набор от ценности на клиентите за изпитване и се добавят на всеки един на масата за клиенти.

```
foreach (string lastName in lastNames)
{
    foreach (string firstName in firstNames)
    {
        //Construct some customer details
        string name = firstName + " " + lastName;
        string address = name + "'s address";
        string bank = name + "'s bank";
        Customer newCustomer = new Customer();
        newCustomer.Name = name;
        newCustomer.Address = address;
        newCustomer.BankDetails = bank;
        newDB.CustomerTable.InsertOnSubmit(newCustomer);
    }
}
```

Тези два вложени цикъла работят през един и същ набор от първите и последните имена, които използвахме за последен път. Основната разлика от предишния генератора тест данни е, че този път завършен клиент се добавят към базата данни:

```
newDB.CustomerTable.InsertOnSubmit(newCustomer);
```

Методът на InsertOnSubmit е как ние да добавяте нови записи в таблица. Ние използваме CustomerTable имуществото на новата база данни. Ако базата данни, съдържаща други таблици бихме могли да се добавят точки към тях по същия начин. Имайте предвид, че това е typesafe, с други думи, ако ние се опитаме да добавите различен вид на CustomerTable програмата няма да компилирате.

Окончателният, и може би най-важното, част от метода е призивът, който всъщност се ангажира промените в базата данни:

```
newDB.SubmitChanges();
```

Това е пряко аналогичен Затвори метод при използване на файлови потоци. Това е точката, в която промените, които са поискали да бъдат извършени до базата данни. До метод SubmitChanges се нарича системата може да запази някои от промените в паметта, за да ускори нещата. Това прави добро чувство, ако една и съща позиция на данни се актуализира многократно. Вместо да променя файла на базата данни всеки път, системата ще използва копие в паметта и работи с това. Само когато този метод се нарича ще копията памет бъдат написани обратно към файла на базата данни.

Ако не представи промените там е един добър шанс, че това ще напусне базата данни в една каша. Кое би било лошо.

Сега имаме база данни, която можем да използваме в нашата молба. Можем също така да се създаде база данни за изпитване със стойности, можем да разгледаме. Контекстът на база данни, която ние ще използваме, ще живее в програмния файл App.xaml.cs заедно с позоваване на активния в момента клиента.

```
public SalesDB ActiveDB;  
public Customer ActiveCustomer;
```

Когато програмата започва да работи на променлива ActiveDB е настроен на базата данни, ние използваме.

```
ActiveDB = new SalesDB("Data Source=isostore:/Sample.sdf");
```

Имайте предвид, че пътят до него има определен формат и идентифицира даден файл в изолираната съхранение. Ние можем да направим използването на няколко бази данни в нашата програма, ако желаем, всеки от тях ще се проведе в друг файл. Тази база данни, файл формат е, че на стандартен SQL база данни. А програма за база данни PC базирани можеше да чете плочите от този файл и може да се управлява от редактор на SQL база данни. Обратното също е вярно, заявлението за Windows Phone може да се възползва от една база данни, която е изготвена на друг компютър и зареден в заявлението.

Обвързването на ListBox към резултата на LINQ Query

Следваща искаме да видим как можем да използваме тази база данни в нашата молба. В проста програма Мениджър Клиент ние използвахме Списък, за да побере всички клиенти. Открихме, че елемент на дисплея ListBox да вземете списък и ще го покаже. Сега искаме да се вземат данни от базата данни и да покаже това.

Ние получаваме информация от базата данни чрез издаване запитвания. По-рано видяхме как може да се използва SELECT команда, за да донесе информация. Това, което сега ще направя, е изграждане на заявки LINQ, за да получите всички клиенти от базата данни.

```
var customers = from Customer customer  
                in thisApp.ActiveDB.CustomerTable  
                select customer;
```

Тази линия на C # създава променлива наречена клиенти. Променливата на клиентите е от тип VAR. Ако не съм виждал този тип, преди да може да изглежда малко объркващо. Какво VAR означава в този контекст е "Връща типа на тази променлива от израза, който се поставя в нея". Това не означава, че C # се отказва от силен пишете, ако някога се опитате да използвате променливите клиентите по неправилен начин нашата програма ще все още не успяват да се съберат.

Останалата част от отчета казва LINQ да получите всички елементи на доставката от държавата CustomerTable от контекста на базата данни ActiveDB. Това се връща като списък от клиенти, които могат да се държат като ObservableCollection. Което е точно това, което ние трябва да дадем на падащ списък, за да покажете на клиентите:

```
customerList.ItemsSource = customers;
```

В този момент ние имаме напълно работеща база данни на клиентите. Единственото нещо, което трябва да добавите към програмата е код, за да представя промените в базата данни, когато потребителят напусне програмата. Най-добрият начин да се организира това е да се сложи на поканата в метода OnNavigatedFrom в MainPage.xaml.cs:

```
protected override void OnNavigatedFrom(  
    System.Windows.Navigation.NavigationEventArgs e)  
{  
    App thisApp = Application.Current as App;  
  
    thisApp.ActiveDB.SubmitChanges();  
}
```

Когато потребителят се движи далеч от тази страница, методът ще намерите текущо активната база данни и да представи всички спорни промени в него.

Разтворът в Demo 01 SalesManagement съдържа заявление за Windows Phone Silverlight, който реализира напълно работеща мениджър на клиентите. Тя не се задържат данните в изолирано съхранение, но тъй като съхранението на данни се създава пресен всеки път, вие няма да видите това. След като стартирате програмата, веднъж, за да се създаде можете да коментирате изявлението в App.xaml.cs че създава нова база данни и да видим тази работа за себе си.

База данни и Данни Обвързващата магия

Мисля, че това всъщност е доста вълшебно . Добавили сме само една малка част от код на нашата молба и как можем да се съхраняват и възстановен за нас автоматично елементите на данните . Промените , които правим в потребителския интерфейс , автоматично се запазили в базата данни , без никакви усилия от нас. Той е на стойност просто става чрез точно това, което се случва тук , така че да можем напълно да разберем как работи. Нека да разгледаме следната последователност:

1. Потребители промени " Роб Майлс " на " Роб Майлс прекрасните " в текст полето провеждане полето Име на CustomerDetailsPage .
2. Защото това TextBox е обвързана до имота Име в CustomerView обвързана към тази страница сега актуализира , че имот в CustomerView към новото име текста.

3. Потребителят натисне бутона Save на страницата CustomerView .
4. Методът saveButton_Click в CustomerDetailsPage работи и призовава метода Save в CustomerView инстанция зад тази страница.
5. Методът на Save copia Свойства на страница от класа на оглед в регистъра на активен клиент .
6. Промени в собствеността име в активните клиентски пожарите имота променени събития в LINQ информирание на базата данни , че стойностите са се променили.
7. LINQ знамена информацията за клиентите, които са се променили, а също така повдига промяна събитие в ObservableCollection която държи нашия списък на клиентите и е свързан с падащ списък на Главна страница.
8. Програмата се връща към екрана на ListBox на Главна страница и на дисплея на името "Rob Miles" е актуализиран, за да "Роб Майлс прекрасните".
9. Когато потребителят излиза от прилагането на събития огньове OnNavigatedFrom и призовава SubmitChanges в базата данни, съхраняване на промените обратно в изолирано съхранение.

Ако не вярвате, че това работи, то се опитам да разбера. Можете да използвате по-горе модел навсякъде, където искате да запаметите автоматично и зареди данни с много малко усилия.

Добавяне на филтри

Чрез леко промяна на формата на заявката LINQ да донесе на данните от базата данни можем да изберете записи, които съответстват на определени критерии:

```
var customers = from Customer customer
                 in thisApp.ActiveDB.CustomerTable
                 where customer.Name.StartsWith("S")
                 select customer;
```

Тази заявка избира само клиентите, чиито имена започват с буквата S.



Това ще бъде в резултат на такава заявка. Това го прави много лесен за извличане на конкретни елементи от базата данни. Бихме могли лесно да добавим поле за търсене на нашата молба, ако искаме да търсим за клиенти с определено наименование.

6.2 Създаване на връзки между данните с LINQ

Сега сме в състояние да съхранява голям брой обекти на клиентите и да се промени и да ги актуализира с помощта на данни задължителен.

Въпреки това, за да направим нашата система за управление на продажбите на работа ние ще трябва да се създадат други таблици и да ги свърже заедно. За да обновите нашите спомени това е дизайн за клиентската маса:

Customer ID	Name	Address	Bank Details
123456	Rob	18 Pussycat Mews	Nut East Bank
654322	Jim	10 Motor Drive	Big Fall Bank
111111	Ethel	4 Funny Address	Strange bank

Ние създадохме един клас , който съдържа необходимите редове и да използвате LINQ да се задържат тези данни в една база данни, файл в изолирана съхранение. Масата на продуктите е много подобен на масата на клиенти, в това, че просто има списък на обекти с определени свойства .

Product ID	Product Name	Supplier	Price
1001	Windows Phone 7	Microsoft	200
1002	Cheese grater	Cheese Industries	2
1003	Boat hook	John's Dockyard	20

Класът на продукти е действително много подобна на Клиента едно , то просто съдържа набор от показатели за данни . Таблицата на Ред е по-сложен един . То всъщност съдържа препратки към реда в другите две класи.

Order ID	Customer ID	Product ID	Quantity	Order Date	Status
1	123456	1001	1	21/10/2010	Shipped
2	111111	1002	5	10/10/2010	Shipped
3	654322	1003	4	1/09/2010	On order

Всеки ред от таблицата за Поръчка описва конкретна цел , която е била поставена . Тя дава ID Продукт на продукт, който е купен , и идентификационен номер на клиент на клиента, че го е купил . В C # това ще бъде лесно да създадете :

```
public class Order
{
    public DateTime OrderDate;
    public int Quantity;
    public Customer OrderCustomer;
    public Product OrderProduct;
}
```

Позоваванията на клиента и ред ще се свързват със заповед на клиента, че е направил поръчката, и продукта, който се нареди .

Въпреки това, когато ние сме с помощта на база данни, ние не разполагат с никакви препратки като такива. Вместо да има препратка към един клиент и продукт , вместо таблицата Поръчка държи ID на елементите, които са свързани с всяка поръчка . Ние след това да използваме стойността в колоната ID Customer да намерим на клиента, който е направил поръчката . Например , ние знаем, че за да се постави една от Роб Майлс , защото той съдържа идентификационен номер на клиент на 123 456 , и така нататък.

В добрите стари дни преди компютри това е как работят нещата . Една компания ще има картотека , пълна с данни за клиенти , друг , пълен с данни за продукта и трета пълна с поръчки. За да разберете подробности за определен ред чиновник ще трябва да се запознаете с информацията, клиент и продукт от информация, написана върху детайлите на поръчката. В релационна база данни, предвидена начин за компютри да управляват информация, която съдържа връзки като тези , това, което ние искаме да направим сега , е да вземе информацията за връзка на базата

данни и да го използвате за създаване на референции на обекти, които я правят лесно да се манипулира в обектно-ориентирано програма.

LINQ Асоциации

В LINQ връзка между една друга таблица и се нарича асоцииране . Това се осъществява в класовете на базата данни от стойността EntityRef .

Обвързването на заповед на клиента, че той е пуснал

От нашите системни изисквания класа Поръчка трябва да държи препратка към клиента, който е направил поръчката . Това се осъществява като връзка между две таблици . The EntityRef е специален клас LINQ можем да използваме, за да свържете две таблици заедно , с помощта на база данни за предоставяне на основната памет

```
[Table]
public class Order : INotifyPropertyChanged,
                    INotifyPropertyChanging
{
    ...

    private EntityRef<Customer> orderCustomer;

    [Association(IsForeignKey = true, Storage = "orderCustomer")]
    public Customer OrderCustomer
    {
        get
        {
            return orderCustomer.Entity;
        }
        set
        {
            NotifyPropertyChanging("OrderCustomer");
            orderCustomer.Entity = value;
            NotifyPropertyChanged("OrderCustomer");
        }
    }
}
```

EntityRef действа като един вид лепило между една проста справка (която бихме искали да се използва) и търсене в таблица (което е това, което ще трябва да направите, когато ние използваме тази референтна LINQ). Добрата новина е, че като програмисти можем да напишем:

```
Customer newCustomer = new Customer();
Order newOrder = new Order();
newOrder.OrderCustomer = newCustomer;
```

Тази проста задача ще има ефект на свързване на заповедта на клиента, така че записа на база данни за тази цел държи идентификатора на клиента поръчката е за. Ние можем да използваме тази техника всеки път, когато искат да приложат една връзка между един ред в една таблица и един ред в друг.

Когато ние определяме имуществото на сдружението ние също да кажете LINQ две други неща. Ние го кажа, че асоциацията е "външен ключ" и ние също така да определи имуществото на класа, в който ще се съхранява стойността. Ако сте използвали бази данни, пред вас ще бъдат запознати с

идеята за външен ключ. Това е първичен ключ от друга база данни. В този случай той е основния ключ от базата данни на клиента, че може да се използва за идентифициране на конкретния клиент, който е направил поръчката.

Точката за съхранение в асоциацията разказва LINQ която частната собственост на нашата база данни действително ще разполагат с информация за свойства.

Бази данни и Плавателен

Плавателен в системи за бази данни е важно. Тя позволява на системата да открие една част от информацията, дадена на друг. В момента можем да започнем от един ред и директно се намери клиента за тази цел. Това е така, защото един ред в таблицата с цел съдържа на клиент на клиента за тази цел. Въпреки това, ние не можем да намерим обратния път. Започвайки със заповед ние нямаме начин да разберем на клиента, който го постави.

За да се позволи правилното плавателност ние трябва да сложите нещо в класа на клиентите, които ще ни позволят да се намери на поръчките, които клиентът е поставил. Въпреки това, нещата вече стават още по-сложно, защото един клиент може да създаде много поръчки. Това означава, че характерът на взаимоотношенията, се променя в зависимост от посоката, в която пътувате:

- А определен ред само някога ще бъде свързана с един клиент, клиентът, който е създаден на поръчката. Това означава, че връзката, пътуващ от поръчка на клиента е едно към едно (една цел един клиент).
- А конкретен клиент ще бъде свързано с много поръчки. Това означава, че връзката, пътуващ от клиент по поръчка е едно към много (един клиент за много поръчки)

Когато ние проектираме нашите бази данни, ние трябва да се помисли за плавателността, по-специално, че трябва да помислим за това как ние ще се движите с помощта на нашите взаимоотношения, както и дали те са едно към едно или едно към много.

Свързването на Клиент за всичките си поръчки

Един клиент може да постави много поръчки и така връзката, която ние трябва да се приложи е "един към много" на връзката. Ако бяхме прилагането на настоящия използвайки C # класове ние ще се използва за събиране на някакъв вид, може би масив или List. В LINQ такава връзка се осигурява от класа EntitySet. Това е малко като EntityRef, освен, че може да управлява набор от елементи, а не само един. Ние може да се свърже с клиента, за да си поръчки чрез добавяне на EntitySet към класа на клиентите.

```

[Table]
public class Customer : INotifyPropertyChanged,
    INotifyPropertyChanging
{
    [Column(IsPrimaryKey = true, IsDbGenerated = true,
        AutoSync = AutoSync.OnInsert)]
    public int CustomerID { get; set; }

    private EntitySet<Order> orders = new EntitySet<Order>();

    [Association(Storage="orders", ThisKey="CustomerID",
        OtherKey="OrderCustomerID")]
    public EntitySet<Order> Orders
    {
        get
        {
            return orders;
        }
        set
        {
            orders = value;
        }
    }
}

```

Това изглежда много подобно на EntityRef, че ние създадохме по-рано, с изключение на това, че има допълнителна информация, в асоциация, за да опише връзката.

ThisKey дава името на имота, който ще бъде използван от Поръчката да се намери клиентът тя е свързана с. Ние можем да използваме на клиент, на първичния ключ за клиента, за да направите това.

OtherKey дава името на елемента в класа на поръчки, която ще реализира асоциацията в другата посока, т.е. позволява да намерите клиентът тя е свързана.

Може би се чудите защо не сме уволнени всички NotifyPropertyChanged събития когато поръчките се променили. Това е така, защото промяната на стойността на EntitySet е нещо, което едва ли някога ще направя. Не забравяйте, че това е контейнер за поръчките, не самите поръчки. Ние ще добавим поръчки на клиента с удар неща в EntitySet, а не чрез замяна на самата EntitySet.

За да направи работата на сдружение правилно, че трябва да се направят някои промени в позоваването в класа на Поръчка. Те ще се свързват двата края на сдружението заедно.

```

[Table]
public class Order : INotifyPropertyChanged, INotifyPropertyChanging
{
    ...

    [Column]
    public int OrderCustomerID;

    private EntityRef<Customer> orderCustomer =
        new EntityRef<Customer>();

    [Association(IsForeignKey = true, Storage = "orderCustomer",
        ThisKey = "OrderCustomerID")]
    public Customer OrderCustomer
    {
        get
        {
            return orderCustomer.Entity;
        }
        set
        {
            NotifyPropertyChanging("OrderCustomer");
            orderCustomer.Entity = value;
            NotifyPropertyChanged("OrderCustomer");
            if (value != null)
                OrderCustomerID = value.CustomerID;
        }
    }
}

```

Сдружението се определя като холдинг външен ключ (на първичния ключ на базата данни клиентът) и с помощта на стойността на OrderCustomerID да държи тази стойност.

Тази версия на препратка прави копие на стойността на клиент в колоната OrderCustomerID в таблицата, когато клиент се възлага поръчката. С други думи, ако аз напиша нещо като това:

```

Customer c = new Customer();
Order o = new Order();
o.OrderCustomer = c;

```

Когато се възлага имота OrderCustomer зададената метода по-горе ще отчита стойността на клиент и го копирайте в стойността OrderCustomerID да записва клиентът за тази цел.

Ако всичко това е боли главата си малко, и аз трябва да призная, че боли мина първия кръг време, не забравяйте, проблемът, че ние се опитваме да решим. The ThisKey и OtherKey описанията в асоциацията казват всяка страна на връзката как да се намерят един друг.

Добрата новина е, че след като сме приложили този модел ние можем лесно да добавите нови поръчки на клиент:

```

Customer c = new Customer();
Order o = new Order();
o.OrderCustomer = c;
c.Orders.Add(o);

```

Класът EntitySet осигурява метод добавя, че ни позволява да добавите поръчки към базата данни. За да работят чрез съдържанието на комплекта лице можем да използваме някоя от конструкциите ние обикновено използваме, за да обработим колекции:

```
foreach (Order o in c.Orders)
{
    // do something with each order
}
```

Това не изглежда толкова специален, но специалното нещо е, че ние пишем C #, за да работите с базата данни.

Поръчки, Продукти и Database Design Сега имаме връзка между клиенти и поръчки, който работи имот. Един клиент може да има много голям брой на поръчките, както и за да е винаги свързан с клиента, който го постави. Следващото нещо, което трябва да направите, е да попълните в реда, с продукти, които клиентът може да са закупени. Това е точката, в която ударихме на камък с нашия оригинален дизайн на нашата таблица за поръчка:

Order ID	Customer ID	Product ID	Quantity	Order Date	Status
1	123456	1001	1	21/10/2010	Shipped
2	111111	1002	5	10/10/2010	Shipped
3	654322	1003	4	1/09/2010	On order

Ако се вгледате внимателно в дизайна се оказва, че няма начин, ние можем да имаме ред, който притежава повече от един продукт. А специално за да може да бъде за повече от един елемент, но всички те трябва да бъдат от една и съща позиция.

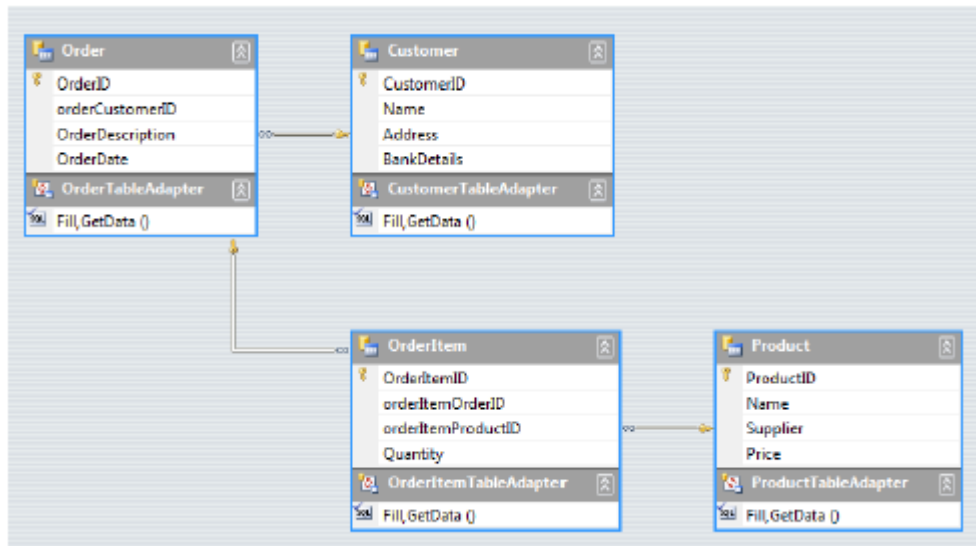
Тя може да бъде, че това е напълно ОК. Човекът, който иска да използва вашата система може да се продават продукти, които са много скъпи и са само със поръчал един в даден момент. Алтернативно това може да бъде пагубно на системата може да се използва от един супермаркет, който желае да се съберат поръчки, които съдържат голям брой различни продукти.

За да реши този проблем, трябва да се добави друга маса, която ще се проведе на всички елементи във всеки ред. Всеки OrderItem ще бъде свързано с определен ред. След това за всяка поръчка ще проведе събиране на тези елементи.

OrderItem ID	Order ID	Product ID	Quantity
1	56	1001	1
2	56	1002	5
3	12343	1003	4

Таблицата по-горе показва, че заповедта 65 е съставена от два продукта, а Телефон Windows и 5 сирене рендета. (Ако се чудите къде продуктите идват от, да погледнем таблицата Products начин обратно в началото на тази глава).

База данни дизайнери обичат да използват диаграми, които показват как различните елементи пасват. Диаграмата за базата данни, която сме изградили изглежда така:



Малките ключовете показват къде външен ключ се използва в единия край на сдружението. Малките безкрайности (∞) показват, че сдружение може да се свърже с всеки брой на тази позиция.

Ако все пак много на дизайна на базата данни ще бъдат използвани за построяване на диаграми, като една по-горе. Това всъщност е създаден от базата данни, която се образува от класове, които ние създадохме.



Разтворът в Demo 02 SalesManagement съдържа заявление за Windows Phone Silverlight, която добавя Order, OrderItem и маси за продукта в базата данни на клиентите. Можете да се движите в списъците на клиенти и продукти и да намерите поръчките, които клиентът е създадал и съдържанието им.

LINQ заявки и Присъединяване

Сега, когато имаме нашите данни в база данни, можем да използваме много мощни механизми за заявки, за да получим информация от него. Например, ние може да искаме да намерим всички поръчки, които са били пуснати на определена дата.

```
DateTime searchDate = new DateTime(2011, 8, 19);

var orders = from Order order
              in activeDB.OrderTable
              where order.OrderDate == searchDate
              select order;
```

Този заявка търси масата за нареждане на активния в момента ПБ и намира всички поръчки, създадени на 19 Август, 2011. Ние не трябва да напиша някоя от търсенето, като резултатът е просто връща в променлива наречена поръчки. Може би се чудите каква е действителната типа на поръчките, е. Всъщност тя е от тип "LINQ заявка". Изявлението по-горе не прави нищо, докато не опиташ да се консумират на данните, които тя притежава. Например, ако ние се опитаме да работят на общата стойност на продажбите за този ден ние може да напише нещо като това:

```
int totalSales = 0;

foreach (Order order in orders)
{
    foreach (OrderItem item in order.OrderItems)
    {
        totalSales += item.OrderItemProduct.Price;
    }
}
```

Първият foreach работи чрез поръчки стават всяка поръчка на свой ред. Тя е само, когато програмата започва действително да използва данните, които LINQ ще скачат в действие и започват да произвеждат резултати.

Имайте предвид, че това е от значение, ако искате да използвате резултата от заявка повече от веднъж. Ако използвате една и съща заявка два пъти вие ще откриете, че вашата програма ще работи малко по-бавно, тъй като LINQ ще бъде привлекателен неща от базата данни на два пъти. Ако искате да използвате даден резултат повече от веднъж, че е най-добре да използвате заявка за продукт списък на елементи, които след това можете да работите през толкова пъти, колкото искате:

```
List<Order> DateOrders = orders.ToList<Order>();
```

Това създава списък нарича DateOrders който съдържа резултатите от търсенето. Ние можем да използваме списъка в нашата програма, без генериране на нови сделки на базата данни. Методът на ToList отнася заявката и генерира списък с резултати.

Присъединявайки се към две таблици с въпрос

LINQ има друг трик в ръкава си. Тя може да създаде заявки, които отиват в повече от една таблица, процес, наречен "присъединяване". Може би бихме могли да искате да създадете пълен списък на всички поръчки, направени от всички клиенти, изброени в поръчка на клиента. Искаме да открие ред и на клиенти, които съответстват на двойки и да ги изброим. Използвайки присъединят дума LINQ може да направи това в едно действие:

```
var allOrders = from Customer c in newDB.CustomerTable
                join Order o in newDB.OrderTable on
                    c.CustomerID equals o.OrderCustomerID
                select new { c.Name, o.OrderDescription };
```

Ако не сте виждали един от тях ще ви бъде простено, се чудех как на земята го компилира. Ключът е в LINQ име, това, че **from**, **join** и **select** ключови думи са всички интегрирани в езика C #, за да направи тези видове заявки работят. В първата част на заявка намира всички клиенти в

клиентската маса. Тази заявка се присъедини към втория, който работи чрез всички поръчки вършат присъединят на двете ID стойности, съхранявани в клиент и ред.

По същество това, което той казва, е "Намери ме на двойки на поръчки и клиенти, където OrderID в клиента съответства на идентификатора на клиента в реда".

След като установи, мач го след това създава нов тип, който съдържа само резултатите, които искаме, в този случай името на клиента и OrderDescription от поръчката. Обърнете внимание на магия `var` отново е тук. В този случай, ние добиваме SQL заявка, която генерира колекция от тип, съдържащ две струни на съответните имена.

```
List<String> OrderDescriptions = new List<String>();
```

```
foreach (var orderDesc in allOrders)
{
    OrderDescriptions.Add(orderDesc.Name + " order: " +
                          orderDesc.OrderDescription);
}
```

Както и при предишната заявка, LINQ всъщност не прави нищо, докато програмата започва да работи чрез резултатите. Имайте предвид, че ние трябва да използваме тип `var` за стойността на `orderDesc`, че ние сме привлекателен от заявката. Той съдържа два елемента, които сме начислили, когато ние създадохме вида в заявката.

В резултат на заявка му може да бъде свързан към елемент на дисплея да се покаже списък:

```
Rob Miles order: Camera Order
Rob Miles order: Cheese Order
Rob Miles order: Barge Pole Order
```

Ключовата дума `VAR`

Ключовата дума `var` може да изглежда малко страшно, ако, като мен, сте свикнали да се обявява променливи от определен тип, и след това просто да ги използвате. Харесва ми начина, по който C # компилаторът fusses за моите програми и ме спира да прави глупави неща, като:

```
int i = "hello";
```

Направен е опит да се сложи низ в цяло число е лошо нещо да се направи, и няма да се компилира. Въпреки това, на пръв поглед `VAR` дума изглежда като начин да се прекъсне този прекрасен мрежа за сигурност. Ние можем да създадем неща, които не изглежда да има вид, свързани с тях, които трябва да бъдат лоши. Въпреки това, можете да бъдете сигурни, че въпреки че ние не даваме име за тип като този, на C # компилаторът знае точно какъв вид е направен от `VAR`, и ще откаже да се състави програма, която се опитва да използва такъв тип в неправилен начин.

Изтриване на елементи от Database

Видяхме как да добавите елементи в база данни. Възможно е също така да изтриване на елементи и. Например, ако клиентът реши, че те не искат даден елемент, за да можем да го изтриете от базата данни:

```
ActiveDB.OrderItemTable.DeleteOnSubmit(item);
```

Можем също така да изтриете колекции от предмети, използващи DeleteAllOnSubmit, включително колекция идентифицирани като резултат от заявка.

Чужди някои основни ограничения и да изтриете

Някои от нашите показатели за данни имат взаимоотношения с другите, например Customer ще съдържа броя на поръчките, както и заповед ще съдържа редица ценности OrderItem. Ако родителския обект се заличава преди децата (т.е., ако се опита да изтриете даден клиент, който съдържа някои поръчки) базата данни ще откаже да изпълни това, и ще се хвърли изключение, като резултат. Ако искате да изтриете даден клиент ще трябва първо да изтриете всички елементи на ред във всеки ред, и след това да изтриете поръчките, и след това да изтриете клиента.

Какво научихме

1. Windows приложения на телефона могат да използват Language Integrated Query (LINQ), за да си взаимодействат с бази данни могат да се съхраняват във файлове в изолирано съхранение в устройството. Самите базите данни се управляват от сървъра SQL база данни , но това не е възможно да се използва SQL команди, за да си взаимодействат с база данни , вместо заявки са изразени с помощта на LINQ .
2. Базата данни се състои от множество от таблици . А маса се състои от колони (различни показатели за данни като име , адрес и информация за банката) и редове (цялата информация за даден елемент - подробно описание на един отделен клиент име, адрес и банкови) .
3. Една колона в таблица действа като " първичен ключ " , който притежава стойност, която може еднозначно да идентифицира този ред (уникалния идентификационен код на този конкретен клиент) . Базата данни може да бъдете помолени да създадете автоматично първични ключове , които са уникални за всеки ред в таблица.
4. Windows приложения на телефона могат да създадат класове, които се съхраняват от LINQ като таблиците в базата данни . Атрибутите [Таблица] и [Колона] се използват в рамките на дефиницията на класа , за да се уточни ролята на компонентите. Допълнителна информация може да бъде добавен към атрибути идентифицира първични ключове .
5. Класове за бази данни , управлявани от LINQ могат да добавят уведомяване поведение към свойствата на данни, така че LINQ автоматично ще актуализира записите в база данни , когато имотите са се променили. Ако тези средства се използват с променените събития в Silverlight елементи на дисплея това може да позволи автоматичното обновяване на базата данни , когато предметите са променени от потребителя .

6. Базата данни може да съдържа връзки , където колона в една таблица съдържа първични ключови ценности, които определят реда в друга . Те са наречени " външни ключове " , тъй като те са първични ключове , но не за масата , която ги съдържа .
7. LINQ осъществява връзки с помощта на класа EntityRef , който съдържа препратка към външен ключ в друга база данни. Това се използва за " 1:1 " връзки . Класът EntitySet се използва, за да се даде възможност на член на една маса , за да се отнасят до голям брой редове в друга таблица . Това се използва за " един към много " връзки . Тези елементи на данни се изменят с атрибута за асоцииране, което се описва как се реализира връзката на таблицата.
8. База данни, често ще има едно към едно отиваш в една посока (заповед е прикрепен към един клиент) и един за мнозина става в другата (един клиент може да постави много поръчки) . За такива сдружения да работят правилно е важно , че сдруженията са конфигурирани правилно и че чужди ключове се съхраняват правилно .
9. LINQ заявка може да се използва за извличане на структурирани данни от базата данни. Заявката е само действително изпълнява, когато резултатите от запитването се консумират от програмата. LINQ заявки могат да използват , за да се присъединят към ключовата дума се комбинират резултатите от няколко запитвания.
10. При изтриване на елементи от база данни на всички деца елементи трябва да бъдат изтрети преди родителския обект се отстранява , в противен случай няма да има елементи в базата данни, съдържаща основни ключови стойности за обекти , които не съществуват .

Глава 9. Създаване на Windows Phone приложения

Знаем достатъчно, за да създадем приложения на Windows Phone. В тази секция ще разгледаме какво взима програмата и я превръща в „истинско“ приложение. Това включва набор от теми – от това как програмата да се сдобие със стартов екран и икони до това как може да създава и управлява задачи, които са на заден план.

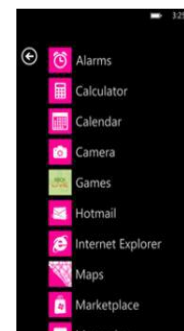
9.1. Windows Phone икони и стартов екран

До момента всичките програми, които написахме, имаха една и съща иконка и стартов екран. Това бяха „празните“, които Visual Studio създава, когато се прави нов проект. Има две изображения, които Windows Phone използва, за да идентифицира програмата, когато тя се запазва в телефона. За да разберем, как те се използват, трябва да научим малко за това как Windows Phone потребителите намират и зареждат програмите на устройството.



Windows Phone има „Start“ екран, който се показва, когато потребителят натисне стартовия бутон на устройството:

Потребителят след това може да натисне всяка от големите плочи на стартовия екран, за да зареи програмата, която стои зад тях. Също така, те могат да се плъзгат по стартовия екран отляво

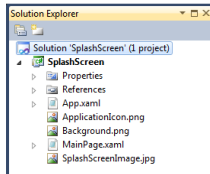


(или да се натисне малката стрелкичка най-отгоре на устройството), за да придвижат списък с приложения, инсталирани на устройството.

В този случай потребителят може да скролира нагоре-надолу в списъка и да избере програма, която желае да зареди. Ако потребителят иска да стартира някоя от стартовия екран, той може да я „прикачи“ там.

Silverlight икони

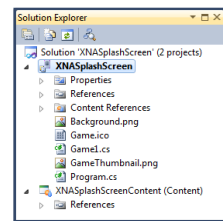
Ако създавате приложение на Silverlight, Visual Studio ще създаде проект, който съдържа версии, които са „под разбиране“ на тези две икони:



Файлът ApplicationIcon.png съдържа изображението, което ще се използва в списъка с приложения. Картинката, която ще се използва, когато приложението е „прикачено“ към стартовото меню е Background.png.

XNA икони

При създаването на XNA приложение Visual Studio ще създаде малко по-различна аранжировка на програмите, а файловете с икони ще бъдат с различни наименования:



Файлът GameThumbnail.png съдържа изображението, което ще се използва в списъка с приложения, а Background.png – картинката, която ще се използва при прикачването към стартовото меню.

Изработване на икони

Ако искате да създадете програмни иконки (а и трябва), можете да отворите и редактирате тези файлове с всяка Paint програма. Запомнете, че те се съхраняват в папката на проекта заедно с файловете от този проект. Освен това, трябва да се уверим, че файловият тип се запазва, както и размерността на оригиналните изображения.

От личен опит се оказва, че рисуването на икони по този начин (особено малките) е нещо като форма на изкуство. По тази причина, ако успеете да намерите художник, който да свърши тази работа за Вас, това би било добра идея. Ако трябва да направите това сам/а, най-добре е да изберете възможно най-опростен дизайн, за да сте сигурни, че дава на потребителя точна представа относно същността на програмата. Запомнете, че иконката на Вашето приложение също така се използва, за да го рекламира в Marketplace, така че колкото по-добра и елегантна е тя, толкова по-голям шанс има програмата Ви да бъде свалена и купена.



и иконки за него.

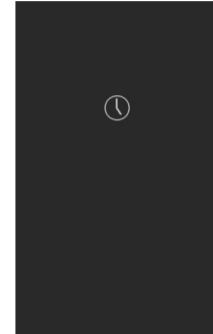
Ако използвате Marketplace тестовия комплект, за да подготвите приложението си за продажда, можете да качите цялото си творчество

Начален екран

Стартовият екран е нещо, с което програмта се показва на екрана, когато е заредена. Това може да се използва, за да отличи приложението Ви, показвайки логото на компанията Ви, когато програмата стартира. Началните екрани са също полезни, за да подобрят потребителската практика, когато програмата започва да се зарежда. Най-малкото, което те ще направят е да предоставят хубава гледка докато програмата се зарежда.

Silverlight стартов екран

Системата за зареждане на Silverlight има вграден начин на показване на начален екран. Новият Silverlight проект всъщност съдържа файлово изображение, което се изобразява по време на интервала, получен между времената, когато програмата се зарежда и когато се появява MainPage на екрана. Ако сте стартирали програмата на емулятор, вероятно сте виждали това за момент съвсем в началото на стартирането на програмата.



Ако програмата Ви стане по-голяма с повече елементи на страница и ресурси, които да се заредят, тази страница може да бъде показвана повече пъти. Можете да направите собствена версия чрез изменение на файла SplashScreenImage на Silverlight проекта.

XNA стартов екран

Програмите XNA нямат начален дисплей, който да е доставен като част от проекта. Вече видяхме, че начинът, по който екрана се създава е доста различен. Когато XNA програма се зарежда, методът LoadContent се извиква, за да извлече неща, които стоят на съхранение и да ги направи достъпни на играта. Веднъж, след като съдържанието бива заредено, методите Draw и Update се извикват. Това може да доведе до проблеми. Рамката XNA настоява играта да показва нещо в рамките на пет секунди след стартирането на програмата. Ако това не се случи, рамката счита, че играта е зациклила и я спира. Това още означава, че играта реално никога няма да се стартира, ако зареждането на съдържанието отнема прекалено много време.

За щастие, XNA не трябва да зареди цялото съдържание, за да стартира. Една игра може да използва управителя на съдържанието, за да зареди различни неща по всяко време. Програмата също може да премахне някои обекти, за извърши зареждането на фона докато играта е пусната. В заключение може да се каже, че за XNA е необходимо да се развие стратегия за зареждане на приложението. Вероятно методът LoadContent за играта може само да зареди текстура за началния екран и това да бъде изобразено докато останалите обекти от съдържанието биват извикани.

Оказва се, че това е често срещан проблем при всички платформи. Основен ограничаващ фактор при развитието на все по-големите конзолни игри е лимитираната скорост на достъп и честотата на пренос на оптичните устройства, използвани при зареждането на информация от хранващото устройство.

9.2. Бързо превключване между приложенията

Windows Phone е много мощно устройство. Компютри с такава мощност са изпълвали цели стаи само преди години. Освен това, сравнявайки с екранните системи, това е нещо смутително. Няма процесор, който е доста мощен, и е ограничен от ограничената вместимост на батерията. Докато телефонът може да изпълни много операции, най-добре е батерията да изразходва минимум енергия възможно най-дълго.

Сега ще открием как приложението може да се програмира така, че да предостави добър потребителски опит в рамките на средата за отделни задачи, осигурена от Windows Phone. Това осигурява добро вникване в това, как на програмистите често им е напала да се възползват от системи, които са направили компромис с ограниченията от страна на платформата и операционната система.

Навигация на задачи в Windows Phone

Операционната система на Windows Phone е наследствено управляваща много задачи. Това означава, може с лекота да зарежа повече от един проес по едно и също време. Ето как може да пуска музика, използвайки програмата Zune докато Вие си проверявате имейла. Въпреки това, поради набор от много технически причини, свързани със звука, тази способност за управление на много задачи не се простира върху приложения, пуснати на системата. Създателите на Windows Phone системата са взели добре обмислено дизайнерско решение да не позволят на повече от едно приложение да бъде активно в кое да е време.

От части, това е хубаво нещо. То спира програмите от това да изглеждат зле, само защото друго приложение, пуснато на телефона краде от паметта, за да свърши нещо друго. Това „отиване и вършене на други неща“ е поведение, което силно се окуражава от дизайна на потребителския интерфейс на Windows Phone. Знаем как приложението може да съхрани данни, използвайки огромно складиране на телефона; онова, което трябва да направим, е да съхраним и си възвърнем данни, за да създадем илюзията, че програмата ни никога не е била спирана.

По-нататък ще видим как можем да създадем приложения, които обработват нещо на заден план от името на потребителя. Освен това, ще открием, че тази способност е внимателно управлявана, за да се осигури факта, че когато програмата прави нещо на заден план, това не засташава нищо потребителския опит.

Бутоните Start и Back

Докато Windows Phone не притежава свойството да управлява много задачи, то има характеристики, проектирани, за да улеснят потребителя при влизане и излизане от програма. Те се базират на хардуера на бутоните Back и Start, рикачени към всяко Windows Phone устройство.

Накратко казано, Start придвижва потребителя към новата програма, която иска да зареди, а Back го отвежда назад към предходната. Потребителят може да натисне Start по всяко време, за да

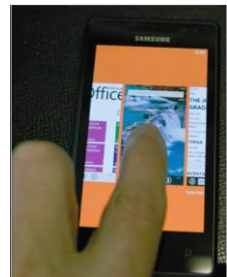
отвори стартовия екран и да избере различна програма, която да зареди. Когато приключи с тази програма, се натиска Back, за да се върне към първоначалната.

Използвани в комбинация, тези бутони значително увеличават потребителската практика. Лесно е да се натисне Start, за да изпратите бързо SMS съобщение и след това - Back, за да се върнете към това, с което се занимавахте преди това. Бутоните Start и Back позволяват на потребителя на навигира през куп наскоро използвани приложения по подобен начин като сърфирането в интернет, където клика върху линкове и използва командата Back, за да се върне към предишната страница.

Натискането на Start бутона не предизвиква премахване на приложението от паметта, за да освободи място за ново; вместо това то се „деактивира“. Можем да считаме такъв вид приложение като “waiting in the wings”, готово да бъде повикано обратно на сцената по някое време по-късно. Разбира се, това връщане на сцената може никога да не дойде (за реални изпълнители на живо) и така, когато приложението се деактивира, то трябва да съхрани всякакви важни данни в случай, че потребителят не се върне към него.

Продължително натискане на бутона Back

В допълнение с натискането и освобождаването на Back бутона потребителят на телефона може да го задържи за половин секунда или да активира екран с приложения, затворени наскоро. Ако потребителят избере някое от тях, то се рестартира точно като че ли потребителят се връща към приложението чрез Back бутона.



Това показва как изчакващите приложения биват избрани. В момента се преминава от приложенията, активни в момента (Office и Bing search) към онези, към които желаем да се върнем.

Създаване на приложение с „добро държание“

Резултатът от това е, че оприложението трябва да умее да запази състоянието си, след което да се възстанови към същото състояние, когато потребителят се върне към него. Системата Windows Phone ни помага да направим това чрез изпращане на съобщения от приложението, които да загатнат какво ще се случи с него. В следващата стъпка ще изучим ситуациите, при които се изпращат съобщения и как да направим приложенията и игрите ни да отговарят коректно, за да изглежда така сякаш те са винаги активни.

Разбиране на бързото превключване между приложенията

Името на този набор от поведения е “Fast Application Switching”. Тъй като деактивирани програми обикновено се държат в паметта, процесът на превключване между тях би трябвало да е бърз. Ако ще разбираме как да правим програми, които се държат както трябва, ще трябва да се научим малко от терминологията, използвана от Windows Phone дизайнерите, за да опишат какво се случва, за да работи това.

Методите за бързо превключване на събития

Файлът App.xaml.cs в Silverlight приложението съдържа методи, които се извикват, когато настъпи бързо превключване между различни приложения:

```
// Code to execute when the application is launching (eg, from Start)
// This code will not execute when the application is reactivated
private void Application_Launching(object sender, LaunchingEventArgs e)
{
}

// Code to execute when the application is activated (brought to foreground)
// This code will not execute when the application is first launched
private void Application_Activated(object sender, ActivatedEventArgs e)
{
}

// Code to execute when the application is deactivated (sent to background)
// This code will not execute when the application is closing
private void Application_Deactivated(object sender, DeactivatedEventArgs e)
{
}

// Code to execute when the application is closing (eg, user hit Back)
// This code will not execute when the application is deactivated
private void Application_Closing(object sender, ClosingEventArgs e)
{
}
```

Видяхме първо файла App.xaml.cs в секция *4.5 Страници и Навигация*, когато го използвахме като място, където да създадем променливи, които да бъдат поделени между всички страници в Silverlight приложението. Именно класът контролира създаването и управлението на Silverlight страниците. Нашата работа е да накараме приложенията да използват тези събития, за да дадат на потребителя представата, че програмата винаги е готова за употреба. Подобен набор от съобщения е предоставен в XNA програмите, така че игрите да могат да запазят състоянието си.

Windows Phone приложението LifeCycle

Сега, след като знаем какво представляват съобщенията, придружаващи събитията и как потребителят се предвижда от едно приложение към друго, можем да обмислим как тези съобщения действат в житейския цикъл на едно приложение.

Стартиране на програма

Потребителят стартира програма, като я избере от тези, които се предлагат на телефона. Когато тя се стартира, програмните файлове се зареждат в паметта и изпълнени от Windows Phone процесора. По време на процеса на стартиране се извиква метода Application_Launching:

```
private void Application_Launching(object sender, LaunchingEventArgs e)
{
}
```

Този метод трябва да заредси всякакви данни от файловата система. Но в случай, че данните, които ще се четат са много, най-добре би било това да се зареди на отделно място (напр. на заден план). Това се налага, тъй като приложението няма да изобрази нищо на екран докато методът не се изпълни. Ако програмта прекарва дълго време в зареждането на данни, това означава, че много бавно ще се зареди.

От тук нататък потребителят може да направи две неща: да излезе в програмата чрез натискане на Back бутна на най-горното ниво, където е менюто или да премине към друга програма, натискайки Start бутона.

Натискане на Back за затваряне на приложението

Приложенията на Windows Phone нямат опция за „излизане“, която може да се използва, за да се затворят. Но това не означава, че е невъзможно да се затвори приложение. Ако потребителят натисне Back бутна на „най-горното ниво“ на приложението (напр. екранът, който се появява, когато приложението е било стартирано), приложението ще бъде затворено в този момент. Когато това стане, то се премахва от паметта, а предишното приложение (ако има такова) се връща. Това създава поведение, където потребителят може да „излезе“ от приложението и да се върне към онова, което са прекъснали.

Тъй като Windows PC приложение това на Windows Phone използва събитие, съпровождащо затварянето на програмата, така че да се запазят всякакви отворени файлове.

```
private void Application_Closing(object sender, ClosingEventArgs e)
{
}
```

Възможно е дори да се даде на потребителя шанс да потвърди, че ще затвори програмата. Приложението за телефон Microsoft Word ще изобрази съобщение „Желаете ли да запазите Вашия текст“, ако се опитвате да излезете от него, след като сте въвели текст.

За жалост, някои от нас, потребителите няма винаги да затворят програмата така „чисто“. По-скоро те ще натиснат Старт бутона и ще отидат да вършат нещо друго отколкото да затворят приложението и да стартират друго. Когато проектираме Windows Phone приложение, трябва да се погрижим да дадем точни пояснения, как програмата ще се държи след като потребителят излезе от нея и се върне по-късно. Трябва да се уверим, че приложението винаги прави онова, което потребителят очаква от него и не прави изненадващи промени или да загуби данни.

Деактивиране на приложение, натискайки Start

Когато потребителят натисне Старт, програмата, активна в момента се деактивира. Това означава, че тя „заспива“. Спящите програми са все още в паметта, но не са активни в момента. Те са готови да бъдат върнати отново, използвайки „Бързо превключване между приложенията“.

```
private void Application_Deactivated(object sender, DeactivatedEventArgs e)
{
}
```

В момента, в който приложението „влезе в покой“, то изпраща „деактивиращо“ съобщение, за да укаже, че е отишло на заден план, за да преотстъпи мястото си на друго. Тук приложението трябва да запамети както състоянието на действието, изпълнявано в момента, така и данните във файловата система в случай, че не е отворено отново. Една програма също може да „влезе в покой“, ако екранът за заключване на телефона не е бил използван и се е изключил. Друг начин това да се случи е, когато потребителят натисне бутона Търсене или започне да използва камерата, за да снима.

Преобразуване на приложение от състояние в покой към задгробно такова

Истински театър би имал проблем, ако голям брой изпълнители биват изпратени зад кулисите, тогава мястото за сцената би се препълнило. Ако това се случи, някои от изпълнителите ще бъдат изпратени обратно към съблекалните си. А в случай, че те отново ще потрябват, ще трябва да им се обадят, което означава, че ще им е необходимо повече време, за да се върнат към изпълнението си.

Точно същото нещо се случва в Windows Phone. Колкото повече приложения биват деактивирани и изпратени в състояние на покой толкова по-бързо ще се препълни паметта. В този момент някои от приложенията трябва да бъдат премахнати от паметта, за да отстъпят място за други. По отношение на Windows Phone те се преобразуват състоянието от „в покой“ (в паметта и на дпстъп за Fast Application Switching) в „задгробно“ (премахнати от паметта). Потребителят все още може да възвърне задгробно приложение, ако натисне бутона Назад достатъчно пъти или го избере от менюто Long Press, но това ще отнеме малко повече време.

Ако задгробна програма се активира наново някога, тя ще се зареди в паметта и всичките ѝ обекти ще трябва да бъдат конструирани наново, точно както, ако тя се зарежда за пръв път.

До този момент може да ни се прости, че мислим, че задгробен означава едно и също с това се излезе за програмата. Но, това не е точно така. Великото нещо за плочата е, че можем да пишем неща на нея. Системата на Windows Phone осигурява някакво пространство в паметта, където приложението може да складира временна информация в случай, че бива деактивирано. Тази информация може да бъде повторно заредена в приложението и поддържана независимо от факта, дали приложението е изпратено в състояние в покой или задгробно такова.

Windows Phone системата също използва такава временна памет. Ако приложението се деактивира на определена страница, то когато се използва обратно, същата страница ще се появи отново. Системата Silverlight използва съхраняването на временни състояния в паметта, за да отбележи активната страница, когато тя е деактивирана.

Подновяване на приложение в състояние на покой или задгробно такова

Когато приложението се стартира, то ще получи или „Launching“ съобщение, ако се зарежда като ново, или „Activated“ съобщение, ако се реактивира.

Приложението може да бъде тествано, за да се провери дали си връща активността от състояние на „покой“ или на „задгробна плоча“.

```
private void Application_Activated(object sender, ActivatedEventArgs e)
{
    if (e.IsApplicationInstancePreserved)
    {
        // Dormant - objects in memory intact
    }
    else
    {
        // Tombstoned - need to reload our objects
    }
}
```

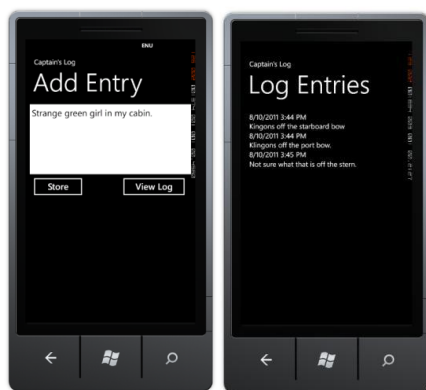
Този метод се изпълнява, когато приложението се активира и тества коя форма на активация се изпълнява. Когато то е активирано наново от състояние на покой, то всичките обекти от паметта ще са непокътнати и зареждането може да продължи. Но когато се реактивира от състояние на „задгробна плоча“, то трябва да презареди всичките данни, тъй като те ще са били премахнати от паметта.

Обърнете внимание, че приложението не получава съобщение, когато преминава от състояние в покой към задгробно такова; може само да каже в какво състояние е, ако се стартира наново.

Fast Application Switching в приложението

Сега ще разгледаме приложението как да го направим да работи като едно цяло, което работи правилно с отношение към Fast Application Switching. В Глава 5 създадохме приложение, наречено „JotPad“, което позволява на потребителя да запазва набързо написани бележи на телефона. За да изследваме Fast Application Switching, ще използваме подобрена версия на „JotPad“, наречена „The Captain’s Log“.

Въведение в „The Captain’s Log“



The Captain’s Log позволява на потребителя да запамети редица от входни данни във входен файл. При всеки вход се указва времето на посещение, което се добавя към данните, а те представляват дълъг текстов низ. Когато програмата се стартира, се изобразява страницата Add Entry. Потребителят може да въведе някакъв текст и да натисне Store, за да запише входните данни или View log, за да види съхранените данни. Ако се натисне View log, програмата се отправя към втората страница, която показва натрупаните данни от входа. Всеки път, когато се дабавят такъв тип данни, към тях се добавя бележка с настоящото време и дата, както беше показано по-

горе.

Искаме да направим програмата Captain's Log да работи по начина, по който потребителя очаква. Тя трябва да изглежда така сякаш е достъпна по всяко време, така че потребителят да е в неведение за начина, по който данните се съхраняват и зареждат при необходимост. Когато потребителят се върне към приложението, данните, които са били запаметени преди, трябва да са налице, давайки впечатлението, че винаги са били там.

Captain's Log и бързото превключване между приложенията

Когато създаваме приложение, трябва да решим как ще се сържи то, когато потребителят го остави. Дали ще трябва да съхрани цялата информация във файловата система или да задържи нещата, ако потребителят затваря програмата. Това решение има въздействие върху потребителския опит. Потребителят може да не очаква запамятаване на информацията, ако излезе от приложението чрез Старт бутона. Всъщност юзърът може да си мисли, че е способен на натисне стартовия бутон и да влезе отново в Captain's Log приложението като резултат от незавършено въвеждане на данни, които са решили да се публикуват.

Забележете, че това не са технически въпроси. Ако правите Windows Phone програма за клиент, трябва внимателно да се консултирате с него, за да разберете какво искат от програмата. Най-лошото нещо, което може да се направи са предположения за това как би трябвало тя да работи.

Когато потребителят натисне Старт бутона, за да излезе от приложението, вероятно няма да е най-добре, ако данните се запазят като „работа в процес“ и реално неприкрепени към основните ресурси. По този начин ще направим да работи програмата Captain's Log.

Данни в програмата Captain's Log

Програмата Captain's Log само съдържа два даннови елемента, които са низ. Те са пълните текстове от входа, които се изобразяват пти страницата „log entries“ и при всеки наполовина завършен вод, показван на add entry страницата.

```
// This is shared amongst all the pages
// It is the contents of the log itself, as a large string
public string LogText;
```

```
// This is also shared, only used by the log entry page
public string LogEntry = "";
```

Най-доброто място да се сложат тези променливи е класа App от файла App.xaml.cs. Този файл е контейнер за цялото приложение и е добро място за слагане на данни, които се споделят от всички страници на програмата. Това е класът, който съдържа методите, използвани за управлението на Fast Application Switching, така че те могат да запамятят и заредят данните на програмата, както ще видим по-късно.

Навигация на страницата в програмата Captain's Log

Приложението има две страници, между които потребителят ще навигира. Важно е страницата винаги да показва коректните данни. Знаем, че Silverlight страниците са без състояние (т.е. не могат да се използват, за да отбелязват данни) и така, когато се изобразява страница, е важно да се уверим, че съдържанието ѝ е правилно. Методът `OnNavigatedTo` може да се използва за целта:

```
protected override void OnNavigatedTo(System.Windows.Navigation.NavigationEventArgs e)
{
    // When we navigate to the page -
    //put the semi-completed log text in the display

    // Get a reference to the parent application
    App thisApp = App.Current as App;
    // Put the text onto the display
    LogTextBox.Text = thisApp.LogEntry;
}
```

Silverlight ще извика този метод за нас, когато потребителят навигира към страницата. Кодът в метода се обръща към класа `App` за това приложение и тогава зарежда текста `LogEntry` от класа и го изобразява в `textbox` на екрана. С това става сигурно, че всеки път, когато страница се показва, те ще съдържа най-новите данни.

Можем да правим още едно подобно нещо, когато потребителят излезе от страницата. В този случай искаме да се уверим, че всякакви промени на съдържанието в `textbox` на екрана се отразяват върху данните на програмата, с която работим.

```
protected override void OnNavigatedFrom (System.Windows.Navigation.NavigationEventArgs e)
{
    base.OnNavigatedFrom(e);
    // When we leave the page -
    // store the semi-completed log text in the application

    // Get a reference to the parent application
    App thisApp = App.Current as App;
    // Store the text in the application
    thisApp.LogEntry = LogTextBox.Text;
}
```

Този метод се извиква от Silverlight, когато потребителят навигира извън страницата. Запазва копие на текста от текстовото поле в данните на приложението. Като резултат това е обратното на предишния код.

Обърнете внимание, че тази техника не е важна само за съдържанието на `Fast Application Switching`; това е и начинът, по който се уверяваме, че когато потребителят се мести от едната на другата страница от нашето приложение, на страниците винаги да се показва вярна информация.

Запазване на данни при затваряне на Captain's Log

Знае, че когато потребителят натисне бутон `Назад` на върха на менюто в приложението (в случая `Add entry` страницата) системата Silverlight ще затвори приложението. Когато това стане, системата ще извика два метода:

1. OnNavigatedFrom се извиква на страницата, която се гледа в момента
2. Application_Closing позволява да се съхранят данни

Методът OnNavigatedFrom копира всякакви данни от настоящата страница в приложението, а Application_Closing трябва да ги съхрани във файловата система.

```
private void Application_Closing(object sender, ClosingEventArgs e)
{
    SaveToIsolatedStorage();
}
```

Приложението съдържа метод, наречен SaveToIsolated, който слага стойностите от lodText и logEntry във файловата система.

Зареждане на данни при отваряне на Captain's Log

Сега, когато кодът ще запаметява данни, когато приложението се затваря, трябва да добавим код, който ще зарежда програмата, когато тя стартира. Може да направим това чрез добавяне на метода Application_Launching:

```
private void Application_Launching (object sender, LaunchingEventArgs e)
{
    LoadFromIsolatedStorage();
}
```

Когато приложението стартира, ще взима данните от файловата система. В случая на Captain's Log тази версия на метода е добре.

Веднъж, след като е завършен метода, Silverlight страницата, на която е бил потребителят ще се зареди, след което ще извика метода OnNavigatedTo. Това ще вземе данните на приложението и ще ги покаже на потребителя.

Управление на Start бутона

Кодът, който написахме до сега е за това, как ще работи приложението на Windows PC. Когатото стартира, ще зареди данни от файл, а когато се затваря – ще ги върне обратно. Но за Windows Phone приложение има малко повече работа. Това е така, тъй като мобилният потребител натиска Start бутона, когато преограмата е пусната, което кара приложението да влезе в състояние на покой.

Когато Start бутона е натиснат Silverlight системата извиква метода OnNavigatedFrom, след което и метода Application_Deactivated. Можем да сложим кода тук, за да съхраним данни, така че потребителят да се върне към приложението.

```
private void Application_Deactivated (object sender, DeactivatedEventArgs e)
{
    SaveToIsolatedStorage();
    SaveToStateObject();
}
```

Този метод изглежда малко странен поради факта, че съхранява данните на две места, файловата система и временната памет. Въпреки това се оказва, че това е правилното нещо, което трябва да

се направи. Данните трябва да бъдат запазени във файловата система, защото потребителят може никога да не се върне към програмата, а той би бил много разстроен, ако е изгубил данните си. Те се съхраняват във временен обект, защото искаме копие на данните в паметта, което можем да използваме, за да възстановим състоянието на приложението, ако потребителят реактивира програмата.

Управление на процеса на връщане активността на програмата от страна на потребителя

Ако потребителят се върне към програмата, тя трябва да изглежда така сякаш никога не е била напускана. Работата на етода `Activated` е да презареди данните при необходимост. Нужно е да запомним, че има два начина да се активира приложението.

- Може да е активира от състояние на покой. В този случай всичките данни в програмата са в наличност и няма нужда да се възстановява каквото и да е, тъй като са в паметта.
- Може да е активира от състояние на плоча. В този случай никой от данните не е в наличност и програмата трябва да ги презареди. Това може да стане чрез използване на данни, съхранени в `State Object` при деактивирането.

Следният метод прави това:

```
private void Application_Activated (object sender, ActivatedEventArgs e)
{
    if (!e.IsApplicationInstancePreserved)
    {
        // We are coming back from a tombstone -
        // need to restore from the state
        object LoadFromStateObject();
    }
}
```

Проверява се флагът за истина, ако приложението се връща от състояние на покой. Ако флагът е лъжа (т.е. приложението се връща от състояние на плоча), цялата програма се зарежда от `State Object`.

Използване на статична памет

Програмата винаги може да използва файловата система, за да съхрани информация, дори когато е в състояние на покой. Въпреки това, не винаги е най-доброто място да се запазват малки количества данни, които ще са нужни за поддържането на потребителския интерфейс. За да се направи по-лесно и бързо управлението на такъв вид данни, системата `Windows Phone` осигурява начин програмата да съхрани малки количества информация в паметта. Телефонът нарича това статична памет и всяка програма има такъв тип памет. Тя се съхранява непокътната, ако програмата е в състояние на плоча.

Програмата Captain's Log може да използва това място да съхранява данни, когато е в замръзено състояние отколкото да ги връща в папките. В случай, че програмата е активирана наново, информацията може да се зареди от това място вместо да се използва папките.

```
private void Application_Deactivated (object sender, DeactivatedEventArgs e)
{
    SaveToIsolatedStorage();
    SaveToStateObject();
}
```

Методът отгоре се извиква, когато програмата е „замразена“. Извиква се метода saveToStateObject в App.xaml.cs, за да запази състоянието на програмата в паметта. Този метод saveToStateObject запазва цялото състояние на входния материал чрез употребата на помощния метод, който може да запаметява низове в статичния обект:

```
private void SaveToStateObject()
{
    SaveTextToStateObject("Log", LogText);
    SaveTextToStateObject("Entry", LogEntry);
}
```

Методът SaveTextToStateObject заема два параметъра, името на обекта и низа:

```
private void SaveTextToStateObject(string filename, string text)
{
    IDictionary<string, object> stateStore = PhoneApplicationService.Current.State;
    stateStore.Remove(filename);
    stateStore.Add(filename, text);
}
```

Директорията на обекта със статична памет за всяко приложение се запазва в свойството PhoneApplicationService.Current.State. За да се сдобие с достъп, програмата трябва да съдържа стандарта Microsoft.Phone.Shell:

```
using Microsoft.Phone.Shell;
```

Статичната памет работи като чист речник на обекти, индексирани от низ и така, за да запазва нашата програма данни, трябва да се създадат променливи на място от този тип, които след това да съхранят новия обект, като премахват първия, който съществува. Зареждането на статичната памет работи по същия начин:

```
private bool LoadTextFromStateObject(string filename, out string result)
{
    IDictionary<string, object> stateStore = PhoneApplicationService.Current.State;
    result = "";
    if (!stateStore.ContainsKey(filename)) return false; result =
    (string)stateStore[filename];
    return true;
}
```

Този код фактически е еднакъв с онзи, който писахме за зареждане на информация от папките с файлове на телефона. Единствената промяна е, че поради внедряването на статичната памет като чист речник, можем да използваме метода `ContainsKey`, за да видим, дали данните са запазени в приложението, когато то е замразено. Фактът, че тези методи изглеждат и функционират по подобен начин е всъщност добър дизайн.

Навигация на формите и бърз преход между приложенията

Системата Silverlight поддържа набор от най-използваните страници за едно приложение, заедно с “скрит стек” страници, използвани, за да се достигне тази страница. Когато потребителят се връща към приложението, правилната страница се изобразява и той ще може да извърши преход назад към предходните страници по правилния начин.

Освен това, трябва да запомним, че дори и прехода между страниците да се запазва, реалното съдържание на всяка страница се губи. За щастие, много е лесно да се уверим, че страницата съдържа правилните данни чрез слагане на код в метода `OnNavigatedTo` на всяка страница.

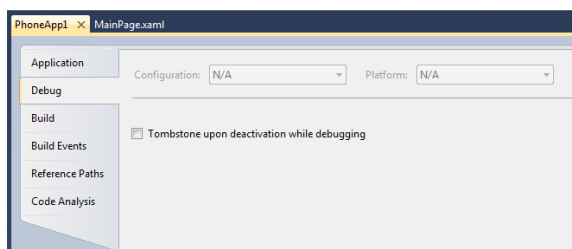
Бърз преход между приложенията и развитие

Можем да се пренесем от едно приложение към друго чрез Visual Studio. Ако приложението се рестартира, всякакви дебъкиращи сесии ще бъдат автоматично подновени в средата за развитие. Това работи, дори когато се отстраняват дефектите на приложението чрез Windows Phone устройството.

Можете да видите как работи това с натискането на Старт бутона в емулятора докато една от програмите Ви е нусната. Стартовата страница се отваря, но Visual Studio не спира отстраняването на дефектите. Ако тогава натиснете Back бутона, ще откриете, че прекрати изпълнението си, след като е била активирана наново.

Принуждаване програмата да се премахне от паметта

Когато тестваме програмата, трябва да се уверим, че тя работи правилно, дори когато е премахната от паметта и заредена наново. Знаем, че Windows Phone само ще премахне програми, когато няма памет. Вместо да ни принуждава да зареждаме памет и да премахваме програмите си, Visual Studio осигурява начин, по който да запитаме програмите да бъдат премахнати от паметта, когато влязат в състояние на покой.



В подпрозореца за отстраняване на дефекти на характеристиките на проекта има текстово поле, което, ако се провери, отправя запитване програмата да бъде премахната от паметта, когато се деактивира след отстраняване на грешките.

Бърз преход между приложенията и дизайн

Не е хубаво, че трябва да правим толкова сложни неща, за да може потребителят да остане с впечатлението, че приложението всъщност никога не спира. Но полагането на усилия, за да се уверим, че приложението работи правилно в това отношение, добавя много към потребителския опит. Струва си да се прекара известно време в работа с това, тъй като е добър начин да осъвършенстваме разбирането си за събитията и речниковото съхранение, заедно с други неща. Също така е интересно да се открие, че пътят към създаването на добър потребителски опит е не винаги гладък, особено ако се пишат програми в скована обстановка.

Игра на XNA също може да се свързва към събития, които ще бъдат генерирани, когато тя се спре или деактивира, а Вие можете да създавате игри, които запазват състоянието на играта и постигат отличен резултат, когато потребителят излезе от тях точно по същия начин.

Проблемът в Demo01 Complete Captains Log съдържа Windows Phone Silverlight log програма, която работи коректно в случай на деактивация.

9.3. Начини за стартиране и избиране

Следващата част в развитието на нашето приложение е свързана с това как да използваме начините за стартиране и избиране в телефона. Това е вграден режим на работа, осигурен, за да позволи на програмите да изпълняват стандартни действия. Устройството за стартиране е средство, чрез което програмата стартира дадена характеристика на телефона, която не връща никакво данни. Устройството за избиране е там, където чрез тази характеристика се избира да се извърши нещо. Приложението в такъв случай може да направи нещо с обекта, който бива върнат.

Устройствата за стартиране и избиране се възползват от деактивирането, така че когато програмата активира стартирането или избирането, тя автоматично деактивира. Когато стартовото или избиращото устройство приключва работата си, програмта се активира наново.

Използване на стартовото устройство

Всичко това са устройства за стартиране, заедно с кратко обяснение на това, какво прави всяко едно:

PhoneCallTask	Стартира телефонното приложение с определен телефонен номер и избрано име. Обърнете внимание, че програмата не може сама да осъществи обаждането, потребителят трябва да се намеси.
EmailComposeTask	Нашата програма може да задава характеристики на имейла и тогава да стартира задачата, за да позволи на потребителя да изпрати съобщение.
SmsComposeTask	Стартира Message и изобразява новото SMS съобщение.

	Забележете, че съобщението не се изпраща автоматично.
SearchTask	Стартира Search приложението, използвайки предоставена от Вас заявка.
WebBrowserTask	Стартира уеб браузъра и изобразява URL, който предоставяте.
MediaPlayerLauncher	Стартира приложението Media Player и пуска даден медия файл.
MarketplaceReviewTask	Вашата програма може да стартира процеса на преглед на приложението.
MarketplaceDetailTask	Програмата може да покаже детайли относно дадено приложение на пазара.
MarketplaceHubTask	Стартиране на клиент от Windows Phone Marketplace.
MarketplaceSearchTask	Програмата може да търси в Windows Phone Marketplace за определено приложение, след което да покаже резултатите от търсенето.
BingMapsDirectionsTask	Програмата може да създаде набор от Bing Maps директории между отделни дестинации при шофиране или ходене.
BingMapsTask	Програмата може да изобрази карта, поставена на настоящото местоположение на телефона или на специфично място.
ConnectionSettingsTask	Програмата може да използва тази задача, за да позволи на потребителя да промени свойствата на свързване на телефона.
ShareStatusTask	Вашата програма може да попълни статус-съобщение, което потребителят може да изпрати към социална мрежа по избор.
ShareLinkTask	Програмата може да задава url, който след това да бъде споделен от потребителя чрез социална мрежа по избор.

Всяка от тези задачи може да бъде стартирана от приложението и когато бъде завършена, приложението отново ще стане активно. Разбира се, няма гаранция, че то ще възвърне контрол в бъдеще. Потребителят може вместо това да се захване с много други неща отколкото да се върне към програмата.

Добавяне на имейл характеристика към JotPad



Може да решим да добавим Mail характеристика към програмата Jotpad. Чрез натискане на бутона Mail потребителят може да изпрати съдържанието на jotpad-а като имейл. Изображението отгоре показва как това работи.

```
private void mailButton_Click(object sender, RoutedEventArgs e)
{
    sendMail("From JotPad", jotTextBox.Text);
}
```

Когато се кликне върху имейл бутона, устройството, което управлява събитията извежда текста от текстовото поле и извиква метода sendMail, за да го изпрати. Също така се добавя съобщението From. Методът sendMail е много прост:

```
private void sendMail(string subject, string body)
{
    EmailComposeTask email = new EmailComposeTask();

    email.Body = body; email.Subject = subject;
    email.Show();
}
```

Създава се отделен случай на класа EmailComposeTask, след което се задават притурките Body и Show към осигурените параметри. Тогава се извиква метода Show на зададения пример. В този момент приложението JotPad е „замразено“, за да преотстъпи мястото си на приложението за имейл. Когато потребителят завърши изпращането на имейли, програмата JotPad отново ще заеме мястото си.

За да може програмата да се възползва от тези задачи, в нея трябва да бъде включен стандарта:

Проблемът в Demo02 Email JotPad съдържа Windows Phone Silverlight jotter pad, който може да изпраща малки бележки като имейл. Обърнете внимание, че това няма да работи както трябва при емулятора на Windows Phone, тъй като той няма прикачен клиентски имейл. Въпреки това, може да се види деактивирания режим на работа докато програмата се опитва да изпрати имейл и показва предупреждение вместо това.

Използване на устройство за избиране

	Стартира Camera приложението, за да може потребителят
--	---

CameraCaptureTask	да фотографира.
EmailAddressChooserTask	Стартира Contacts приложението и позволява на потребителя да избере имейл адрес от списъка с контакти.
PhoneNumberChooserTask	Стартира Contacts приложението и позволява на потребителя да избере телефонен номер от списъка с контакти.
PhotoChooserTask	Стартира Photo Picker приложението, където потребителят може да избере снимка.
SaveEmailAddressTask	Запазва предоставените имейл адреси в списъка с контакти; като резултат връща дали процесът е бил завършен успешно или не.
SavePhoneNumberTask	Запазва предоставения телефонен номер. Като резултат връща дали процесът е бил завършен успешно или не.
AddressChooserTask	Програмата може да изисква от потребителя да избере адрес от контактите в телефона.
GameInviteTask	Програмата може да покани друг мобилен потребител за игрална сесия с повече играчи.
SaveContactTask	Програмата може да подвключи входни данни от контакт и да позволи на потребителя да запази в списъка с абонати на телефона. Като резултат връща дали процесът е бил завършен успешно или не.
SaveRingtoneTask	Програмата може да предостави на потребителя опцията за съхрани аудио файл в подходящия формат като мелодия за звънене за телефона.

Устройството за избор работи по същия начин като това за стартиране с изключение на една разлика. Този тип устройства мога да връщат резултат към приложението, което ги е предизвикало. Това се прави по вече познат начин, нашето приложение се свързва с метод за управление към приключилото събитие, което е предоставено то устройството за избиране.

Изобразяване на картинка



Можем да изучим как това работи чрез създаване на приложение за изобразяване на обикновена картинка, което ще даде възможност на потребителя да избере изображение от медийната библиотeka на телефона, след което да го изобрази на екран. За целта устройството, което управлява избора трябва да бъде създадено в конструктора за MainPage на нашето приложение:

```
PhotoChooserTask photoChooser;
```

```
// Constructor public MainPage()  
{  
    InitializeComponent();  
  
    photoChooser = new PhotoChooserTask();  
    photoChooser.Completed += new EventHandler<PhotoResult>(photoChooser_Completed);  
}
```

Този конструктор създава новата стойност PhotoChooser и управляващо устройство към завършеното събитие, което се генерира. Устройството, използващо приключеното събитие използва резултата, върнат от устройството за избор и го изобразява като картинка.

```
void photoChooser_Completed(object sender, PhotoResult e)  
{  
    if (e.TaskResult == TaskResult.OK)  
    {  
        selectedImage.Source = new BitmapImage(new Uri(e.OriginalFileName));  
    }  
}
```

Този метод задава източника на изображението да бъде BitmapImage, който е създаден, използвайки адреса на резултата на снимката. Забележете, че свойството taskResult на резултата позволява на програмта на определи дали потребителят ще избере да върне нещо или не.

Последната връзка, която трябва да предоставим и устройство за поддържане на събитията за бутона Load, който стартира процеса.

```
private void loadButton_Click(object sender, RoutedEventArgs e)  
{  
    photoChooser.Show();  
}
```

Този метод само извиква метода Show от устройството за избор, който да започне зареждането на процеса на избиране.

Проблемът в Demo03 PictureDisplay съдържа Windows Phone програма, която може да се използва, за да се избере изображение от медийното хранилище. Тази програма няма да работи на Windows Phone емулятора, който няма съхранени изображения. Също така няма да тръгне и на

Windows Phone устройството, ако се използва Zune софтуер, за да се свърже телефона към Visual Studio. Това е така, тъй като когато телефонът се свързва към Zune, съдържанието на медията е недостъпно. В случай, че връзката на телефона бива прекъсната, след което приложението се стартира отново, обаче, то ще заработи правилно.

9.4. Обработване на данни на заден план

От опитите знаем, че Windows Phone ще позволи само на една програма да бъде активна в дадено време на телефона. Въпреки това, има случаи, когато би било много полезно за някоя част от приложението да бъде в активност, дори когато не е видима за потребителя. Операционната система на Windows Phone осигурява това под формата на „фоновая обработка“ където разработчикът може да създаде код, който да работи, когато приложението не е активно.

Обърнете внимание, че това не е същото като да се позволи на дадена програма да действа на заден план. Фоновите компоненти са напълно отделени, нямат потребителски интерфейс, а стриктни ограничения относно какво могат да извършват и какво – не. Те се зареждат единствено в специфични ситуации, така че операционната система на телефона да се увери, че те ще имат въздействие върху потребителския опит. Операционната система на телефона ще затвори заачите, които са на заден план, ако се пести по отношение на паметта, батерията или мрежовата свързаност. Потребителят също може да извади от действие някои задачи, които са на заден план за определено приложение.

По тази причина фоновата задача трябва да се счита като част от „глазурата“ на приложението. Не би трябвало да е основна част от принципа на работа, тъй като съществува вероятността никога да не се пусне да работи. Като пример може да се даде приложение за продажби, което изобразява съобщение, когато желаният артикул е в наличност, или приложение за новини, показващо последните заглавия на началния екран.

Задачи на заден план и периодично планирани задачи

Има няколко различни вида процеси на заден план; ще започнем с разглеждането на две типа с общо преданзначение – периодични и интензивни по отношение на ресурс. Тези задачи могат да се създадат по един и същ начин, но ситуациите, при които кодът ще работи в тях са малко по-различни.

- Периодичните процеси могат да бъдат заредени за кратък период от време (до 15сек.) на редовни интервали – обикновено на всеки половин час.
- Интензивните задачи по отношение на ресурса могат да бъдат заредени за повече време (до 10мин.), когато телефонът е заключен (т.е. когато потребителят не го използва за нищо), свързан към източник на мощност и има на разположение висока мрежова свързаност.

Периодичната опция е полезна за обновяване на дисплей за времето, статусна информация или засичане на местоположението. Интензивната възможност е от полза за качване/сваляне на обновявания на базата данни, извършване на обработка на данни или тяхното свиване.

Агенти и задължения

Когато се говори за такъв тип посредници, терминологията трябва да се разграничи. Задачата представлява контейнер, който се управлява от операционната система и бива зареден, когато условията го позволяват. Агентът е същинският код, който извършва работата. Когато се създава този агент, се създава един артикул, който ще се зарежда или в периодично, или в интензивно състояние по отношение на ресурса.

Локализиране при Captain's Log

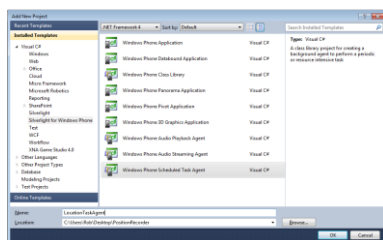
За да се разбере как се използва агента в приложението, трябва да се върнем към програмата Captain's Log.



Програмата може да се осъвършенства чрез добавяне на особеност за засичане. Когато тя се прикачи, мястото, на което се намира телефона ще бъде запазено, използвайки периодичен принцип, за да се чете редовно информацията за позиционирането от GPS устройството на телефона, добавя се времева марка и това се запазва във файлът за входни данни. Идеята е, че фоновата задача ще запази информацията за позиционирането на мобилния апарат, когато програмата Captain's Log не е активна.

Добавяне на задача на заден план в Captain's Log

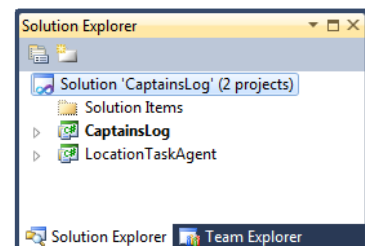
Фонова задача се добавя като допълнителен проект в Windows Phone проблема.



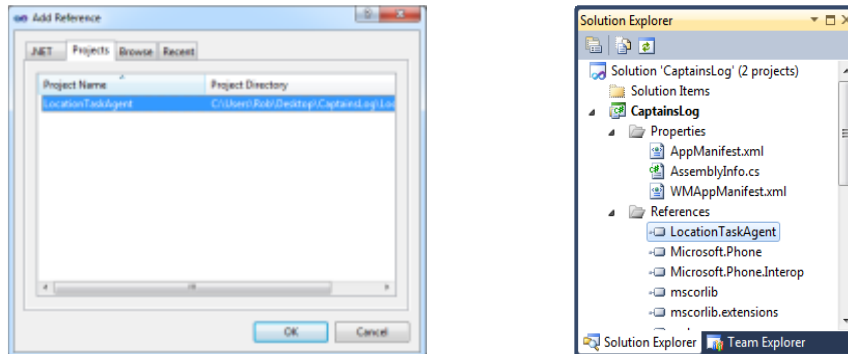
Обърнете внимание, че се добавя като Scheduled Task артикул, който може да бъде периодичен, интензивен или и двата едновременно. Веднъж, след като се добави, задачата, стояща на заден план ще покаже друг проект към създаването на приложението.

Даденото приложение може да има прикачен само Scheduled Task. Когато то се създава, проектът, който съдържа кода за списъчните задачи се добавя към другите части на приложението.

Основната част от програмата Captain's Log ще създаде задачи в списък, използвайки кода от проекта-агент. За да проработи това, е необходимо главното



приложение да е способно да използва действащия код. Във Visual Studio това означава добавяне на изходното устройство на проекта LocationTaskAgent към връзките, използвани от проекта на преден план. Вече видяхме как това става, когато се занимавахме със задачи, съставени от множество проекти, а това е друг пример на тази техника в действие.



Ето как проектът бива добавен към справките, а отдясно може да се види как главният код е добавен към същите тези допълнителни връзки на проекта.

Действащ код към приложение на заден план (Background Task Agent Code)

При създаването на фоновата задача Visual Studio предоставя празен шаблон, в който може да се напише същинския код.

```
namespace LocationTaskAgent
{
    public class ScheduledAgent : ScheduledTaskAgent
    {
        protected override void OnInvoke(ScheduledTask task)
        {
            //TODO: Add code to perform your task in background
            NotifyComplete();
        }
    }
}
```

Пише се кодът, който изпълнява заданието на метода OnInvoke. Когато този метод приключи, трябва да се извика NotifyComplete, за да спре протичането на процеса. Обърнете внимание, че не се контролира кога и дали изобщо този код ще се изпълни. Когато Agent бива регистриран чрез телефона ще се изисква периодичния, интензивния сценарий или и двата заедно. В случай, че потребителят е разрешил фоновото задание в точното време, програмата ще се зареди.

Основният код може да определи типа задача, която е пусната, като провери вида на ScheduledTask, предаден по OnInvoke:

```
if (task is PeriodicTask)
{
    // Is running as a periodic task
}
```

Зареждане на данни от приложението в агента на фоновото задание

В случая на програмата Captain's Log заданието трябва да зареди низ от файловата система, да добави информация за локализацията към него, след което да съхрани въведения низ отново. Приложението и задната задача споделят едно и също пространство за файловата система. Забележете, че не става въпрос за конфликт на достъп към този ресурс, тъй като заданието на заден план няма да се задейства, ако приложението е вече активно.

```
protected override void OnInvoke(ScheduledTask task)
{
    string message = "";

    string logString = "";

    if (loadTextFromIsolatedStorage("Log", out logString))
    {
        message = "Loaded ";
    }
    else
    {
        message = "Initialised ";
    }

    //When we get here we have a log string - add location to it

    NotifyComplete();
}
```

Тази част от основния код на фоновата задача ще извика метод, който да зареди въведения текст от файловата система. Това е същият код, който се използва за съхранение и зареждане на данни, когато се зарежда програмата на преден план. Ако входа се окаже празен (т.е. агентът е зареден преди приложението да е съхранило нещо във файловата система), тогава низът в променливата на съобщението се задава както трябва.

Достъп до информацията за местоположение в агента на фоновото задание

Сега, когато входният низ е на разположение, следва да се открие позицията на телефона и да се добави към входния текст.

```
protected override void OnInvoke(ScheduledTask task)
{
    //When we get here we have a log string - add location to it

    GeoCoordinateWatcher watcher = new GeoCoordinateWatcher();
    GeoPosition<GeoCoordinate> position = null;

    watcher.StatusChanged += new EventHandler<GeoPositionStatusChangedEventArgs>
    (delegate(object sender, GeoPositionStatusChangedEventArgs e)
    {
        if (e.Status == GeoPositionStatus.Ready) {
            position = watcher.Position;
            GPSDoneFlag.Set();
        }
    })
}
```



```

});

GPSDoneFlag.Reset();

watcher.Start();

// Wait for 10 seconds to give time to get position
GPSDoneFlag.WaitOne(10000);

// When we get here we the position or we timed out
}

```

Тази част от фоновия код създава нов GeoCoordinateWatcher и го стартира. GeoCoordinateWatcher показва съобщение StatusChanged, към което можем да се свържем. Това събитие протича, когато GPS-четеца промени своето състояние. Кодът отгоре се свързва с управляващото устройство на този метод, който търси промяна в състоянието за готовност. Когато GPS достигне това състояние, се прочита стойността на позицията и се задава указател, който позволява чакащия процес да продължи. Същата техника беше използвана, когато програмите чакаха за заявление относно TCP мрежата в Глава 7. Това позволяваше да се напишат програми, които могат да изчакат приключването на системното действие. Ако GPS позицията е на разположение, стойността се копира в променливата position.

Обърнете внимание, че основният код на фоновото задание при GPS е малко по-различен. Тъй като би било непозотворно да се стартира GPS система за всеки процес на заден план, системата вместо това прави копие на информацията за локализирането на чести интервали и я използва за агентите, които се имат нужда от нея. Това означава, че вместо да чака дадено събитие да заяви, че информацията е готова (както би трябвало да се използва GPS), тази програма може да прочете тези данни моментално, за да създаде входен низ.

Обърнете внимание, че кодът отгоре работи добре на Windows Phone устройството, но може да възникнат проблеми при емулятора.

Създаване на входно съобщение

Сега, след като имаме информацията за позиционирането, трябва да създадем входно съобщение и да го съхраним във файловата система. В момента, в който данните се запамятят като обикновена двойка от стойности, ще бъде лесно да се съхранят като XML.

```

protected override void OnInvoke(ScheduledTask task)
{
    // When we get here we got the position or we timed out
    string positionString ;

    if (position != null)
        // If we get here we got the position or we timed out
        positionString = position.Location.ToString();
    else
        positionString = "Not Known";

    DateTime timeStamp = DateTime.Now;

```

```

        string timeStampString = timeStamp.ToShortDateString() + " " +
        timeStamp.ToShortTimeString() + System.Environment.NewLine;

        logString = logString + timeStampString + positionString;

        saveTextToIsolatedStorage("Log", logString);

        // When we get here we can display a toast message
    }

```

Кодът отгоре създава текстово съобщение, което съдържа позиционна стойност "Not Known" в случай, че не е в наличност. Тогава това се добавя към края на входния низ и го запазва обратно във файловата система.

Показване на „моментално“ съобщение от основния фонов код

Веднъж, след като приложението бива запазено, локализиращото устройството за входни данни може да изведе ползите от извършеното действие. Фоновият задание няма възможност да се свърже директно с потребителя, но може да изобрази изкачащи съобщения или да промени изгледа на плочите от стартовото меню.

Снимката показва елементарен рорип, създаден чрез кода отгоре. Телефонът ще пусне звуков предупредителен сигнал, а потребителят може да стартира приложението чрез натискане върху това съобщение. Кодът, който изобразява изкачащите съобщения е много лесен. Веднъж, след като създадем пример, само трябва да се зададе заглавието и съобщението, което да се изобрази, след което да се извика Show метода.



```

protected override void OnInvoke(ScheduledTask task)
{
    // When we get here we can display a toast message

    ShellToast toast = new ShellToast();
    toast.Title = "Captain's Log";
    toast.Content = message + positionString; toast.Show();

    // Now we need to schedule the task for debugging
}

```

Този тип съобщение може да се измени, като се зададе uri, което идентифицира определена страница в приложението, към която трябва се отиде, ако потребителят я избере. Това ще подпомогне дадено сигнализиращо съобщение, стоящо на заден план да отведе потребителя към част от програмата, която е засегната.

Отстраняване на дефекти от главния фонов код

Може да си мислите, че отстраняването на дефекти от фоновия главен код е трудно, но реално не е така. Visual Studio позволява използването на опорни точни и последователни стъпки в кода по същия начин както това би се случило за друга част от програмата.

Когато отстраняваме грешки, не искаме да чакаме подходящия момент, за да се стартира фоновата задача. Поради тази причина програмата осигурява метод, който може да зареди задностоящия процес и да го тества. Условното компилиране, от своя страна, засича обаждане от метода, което ще задейства фоновия основен код при отстраняването на грешки.

```
#define DEBUG_AGENT
```

Чрез дефиниране на този символ компилаторът може да преработи кода, който определя кога да се заредни главния код.

```
protected override void OnInvoke(ScheduledTask task)
{
    // Now we need to schedule the task for debugging
#ifdef DEBUG_AGENT
    ScheduledActionService.LaunchForTest(task.Name, TimeSpan.FromSeconds(60));
#endif

    NotifyComplete();
}
```

Кодът отгоре може ще накара агента, стоящ на заден план да се изпълнява на всеки 60сек. Обърнете внимание, че не се изисква повтарящо се изпълнение на главния код, тъй като се налага зареждане на всеки 60сек.

Последното приложение на NotifyComplete и да установи как агента по задание уведомява Windows Phone системата, която се изпълнява в момента, че обработката е приключила.

Управление на заданието на фоновия агент

След като е създаден кода на фоновия изпълнител, можем да се захванем с това да го накараме да проработи. Операционната система на Windows Phone съдържа клас, наречен ScheduledActionService, който управлява всички подредени задания. Можем да извикаме методи от този клас, които да търсят активни задачи, за да ги спрем или стартираме. Реализиращата се в момента задача има определено име, по което да я търсим. Разбира се, можем само да контролираме заданията, които се изпълняват от нашето приложение; не ни е позволено да търсим задачи на други приложения.

```
PeriodicTask periodicTask = null;
string periodicTaskName = "CaptainTracker";

private void StartTracking()
{
    periodicTask = ScheduledActionService.Find(periodicTaskName) as PeriodicTask;

    // If the task already exists and the IsEnabled property
    // is false, background agents have been disabled by the user
    if (periodicTask != null && !periodicTask.IsEnabled) {
        MessageBox.Show( "Background agents disabled by the user.");
        return;
    }
}
```

```

// If the task already exists and background agents are enabled
// for the application, we must remove the task and then add it
// again to update the schedule
if (periodicTask != null && periodicTask.IsEnabled) {
    RemoveAgent(periodicTaskName);
}

periodicTask = new PeriodicTask(periodicTaskName);

// The description is required for periodic agents. This is the
// string that the user will see in the background services
// Settings page on the device.
periodicTask.Description = "Log Tracker";
ScheduledActionService.Add(periodicTask);

// If debugging is enabled, use LaunchForTest to launch the
// agent in one minute.
#if(DEBUG_AGENT)
    ScheduledActionService.LaunchForTest(periodicTaskName, TimeSpan.FromSeconds(60));
#endif
}

```

Горепосоченият код ще стартира агент като периодично задание, след което ще го накара да се изпълнява на всеки 60сек., в случай че се отстраняват дефекти. Обърнете внимание, че задачата трябва да има определено име и че потребителят може да прекъсне агентите на заден план за определено приложение, ако пожелае. Можем да стартираме интензивно задание по отношение на ресурса, като създадем примерен `ResourceIntensiveTask` вместо `PeriodicTask`. Ако създаваме задача, която се изпълнява и в двата случая, можем да направим `ScheduledTask`.



Проблемът от Demo04 Location Logger съдържа Windows Phone програма, която изпълнява въвеждане на местоположение, използвайки фоново задание. Когато програмата започва да се изпълнява, тя проверява за съществуващи засичащи агенти и позволява на потребителя да спре или стартира засичането. Използва се дебъгиращото състояние "LaunchForTest" за обновяване на местонахождението на всяка минута.

Пренос на файлови задачи

Възможно е да се създадат задачи на заден план, които да прехвърлят файлове към и от файловата система на приложението. Трансферът ще се осъществи, когато приложението не е действащо, а когато то се стартира наново, ще може да покаже до къде е стигнало зареждането на данни и да изобрази състоянието им. Файловете могат да бъдат извикани от HTTP или HTTPS домейни, но за момента FTP не се поддържа от системата.

Принцип на трансфера на файлове от заден план

Съществуват няколко правила, които управляват строго количеството данни, което може да бъде пренесено от приложението чрез агента за пренос на файлове:

- Максимален размер при качване: 5 Mb

- Максимален размер на файла при сваляне през клетъчни (при мобилни устройства) данни: 20 Mb
- Максимален размер на файла при сваляне през WiFi: 100 Mb

Даден пренос може да промени тези стойности чрез `TransferPreferences`, ако се изискват различни настройки.

Създаване на заден трансфер

Услугите при задния трансфер са дефинирани в стандарта за този случай:

```
using Microsoft.Phone.BackgroundTransfer;
```

Прехвърлянето започва чрез създаване на обект `BackgroundTransferRequest`, след което той се добавя към онези, които се управляват от `BackgroundTransferService`. На трансфера се дава име, така че следващият път, когато приложението го зареди, да може да го локализира и да открие процеса на пренос.

Също така е възможно да се свържат събития, породени от трансфера, така че пуснатото в момента приложение да изобрази информация, до къде е стигнал процесът.

Забележете, че от приложението зависи управлението на активни трансфери и премехването на онези, които вече са приключили.



Проблемът в `Demo05 File Transfer` съдържа Windows Phone програма, която ще извика файлово изображение от интернет и ще го изобрази. Програмата също така показва информация за състоянието докато файлът се зарежда.

Таблични известия

Докато приложението не може да създаде календарни бележки, то има възможността да направи таблични известия. Те се запомнят, когато телефонът е изключен и могат да предизвикат иккачащи съобщения или да променят Live Tile. Обърнете внимание, че този вид известия не могат да стартират приложението, но в случай, че потребителят ги натисне, то приложението ще стартира. Известията могат да се програмират да се покажат веднъж или на определени интервали.

Напомнящите бележки са създадени по подобен начин на файловите трансфери и могат да бъдат локализирани или управлявани от приложението чрез указаните им имена.



Проблемът от Demo06 съдържа Windows Phone програма, която ще отбоява от 1 до 5 минути.

Агент за аудио плейбек (Audio Playback Agent)

Приложението може също така да създаде аудио плейбек агент. Той се обединява с нормалните контролери на плейбека и позволява пускането на музика, когато приложението не е активно. Такъв проект може да бъде създаден и добавен към Windows Phone програмата по същия начин като фонова задача.

Плейбек агента може да пуска музика от файловата система на приложението, но няма достъп до медийната библиотека на телефона. Въпреки това, може да се използва да възпроизведе файлове от мрежовото медийно пространство.

Какво научихме

1. Windows Phone приложението бива представено от две иконки на устройството. Едната се използва, за да пресъздаде приложението в списъка Application на телефона, а другата, по-голямата, иконка се използва, ако приложението се прикачи към Start менюто на мобилния апарат. Качествените иконки да от значение, тъй като се използват в Windows Phone Marketplace, за да рекламират приложението.
2. Silverlight приложенията също имат “splash screen” изображения, показвани, когато приложението стартира. Началните екрани, зададени по подразбиране са предоставени като част от новия Silverlight проект, но е по-добре, ако разработчиците на приложения сами ги изработят. Проектите на XNA нямат вграден механизъм за начален дисплей, но разработчиците на игри трябва да създадат такъв, ако на съдържанието е необходимо повече от няколко секунди, за да се зареди.
3. Операционната система на Windows Phone осигурява модел за извеждане на задачите една по една за всяко приложение. Дадено приложение може да бъде деактивирано по всяко време и заменено с друго. Бутоните Start и Back на мобилното устройство позволяват на потребителя да спре активното в момента приложение и да стартира друго, натискайки Start, а след това да се върне към деактивираното приложение, като натисне Back. Освен това, той може да задържи за по-дълго Back бутона, за да се покаже списък със спрени приложения, от които да избира.

4. Деактивирано приложение може да остане в паметта (в състояние на покой) или не („да бъде замръзено“). Когато се активира отново от състояние на покой, то има цялата памет в наличност. Във втория случай, обаче, данните трябва да бъдат заредени наново.
5. Съществуват четири събития, които се получават в приложението докато то е пуснато на телефона. Това са “Launched”(изпълнено, когато приложението стартира), “Closed”(когато то се затваря), “Deactivated” (при деактивиране) и “Activated” (активиране от състояние на покой или сън).
6. Windows Phone предоставя механизъм за съхранение на паметта, който може да се използва, за да запазва активни данни, когато приложението се деактивира. Тази информация може да се прочете отново, ако приложението се стартира.
7. Програмите могат да използват Launchers и Choosers, за да комуникират с подсистемите на телефона. Първото устройство стартира външно приложение (напр. изпращане на имейл), второто – зарежда външно устройство, което връща резултат (напр. избиране на картинка от медийното пространство). Програмата може да бъде деактивирана докато се зарежда външното приложение.
8. Дадено приложение може да създаде фонові агенти, които да бъдат активни, когато то не се използва. Те могат да бъдат периодични (заредени за кратко време на всеки 30мин.) или интензивни по отношение на ресурса (стартирани, когато телефонът не се използва, външно захранвани и с добра мрежова свързаност). Агентът представлява отделен проект, който се добавя към общата реализация на приложението.
9. Приложението може да създаде прехвърляна на файлове, стоящи на заден план, когато то не е активно.
10. Приложението може да създаде таблични известия, които да се зареждат на определени интервали. Те ще се появяват, дори когато приложението не е в употреба.

Глава 10. Windows Phone Marketplace

Ние вече знаем достатъчно, за да се извършват пълни приложения и игри, които ще работят правилно в рамките на Phone околната среда и да използват вградените характеристики на системата на телефона. Ние сега ще разгледаме как можем да се уверим, че нашите програми са готови за пазара, а след това ние ще разгледаме процедурата по подаване и в крайна сметка ние ще обмислим някои неща, които можем да направим, за да направим нашите програми се открояват навън, и се надяваме, увеличаване на продажбите.

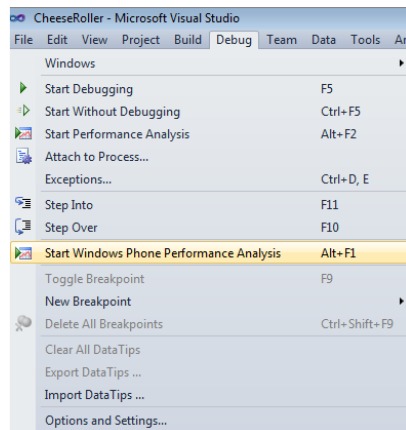
10.1 Изготвяне на Заявление за продажба

В този раздел ние ще се съсредоточим върху нещата, които можем да направим, за да се подготви заявление за продажба.

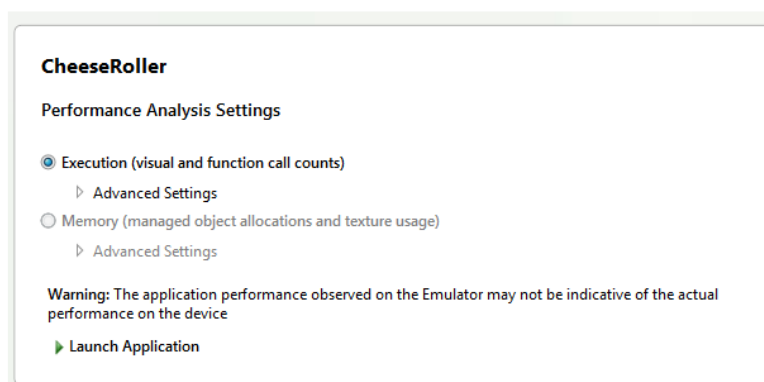
Анализ на ефективността

Съвременните компютри са толкова мощни, че те често са в състояние да компенсират зле написан софтуер. Това е често рентабилно да си купите по-бърз компютър, а не се оптимизира дадена програма. Въпреки това, когато се разработват за Windows Phone ще е нужно да се притеснявате за изпълнение много повече, отколкото сме направили за десктоп приложение. Това е така, защото силата на процесора е ограничен и самото устройство е по-малко памет от компютър.

Windows Phone SDK предоставя набор от инструменти за анализ на производителността, които можем да използваме, за да разберете какво ни програма се прави и как ще го изпълнява. Те могат да дадат представа за натоварването на нашите програми пускат на процесора, скоростта на обновяване на екрана (колко бързо нашата програма е актуализирането на екрана) и използването на паметта. Ние дори може да разберете много ниско ниво на детайлност за кои конкретни методи се наричат в нашата програма. Това е много полезно, тъй като това означава, че ние може да се съсредоточи нашата оптимизация на онези части от програмата, които консумират най-много от времето на процесора.



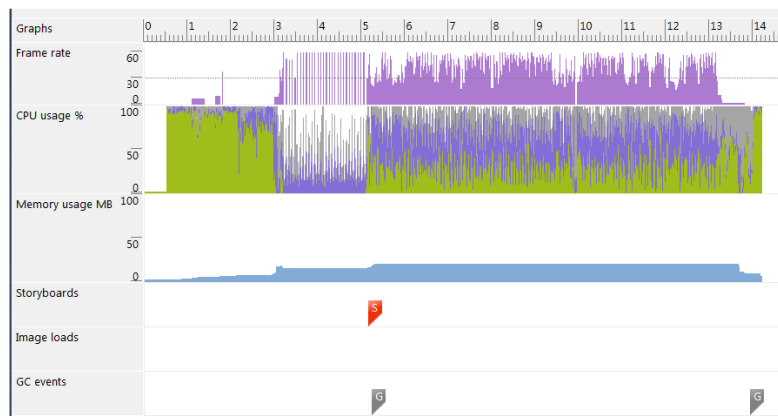
Инструментите за анализ са започнати от менюто Debug в Visual Studio. Когато те се отворят те покаже екрана за конфигуриране:



Има няколко различни опции, които можем да изберем, в зависимост от това, дали ние сме заинтересовани в работата или използването на паметта. Когато сте избрали нашите опции на

програмата ще се проведе в рамките на анализа на изпълнението също. Програмата може да работи вътре в емулятора или хардуерно устройство. И в двата случая системата произвежда в лог файл на сесията, която се съхранява като част от Visual Studio проект. Това го прави лесно да се върнем към предишните тестове на изпълнението и да се определи ефектът от промените, които ние правим.

Когато програмата е спряна в края на анализ на изпълнението на сесията Visual Studio ще напише лог файл и след това да ни позволи да проучи резултатите.

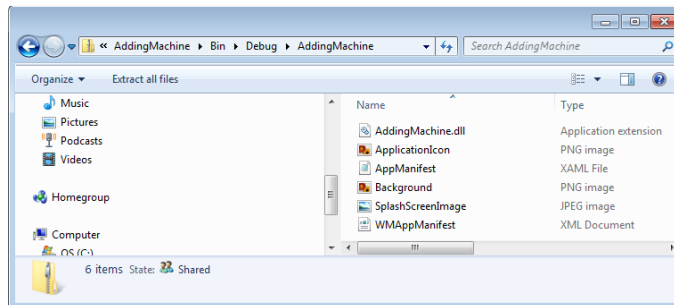


Това е лог файла за Silverlight приложение, което извършва обработка на изображения. Ние можем да видим, че за първите няколко секунди програмата се Започва устройството на камерата. След това той започва да обработва видео потока и увеличава натоварването на процесора. Програмата показва също така, честотата на кадрите, че заявлението е постигано. Самата програма не използва много памет, въпреки че тя разпределя някои видео буфери, след като камерата е инициализиран.

Това е мнението на много високо ниво на информацията, че е възможно да се пробие и да вземе много по-подробен оглед. Неца, за да гледате за са резки скокове в използването на процесора, и постоянното нарастване на използване на паметта или голям брой на Garbage Collection (GC) събития. Това може би показва, че такова заявление не е много разумно с начина, по който тя използва телефонните ресурси.

Създаване на XAP Файл за Приложение Разпределение

Ние вече видяхме, че молбата е описан от Visual Studio разтвор. В глава 3 "Работещи Windows Phone приложения" видяхме, че цялото приложение се съхранява в един XAP файл. Това е файл, който се прехвърля в Phone устройство Windows, когато програмата се осъществяват. Също така е файла, който се подава в Windows Phone Marketplace, когато искаме да продаваме нашата молба.



Този файл съдържа всички възли и средствата по програмата, заедно с явни файлове, които описват заявлението. Файлът WMAAppManifest.xml изброява възможностите на телефона, която използва приложението и също така определя жанра на заявлението. Това определя къде по телефона се съхранява програмата. Програми с жанра на играта се съхраняват в игра главината, приложения се съхраняват в главината приложения.

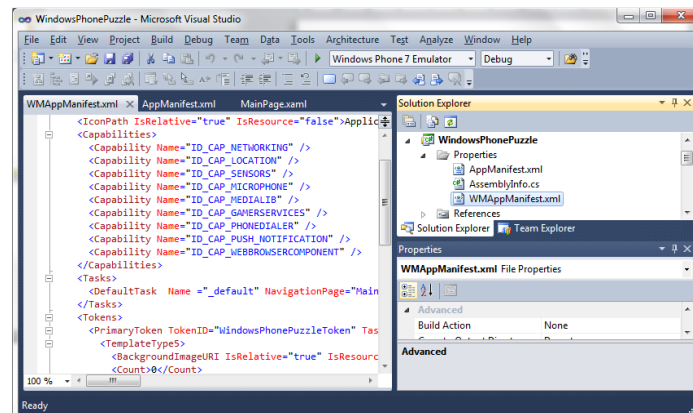
```
<?xml version="1.0" encoding="utf-8"?>

<Deployment xmlns="http://schemas.microsoft.com/windowsphone/2009/deployment"
AppPlatformVersion="7.0">
  <App xmlns="" ProductID="{b92ce770-1b22-4d52-998e-113c2784067a}" Title="PictureDisplay"
RuntimeType="Silverlight" Version="1.0.0.0" Genre="apps.normal"
Author="PictureDisplay author" Description="Sample description" Publisher="PictureDisplay">
    <IconPath IsRelative="true" IsResource="false">ApplicationIcon.png</IconPath>
    <Capabilities>
      <Capability Name="ID_CAP_GAMERSERVICES"/>
      <Capability Name="ID_CAP_IDENTITY_DEVICE"/>
      <Capability Name="ID_CAP_IDENTITY_USER"/>
      <Capability Name="ID_CAP_LOCATION"/>
      <Capability Name="ID_CAP_MEDIALIB"/>
      <Capability Name="ID_CAP_MICROPHONE"/>
      <Capability Name="ID_CAP_NETWORKING"/>
      <Capability Name="ID_CAP_PHONEDIALER"/>
      <Capability Name="ID_CAP_PUSH_NOTIFICATION"/>
      <Capability Name="ID_CAP_SENSORS"/>
      <Capability Name="ID_CAP_WEBBROWSERCOMPONENT"/>
    </Capabilities>
    <Tasks>
      <DefaultTask Name="_default" NavigationPage="MainPage.xaml"/>
    </Tasks>
    <Tokens>
      <PrimaryToken TokenID="PictureDisplayToken" TaskName="_default">
        <TemplateType5>
          <BackgroundImageURI IsRelative="true"
IsResource="false">Background.png</BackgroundImageURI>
          <Count>0</Count>
          <Title>PictureDisplay</Title>
        </TemplateType5>
      </PrimaryToken>
    </Tokens>
  </App>
</Deployment>
```

Над можете да видите WMAAppManifest.xml файла по подразбиране, който е създаден за прилагането PictureDisplay. Тази версия на файла показва, че приложението ще използва всички възможности на телефона. Всъщност единствената функция, която той използва, е ID_CAP_MEDIALIB един. Когато е подадено заявление за Marketplace съдържанието на този файл се проверява срещу призивите, че заявлението прави. Ако приложението използва ресурси, които не са посочени в този файл, той ще се провали валидиране.

Когато клиент се инсталира програма, им се казва, характеристиките на телефона, че ще използват и са помолени да потвърдят това. Идеята е да се спре програми с неочаквани поведение се използва от нищо неподозиращите клиенти, например програма за показване на картината, която също следи позицията на потребителя.

Преди да изпратите този файл трябва да настроите полетата на автора, описание и издател.



Ние може да редактирате явна файл от рамките на Visual Studio. Когато изграждане на прилагането е включен във файла XAP, който се произвежда. Ние можем да намерим това в папката бин, че Visual Studio създава като част от проекта за кандидатстване.

Имайте предвид, че когато ние произвеждаме версия на програмата за качване на пазара трябва да се уверете, че тя е съставена на базата в режим на издаване, не Debug.

Създаване Приложение Плочки и Произведение

Преди да можем да подадат заявление ние също трябва да се подготви на плочките , които ще означават прилагане на устройството, и ние също трябва да се подготвят някои произведения на изкуството за показване на Windows Phone Marketplace . Иконите трябва да имат конкретни размери и да се съхраняват в Портативни Network Graphics (PNG) файлове . Действителните размери от които имате нужда , са:

- А малък мобилен икона ап плочки (задължително) , която се използва в телефона Windows Phone Marketplace , 99 x 99 пиксела по размер.
- Голяма икона мобилно приложение за плочки (по желание) , която се използва в телефона Windows Phone Marketplace , 173 x 173 пиксела по размер.
- Голяма икона PC ап плочки (задължително) , която се използва в телефона Windows Phone Marketplace , 200 x 200 пиксела по размер.
- Фон изкуство (по желание) , която се използва във фонов режим панорама във вписването Marketplace за вашата кандидатура , 1000 x 800 пиксела по размер.

Можете също така да създадете поне една снимка , която е 800x480 пиксела по размер. Всъщност можете да качите няколко снимки на екрани, това е една много добра идея , тъй като дава на потенциалните купувачи по-добра представа за това какво вашата програма е искал/

Това е много добра идея , за да получите помощта на графичен дизайнер, който да подготви иконите и фон изкуството за вашата програма .

Изследване на Вашата Приложение

Отключване Устройства

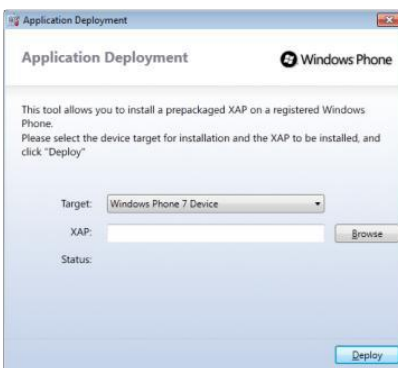
Windows Phone може нормално само стартиране на програми, които са били получени чрез Marketplace. Все пак държавата-базар може да отключите телефона, така че да може да се използва за стартиране на програми, които се зареждат в нея от Visual Studio. Платените членове могат да отключите Marketplace до три устройства, докато членовете студентски, които се присъединиха през DreamSpark могат само да отключите едно. Отключването се извършва с помощта на приложение, което се инсталира като част от софтуера на телефона Комплектът за развитие на Windows. Това приложение може да заключите и отключите телефони



Приложения, които са разположени към телефона ще остане в паметта на телефона и може да бъде стартирана от приложение или игра центрове по същия начин като тези, получени от Marketplace. Въпреки това, вие може да има само до 10 от вашите собствени програми на дадено устройство, по всяко време.

Разпределянето XAP файлове

Ако искате да изпратите програма за друг разработчик, можете да ги изпратите файла XAP и те могат да използват програмата за разполагане, за да изпратите файла XAP да си отключена устройство или емулаторът на тяхната Windows PC.



Това ви позволява да разпространявате приложения за тестване, преди да пусне на готовия продукт.

Програма Обфускация

Ако изпратите някой файл XAP е много лесно за тях да се отвори това и да разгледаме всички неща в него , в това число всички кода на програмата , че прекарват толкова дълго писане. За съжаление това е просто въпрос да отвори монтаж и да разгледаме как тя работи . Направихме това в глава 3 с помощта на програмата ildasm . Това означава, че вие трябва да бъдете внимателни при изпращане на програми, които не се раздават всички упорита работа, която ще ви постави в правенето на програмата . Всеки път, когато ви изпращам програмата, която трябва да предприеме стъпки , за да я направи по- трудно за някой, който да разплитам съдържанието .

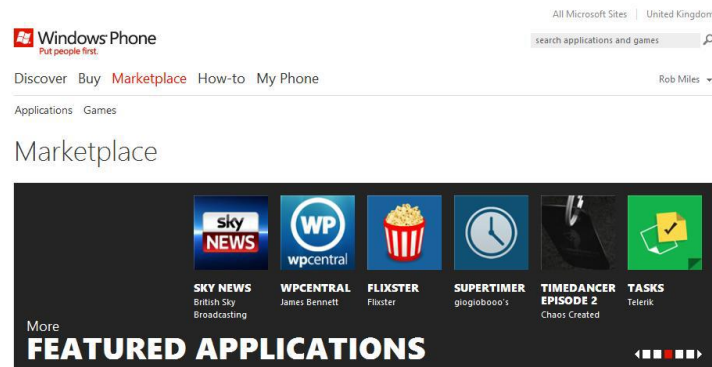
В разработката на софтуер на този процес се нарича объркване . Това включва използването на специален инструмент, който се използва в процеса на изграждане , за да го направи много трудно за някой, който търси по вашата програма да направи извод как работи и какво прави той. Инструментът променя имената на променливите и добавя фалшив контрол на потока , за да се направи програмата, много трудно да се разбере.

Има редица безплатни инструменти злотворци , които можете да използвате, за да направите вашата програма по-трудно да се разбере. Можете да използвате тези на вашия програмен код , преди да изпратите файла XAP . Телефон програмата разработчик на Windows също така включва достъп до злотворци инструменти от превантивна Solutions. Те се предоставят на много ниска цена (тези системи са изключително скъпи) и трябва да се обмисли възможността за използване тях, ако сте загрижени за този въпрос. Тези инструменти могат да бъдат използвани също за инструмент на вашия код и ще ви позволи да разберете каква полза се прави от най- различни части на вашата програма.

10.2. Разпределителни Windows Phone приложения и игри

Единственият начин, че собственик на Windows Phone може да получи програма върху тяхното устройство е чрез изтегляне от Windows Marketplace. Само програми, които са преминали през процеса на валидиране Marketplace може да бъдат натоварени на телефон.

Получаване на Windows Phone приложения



Собственик на Windows Phone могат да използват своя Windows Live ID, за да влезете в телефона си и да получат достъп до услуги, предоставяни от Windows Live, Zune и Xbox Live, заедно с Marketplace.

Собствениците на Windows Phone могат да изтеглят приложения чрез приложението Marketplace на устройството (чрез WIFI или клетъчната мрежа), на Zune софтуера на Windows PC или от раздела Windows Phone Marketplace на Windows Phone сайта на www.windowsphone.com. Много големи приложения, които не могат да бъдат изтеглени чрез клетъчната мрежа и трябва да бъдат прехвърлени с помощта WIFI или програмата Zune.

А даден Windows Live ID може да се използва за до пет телефонни устройства. Ако премахнете дадено приложение от устройство, можете да го презаредите по-късно, без да го купуват отново, тъй като на Marketplace съхранява списък на всичко, което сте купили. Пазарът също така осигурява система за рейтинг, която позволява на собствениците на приложения, за да дават оценки и отзиви за тях.

Създаване на Windows Phone приложения и игри

Регистрираният Windows Phone разработчик може да качите своите приложения и игри в Marketplace за дистрибуция. Те правят това чрез "App Hub" в create.msdn.com



Тестване Marketplace Одобрение

Преди приложение, могат да бъдат разпространявани от Marketplace тя трябва първо да премине през процес на одобрение. Това гарантира, че прилагането се държи правилно по отношение на

неща като бутоните Старт и обратно, а също така, че програмата не прави нищо глупаво като вземете всички на паметта или да вземат двамадесет минути, за да започнат да действат. Този процес също така гарантира, че програмите, които са обидни не са направени общодостъпни.

Тестването е частично автоматично и отчасти човек. Един инженер ще стартира програмата и да се гарантира, че тя работи правилно. Приложението ще бъде тестван със светли и тъмни фонове по телефона, за да се уверите, че всички менюта са видими.

Ако вашето приложение не е одобрено, ще получите писмен доклад, който дава подробна информация за частите на програмата, които изискват внимание. Когато подадете повторно теста ще се фокусира върху областите, които са били идентифицирани, а не на цялата програма отново. Понякога една кандидатура ще бъде одобрена с някои "предупреждения". Това означава, че въпросите са били идентифицирани, които трябва да бъдат разгледани, но положението не е толкова сериозно, че да изисква, че заявлението не може да бъде одобрен. В този случай ще ви е да се очаква да пусне подобрена версия по-късно.

След подаването на заявление е било одобрено, то е включено в една от категориите на приложение и е на разположение за изтегляне. Разработчикът може да произведе една подобрена версия на приложение, което, след като е бил през процеса на одобрение, ще бъде на разположение като надстройка на всички съществуващи собственици.

Някои приложения са безплатни, а други са платени. Когато един собственик на телефон купува платени приложения, цената може да се добавя към сметката за телефон или те могат да се регистрират на кредитна карта, за да се използва за заплащане на покупки. Много приложения имат режим "пробна", което е безплатно. Това по-късно може да бъде обновена, за да пълнофункционален версия. Когато една програма работи, той може лесно да се провери дали той се използва в "пълна" или режим "пробна".

Регистрирани разработчиците право 100 изявления на безплатни приложения годишно. Ако всеки от тях е одобрен, това означава, че те могат да предоставят 100 безплатни приложения. Ако един предприемач иска да представи каквито и да било повече безплатни приложения всяка кандидатура ще им струва \$ 20. А разработчик може да представи неограничен брой на "платените" приложения.

Заявленията могат да бъдат заредени посока към телефона "по въздуха", като използвате телефона от оператора на мобилна или WiFi. Те могат да бъдат натоварени с помощта на Zune софтуер, който също осигурява управление среда за телефона, както добре. Marketplace членство

Преди един разработчик може да подават заявления до Marketplace те първо трябва да станете член. Членството е застопорена към Windows Live ID на члена на. Тя струва \$ 99 на година, за да бъде член на пазара. Ако спреш да бъдеш член на пазара всички приложения са отстранени. Студентите, които имат достъп до програмата DreamSpark могат да получат безплатно членство в Windows Phone Marketplace.

Членове на пазара имат своята идентичност валидиран, когато те се присъединят и Marketplace им дава уникален ключ, който се използва за подписване на техните приложения. The Marketplace също запазва своите банкови и данъчни данни, така че да може да се плаща за продажби на приложения.

Членове на Marketplace могат да зададат на продажната цена на тяхното прилагане и да получават 70% от платената от клиента цена. Това е платена директно в банковата си сметка, след като те са достигнали прага от \$200 продажби.

Пробен Режим

Едно приложение или игра може да бъде изтеглена като безплатна демо. Една програма може лесно да се определи дали или не работи в демо режим:

```
LicenseInformation info = new LicenseInformation();  
  
if ( info.IsTrial() )  
{  
    // running in trial mode  
}
```

Класът LicenseInformation е в пространството от имена Windows.Phone.Marketplace. Ако програмата работи в пробен режим тя може да ограничи действията на потребителите, например тя може да забрани някои функции или да спре изпълнението на програмата след ограничение във времето. Ако потребителят желае да ъпгрейд на заявлението той може да използва механизма Launcher да насочи потребителя към пазара.

Въпреки това, трябва да се помни, че плаща за програма с режим на съдебен процес все още ще бъде видима само в "платени" молба част от Windows Phone Marketplace. Това означава, че клиентите, които само съм търсят безплатни приложения никога няма да видят вашата програма. Може би по-успешна стратегия би било да се произвежда две версии на вашата програма, ограничен "свободен" и един напълно функционален "платени" версия. По този начин потенциалните клиенти имат по-голям шанс за намиране и използване на вашата програма.

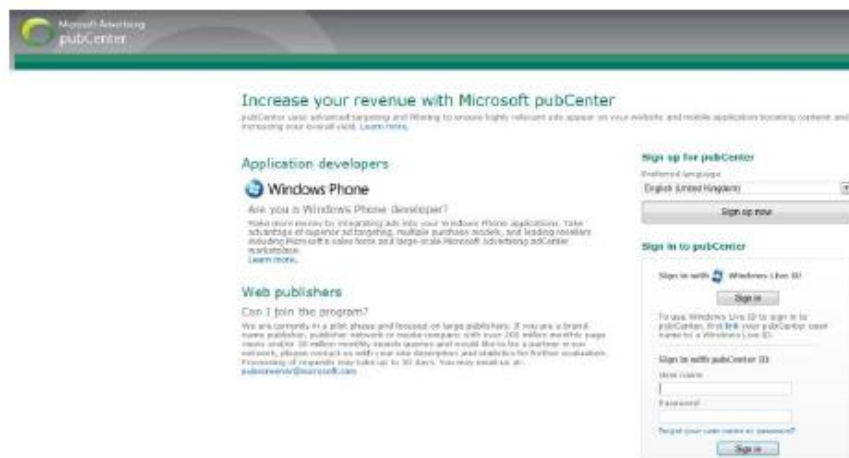
Добавянето Рекламирање

Ако не искате да продавате вашето приложение, но все пак искате да направите пари от него, можете да добавите в-приложни реклами. Те могат да бъдат добавени към XNA или Silverlight програми и ще покаже рекламно съдържание за потребителите. Потребителят може да се отговори на реклама чрез свързване чрез уеб сайта на рекламодатели или поставяне на телефонен разговор с тях. И в двата случая вие ще получите 70% от приходите от реклама, която се генерира.



Рекламите се показват в определени области на екрана на телефона.

Можете да се запишете за една сметка в PubCenter Microsoft:



Сега е възможно да се включи реклама в приложения в голям брой региони в целия свят. Advertising SDK е част от Windows Phone SDK и предоставя набор от тест обява сървъри, които можете да използвате в XNA игри и Silverlight приложения.

Процесът на подаване и одобрение

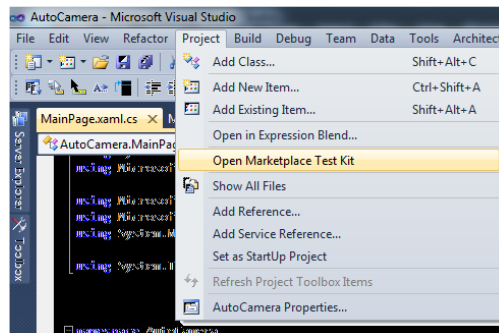
Процесът на кандидатстване за програми, които не е трудно да се използва. Това е всичко, управляван от сайта на разработчика за Windows Phone и XNA игри: <http://create.msdn.com> Посетителите на сайта могат да изтеглят документация и инструменти за развитие. Членове на Marketplace могат да използват страниците на таблото, за да видят напредъка на изявления и да представят нови програми. Съществуват известен брой на подробни Запознай който ще ви преведе през процеса на членство и подаване, трябва да мине през тях, за да се уверете, че сте разбрали какво се случва и какво се изисква от вас и всички програми, които представят.

Windows Phone Приложение сертификация Насоки

Windows Phone разработчик сайт предоставя достъп до много подробен документ, който описва как да създавате приложения. Много е важно да прочетете най-новата версия на този документ и следват насоките, изложени в него. Той е бил през множество версии като процесът се е развила, така че трябва да сте сигурни, че използвате най-актуална версия на текста. Можете да намерите указанията тук: msdn.microsoft.com/en-us/library/hh184844.aspx

The Marketplace Test Kit

Един от начините да се уверите, че молбата преминава одобрения процес първи път е да се възползват от Marketplace Test Kit. Това е вградена в инструменти за развитие и осигурява достъп до един и същ набор от инструменти и процедури, използвани от тестерите. Visual Studio ще проверява за промени в процедурите и актуализира комплект Test автоматично, така че винаги се отразява на политиката одобрения.



Той може да бъде намерен в менюто Project.



Тестовите дори проверяват за правилния тип файл и размера на плочките на приложение. Има също така някои автоматизирани тестове и комплект Test също осигурява детайлизирана репетиция на проведените тестове от тестерите, така че можете да изпълнявате тези тестове, преди да подадете молбата си.

Run Tests		
Passed: 2 Failed: 2		
Result	Test Name	Test Description
Passed	XAP Package Requirements	Validation of XAP file size and content files
Passed	Capability Validation	Validation of application capabilities
Failed	Iconography	Validation of Application Icons
Failed	Screenshots	Validation of Screenshots

Това показва, някои от изпитванията са преминали и други неуспешни.

Частен Бета Тестване

Ако искате да получите обратна връзка от потребителите, преди официално освобождаване на вашето приложение можете да създадете "частен" освобождаване бета тест. Вместо да се направи на приложение видими за всички клиенти на пазара, тази форма на освобождаване изпраща уеб връзка на до 100 тестери, които могат да следват линка до версия на вашата програма, която може да се изтегля на един телефон и да се използва за до 90 дни, след което на приложение се сваля от телефона си.

submit an app!

Let's get started. Distribute your app by giving it a name and uploading the app package.
You can also learn what to expect during this [submission and certification process](#).

* Required fields

* App name for App Hub:
App name only visible in App Hub

* Distribute to: ☐ Public Marketplace
☒ Private Beta Test. Learn more about [beta testing](#).

* Browse to upload a file: Browse
Max size: 225 MB
Expected format: *.xap

* App version number: .

Тази опция се намира в началото на процеса на подаване.

10.3. Осъществяване вашия Забележима Приложение

В Windows Phone Marketplace вече е придобиване на доста голям брой приложения и така , ако искаме нашите програми, за да бъдат тези, които са изтеглили и ние трябва да направим малко допълнителна работа, за да бъдете забелязани . Ето някои съвети, които можете да намерите полезни .

Дизайн да се продават

Атрактивен търси приложение ще има много по-голям шанс да бъдат забелязани. Плочките, фонове и снимки на екрани, които се показват на Marketplace всички трябва да бъдат проектирани, за да помогне продават на заявлението. Вие също трябва да се уверете, че да ви осигури много снимки, а също и голям фон графика. Не правете описанието Marketplace точно преди да публикува заявлението. Уверете се, че можете да прекарат известно време получаване на думите и на дисплея точно.

Наистина атрактивен приложение може дори да се качват като специално подбрани и маркира на Marketplace.

Насочени към различни локализации

Докато може да има повече от едно заявление, като този, който се продава, може да няма нито един на немски език. Така, чрез производството на локализирана версия на програмата, която може да има, че пазар за себе си. Налице е много добра поддръжка локализация на разположение на Windows Phone разработчици, можете да научите повече тук: msdn.microsoft.com/en-us/hh336287

Използвайте App Connect

Функцията App Connect в Windows Phone дава възможност на клиентите да намерят вашето приложение дори да когато те търсят за други неща. Например търсене Bing за "плетене вълна" може да се свърже чрез за вашето приложение плетиво модел. Процесът на свързване на заявление по този начин е свободен и всичко, което трябва да направите е да добавите подробности за категориите, при които приложението ви може да се появи. Можете да научите повече тук: msdn.microsoft.com/en-us/library/hh202969.aspx

Дайте вашата програма навън

Ако ви осигури безплатна версия има по-голям шанс, че потребителите ще го изтеглите и да го използват. Безплатната версия може да съдържа реклами или "конче" съобщения. Защото е лесно да се създаде стартер, който ще отведе потребителя в Windows Phone Marketplace можете да се осигури "купуват ъпгрейд" опция, която може да се насочите платена версия на програмата или играта.

Издаване ъпгрейди / Епизоди

Вместо да се освобождаване игра 15 ниво или напълно функционален програма бихте могли вместо пусне "стартер" версия и след това предоставя ъпгрейди на редовни интервали от време. Има редица причини, поради които това е добра идея.

- Вие ще получите на пазара по-рано от вашата конкуренция.

- Вашата кандидатура ще се появяват редовно в списъка "Нови продукти" на Marketplace. Тъй като много от продажбите на заявление са направени в няколко дни, след като новото издание, което прави по-нови версии ще подобрят продажбите си.
- Можете да получите клиенти, ангажирани в предоставянето на идеи за ново съдържание и функции. Това означава, че може да се изгради добри отношения с тях.

Променете Категории

Когато подава заявление, ще бъдете помолени да изберете категорията, в която тя трябва да бъде показан в Marketplace. Това често се случва, че една специална категория не наистина работят. Една игра може да се мисли като спортен симулация, аркадна игра или пъзел. Ако не сте сигурни кой е най-добрата категория, изпробвайте ги движат на мача и да видим как това се отразява на продажбите. Това също така ще получите експозиция вас в други категории, които също ще бъде от полза.

Насърчаване на добри отзиви

Една добра връзка с клиентите е нещо, което трябва да работим усилено, за да се култивира. В малко вероятния случай на вашата кандидатура трябва да покаже, че при липса на населено поща и предоставя на потребителя с един лесен начин да го изпратите. След това, ако получите такова съобщение, което трябва да отговаря конструктивно. Изграждане на връзка по този начин прави клиенти в дебъгери и защитници на вашите програми, което е наистина полезно.

10.4. Какво да направите после

В този момент в документа, който трябва да има всички умения, които трябва да направите нещо, което може да бъде публикувана на Windows Phone. Ето списък на нещата, които можете да направите, за да направим следващата стъпка.

Регистрирайте се като Разработчик

Ако все още не сте направили това, трябва да минете през create.msdn.com и да се запишете. Ако сте студент, трябва да спрете в dreamspark.com и да получите безплатна регистрация.

Вземете Инструментариумът

Инструментите могат да бъдат намерени в create.msdn.com и ще ви осигури всичко необходимо в един инсталирате. Това включва Visual Studio, емулятора, Expression Blend, Рекламна SDK и програми, за да отключите устройството си.

Публикуване Нещо

След като сте ли си среда за разработка работи трябва да сложите нещо в Marketplace. Всъщност няма значение дали това е нещо, което са масово гордост, и винаги можете да го премахнете по-късно, но никога не се знае, може да има някой там, отчаяно се нуждае от приложението, което направи.

Направете предложения

Ако откриете нещо, което не ви харесва, или имате идея, която може да направи нещата по-добре, можете да използвате форумите UserVoice да предложите неща и да гласувате за тях. The Phone екипа Windows наистина четат и да реагират на тези предложения, както и доста по-малко от тях са намерили своето място в устройството. Можете да намерите форуми тук: wpdev.uservoice.com

Ресурси

Има много добри ресурси за телефонни разработчиците.

Учене Windows Phone развитие

Windows Phone Jump Start

create.msdn.com/en-US/education/catalog/article/wp7_jump_start

Windows Phone Code Samples

msdn.microsoft.com/en-us/library/ff431744.aspx

Application Features for Windows Phone

msdn.microsoft.com/en-us/library/ff402551.aspx

XNA

XNA Game Studio 4.0 on MSDN

msdn.microsoft.com/en-us/library/bb200104.aspx

XNA Game Development Resource Page

create.msdn.com/en-us/education/gamedevelopment

Farseer Physics Engine

farseerphysics.codeplex.com

Silverlight

Windows Phone Silverlight, Development Quickstarts

create.msdn.com/en-us/education/quickstarts

Royalty Free Icons

thenounproject.com

The Silverlight Toolkit

silverlight.codeplex.com

Design toolbox

www.microsoft.com/design/toolbox

Windows Phone Azure Toolkit

watoolkitwp7.codeplex.com

Идеи Програма

Ние вече знаем как да се направи програми за Windows Phone. Ние можем да създадем много страници Silverlight интерфейси, които съвпадат с тези на самия телефон и ние също сме имали

поглед към това как да се създаде игри, които използват XNA. Ние също така знаем, как да създадете свързани приложения, които се възползват от данните, хоствани в интернет. Ние открихме някои от трудностите, срещани от програмисти, когато те пишат код за малки, ограничени ресурси, устройства и сега ние също така знаем как Phone операционната система Windows ни позволява да създадем използваеми приложения, независимо от тези въпроси. Накрая ние знаем как да използваме основната Windows Phone системата, така че нашите приложения могат да се възползват от функции в самия телефон.

В този момент ние имаме великолепен набор от инструменти, следващата ние трябва да се намери проблем да се реши с тях.

Идеи Приложение

Приложения идеи не винаги се появяват напълно оформени. Доста често ви ще започне с една малка идея за разрешаване на проблема и след това да добавите функции или открие ситуации, в които биха могли да бъдат направени на разтвора още по-полезен. Това не е лош начин да се излезе с нещо оригинално, но трябва да се пази от добавянето на твърде много игрални идеи, преди да са направили нещо на работа, в противен случай всичко да рухне под собствената си тежест, преди да са построили нищо.

Можете да получите добри идеи за приложения, като говоря с много хора и се опитвам да разбера какво би било полезно за тях. Не всеки е наясно за това колко много можете да направите с модерни устройства, и така че ако ви кажа, че можете да направите нещо, мобилен, които могат да проследят позиция, да правите снимки и да се възползват от интернет услуги, които те биха могли да имат представа за ситуация, в която някои от които енергия ще бъде полезна за тях.

Идеи за игра

В някои начини на игра идеи са по-лесно да дойда, отколкото идеи за кандидатстване. Системи като XNA ви позволяват "леляща" с игра код, за да разберете какво го прави. Аз вече споменах, че е важно playtesting. Това е мястото, където ще ви даде играта си с други хора, за да разберете дали играта е годен за игра. Playtesting също могат да хвърлят идеи за разработка на видеоигри. Често playtester ще предложи промени в програмата, която ще го направи по-интересно и забавно. Не се страхувайте да направите глупави неща в една игра (да се добавят 10 пъти повече чужденци, обърнете гравитацията, да направи всичко по-голям / по-малки и т.н. и т.н.) и да видим какво ще се случи, за да опита играта.

Голямото нещо за една игра е, че по-скоро, отколкото решаване на конкретен проблем, като със заявление, една игра, просто трябва да бъде забавно да се играе.

Забавлявайте се с Windows Phone

Windows Phone осигурява количество процесорна мощ, че е била недостъпна в света преди няколко години, да не говорим в джоба на всеки. Той също така осигурява невероятно ниво на

свързаност, висока графична производителност и устройства като камери и сензори за местоположение, които правят възможно да се изгради наистина нови приложения.

Лично аз смятам, че просто е в състояние да се програмира за това устройство, нека заедно се продават на резултатите, е вдъхновяващо достатъчно. Надявам се, че са достатъчно поуки от тези бележки, за да получите себе си започват веселбата можете да имате с тази платформа.

Какво научихме

1. Собствениците на Windows Phone получат своите приложения от Windows Phone Marketplace .
2. Регистрирани разработчиците могат да подават заявления за одобряване и разпространение чрез Marketplace .
3. Разработчиците се регистрират и следят напредъка на техните приложения чрез Телефон сайта разработчик на Windows .
4. Регистриране като разработчик струва \$ деветдесет и девет годишно, студентите могат да се регистрират безплатно чрез програмата DreamSpark .
5. Разработчиците могат да произвеждат безплатни или платени приложения, но се ограничават до подаване 100 безплатни приложения за година . По-нататъшни изявления на безплатни приложения струват 20 \$ всеки .
6. Регистрираният разработчик може да отключите телефона , така че те могат да стартират програми , съставени в Visual Studio.
7. Заявленията са разпределени както ХАР файлове, които съдържат всички необходими ресурси и съдържанието заедно с явна файл, който описва съдържанието и дава информация за настройките на телефона за приложението.
8. ХАР файлове могат да се зареждат директно в отключени устройства от разработчиците.
9. Програмистите трябва да обърквам техните програми, така че не е лесно за всеки да получи файл ХАР , за да разберете как работи кода. Има процесор достъпно еквилибристика чрез Телефон разработчик на уеб сайта на Windows .
10. А разработчик трябва да прочетете насоките за Windows Phone сертификация преди представяне на програмите за процеса на одобрение.

Съдържание

За тази книга.....	0
Колектив	2
Глава 1. Windows Phone	3
1.1. Платформата Windows Phone	3
Windows Phone като компютър	3
Windows Phone Hardware	3
The Windows Phone Processor	4
1.2. Операционна система Windows Phone	5
Графичен дисплей	5
Графичният процесор на Windows Phone	6
Touch input	6
Локационни центрове	7
Акселометър	7
Компас	8
Жироскоп.....	8
Сензорна интерграция и симулация.....	8
Камера	8
Хардуерни бутони.....	8
Памет и съхранение	10
Мрежова свързаност	10
Платформени предизвикателства.....	11
Добрите новини.....	11
1.3. Windows Phone екосистема	11
Zune Media софтуер за управление	11
Windows Live и Xbox Live	12
Bing Maps.....	12

Windows услугата известяване	12
Windows Phone и Windows Azure	12
Свървърни приложения.....	12
Съхранение на база данни	13
BLOB съхранение	13
Authentication Services.....	13
В Windows Azure Toolkit	13
Използването на екосистемите	13
1.4. Изпълнение на програма в Windows Phone.....	13
Заявление включване на телефона Windows	14
Background обработка	14
Windows Phone и код на управление	15
The Microsoft Intermediate Language (MSIL).....	15
Компиляция Just In Time	16
Код на управление	16
Ролята на разработчика	17
1.5. Разработка на приложения за Windows Phone.....	17
Windows Phone емулатор	17
Достъп до Windows Phone съоръжения	18
Windows Phone свързаност.....	18
Silverlight и XNA развитие.....	18
Комбиниране на Silverlight и XNA	19
Съхранение на данни в Windows Phone.....	19
Виртуална файлова система	19
Локална база данни	19
Инструменти за разработка.....	20
Профилирано изпълнение.....	20
Windows Marketplace	20

Тестващ инструмент	21
Частни бета версии.....	21
Какво научихме	21
Глава 2. Въведение в Silverlight	22
2.1. Програмен дизайн със Silverlight.....	22
Инструменти за разработка	23
The Metro Design Style	23
Silverlight елементи и обекти.....	24
The Toolbox and Design Surface	27
Managing Element Names in Visual Studio.....	29
Properties in Silverlight Elements	30
Silverlight properties and C# properties	31
Създаване Get и Set методи	32
Използване на свойства	32
Валидиране на данни в свойства	33
Множество начини за четене на свойства	34
Свойства и уведомления.....	34
Page Design с Silverlight.....	34
2.2. Разбиране на XAML.....	35
Защо трябва XAML ?	35
Съдържание XAML файл	36
Extensible Markup Languages.....	36
XML Schema	38
XAML и страници.....	38
2.3. Създаване на Silverlight Application	39
Изграждане на Заявление	41
Изчисляване на резултата.....	41

Събития и програми	42
Събития в Silverlight	42
Управление на имоти събития	45
Събития и XAML	46
Какво научихме	47
Глава 3. Въведение във Visual Studio	47
3.1. Проекти и решения.....	48
Създаване на програми	48
THE NET SDK.....	48
Visual Studio	49
Прост проект на Visual Studio.....	49
Създаването на програма от няколко изходни файла	50
Именно пространство и проекти.....	50
Добавяне на ресурси с проект.....	51
Свойствено съдържание	51
Вграждане на съдържание в самата искова молба.....	52
Файлово събрание и изпълними файла	52
Използването ildasm да погледнем вътре файла за сглобяване.....	52
Библиотека монтаж.....	53
Системни библиотеки и референции	53
Процесът на настройване	54
Visual Studio решения.....	54
Създаване решение мулти-проект.....	55
Обвързването Проекти.....	55
Обвързването Проекти.....	55
Решения Мултипроекта	56
Windows Phone решения	56

Silverlight Windows Phone проекти	56
XNA Windows Phone Project	56
Работещи Windows Phone приложения	57
Файлът XAP	57
3.2 Отстраняване на грешки в програми	58
Използване на Windows Phone емулатор	58
Разполагането на емулятора	58
Функции на емулятора	58
Изпълнение на емулятора	59
Програми в емулятора	59
Отстраняване на грешки Visual Studio	59
Настройка и програмиране	59
Добавяне на точка на прекъсване в декларация	60
Стартиране на програмата	60
Единично засилване	60
3.3. Контролиране на изпълнението на програмата	61
Възобновяване на изпълнението	61
Временно спиране на изпълнение на програмата	61
Спиране на програмата	61
Управление Гранични стойности	61
Използване на прозореца на непосредствените	62
Дизайн за отстраняване на грешки	62
Какво научихме	63
Глава 4. Конструирание на програма с помощта на Silverlight	63
4.1 Подобряване на работата на потребителя	64
Манипулиране настройките на елемента	64
Търсене на грешки	64

Промяна цвета на текста	65
Редактиране на XAML за Silverlight елементи	67
Конфигурация на TextBox и използване на клавиатура с цифри	67
Елемент без екстри	68
Елемент с екстри	69
XAML атрибути и екстри - истината	70
Стойностни характеристики на C#	71
Изобразяване на MessageBox	71
Поле за съобщения със селектиране	72
Добавяне и използване на придобивки	72
Заден екран на машина за събиране	73
Принадлежности за Windows Phone	73
Добавяне на изображения като част от съдържанието	73
Линкове към екстрите	73
“Build Action” на придобивката	73
Добавяне на придобивка като съдържание	74
Използване на Image Content в програма	74
Добавяне на изображения като ресурс	75
Съдържание – Ресурси	76
4.2. Манипулация на данни и изобразяването им	77
Събитието TextChanged	77
Двойно променени събития	78
Свързване на данни	78
Създаване на обект, който да бъде свързан	79
Добавяне на свойство за изместване	80
Добавяне на стандарт, съдържащ нашия клас в главната страница	82
Добавяне на класове към елемент от страницата	82
Свързване на Silverlight елемент към дадено свойство	83

Свързване на данни чрез използване на даннов контекст.....	83
Задаване на Data Binding в XAML	84
Настойване на DataContext.....	84
4.3. Управление на външния вид на страницата на приложението	84
Програми за пейзаж и портрет.....	85
Избор между пейзаж и портрет в MainPage.Xaml.....	85
Събитие при промяна на посоката.....	86
Използване на контейнери за изобразяване на обложка	87
4.4. Изобразяване на списъци с данни	88
Създаване на потребителски списък	88
Създаване на примерни данни	89
Използването на StackPanel за изобразяване на списък	90
Използване на Listbox за изобразяване на списък	91
Свързване на данни на единични обекти	92
Създаване на Data Template	92
Употреба на DataTemplate в списък.....	93
Добавяне на стил към екрана.....	94
Избиране на обекти от ListBox.....	94
4.5. Страници и навигация	95
Добавяне на нова страница.....	95
Навигация между страниците	96
Използване на бутона Back.....	96
Предаване на данни между страниците	97
Използване на събития за навигация между страниците.....	98
Споделяне на обекти между страниците	100
App.xaml страница	100
Дизайн на данните	101

4.6. Употреба на ViewModel класове	101
Дизайн с клас ViewModel.....	102
Разделяне на целите	102
Осигуряване на валидация и преобразуване на данни	102
Предоставяне на поведение Undo.....	102
Създаване на клас ViewModel	103
Местене на данни в и извън Viewmodel.....	104
ViewModel и тестване.....	104
Навигация между страниците, използваща метода GoBack	105
Видими колекции	106
Съхранение на данни	107
Видими колекции и ефикасност.....	107
Какво научихме	108
Глава 5. Isolated Storage на Windows Phone	109
5.1 Съхраняване на данни на Windows Phone	109
Използване на Isolated Storage File System.....	110
Използване на Isolated Storage съхранение настройки	112
Въведение в света на речници	113
Речници и Isolated Storage	114
Спасяването на текст в настройките на изолирани съхранение	114
Зареждане на текст от настройките на изолирани съхранение	115
The Isolated Storage Explorer	115
Какво научихме	116
Глава 6. Използване на бази данни на Windows Phone	117
6.1 Преглед на База данни за съхранение	117
Бази данни и заявки	118
Свързване към база данни	119

Бази данни и класове	119
Използването LINQ за да се свържете с Данни на обекти	120
Библиотеките LINQ	120
Създаване на LINQ Таблица	122
Създаване на Primary Key.....	126
Създаване на Context LINQ Data	127
Създаване Sample Data	127
Обвързването на ListBox към резултата на LINQ Query.....	130
База данни и Данни Обвързващата магия	131
Добавяне на филтри.....	132
6.2 Създаване на връзки между данните с LINQ	133
LINQ Асоциации	135
Обвързването на заповед на клиента, че той е пуснал.....	135
Бази данни и Плавателен.....	136
Свързването на Клиент за всичките си поръчки	136
LINQ заявки и Присъединяване	140
Присъединявайки се към две таблици с въпрос	141
Ключовата дума VAR	142
Изтриване на елементи от Database.....	143
Чужди някои основни ограничения и да изтриете.....	143
Какво научихме.....	143
Глава 9. Създаване на Windows Phone приложения	144
9.1. Windows Phone икони и стартов екран.....	144
Silverlight икони.....	145
XNA икони.....	145
Изработване на икони.....	145
Начален екран.....	146

Silverlight стартов екран	146
XNA стартов екран	146
9.2. Бързо превключване между приложенията	147
Навигация на задачи в Windows Phone	147
Бутоните Start и Back	147
Продължително натискане на бутона Back.....	148
Създаване на приложение с „добро държание“	148
Разбиране на бързото превключване между приложенията.....	148
Методите за бързо превключване на събития	149
Windows Phone приложението LifeCycle	149
Стартиране на програма	149
Натискане на Back за затваряне на приложението	150
Деактивиране на приложение, натискайки Start	150
Преобразуване на приложение от състояние в покой към задгробно такова	151
Подновяване на приложение в състояние на покой или задгробно такова.....	151
Fast Application Switching в приложението.....	152
Въведение в “The Captain’s Log”	152
Captain’s Log и бързото превключване между приложенията	153
Данни в програмата Captain’s Log	153
Навигация на страницата в програмата Captain’s Log	154
Запазване на данни при затваряне на Captain’s Log	154
Зареждане на данни при отваряне на Captain’s Log	155
Управление на Start бутона	155
Управление на процеса на връщане активността на програмата от страна на потребителя...	156
Използване на статична памет	156
Навигация на формите и бърз преход между приложенията.....	158
Бърз преход между приложенията и развитие	158
Принуждаване програмата да се премахне от паметта	158

Бърз преход между приложенията и дизайн	159
9.3. Начини за стартиране и избиране.....	159
Използване на стартовото устройство	159
Добавяне на имейл характеристика към JotPad.....	161
Използване на устройство за избиране.....	161
Изобразяване на картинка.....	163
9.4. Обработване на данни на заден план	164
Задачи на заден план и периодично планирани задачи	164
Агенти и задължения	165
Локализиране при Captain's Log.....	165
Добавяне на задача на заден план в Captain's Log.....	165
Действащ код към приложение на заден план (Background Task Agent Code).....	166
Зареждане на данни от приложението в агента на фоновото задание	167
Достъп до информацията за местоположение в агента на фоновото задание	167
Създаване на входно съобщение.....	168
Показване на „моментално“ съобщение от основния фонов код	169
Отстраняване на дефекти от главния фонов код.....	169
Управление на заданието на фоновия агент	170
Пренос на файлови задачи	171
Принцип на трансфера на файлове от заден план	171
Създаване на заден трансфер	172
Таблични известия.....	172
Агент за аудио плейбек (Audio Playback Agent)	173
Какво научихме.....	173
Глава 10. Windows Phone Marketplace.....	174
10.1 Изготвяне на Заявление за продажба	174
Анализ на ефективността	175

Създаване на XAP Файл за Приложение Разпределение	176
Създаване Приложение Плочки и Произведение	178
Изследване на Вашата Приложение	179
Отключване Устройства	179
Разпределянето XAP файлове	179
Програма Обфускация.....	180
10.2. Разпределителни Windows Phone приложения и игри.....	180
Получаване на Windows Phone приложения	181
Създаване на Windows Phone приложения и игри.....	181
Тестване Marketplace Одобрение	181
Пробен Режим	183
Добавянето Рекламирање	183
Процесът на подаване и одобрение.....	184
Windows Phone Приложение сертификация Насоки.....	185
The Marketplace Test Kit	185
Частен Бета Тестване	186
10.3. Осъществяване вашия Забележима Приложение.....	186
Дизайн да се продават	187
Насочени към различни локализации	187
Използвайте App Connect	187
Дайте вашата програма навън	187
Издаване ъпгрейди / Епизоди.....	187
Променете Категории	188
Насърчаване на добри отзиви.....	188
10.4. Какво да направите после.....	188
Регистрирайте се като Разработчик	188
Вземете Инструментариумът	188

Публикуване Нещо	188
Направете предложения.....	189
Ресурси	189
Идеи Програма	189
Идеи Приложение	190
Идеи за игра	190
Забавлявайте се с Windows Phone	190
Какво научихме.....	191
Съдържание	192