به نام خدا
همانطور که قول داده بودم، قصد دارم از این لحظه آموزش WPF ویا همان Windows Presentation Foundation را شروع کنم. قبل از شروع اموزش لازم است که چند تا نکته را یاداورشوم:

قبل از هر چیز یک نکته اخلاقی:
هرگونه کپی برداری از مطالب این بخش همراه با ذکر کامل منبع بلا مانع می باشد...

(1به دلیل اینکه اموزش به صورت متن می باشد، طبیعتا سرعت آن نسبت به کلاس های حضوری کمتر خواهد بود. چون مجبور خواهم بود که علاوه بر ترجمه منابع مختلف، جهت جمع آوری یک مطلب، عکس های مربوطه را نیز با نرم افزار های مربوطه ایجاد و چنانچه مبحث دارای کد برنامه هم باشد، آن ها را هم مهیا کنم. پس خواهشا صبر به خرج دهید و انتظار تکمیل شدن این آموزش را در چندین روز نداشته باشید. چون این مبحث جدیده و منابع فارسی برای آن وجود نداره.. انشاء الله بتوان یک منبع خوب فارسی در این زمینه پس از این آموزش داشته باشیم

(2به همان دلیلی که در ابتدای گزینه اول گفته شد، امکان توضیح هر مطلب به صورت بسیار و کامل وجود ندارد (کمبود وقت) .. پس مطالب در حدی توضیح داده خواهند شد، که خواننده مطلب را دریافته و بتواند خودش در این زمینه به مطالعات بیشتری بپردازید.. چنانچه در هر کجای مطالب اشکال دارید، یا مطالب برای شما نا مفهوم هستند، به تاپیکی که در گزینه 5 بیان شده است، مراجعه نمایید.

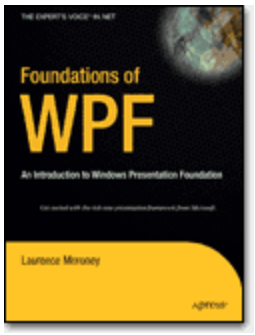
(3به دلیل برنامه ریزی هایی که انجام شده، مطالب به صورت سلسله وار و از مباحث مبتدی شروع میشه و به تدریج مباحث پیشترفته تدریس خواهند شد. چون این جا یک مکان عمومی است و هر برنامه نویسی با هر سطحی ممکن است، وجود داشته باشد. پس دوستانی که بر مباحث ابتدایی اشراف دارند، می توانند این تاپیک را در زمانی که به مباحث پیشترفته تر رسید، دنبال کنند.

(4دوستانی که قصد همکاری در آموزش را دارند، مطالب خود را یا با پیغام خصوصی و یا با ایمیل من که در امضای من موجوده، به دست من برسانند، تا در زمان معین آن را در تاپیک قرار دهم. پس خواهشا از قرار دادن لینک ها و نکته های مختلف و متعدد در این تاپیک خود داری کنید و آن ها را در صورت لزوم در تاپیک نکات که در اعلانات این بخش قرار داره، قرار دهید. این به این دلیل است که می خواهم مطالب به صورت سلسله وار بیان شوند و از هرج و مرج در تاپیک جلوگیری شود.

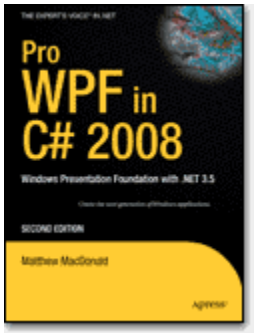
(5هر گونه، سوال و جواب، بحث و گفتگو در رابطه با مفاهیمی که در این تاپیک مطرح می شود، خود داری کنید و آن ها را در تاپیکی که به همین منظور ایجاد شده است و در
ادرس http://barnamenevis.org/forum/showth...769#post499769 قرار دارد، قرار دهید.

(6سعی خواهد شد که پس از پایان هر قسمت، مطالب در قالب یک فایل pdf نیز در اختیار دوستان قرار بگیرد

(7منابع مورد استفاده این اموزش عبارتند از:
الفFoundations of WPF: An Introduction to Windows Presentation Foundation)



ب) Pro WPF in C# 2008: Windows Presentation Foundation with .NET 3.5, Second Edition



ج) Programming WPF
Building Windows UI with Windows Presentation Foundation

د) MSDN Library

) WPF Articles In Code Project۵

بخش اول : مقدمه ای بر تکنولوژی Windows Presentation Foundation

قسمت اول : تکنولوژی های جدید دات نت

قسمت دوم : مشکلات تکنولوژی های قبل در شخصی سازی ابزارها

قسمت سوم : نمونه ساده ای از کاربردWPF

قسمت چهارم WPF :و ارتباط آن با Directx ، GDIوGDI+

قسمت پنجم :عدم وابستگی به رزولوشن صفحه نمایش درWPF

قسمت ششم (آخر :)معماریWPF

بخش دوم : زبان XAML و کاربرد آن درWPF

قسمت اول : مقدمه ای بر زبانXAML

قسمت دوم : کاربرد های مختلف زبانXAML

قسمت سوم : کامپایل XAML به فایل های BAML تزریق شونده به اسمبلی ها

قسمت چهارم : ساختار فایل هایXAML

قسمت پنجم :شکل ساده یک سندXAML

قسمت ششم : فضای نام ها درXAML

قسمت هفتم :خواص و رویداد ها درXAML

قسمت هشتم (آخر :(رویداد ها درXAML

بخش سوم: چیدمان و طراحی کنترل ها

قسمت اول : مقدمه

قسمت دوم :چیدمان عناصر درWPF

قسمت سوم : کنترلStackPanel

قسمت چهارم :ادامه کنترلStackPanel

قسمت پنجم :کنترلCanvas

قسمت ششم :کنترلDockPanel

قسمت هفتم :کنترلWrapPanel

قسمت هشتم :کنترلUniformGrid

قسمت نهم :کنترلGrid

قسمت دهم :ادامه کنترل گرید

قسمت یازدهم :ادامه کنترل گرید

قسمت دوازدهم :ادامه کنترل گرید

قسمت سیزدهم :محدوده سطر و ستون ها در کنترل گرید

بخش چهارمContent Controls :

قسمت اول :مقدمه

قسمت دوم :خاصیتContent

قسمت سوم:کنترل های محتوا با خواص ویژه - کنترلScrollViewer

قسمت چهارم :ادامه کنترلScroll Viewer

قسمت پنجم :کنترلGroupBox

قسمت ششم:کنترلTabControl

قسمت آخر:کنترلExpander

تکنولوژی های جدید دات نت

از زمان ظهور دات نت، با اولین نسخه آن یعنی دات نت فریم ورک 1.0 که همراه با ویژوال استودیو 2002 همراه بود، تا به امروز که شاهد نسخه 3.5 از این تکنولوژی می باشیم، تغییرت بسیاری در آن به وجود آمده است. افزوده شدن کلاس های جدید در غالب فایل های DLL ای که ما آن ها را دات نت اسمبلی می نامیم، همچنین اضافه شدن تکنولوژی های جدید به این مجموعه باعث گسترش کاربرد این مجموعه شده است .

همزمان با ظهور نسخه 3.0 دات نت فریم ورک، تکنولوژی های جدیدی نیز به وجود آمد. این تکنولوژی ها، که بر خلاف تصور سطحی و ابتدایی بسیاری از برنامه نویسان در ابتدای ظهور آن ها، صرفا اضافه شدن تعدادی دات نت اسمبلی به دات نت اسمبلی های قبلی، تلقی می شد، تغییرات بسیاری را در امر برنامه نویسی دات نت به وجود آورد. تکنولوژی WPF به همراه تکنولوژی های WCF و WWF با نسخه 3.0 دات نت فریم ورک توسط شرکت ماکروسافت معرفی شدند.

در ادامه توضیح مختصری راجع به WCF و WWF خواهیم دید و سپس به بحث اصلی، یعنی WPF خواهیم پرداخت.

Windows Communication Foundation

تکنولوژی WCF که مخفف Windows Communication Foundation می باشد، ترکیب شده تکنولوژی های ارتباطی مختلفی که در دات نت فریم ورک 2.0 وجود داشت، می باشد. در دات نت فریم ورک 2.0 ، تکنولوژی های ارتباطی بین سیستم ها عبارت بودند از، ارتباطات بر پایه Soap ، ارتباطات دودویی بهینه شده و... . تکنولوژی WCF که با نام Indigoنیز شناخته می شود، تمامی جنبه های ارتباطی بین سیستم ها را درون خود دارد. جهت مطالعه بیشتر به آدرس

Windows Communication Foundation

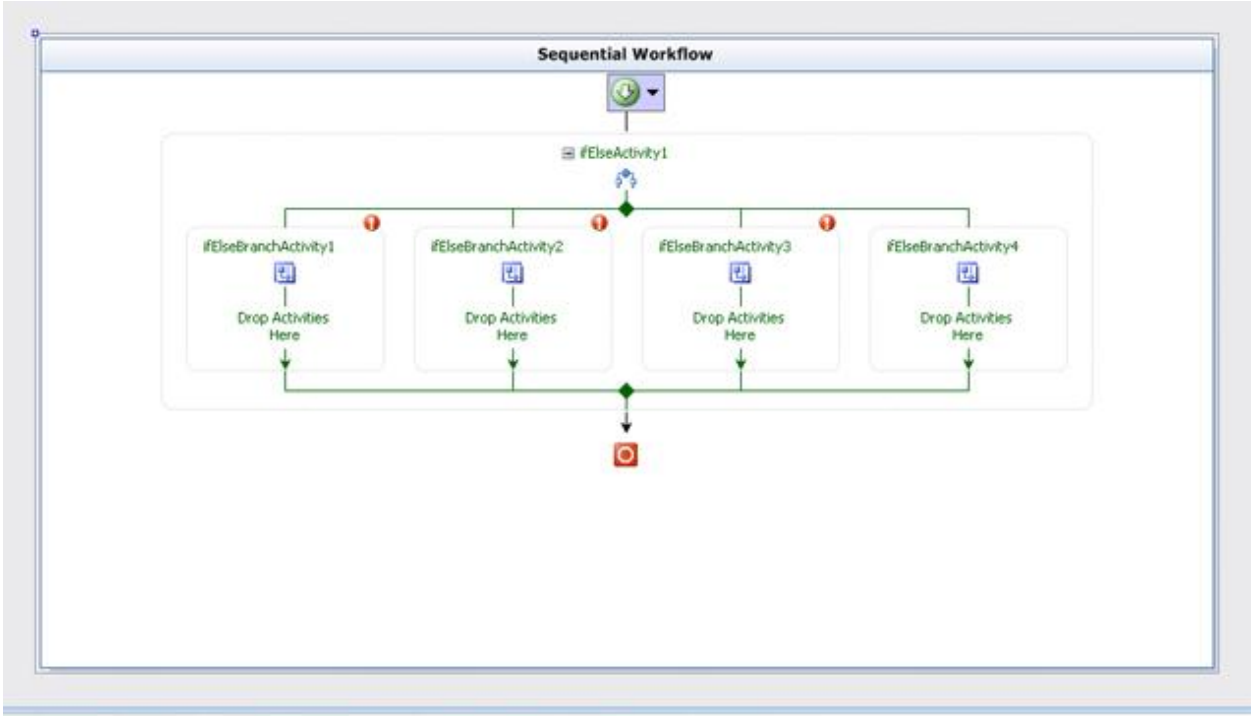
WCF

مراجعه نمایید.

Windows Workflow Doundation

تکنولوژی WWF که مخفف Windows Workflow Foundation می باشد و بیشتر به صورت مخفف WF نشان داده می شود، امکان پیاده سازی و حل مسائل پیچیده دنیای پیرامون خود را که در حالت عادی ممکن است حل آن بسیار پیچییده و دشوار به نظر آید، به صورت بصری و بسیار ساده ارائه می کند. در کل دو شکل Sequential و State Machineرا می توانید با WF پیاده سازی کنید. به عنوان نمونه بسیار ساده به راحتی می توانید یک دستور چند شرطی را به صورت کاملا انتزاعی و با امکاناتی که برای طراحی آن موجود است، پیاده سازی نمایید. به عنوان

مثال، نمونه زیر، پیاده سازی یک دستور چهار شرطی در سیستم WF از نوع Sequential می باشد .



جهت اطلاعات بیشتر در مورد WF به آدرس های زیر مراجعه نمایید

Windows Workflow Foundation

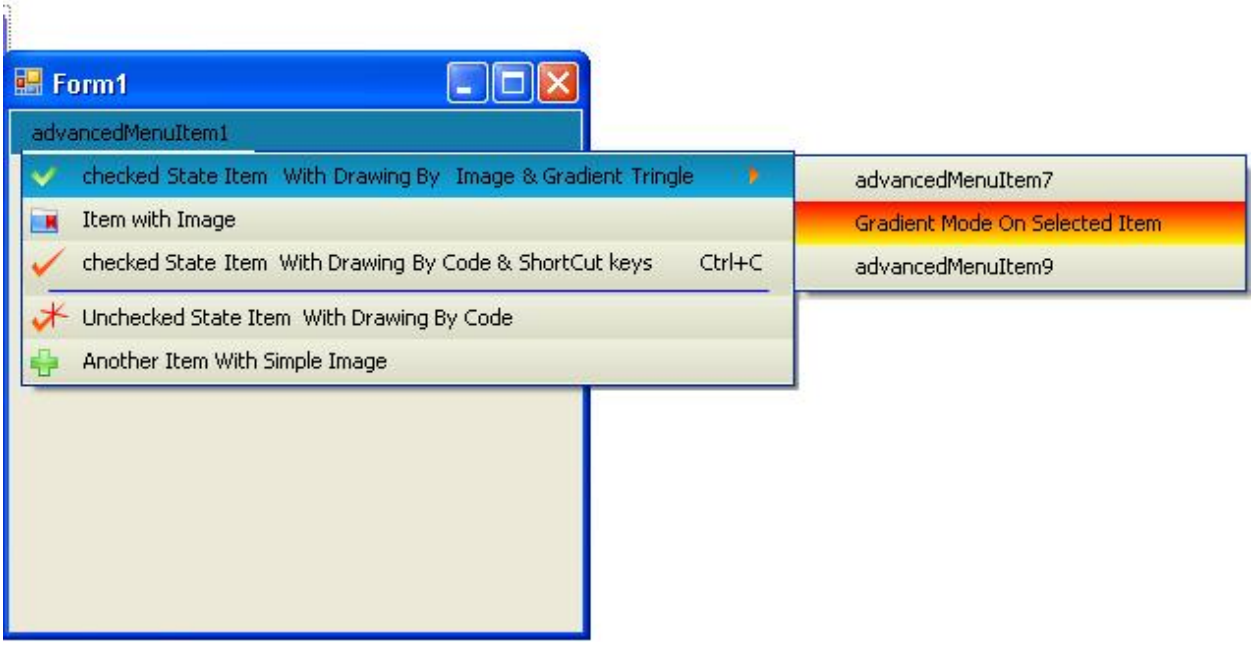
WWF

نکته:

تکنولوژی دیگری که همراه با دات نت فریم ورک 3.0 منتشر شد، Windows CardSpaceبود که با نام info Card نیز معروف می باشد. که جهت اطلاعات بیشتر می توانید به آدرس های زیر مراجعه نمایید

Info CardSpace

س از توضیح و اشارات مختصری درباره تکنولوژی های همپای تکنولوژی WPF که ارتباط بسیار نزدیکی نیز با هم دارند، اینک به معرفی تکنولوژی WPF خواهم پرداخت .سر آغاز سه کلمه Windows Presentation Foundationمی باشد. هر کسی که تا به حال در محیط های گرافیکی و یا به اصلاح برنامه نویسان، محیط های ویژوال، برنامه نویسی کرده باشد، یقینا با مفاهیم Windows Application ها که گاها به صورت مخفف WinApp نیز نامیده می شوند، آشنا می باشد. این نوع برنامه نویسی همزمان با ورود سیستم عامل های ویندوز در دنیای کامپیوتر شروع شد و روز به روز با به وجود آمدن زبان های متفاوت جایگاه محبوبتری نزد برنامه نویسان پیدا کرد. در اینجا قصد توضیح دادن این نوع برنامه نویسی را ندارم. فقط نگاهی گذرا به آن خواهم داشت تا مفهوم تکنولوژی WPFبرایتان روشن تر گردد. همانطور که می دانید، Windows Applicationها، از API های سیستم عامل مربوطه (که اکثرا ویندوز XP نیز می باشد) برای ترسیم عناصر گرافیکی یا همان عناصر ویژوال، استفده می کنند. به عنوان مثال برای ترسیم انواع دکمه ها، فرم ها و بسیاری از عناصردیگری که با آن ها آشنا هستید، از توابع API ویندوز کمک گرفته می شود. همین مسئله باعث ایجاد محدودیت برای برنامه نویسان در ایجاد کنترل های سفارشی با ظاهر دلخواه خود شده بود. اگر چه با ابزار های گرافیکی که در دات نت فریم ورک 2.0 نیز وجود داشت، می توانستیم تا حد خوبی اقدام به ایجاد کنترل های مورد دلخواه خود را بکنیم، اما این موضوع نیاز به دانستن اطلاعات زیاد در مورد ایجاد کنترل های سفارشی و همچنین نوشتن گاها کد های بسیار زیادی جهت ایجاد کنترل مورد نظر می بود. این به آن دلیل بود که قالب و اساس اولیه کنترل ها بسته بود و نمی توانستید به راحتی کنترل ها را شخصی سازی نمایید. در بهترین حالت، یک برنامه نویس ماهر میتوانست با ارث بری از کلاس Control اقدام به ایجاد یک کنترل جدید با ظاهر و امکانات مورد نظر خود بکند. به عنوان مثال کنترل منوی زیر را اخیرا برای شرکتی طراحی کردم. تنها، یکی از کلاس های این منو دارای 1000 خط کد به غیر از کدهای تولید شده توسط خود دات نت می باشد. شاید 1000 خط، برای یه برنامه نویس بسیار ناچیز باشد. ولی چنانچه بخواهید تمامی کنترل های برنامه های خود را، خودتان طراحی کنید، می بینید که زمان زیادی از وقت شما صرف نوشتن کد ها می گردد



این مسئله زمانی نمود بیشتری پیدا میکند که بخواهید، اکثر جنبه های یک کنترل را در کنترل سفارشی خود قرر دهید. به عنوان مثال به دلیل قرار گرفتن حالت های مختلف گرادیان بر روی منو، استفاده از امکانات قبلی مانند ترسیم متن آیتم به صورت اتوماتیک توسط خود منو و یا ترسیم کلید های میان بر آیتم و .... از بین می رود و تمامی این موارد بایستی با کد و توسط شما ایجاد گردد. درست است که می توان از کنترل های ایجاد شده توسط خودتان به کرات و در برنامه های مختلف استفاده کنید و لی تجربه نشان داده است که گاهی نیز مجبور به ایجاد کنترل دیگری شوید. این به این دلیل نیست که شما الزاما کنترل قبلی خود را خوب طراحی نکرده اید. عوامل زیادی می توانند باعث بروز این مورد شوند که پرداختن به آن ها از حوصله این بحث خارج است.
حال که تا حدودی با مشکلات برنامه نویسی های WinApp به روش جاری شدید، در ادامه به معرفی WPF خواهم پرداخت و در ادامه بحث های این آموزش، خواهید دید که WPF چگونه بسیاری از مشکلات موجود را مرتفع می کند. تکنولوژی WPF به روشی دیگر عمل می کند. در واقع علاوه بر اینکه این تکنولوژی همچنان دارای کنترل های سابقی که آن ها را می شناسید، می باشد، می تواند دسترسی به بیشتر جنبه های کنترل ها را برای شما فراهم کند. در واقع قدرت WPF در این است که اساس و پایه هر کنترلی مانند برنامه نویسی قبل، بسته نیست و این شما هستید که به WPF خواهید گفت که متن روی کنترل را به چه صورتی طراحی کنید. یا پس زمینه کنترل یا کناره های آن را به آن صورتی که شما می گئیید طراحی کند. به همین منظر نیز دارای ابزارهای بسیار زیادی جهت کار برای طراحی کنترل های شما مهیا می کند. ابزارهایی مانند قلم مو های گرادیان با تعداد رنگ های نامحدود، انواع ابزار های گرافیکی برای ترسیم شکل دلخواه شما، امکان ایجاد افکت های بسیار زیبا و متنوع بر روی هر قسمتی از کنترل که بخواهید، وجود افکت های از پیش تعریف شده، امکان طراحی های 2 بعدی و نیز 3 بعدی، امکان ایجاد انیمیشن و بسیاری از امکانات دیگر که به مرور با آن ها آشنا خواهید شد.

پایه و اساس WPF بر Directx استوار می باشد. این موضوع سبب می شود که بتوان از بسیاری از جنبه های گرافیکی بدون ایجاد سربار اضافی بر روی برنامه بهره برد و در واقع برنامه هایی با ظاهر هایی بسازید که ساختن آن ها با برنامه نویسی های پیشین یا غیر ممکن و یا متحمل کار بسیار زیادی بوده است. اگرچه نقطه قوت این تکنولوژی اعمال گرافیکی، انیمیشن و .. می باشد، ولی این بدان معنی نیست که نمی توان با WPF اقدام به ایجاد فرم ها و کنترل های سابق نمود. این تکنولوژی به شما امکان استفاده از کنترل های پیشین را می دهد و همچنین برنامه نویسی WinApp را به همان شکلی که می شناسید، برای شما مقدور می سازد. علاوه بر این موارد، WPF امکان کار با اسناد متنی، کنترل کردن بر روی نحوه Print شدن آن ها و ... را برای شما مهیا می سازد.
نکته دیگری که در مورد WPF باید بدان اشاره کرد، امکان برنامه نویسی به شیوه ای است که شاید تاکنون امثال آن را یا ندیده اید و یا بسیار کم دیده اید و آن هم برنامه نویسی بر اساس عناصری در WPF می باشد که به آن ها Pageمی گویند. این نوع برنامه نویسی را می توان به نوعی شبیه سازی برنامه های وب نامگذاری کرد. این نوع برنامه نویسی WPF Browser Application نام دارد که در بخش های بعدی تفاوت آن را با برنامه نویسی معمولی WPFخواهید آموخت. توسط این مدل برنامه نویسی، می توانید اسمبلی های ایجاد شده را در مرورگر وب نظیر Internet Explorerبدون پیغام های امنیتی که معمولا در صفحات وب وجود دارند، نمایش دهید. به عنوان مثال عکس زیر نمونه ای از نحوه استفاده از Page جهت نمایش وب سایت ها در یک برنامه WPF می باشد

البته استفاده از امکانات مختلف گرافیکی به مانند آنچه در برنامه های WPF امکان پذیر است، در برنامه های بر پایه صفحه . امکان پذیر نیست. دلایل این موضوع را در بخش های آتی خواهید دانست.

User32، GDI، +GDIوDirectX

به طور کلی برنامه های ویندوزی از دو امکان، توابع User32 و GDI/GDI+ برای ترسیم عناصر گرافیکی استفاده می کنند که User32 امکان ترسیم عناصر ویژوال را با ظاهر عادی مهیا می کند. عناصری مانند فرم ها، دکمه ها و ... و GDI/GDI+امکانات گسترده تری را جهت ایجاد برخی اعمل گرافیکی مانند ایجاد گرادیان ها و ... را مهیا می کنند.

شرکت ماکروسافت به دلیل محدودیت هایی که در هر یک از دوبخش فوق، وجود داست، اقدام به ایجاد کتابخانه سطح بالایی به نام DirectX کرد.(حرف X می تواند جایگزین کلماتی مانند Sound و .. شود). این ابزار که امروزه نیز از آن استفاده های زیادی میشود،(از جمله در ایجاد بازی های سه بعدی و ...) با بهره گیری از توان کارت های گرافیکی با بهره بری بالا، حداکثر توان آن را برای ایجاد گرافیک های قوی به کار می برد .

اما با قدرت زیاد این کتابخانه، به دلیل برقراری ارتباط مشکل با ان و نیاز به کد نویسی های زیاد، این ابزار بیشتر در

تهیه بازی ها و برنامه های گرافیکی مورد استفاده قرار گرفت و جایگاه زیادی در توسعغه برنامه های تجاری پیدا نکرد .

تکنولوژی WPF تمامی این مشکلات را مرتفع کرد و در واقع کاربر را از درگیر کردن نوشتن کدهای زیاد و گاه طاقت فرسا به صورت مستقیم در DirectX ، رهایی داد .از تمامی قدرت DirectX جهت ایجاد گرافیک های 2 بعد، 3 بعدی، ایجاد انیمیشن ها، استفاده می کند. همچنین ابزار های بسیاری را جهت طراحی کردن در اختیار شما قرار می دهد. علاوه بر این DirectX به جهت اینکه به خوبی با مفاهیمTexture ، Gradientو ... تطبیق پیدا می کند، دارای سرعت بالاتری نسبتGDI و +GDI می باشد. به این دلیل که این تکنولوژی ها برای رندر کردن از روش پیکسلی و الگوریتم های آن که اصطلاحا Pixel By Pixel Instruction گفته می شود، استفاده می کنند .

یکی دیگر از مشکلاتی که کار کردن با DirectX به صورت مستقیم وجود داشت(دارد) به دلیل نوع بهینه سازی و نحوه رندر کردن اشکال توسط کارت های ویدیویی متفاوت بود، که با WPF این مشکل نیز مرتفع شده است.

یکی از مهمترین اهداف WPF استفاده از GPU به جای CPU جهت انجام روتین های پیچیده گرافیکی می باشد که این امر باعث آزاد بودن CPU بوده که میتواند به پردازش های دیگر در سیستم رسیدگی کند.

WPFبه عنوان یک API سطح بالا

همناطور که پیشتر توضیح داده شد، WPFقادر به انجام کار های بسیاری برای شما خواهد بود که قبل از آن، انجام آن ها بسیار مشکل و زمان بر ونیاز به نوشتن کد های بسیاری می بود. در ادامه به صورت لیست وار، تعدادی از امکانات این تکنولوژی همراه با توضیح مختصر آمده است:

طرح بندی اجزا و عناصر برنامه شبیه برنامه های تحت وب WPF :از عناصر جدید و بسیاری دی تراز بندی و چیدمان کنترل ها و عناصر مختلف بر روی فرم های برنامه شما استفاده میکند. توسط این ابزار ها که از کلاس پایه Panel ارث بری می کنند، قادر خواهید بود که چیدمان عناصر خود را چنان تنظیم کنید، که برنامه شما در رزولوشن های مختلف به خوبی قابل نمایش باشد.

نکته مهم و بسیار حیاتی در هنگام کار کردن با تکنولوژی : WPF

یک برنامه نویس WPF حرفه ای حتی المقدور از خواص Width و Height اشیاء برای چیدمان آن ها استفاده نخواهد کرد. یقینا برایتان غیر قابل تصور است. به این دلیل که تا الان هر عنصری که در برنامه خود استفاده کرده اید، پس از نامگذاری آن اقدام به ایجاد سایز مناسب آن نمده اید. اما در نمونه برنامه ها و بخش های آتی خواهید دید، که کمترین استفاده را از این دو خاصیت خواهیم کرد. این موضوع به دلیل ماهیت WPF و غیر وابسته بودن به رزولوشن صفحه نمایش می باشد که در قسمت بعدی بیشتر به شرح آن خواهم پرداخت.

برخی از امکانات و چنبه های برنامه نویسی با: WPF

مدل قدرتمند و قوی طراحی :

توسط WPF از درگیر شدن با پیکسل ها و کار کردن بر روی ان ها رهایی خواهید یافت و در واقهع با ابجکت ها و اشکال سطح بالا تعامل خواهید داشت. همچنین قادر به ایجاد اشکال سه بعدی و... خواهید بود.

نکته:

یکی از محدودیت هایی که WPF داراست، کار کردن با اشکال سه بعدی می باشد. در واقع گرچه با WPF به خوبی

می توانید اقدام به ترسیم این نوع اشکال نمایید، اما از لحاظ کارایی، اشکال سه بعدی ایجاد شده با WPF کارایی کمتری نسبت به نوع های مشابه و تولید شده با DirectX و یا OpenGL به صورت مستقیم می باشد. به همین دلیل چنانچه قصد نوشتن بازیهای سه بعدی Real Time را دارید، WPFممکن است انتخاب خوبی نباشد. چون ممکن است آن کارایی را که انتظار دارید برای شما فراهم نکند.در این موارد می توانید از محیط های دیگر و مناسب اینگونه برنامه ها استفاده نمایید.

انیمیشن، صدا و تصویر :

همانطور که پیش تر نیز توضیح داده شد، علاوه بر انجام اعمال بسیاری که می توانید، با اشکال انجام دهید، اعم از چرخش، بزگر نمایی، کوچک نمایی و ...، نیز می توانید اقدام به ایجاد انیمیشن های زیبا توسط WPF نمایید.
همچنین قادر خواهید فایل های صوتی و ویدیویی را به خوبی به کار بگیرید.

استایل ها و قالب ها :

همواره یکی از دغدغه های برنامه نویسان ویندوز، ایجاد ظاهری زیبا برای فرم ها و عناصر خود بوده است. تا آن جا که اکثر برنامه نویسان به سراغ کامپونت های شرکت های ثالث که آن ها را Third party Components می نامیم، رفته و از آن ها به کرات در برنامه های خود استفاده می کرده و می کنند. من جدای از اینکه این کامپونت ها چقدر در عمل و کارایی درست و حساب شده عمل می کنند، و اینکه با معیار های زبان فارسی متناسب هستند (که اکثرا نیستند)، دلیل دیگری برای استفاده نکردن از این نوع کامپونت ها دارم و ان هم وابستگی برنامه شما به آبجکت ها و عناصر شرکت های دیگر خواهد بود. این موضوع می تواند در طولانی مدت و استفاده مکرر از این نوع ابزار ها، ضررهای جبران ناپذیری به برنامه نویسیان وارد نماید. با تکنولوژی WPF تقریبا تمامی این مشکلات رفع شده و به راحتی می توانید اقدام به ایجاد استایل ها و قالب های متناسب با معیار خود و برنامه خود، نمایید. چنانچه برنامه نویسی مسلط به این موارد گردد، مطمئن هستم که دیگر به هیچ عنوان به دنبال ابزار های ظاهر سازی برنامه ها و کامپونت های متفاوت نخواهد رفت.

دستورات : (Commands)

یکی از جنبه ها و امکانات فوق العاده زیبا و قدرتمند WPF استفاده از Command ها برای هماهنگ سازی واکنش های مختلف کاربر و هماهنگ سازی قسمت های مختلف برنامه به کار می رود که در جای خود، مفصلا به شرح آن خواهم پرداخت. فعلا به همین قدر بسنده کنم که با یادگیری و استفاده از این ابزار، فوق العاده شگفت زده خواهید شدد و خواهید دید که برنامه های شما با این ابزار به چه درصد بالایی از کارایی خواهد رسید.
برنامه های بر پایه صفحه : کمی پیش تر در این مورد صحبت کردم و نمونه عکس برنامه ای را هم که از صفحات استفاده شده بود را مشاهده کردید. در موقعیت مناسب تری بر روی این نوع برنامه نویسی نیز تمرکز بیشتری خواهیم کرد.

ایجاد واسط کاربر به صورت توصیفی :

زمانی که نامی از تکنولوژی WPF برده میشود، در ادامه آن نامی هم از XAML می آید XAML .که یک زبان توصیفی و XML Basedمی باشد، توسط ویژوال استودیو به کار گرفته می شود تا شما بتوانید فرم ها و عناصر خود را با سرعت بیشتری ایجاد نمایید. به جرات می توانم بگویم که استفاده از XAML در سرعت تولید برنامه های شما، تاثیر چشمگیری خواهد داشت. در ابتدا ممکن است در استفاده از آن کمی دچار سردرگمی شوید، تا آن جایی که بخواهید آن را رها کنید و اقدام به ایجاد محیط واسط برنامه خود با کد نمایید. اما با کمی تلاش و مسلط شدن بر آن ، لز کار کردن با آن لذت خواهید برد، تا جایی که هیچ وقت دوست ندارید دیگر سراغ کد نویسی بروید!!! (البته این یه امر محال خواهد. چون حتما نیاز به کدنویسی هم خواهید داشت(

عدم وابستگی WPF به رزولوشن:

بدون شک یکی از جنبه های فوق العاده مفید و قوی WPF عدم وابستگی آن به رزولوشن صفحه نمایش است. اگر

به خاط داشته باشید، کمی پیش در یک نکته مهم، این موضوع را یادآور شدم که یک برنامه نویس حرفه ای در WPF حتی المقدور از خواص Width و Height عناصر برای چیدمان آن ها استفاده نخواهد کرد. دلیل این گفته را در ادامه متوجه خواهید شد.

برنامه های تحت ویندوزی که تا کنون و با تکنولوژی های موجود نوشته می شدند(می شوند) وابستگی زیادی به رزولوشن صفحه نمایش دارند. یه عنوان مثال فرم های شما، که در صفحه نمایش شما با رزولوشن 1024 * 768 به خوبی طراحی شده اند، ممکن است در یک کامپیوتر دیگری با رزولوشن بالاتر از ان (این امر در Laptop ها بسیار معمول می باشد. علاوه بر اینکه ان ها در بیشتر مواقع از تراکم 120 DPI استفاده می کنند. در صورتی که مونیتور های CRT معمولا از تراکم 96 DPI استفاده می کنند. "گر چه قابل تغییر می باشد") کوچک شود، و بر عکس، در یک سیستم با رزولوشن پایین، قسمتی از فرم های شما از صفحه نمایش خارج گردد.

اما با WPF این مشکلات مرتفع می گردد. دلیل آن هم استفاده از سیستم خاصی برای اندازه گیری اجزاء و عناصر برنامه شما، می باشد. عناصر، اعم از دکمه ها، فرم ها و هر شی قابل اندازه گیری با واحدی با نام DIU (Device Independent Unit) اندازه گیری می شوند. هر یک DIU معادل با 1/96 (1 تقسیم بر 96) هر اینچ می باشد. در واقع می توان گفت هر DIU در صفحه نمایشی با تراکم پیکسل استاندارد یعنی 96 DPI ، دقیقا برابر با 1 پیکسل فیزیکی در صفحه نمایش می باشد. حال اگر از DPI بالاتری استفاده گردد، طبیعتا هر یک DIU در همان رزولوشن قبلی)کمتر از 1 پیکسل خواهد شد(چرا؟(

حال WPF با اندازه گیری DPI در هر رزولوشنی که با فرمول مشخصی محاسبه می شود، می توانید سایز مناسب عناصر شما را محاسبه کند. این روش باعث می شود که نمایش یک کنترل مانند Button در رزولوشن 1024*786 و با 96 DPI تراکم، با نمایش آن در رزولوشن 1600*1200 و با تراکم 120 DPI یکسان باشد.
حال باید دلیل اینکه چرا نباید حتی الامکان عرض و ارتفاع کنترل ها را به صورت مطلق و دستی تعیین کرد را متوجه شده باشید. (چرا؟(

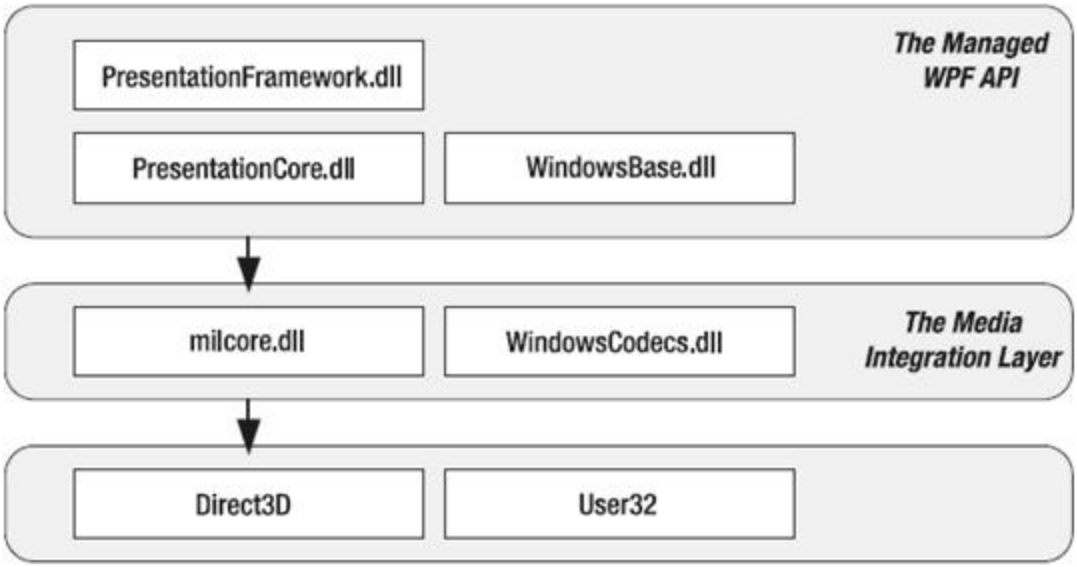
معماریWPF :

تکنولوژی WPF یک تکنولوژی چند لایه می باشد.در بالاترین لایه آن اسمبلی های پایه ای و اسای WPF قرا گرفته اند که تماما به صورت کد های مدیریت شده سی شارپ می باشند. این لایه شامل API های
PresentationFramework.dll، WindowsBase.dllو PresentationCore.dll می باشد که در واقع برنامه شما با این اسمبلی ها ارتباط خواهد داشت .

در لایه زیر آن، کامپوننت مدیریت نشده milcore.dll قرار دارد. تمامی کدهای نوشته شده توسط شما، از طریق لایه اول و ارتباط لایه اول با لایه دوم و کامپوننت مذکور تبدیل آبجکت های مورد نظر می گردد .
در واقع دلیل اینکه کامپوننت milcore.dll به صورت مدیریت نشده می باشد، این است که این کامپوننت بایستی ارتباط تنگاتنگی و مجتمع شده ای با Direct3D داشته باشد و نیز دارای کارایی بسیار بالایی از هر لحاظی باشد .

Direct3Dدر لایه زیرین milcore.dll قرار گرفته است که به صورت یک API سطح پایین می باشد و در واقع به نوعی موتور WPF به همراه milcore نیز به حساب می آید.

در شکل زیر بخش های مختلف معماری WPF نشان داده شده اند



همانطور که گفته شد، برنامه شما در بالاترین سطح با API های سطح بالا که در واقع پایه واساس WPF را تشکیل می دهند، ارتباط بر قرار می کنند. در ادامه به تشریح هر یک از این کامپوننت ها و ابزار ها خواهم پرداخت:

PresentationFramework.dll : این اسمبلی در واقع تمامی آبجکت های سطحه بالا و در واقع به نوعی بالاترین سطح از آبجکت های WPF مانند Windows ها (که بالاترین سطح در برنامه های WPF را در مدل برنامه نویسی WPFApplicationدارا می باشد) و Panel ها که از دیگر اجزاء اساسی برنامه های WPF می باشند،را نگه داری می کند.

می توانید Windows ها را به مانند Form ها در برنامه های معمولی در نظر بگیرید. همچنینPanel ، کلاس پایه برای تمامی کنترل های Container از جمله) Grid که مهمترین آن ها و پر کاربرد ترین آن ها می باشد(، StackPanel، Canvasو ... می باشد.

Presentationcore.dll : شامل نوع های پایه از جمله UIElement و Visual می باشد که تمامی اشکال و کنترل های از این کلاس ها ارث بری می کنند. در قسمت بعدی نمودارسلسله مراتبی کلاس های WPF را مشاهده خواهید کرد.

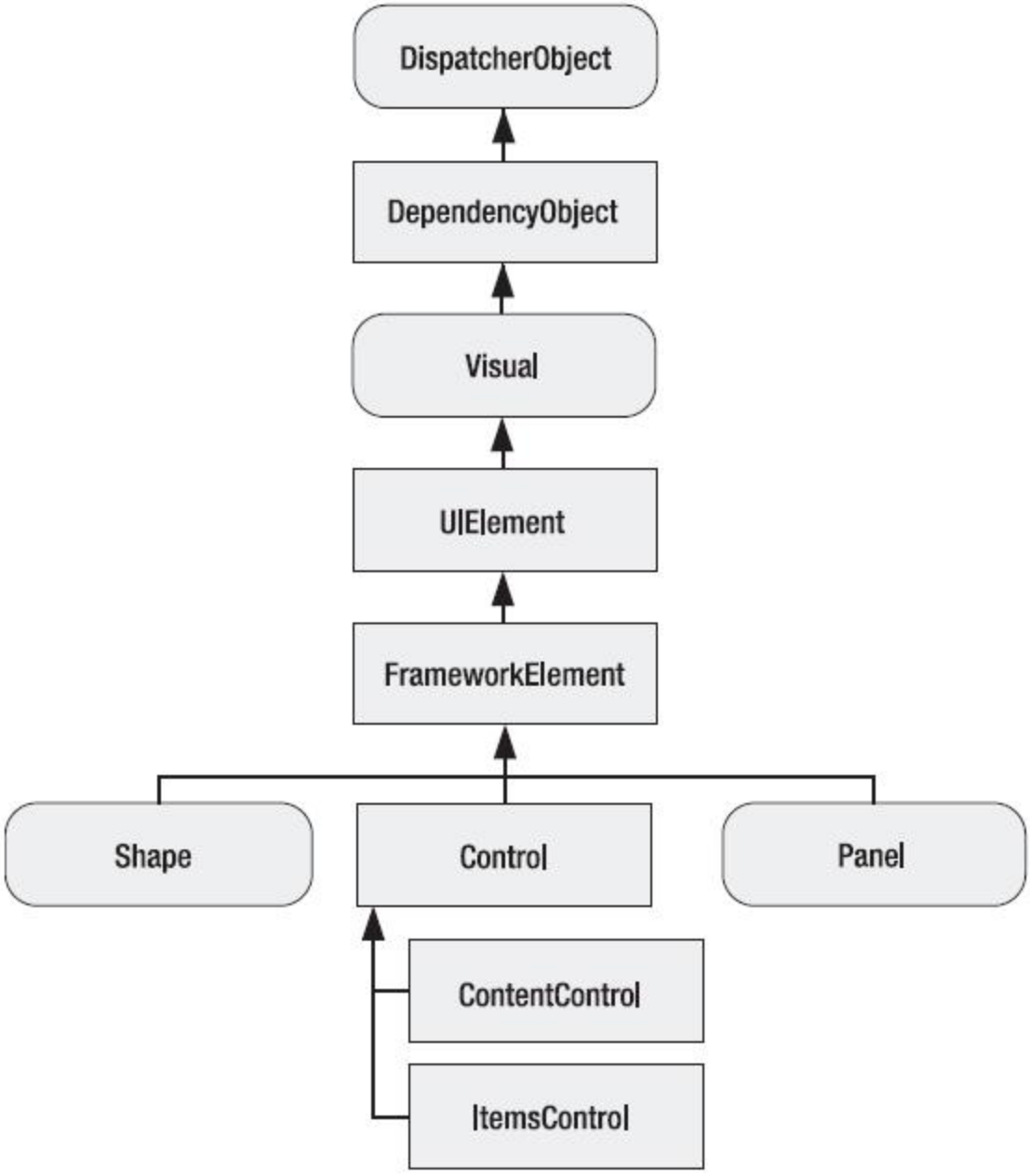
Milcore.dll : در واقع هسته اصلی WPF در رندر کردن آبجکت ها به آبجکت هایی که لایه زیرین خودش یعنی Direct3Dنیاز دارد، می باشد. علاوه بر این در ویندوز ویستا، مدیر پنجره های دسکتاپ یعنی Desktop Windows manager) که عمل مدیریت پنجره های دسکتاب را بر عهده دارد) از همین کامپوننت استفاده می کند. در واقع شما می توانید با فراخوانی DWM ، به فرم ها، یا صحیح تر بگویم به پنجره های برنامه خود، افکت هایی که پنجره های ویندوز ویستا دارا هستند را اضافه نمایید .

نکته:
دقت کنید که این افکت ها بر روی ویندوز ویستا به تنهایی قابل پیاده سازی هستند. گرچه ابزار ها و کامپوننت های Crossنیز برای این کار نوشته شده اند و لی به صورت عادی برنامه هایی که بر روی ویندوز ویستا اجرا می شوند، می توانند قابلیت افکت های ویندوز ویستا را دارا باشند

WindowsCodec.dll :یک API سطح پایین می باشد که قابلیت اعمال، کارهای زیادی را بر روی عکس ها، از قبیل بزرگ نمایی، چرخش و .. را دارد.
Direct3Dنیز یک API سطح پایین است که شامل تمامی گرافیک های رندر شده در WPF می باشد. :

ساختار سلسله مراتبی آبجکت ها درWPF

شکل زیر ساختار سلسله مراتبی آبجکت های مختلف را در تکنولوژی WPF نشان می دهد.



به عنوان مثال کلاس Button را در نظر بگیرید:
این کلاس در اولین سطح از کلاس ButtonBase ارث بری می کند و به ترتیب از کلاس های زیر ارث بری کرده تا نهایتا به کلاس DispatcherObject برسد.

DispatcherObject

.

.

DependencyObject

.

.

Visual

.

.

UIElementt

.

.

FrameWorkElement

.

.

Control

.

.

COntentControl

.

.

ButtonBase

.

.

Button

ترتیب ارث بری ها از پایین به بالا می باشد.

بسیاری از این کلاس ها را در بخش های آتی شرح خواهم داد. اما می توانید جهت اطلاعات بیشتر به کتاب Prof WPF in C# 2008نوشته Matthew macDonald ، صفحه 50 مراجعه کنید.

****
با این قسمت، بخش اول، یعنی مقدمه ای بر WPF پایان می پذیرد. از پست های بعدی، بخش دوم، که مربوط به تشریح XAML می باشد، شروع خواهد شد.

در این بخش قصد دارم، زبان XAML را که نوعی زبان نشانه گذاری می باشد را تشریح کنم و نحوه استفاده از آن و جایگاه آن را در تکنولوژی WPF و نیز در ویژوال استودیو را بیان کنم.
همانطور که در بخش قبلی (مقدمه ای بر (WPF اشاره ای مختصر کردم، XAMLمخفف عبارت
EXtensible Application MarkupLanguageمی باشد. زبان XAML که یک زبان توصیفی می باشد، بر پایه قواعد XMLمی باشد. این زبان در ایجاد برنامه های WPF نقش بسیار موثری را بازی می کند.
این زبان که همراه با ویژوال استودیو 2008 درون آن موجود و نصب شده می باشید، برای نمونه سازی وتعریف آبجکت های WPF به کار می رود. منظور از آبجکت، در اینجا یک واژه کلی می باشد. از یک خط ساده گرفته تا تولید و ایجاد کنترل های پیچیده، همگی قابل پیاده سازی با این زبان توصیفی می باشند. در واقع WPF این زبان را برای ایجاد واسط های کاربری برنامه های خود به کار می گیرد.
اگر چه در ابتدا ممکن است اینگونه به نظر آید که استفاده از XAML برای طراحی پنجره ها و یا صفحات و یا هر آبجکت دیگری درون WPF مشکل تر از نحوه ایجاد فرم های ویندوزی در مدل های برنامه نویسی پیشین باشد، ولی به واقع اینگونه نیست. با کمی تلاش و استفاده از این زبان، پس از مدتی متوجه خواهید شد که توسعه برنامه ها و طراحی پنجره های برنامه به همراه محتویات درون آن ها، توسط XAML بسیار سریعتر و روان تر از روش های پیشین که معمولا به صورت Drag کردن کنترل ها و اشیاء بر روی فرم ها بود، می باشد. علاوه بر این پس از مدتی خواهید دید که ایجاد آبجکت ها و به ویژه ایجاد انواع اشکال با انواع افکت های گوناگون بر روی آن ها، توسط XAML به راحتی صورت می پذیرد. یکی دیگر از ویژگیهای این زبان این است که در ویژوال استودیو، تقریبا بیش از 99% موارد دارای Intellisenceبسیار موثر و کارا می باشد که عمل کد نویسی در این زبان را بسیار راحت تر می کند. در ادامه نگاهی گذرا به روش های پیشین طراحی خواهیم انداخت و مقایسه خواهیم کرد که استفاده از XAML در تولید برنامه های WPFچه اثراتی دارد.

طراحی واسط های گرافیکی کاربر قبل از: WPF

طراحی واسط های کاربری در مدل های برنامه نویسی قبل از) WPF برنامه های ویندوزی) همیشه با بخش کد و منطق برنامه درگیر بوده است. در بهترین حالت، در دات نت فریم ورک 2.0، هر فرم که به عنوان بالاترین آبجکت و به عنوان پدر تمامی آبجکت ها در برنامه های استفاده می شد، دراای دو کلاس مجزا بود.(هست) یکی از این کلاس ها که دارای متدی به نام InitializedComponents بود، (هست). این متد وظیفه طراحی فرم و آبجکت های درون آن را بر عهده داشت. به محض قرار گیری آبجکتی مانند Button بر روی فرم، کدهایی درون متد مذکور به صورت اتوماتیک و توسط خود محیط برنامه نویسی ویژوال استودیو نوشته می شد. این کد ها مربوط به نحوه قرار گیری آبجکت مورد نظر بر روی فرم بود.(هست). و کلاس دیگر معمولا برای کد نویسی و ایجاد منطق برنامه و مشخص کردن عملکرد فم مربوطه و آبجکت های مربوطه به کار می رفت.(می رود). این مسئله ممکن است هیچ ایرادی در یک نگاه سطحی به همراه نداشته باشد. اما در گروه های برنامه نویسی، این یک معضل می باشد. به این دلیل که همیشه طراح با کد نویس درگیر است. این مشکل زمانی بیشتر خود را نشان می دهد که طراح برنامه، (منظور از طراح، گرافیست برنامه می باشد) از کد نویسی و منطق های برنامه نویسی اطلاعات چندانی نداشته باشد
.
این موضوع با ورود ASP.NET 2.0 و به وجود آمدن مبحث Code Behind که منطق برنامه را از طراحی آن جدا می کرد، تا حدی مرتفع گردید. البته کماکان برای برنامه های ویندوزی هیچ راه حل مناسبی وجود نداشت.

این مسئله با آمدن تکنولوژی WPF و همراه ان زبان نشانه گذاری XAML به خوبی مرتفع شده و بسیاری از مشکلات

را کاهش داده است. در این روش، گرافیست برنامه بدون داشتن دغدغه هایی از منطق و نحوه عملکرد برنامه، اقدام به ایجاد و طراحی پنجره های برنامه نماید.
در واقع نکته کلیدی و تفاوت WPF با فرم های ویندوزی این است که WPF طراحی پنجره ها و آبجکت ها را بر خلاف فرم های ویندوزی به کد تبدیل نمی کند. و در نتیجه طراحی برناکمه از کد نویسی و منطق آن کاملا جدا سازی شده است. این موضوع همیشه ذهن برنامه نویسان را درگیر کرده بود. در واقع برنامه نویسان حرفه ای، همواره در تلاش برای تولید برنامه هایی بودن که منطق برنامه از طراحی آن جدا باشد.

نکته:
البته توجه به این نکته بسیار مهم است که وجود XAML به این معنا نیست که WPF حتما به این زبان نیاز دارد. اگر چنین تصوری دارید باید بگم که در اشتباه هستید. هم اکنون هم می توان طراحی برنامه را تماما به صورت کد نویسی انجام داد. اما این مسئله مشکلاتی را که پیش تر به آن ها پرداختم را به وجود می آورد .

عادت بد:
با وجود نر م افزار های گرافیکی که روز به روز بر تعداد آن ها هم افزوده میشه و با اضافه کردن plug in هایی به اون ها و گرفتن خروجی XAML از آن های نباید موجب این شود که نحوه کار کردن با XAML را کنار بگذاریم و از برنامه های آماده استفاده کنیم.. شدیدا و با تاکید بسیار توصیه می کنم که زیاد وابسته اینگونه نرم افزار ها نشوید.
تعدادی از این نرم افزار ها را در بخش های بعدی معرفی خواهم کرد. (شاید هم معرفی نکنم(!!!

کاربرد های مختلف :XAML

زمانی که صحبت از نام XAML به میان می آید، ذهن خیلی از افراد به سمت WPF سوق داده می شود. اگر چه صحیح است که XAML یک ابزار قدرتمند و کارامد در هنگام کار کردن با برنامه های WPF می باشد، اما این موضوع صرفا به این معنی نیست که از XAML تنها می توان در WPF استفاده نمود .
در زیر لیستی از کاربرد های XAML به همراه توضیح مختصری در باره آن آمده است.

XAMLدر : WPF
همانطور که قبلا و از ابتدای موضوعات تا بدین جا چندین بار متذکر شدم، یکی از کاربرد های XAML در هنگام برنامه نویسی WPF می باشد که امکانات بسیاری را برای شما فراهم می کند. هر سند XAML در WPF می تواند نگهدارنده آبجکت های WPF باشد. این آبجکت های می توانند در بالاترین سطح، پنجره های باشند و یا تنها یک آبجکت خط و یا یک مستطیل طراحی شده توسط شما باشد. با ساختار XAML در WPF بیشتر آشنا خواهید شد.

XAMLدر : WF
همانطور که XAML در WPF می تواند نگهدارنده آبجکت های WPF باشد، XAMLدر WF نیز می تواند نگهدارنده آبجکت ها در WF باشد .نیز مانند WPF درای آبجکت های بسیاری از جمله Activity Class ها و .. باشد

XAMLدر : SilverLight نسخه ای دیگر از WPF وجود دارد به نام WPF/E که به نام SilverLight نیز معروف است و نام آن را بار ها شنیده اید. در واقع توسط WPF/E یا همان SilverLight می تواند بسیاری از کارهایی را که با WPF قادر به انجام آن ها در برنامه های ویندوزی هستید، مانند اشکال دو بعدی، صدا، تصویر، انیمیشن و ... را در برنامه های تحت وب به کار ببرید.

نکته:
XAMLکاربرد های دیگری در زیمنه های مختلف دیگری دارد. به عنوان مثال XPS Documentation ها یکی دیگر از زمینه هایی است که XAML در آن استفاده می شود .XPS مخفف Xml Paper Specification می باشد. برای آشنایی با این نوع document ها، خواندن این مطلب خالی از لطف نیست

کامپایل XAML به فایل های BAML تزریق شونده به اسمبلی ها:

زمانی که طراحی های برنامه انجام گرفت، در زمان کامپایل برنامه، ویژوال استودیو تگ های XAML را به فرمتی جدید به نام BAML که مخفف Binary Application Markup Language می باشد، ترجمه می کند. فایل های BAMLدرواقع فرمت دودویی شده فایل های XAML می باشند. علاوه بر این فایل های BAML چون به صورت مجموعه ای از نشانه های می باشند، هم از لحاظ حجم کمتر از فایل های XAML می باشند و هم از لحاظ سرعت Loadشدن، سریعتر از فایل های XAML می باشند. درواقع تعداد خطوط زیادی در فایل های XAML به چندین توکن در فایل های BAML تبدیل می شوند. البته تصمیم گیری در مورد اینکه چه تعداد خطوط از فایل های BAML به چه تعدادی توکن از فایل های BAML و به چه نوع توکن هایی تبدیل می شوند، بر عهده کامپایلر است.

پس از ایجاد فایل های BAML آن ها به اسمبلی های برنامه شما (فایل exe یا (dll ملحق می شوند. (این عمل را اصطلاحا Embed کردن BAML به اسمبلی می گویند.

نکته: (مهم (

از بزرگترین محاسن فایل های BAML این است که می تواند هر نوع فایلی را درون خودش نگه داری کند. به عنوان مثال فرض کنید که برنامه شما از فونتی به نام X استفاده می کند. در مدل های برنامه نویسی پیشین، چنانچه فونت X بر روی سیستم مقصد وجود نداشت، برنامه در قسمتی که از آن فونت استفاده می کرد، دچار اخلال می شد(هر نوع اخلالی اعم از جایگزین شدن با یک فونت دیگر و...) البته راه هایی برای مقابله با این مشکل وجود داشت. به عنوان مثال می توانستید، در هنگام ایجاد فایل های Setup فونت های مشخصی را به آن اضافه کنید تا در هنگام نصب برنامه در سیستم مقصد، فونت های مربوطه به پوشه فونت در سیستم مقصد کپی شوند .
اما به روشی که به زودی خواهم گفت، می توانید فونت ها و یا فایل های دیگری را به فایل exe خود تزریق کنید. در نتیجه دیگر نیاز نیست که نگران وجود فونت خاصی بر روی سیستم مقصد باشید و مطمئن خواهید بود که هر جا که فایل exe شما وجود داشته باشد، آن فونت نیز وجود خواهد داشت. البته این تنها نقطه قوت BAML ها نیست.
دستورات بسیار کوتاهی که قابل استفاده در XAML هستند برای دسترسی ساده و آسان به فایل های Resource شما، از دیگر مزایای آن ها می باشد ک هدر این مورد در بخش های آتی بیشتر خواهید دانست.

ساختار فایل های: XAML

همانطور که قبلا نیز بدان اشاره شد، زبان XAML یک زبان XML base می باشد. پس طبیعتا شباهت بسیار زیاد و نزدیکی به قایل های XML دارد که تاکنون زیاد با آن ها سرو کار داشته اید. اما توجه به چهار نکته زیر که به نحوی قوانین فایل های XAML را بیان می کنند، می تواند در درک ساختار XAML هها بسیار مفید باشد.

همواره چهار نکته زیر را به خاط ر داشته باشید:

الف :هر تگ آغاز شونده XAML نشان دهنده به نمونه ای از کلاس خاص در WPF نگاشت خواهد شد به عنوان مثال تگ <TextBlock>نشان دهنده آبجکتی از کلاس TextBlock می باشد. منظور از کلاس، هر کلاسی می تواند باشد. بهعنوان مثال فرض کنید، کلاسی با نام MyCustomTextBox ایجاد کرده اید. حال می توانید با دستوری مشابه <cl:MyCustomTextBox>نمونه ای از آبجکت MyCustomTextBox را مشخص نمایید.

نکته:

کلمه cl به کار رفته در دستور فوق، جزء کلمات کلیدی نیست. در واقع نباید این تصور را بکنید، برای ایجاد نگاشت به

کلاس های ساخته شده توسط خودمان، حتما کلمه cl را قبل از به کار ببریم. در واقع در این مثال من فرض کرده ام، که cl ، اشاره به فضای نامی دارد که کلاس MyCustomTextBox درون آن قرار گرفته است. نحوه مشخص کردن و اضافه کردن اسمبلی ها را در اسناد XAML را به زودی فرا خواهید گرفت.

ب :به دو صورت می توانید پایان تگ های XAML را مشخص نمایید .
حالت اول : در این حالت از علامت (/>) در پایان تگ استفاده می کنید. که بیانگر پایان تگ می باشد.

قاعده نحوی:

کد:

```
<[Object Name] [ Object Attributes ] />
```

نمونه:
کد:

```
<TextBlock Text="this is a sample textBlock"/>
```

حالت دوم : از تگ (</[ObjectName]>) در پایان کد استفاده نمایید.

قاعده نحوی:

کد:

```
<[Object Name] [ Object Attributes ] ></[ObjectName]>
```

نمونه:
کد:

```
<TextBlock Text="this is a sample textBlock"></TextBlock>
```

ج :پس از نام آبجکت در تگ شروع کد، می توانید صفات آن آبجکت را مشخص نمایید. به عنوان مثال تکه کد زیر، یک Buttonرا مشخص می کند که خواصی برای آن تنظیم شده است. همچنین رویداد کلیک برای آن تعریف شده است.

کد:

```
<Button Name="btnSum" Content="Calculate" Click="btnSum_Click"></Button>
```

د :می توانید خواص هر آبجکت را بین تگ های آغازین و پایانی آبجکت مورد نظر قرار دهید. به عنوان مثال کد فوق، با کد زیر برابر است:

کد:

```
<Button Click="btnSum_Click">
<Button.Name>btnSum</Button.Name>
<Button.Content>Calculate</Button.Content>
</Button>
```

عادت خوب:
سعی کنید، عادت به استفاده از روش دوم (روش د) در تنظیم خواص آبجکت ها کنید. البته این موضوع بیشتر برای زمانی استفاده می شود که بخواهید از خواص پیچیده و ترکیبی برای یک آبجکت استفاده کنید.(این موضوع را کمی جلوتر خواهید دید). ولی به عنوان نمونه برای مثال فوق، بهتر است که از روش (ج) به جای روش (د) استفاده گردد.

شکل ساده یک سند: XAML

در قطعه کد زیر، ساده ترین شکل یک فایل XAML را می بینید. این کدها هنگام ایجاد Window های جدید به برنامه، برای هر Window توسط خود ویوژوال استودیو ایجاد می گردد.

کد:

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

</Grid>
</Window>
```

حال در ادامه اجزای قطعه کد فوق را با هم خواهیم دید:

اگر خوب دقت کنید، کد فوق دارای دو تگ کلی می باشد. یکی تگ Window و دیگری تگ Grid می باشد.
تگ Window بیانگر این است، که این فایل اشاره به یک آبجکت Window دارد. همانطور که قبلا گفتم Window ها در WPFبه مانند Form ها در WinApp می باشند.
تگ Grid اشاره به آبجکتی به نام Grid دارد که یکی از پر کاربردترین آبجکت های WPF می باشد که جزء کنترل های Containerاست. این کنترل به همراه کنترل های دیگر که همه از کلاسی به نام panel ارث بری می کنند، وظیفه

طرح بندی (Layout) پنجره های شما را دارند در مورد این کنترل به همراه کنترل های مربوطه، در بخش Layout مفصل صحبت خواهم کرد.

در خط اول قطعه کد فوق، دستور زیر را مشاهده می کنید:

کد:

```
x:Class="WpfApplication1.Window1
```

این دستور، بیانگر این است که فایل XAML جاری، مربوط به کلاس Window1 که در فضای نام WpfAplication1 قرار گرفته است می باشد. به عنوان مثال چناچه یک Button به فایل XAML فوق اضافه کنید، و برای آن رویدادی (مثلا Click)تعریف کنید، کد های این رویداد درون کلاس Window1 در درون فضای نام WpfApplication1 خواهد بود. این نوع تعریف کلاس و ربط دادن آن به کد های مروبوط به طراحی فرم، برای برنامه نویسان ASP.NET 2.0 آشنا می باشند. (اصطلاحا به آن Code-Behind Class گفته می شود) در آن جا هم از تکنیکی مشابه این استفاده می گردد .

این موضوع باعث جدایی واسط کاربری ما از منطق برنامه می شود. علاوه بر این شما می توانید، فایل های XAML را به صورت پویا و در هنگام زمان اجرای برنامه، فراخوانی کرده و توسط ان ها، واسط کاربری پنجره مربوطه را ایجاد نمایید.

حال به دوخط کد زیر که در قطعه کد فوق قرار گرفته اند توجه کنید:

کد:

```
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
```

این دوخط، دو فضای نام کلی را در برنامه های WPF مشص می کند. در واقع هر سند XAML مربوط به WPF شامل این دو فضای نام خواهد بود. توضیحات بیشتر در مورد فضای نام ها در XAML را در بخش بعدی توضیح خواهد داد. همچنین در آن بخش خواهید دید که دو فضای نام فوق شامل چه عناصری در WPF می باشند.

در نهایت، سه خاصیت برای Window تعریف شده است. خاصیت اول مربوط Tilte پنجره می باشد. این خاصیت مانند خاصیت Text در فرم های ویندوزی می باشد. در واقع متنی که به عنوان مقدار در این خاصیت قرار بگیرد، به نوار عنوان پنجره و همچنین در زمانی که پنجره در حالت minimize قرار دارد، در Taskbar ویندوز، نشان داده خواهد شد. دوخاصیت بعدی هم به ترتیب ارتفاع و عرض Window را مشخص می کند.

فضای نام ها درXAML

در بخش قبل با دو دستور زیر آشنا شدید.

کد:

```

```

```
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
```

دانستید که آن ها فضای نام های اساسی WPF می باشند و بنابراین، هر سند XAML مربوط به WPF بایستی این دو فضای نام را در خود تعریف کند. حال ببینیم تعریف یک فضای نام در XAML به چه نحوی می باشد.

اعلان فضای نام در: XAML

با به کار بردن کلمه xmlns می توانید فضای نام های مورد نظر را در اسناد XAML خود، تعریف کنید. قاعده نحوی تعریف فضای نام ها به صورت زیر می باشد:

کد:

```
xmlns="clr-namespace:[NameSpace Name];assembly=[Assembly Name]"
```

در تعریف فوق، به جای [NameSpace Name] بایستی نام فضای نام مروبطه را قرار دهید. و به جای [Assembly Name]بایستی نام فایل اسمبلی را که آن فضای نام در آن قرار گرفته است را قرار دهید. برای درک بهتر موضوع به مثال زیر توجه کنید:

فرض کنید که یک کلاس اختصاصی با عنوان AdvanceMathClass ایجاد کرده اید که این کلاس در فضای نام MathClassesو در اسمبلی MyCustomClassesقرار دارند. حال برای دسترسی به کلاس AdvanceMathClass بایستی فضای نام آن را و نیز اسمبلی که آن کلاس در آن قرار گرفته است را مشخص نمایید.
با توجه به توضیحات پیشین، بایستی دستور زیر را در ابتدای فایل XAML خود اضافه نمایید:

کد:

```
xmlns="clr-namespace:MathClasses;assembly=MyCustomClasses"
```

حال می توانید به راحتی از کلاس AdvanceMathClass در سند XAML خود استفاده کنید.

نکته:
اگر به خاطر داشته باشید، می توانستید برای فضای نام های موجود، یک اسم مستعار معرفی کنید و از آن اسم در برنامه خود استفاده نمایید. به عنوان مثال می توانید کدی به صورت زیر داشته باشید:

کد:

```
using sys=System;
```

توسط این کد، شما نام مستعار sys را برای System انتخاب کرده اید. حال به راحتی می توانید از کلمه sys به جای کلمه Systsem در برنامه خود استفاده کنید و به فضای نام ها و کلاس های داخلی آن دسترسی داشته باشید .
در فایل های XAML نیز می توانید، عملی مشابه به این را انجام دهید. به قطعه کد زیر توجه کنید:
کد:

```
xmlns:cc="clr-namespace:MathClasses"
```

همانطور که مشاهده می کنید، از کلمه cc به عنوان نام مستعار برای فضای نام مذکور، استفاده شده است. حال می توانید توسط این کلمه به کلاس های درون فضای نام خود دسترسی داشته باشید.

فضای نام های اساسی:

همانطور که در بخش قبل نیز یاد آور شدم، دو فضای نامی که به طور پیش فرض در اسناد XAML در برنامه های WPFوجد دارند، عبارتند از

کد:

```
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
```

که در زیر به توضیح هر یک خواهم پرداخت:

فضای نام: xmlns=http://schemas.microsoft.com/winfx/2006/xaml/presentation

این فضای نام، در برگیرنده تمامی کلاس های WPF و کنترل های و ... که جهت ایجاد واسط های کاربری، به کار گرفته می شوند، می باشد.

فضای نام:xmlns:x=http://schemas.microsoft.com/winfx/2006/xaml

این فضای نام در واقع شامل انواع کاربردهای XAML می باشد که به شما اجازه نظارت بر تولید و شبیه سازی اسناد XAMLرا می دهد. همانطور که مشاهده می کنید، این فضای نام دارای پیشوند x می باشد. این بدان معنی است که در هر جایی از سند XAML که بخواهید از عناصر درون این فضای نام استفاده نمایید، بایستی کدی مشابه کد زیر بنویسید:

کد:

```
<x:[ElementName]>
```

عناصری که از این فضای نام قابل دسترسی خواهند بود، بسته به عنصری که از این فضای نام استفاده می کند، متفاوت می باشد. به عنوان مثال یکی از کاربرد های آن قرار دادن شناسه هایی برای کنترل ها می باشد که در localizableکردن، برنامه ها، نقش اساسی را بازی خواهند کرد

بخش دوم : زبان) XAML قسمت هفتم(

خواص و رویداد ها در: XAML

اگر به خاطر داشته باشید، در زمانی که ساختار ساده یک سند XAML را توضیح می دادم، اشاره ای به سه خاصیت Title، Widthو Height از کلاس Window کردم و اشاره شد که این مقادیر، خواصی را برای پنجره شما مشخص می کنند که به ترتیب عبارت بودند از عنوان فرم، عرض و ارتفاع فرم.

به طور کلی در اسناد XAML ، به دو صورت می توانید خواص یک عنصرا را مشخص کنید:

روش اول : اضافه کردن خواص عناصر در تگ آغازین کنترل مربوطه می باشد. معمولا زمانی از این روش استفاده می کنیم که بتوان مقادیر خو.اص را به راحتی تنظیم کرد. به عنوان مثال به کد زیر توجه کنید:

کد:

```
<TextBox Name="txtNum1" HorizontalAlignment="Center"  VerticalAlignment=
          "Center" Background="Green" Foreground="White">this is sample
                                                    TextBox</TextBox>
```

کد فوق، یک نمونه از آبجکت TextBox تعریف می کند و تعدای خواص آن را از جمله رنگی به عنوان پس زمینه و رنگ پیش زمینه و .. را مشخص می کند.(نگران کد های نوشته شده نباشید، به زودی معنای تمامی آن ها را متوجه خواهید شد(.
پنجره ای که فقط شامل کنترل فوق باشد، ظاهری شبیه با ظاهر شکل زیر خواهد داشت

روش دوم :اضافه کردن خواص کنترل به صورت تگ های داخلی، و بین تگ آغازین و پایانی کنترل مورد نظر می باشد. به عنوام مثال می توان قطعه کد فوق را به صورت زیر نوشت:

کد:

```
<TextBox>
<TextBox.Name>txtNum1</TextBox.Name>
<TextBox.HorizontalAlignment>Center</TextBox.HorizontalAlignmen
<TextBox.VerticalAlignment>Center</TextBox.VerticalAlignment>
<TextBox.Background>Green</TextBox.Background>
<TextBox.Foreground>White</TextBox.Foreground>
<TextBox.Text>this is sample TextBox
</TextBox.Text>
</TextBox>
```

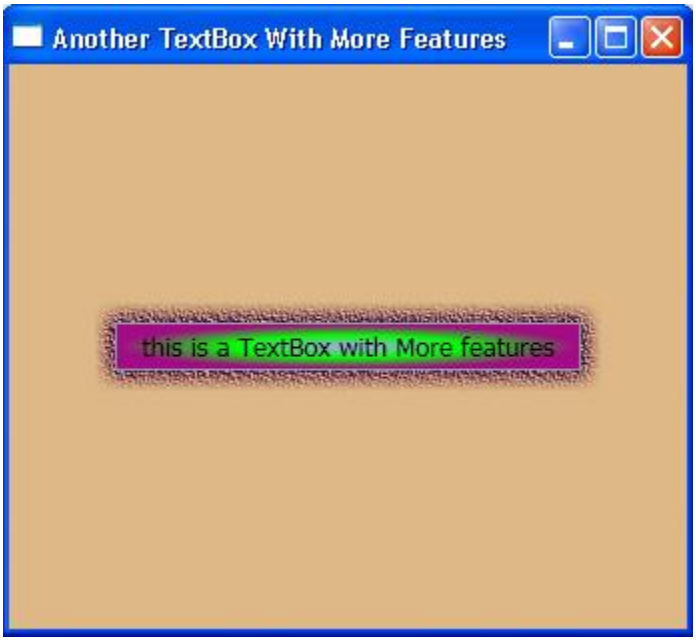
اجرای کد فوق، با کد قبل از آن یکسان می باشد. حال ممکن است که این سوال برایتان پیش آید که حالت دوم نیاز به کدنویسی بیشتری دارد. پس چه نیاز است که کد اول را به این شکل بنویسیم؟
در جواب این سوال باید بگویم که، بسیاری از مواقع، تنظیم مقادیر پیچیده و پیشرفته برای یک خاصیت، در تگ آغازین سخت و گاها غیر ممکن است. به عنوان مثال فرض کنید که بخواهید ظاهر TextBox فوق را با تغییر خاصیت BackGroundو ForeGround آن کمی تغییر بدهید. به قطعه کد زیر دقت کنید:

کد:

```
<TextBox ... >
...
<TextBox.Background>
<RadialGradientBrush >
<RadialGradientBrush.GradientStops>
<GradientStop Color="#b1a4fb" Offset="0"/>
<GradientStop Color="Lime" Offset=".5"/>
<GradientStop Color="#a30c85" Offset="1"/>
</RadialGradientBrush.GradientStops>
</RadialGradientBrush>
</TextBox.Background>
...
```

```
</TextBox>
```

در این کد، خاصیت BackGround تغییر پیدا کرده، و نیز افکتی به آن اضافه شده است. همانطور که مشاده می کنید، خاصیت BackGround در این کد، مانند کد قبل تنها شامل یک رنگ نمی باشد، بلکه یک شی گرادیان می باشد که خود نیز شامل خواص بسیار زیادی می باشد. پس در این حالت نمی توان این خاصیت را در تگ آغازین قرار داد.
)در بخش های آتی با کد های فوق و نحوه عملکرد آن ها بیشتر آشنا خواهیدشد(
شکل حاصل از اجرای این کد، مشابه زیر خواهد بود.



خواص پیوست شده(Attached Properties)

هر کنترلی علاوه بر خواصی که خودش دارا می باشد، بر اساس نحوه قرار گیری آن بر روی کنترل نگهدارنده خودش (کنترلی که این کنترل را در بر گرفته است که اصلاحا به آن کنترل Container گفته می شود.) خواص جدیدی به آن اضافه می گردد که به این خواص، خواص پیوست شده می گویند. به این دلیل این نام برای آن انتخاب شده است که این خواص در حالت عادی برای کنترل مجود نیستند و بسته به کنترل نگهدارنده آن، این خواص اضافه می شوند. به عنوان مثال، اگر TextBox فوق که کد ان را با هم دیدیم، بر روی کنترل Grid که یکی از کنترل های) Container و در واقع مهمترین و پر کاربرد ترین) می باشد، خواصی جهت تنظیم TextBox بر روی Grid به کنترل TextBox اضافه می گردد. نحوه استفاده از این خواص به صورت زیر می باشد:

کد:

```
DefiningName.PropertyName ="Value";
```

به عنوان مثال با اضافه کردن کد زیر به کد های TextBox قبل، TextBoxدر سطر و ستون دوم کنترل گرید، قرار خواهد گرفت.

کد:

```
<TextBox ... Grid.Row="1" Grid.Column="1">


...
...
</TextBox>
```

رویداد ها در: XAML

خوشبختانه، XAMLتمهیدات خوبی برای تعریف و استفاده از رویداد های مختلف کنترل ها فراهم کرده است. نحوه تعریف یک رویداد در اسناد XAML به صورت زیر می باشد:

کد:

```
EeventName="MethodName"
```

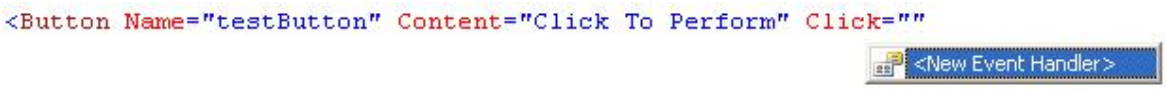
به عنوان مثال در قطعه کد زیر رویداد کلیک برای یک دکمه تعریف شده است:

کد:

```
<Button Name="testButton" Content="Click To Perform"
Click="testButton_Click"></Button>
```

همانطور که قبلا نیز اشاره شد، یکی از بزرگترین قابلیت هایXAML ، هوشمند بودن آن می باشد. به عنوان مثال در هنگام تعریف رویداد، XAMLبه صورت اتوماتیک، لیست تمامی رویداد های تعریف شده برای کنترل های قبلی را که قابل بایند شدن، برای کنترل جدید، باشند را به صورت لیست شده در اختیار شما قرار می دهد. در نتیجه به راحتی می توانید، چندین کنترل را به یک رویداد، بایند کنید.

علاوه بر لیست رویداد های از قبل تعریف شده، گزینه دیگری نیز با عنوان <New Event Handler>موجود است، که با انتخاب آن می توانید، یک رویداد جدید برای کنترل مورد نظر ایجاد کنید.
برای درک بهتر این موضوع به شکل زیر دقت کنید:



همانطور که مشاهده می کنید، لیست کشویی در هنگام تعریف رویداد، برای دکمه قبل باز شده است. اما به دلیل اینکه قبلا هیچ رویدادی تعریف نشده است، که قابل بایند شدن به کنترل Button باشد، تنها گزینه موجود، تعریف یک رویداد جدید می باشد که با انتخاب گزینه <New Event handler>امکان پذیر می باشد. به محض فشردن کلید Enterبر روی این گزینه، یک رویداد در کلاس مربوطه ساخته می شود.

حال اگر بخواهید، برای یک Button ، یک رویداد کلیک تعریف کنید، با لیستی که در شکل زیر نشان داده شده است، مواجه خواهید شد.

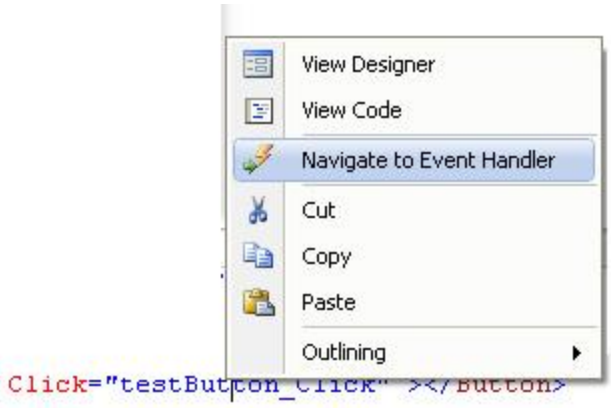
همانطور که می بینید، به لیست قبلی یک گزینه دیگر اضافه شده است، که در واقع رویداد تعزیف شده برای button قبل می باشد. در این حالت، شما دو انتخاب می توانید انجام دهید:

انتخاب اول : گزینه اول، یعنی <New Event handler>را انتخاب کنید، که در این صورت، رویداد جدیدی، صرف نظر از کلیه رویداد های قبلی برای دکمه دوم ایجاد می شود.

انتخاب دوم : با انتخاب گزینه دوم، یعنی textButton_Click ، دکمه دوم را نیز به رویداد کلیک دکمه اول بایند کنید. که در این صورت رویداد جدیدی برای دکمه دوم ایجاد نخواهد شد.

نکته:

برای رفتن به رویداد تعریف شده موجود، دو راه وجود دارد. یا اینکه به کلاس مربوطه که رویداد، در آن تعریف شده است بروید، و رویداد مورد نظر را جهت کد نویسی پیدا کنید. راه دوم، راست کلیک کردن بر روی نام متد مشخص شده برای رویداد، و انتخاب گزینه Navigate To Event handler می باشد. این موضوع در شکل زیر نمایش داده شده است



****

با اتمام شدن این پست، مقدمات لازم برای شروع کار و برنامه نویسی با WPF را فرا، گرفتید. از پست بعدی انشاءالله، با شروع از مبحث طرح بندی (Layout) که از اولین و اصولی ترین و همچنین مهمترین مفاهیم WPF می باشد، برنامه نویسی با WPF را آغاز خواهیم کرد.

****

چنانچه می خواهید نمونه کد هایی را که از قسمت های بعدی ارائه میشوند را بر روی کامپیوتر خود اجرا کنید و نتیجه را مشاهده کنید، ویژوال استودیو 2008 را چنانچه هنوز بر روی PC خودتان نصب نکرده اید، نصب کنید.

در بخش های قبلی، مقدماتی در مورد تکنولوژی WPF و زبان XAML که در این تکنولوژی بسیار مورد استفاده قرار می گیرد، صحبت کردم. در این بخش و بخش های بعدی، نحوه استفاده از کنترل های Container را جهت چیدمان سایر کنترل ها بر روی پنجره ها مورد بررسی قرار خواهم داد .

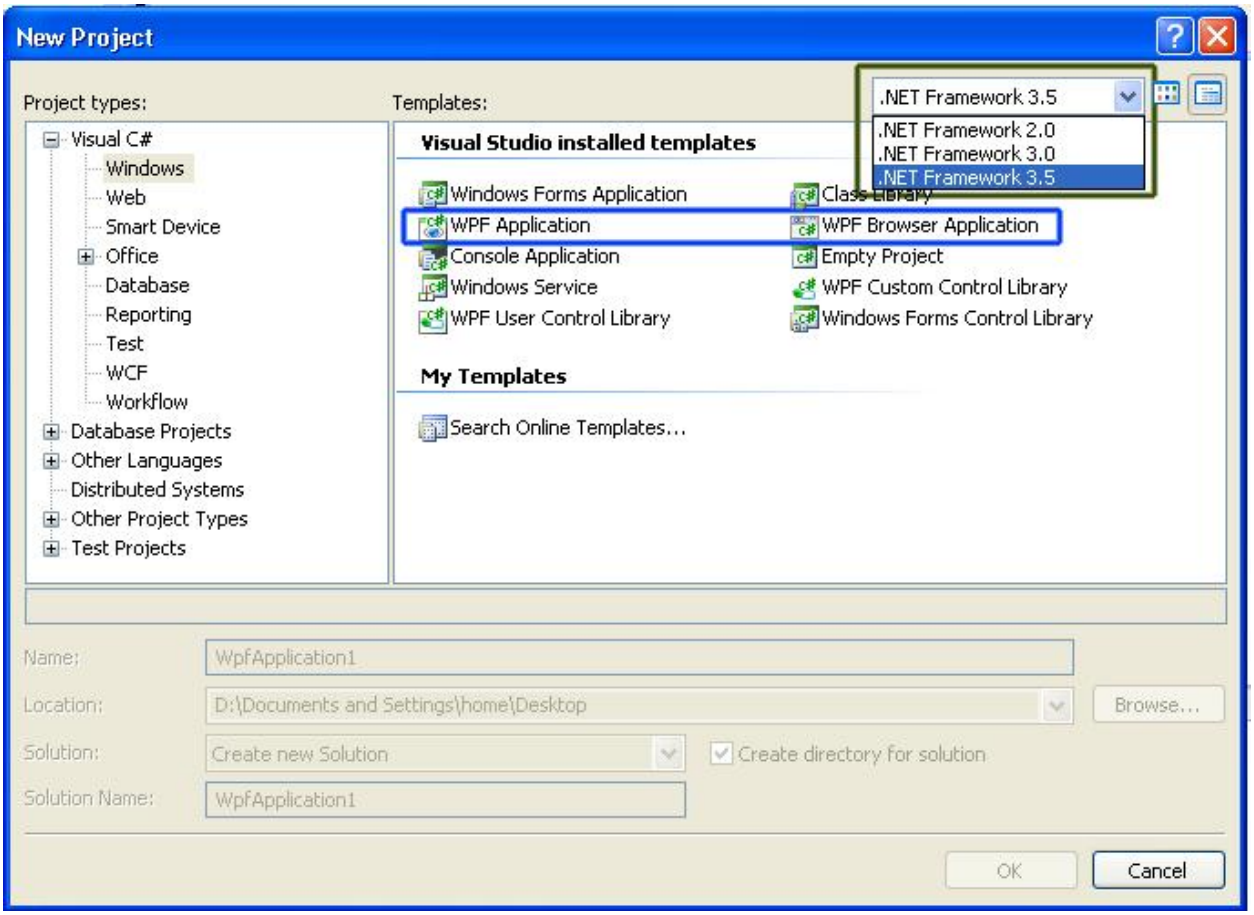
اما قبل از آن نگاهی گذرا به نحوه ایجاد یک برنامه WPF در محیط ویژوال استودیو 2008 خواهیم داشت.

ایجاد برنامه های: WPF

نحوه ایجاد یک پروژه WPF دقیقا مانند نحوه ایجاد پروژه های WinApp می باشد که قبلا نیز بسیار از آن استفاده کرده اید. تنها ذکر چند نکته ضروری می باشد که در ادامه خواهیم دید.

ابتدا برای ایجاد یک پروژه WPF بایستی به پنجره New Project بروید. این پنجره را به روش ها مختلفی می توانید باز کنید (که حتما با آن ها آشنایی دارید .(

این پنجره را در شکل زیر مشاهده می کنید:

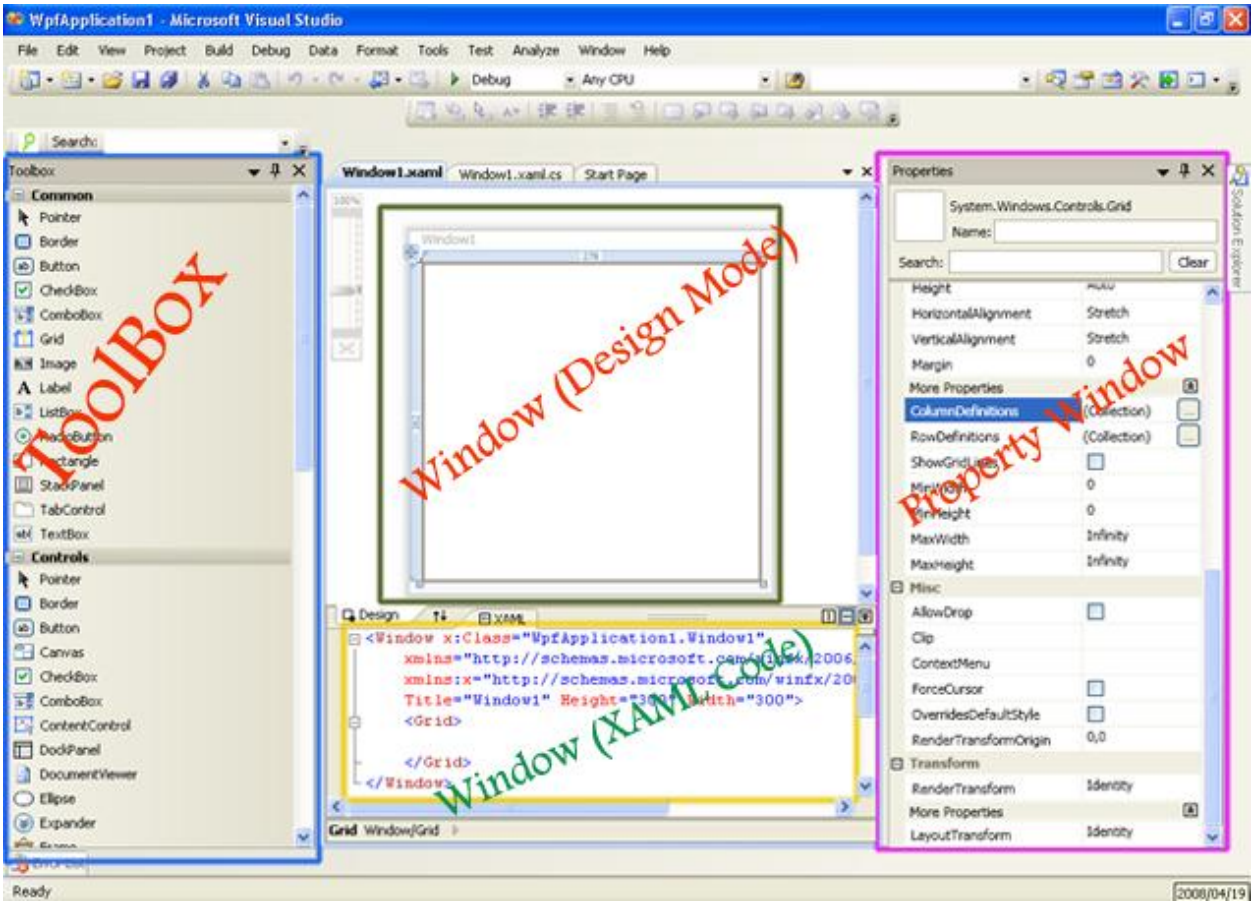


همانور که در شکل نشان داده شده است، دو قالب کلی برای ایجاد برنامه های WPF موجود می باشد. اولین قالب گزینه WPF Application می باشد. که موضوع اصلی ما نیز همین گزینه است. و دیگری گزینه WPF Browser Applicationمی باشد. در ادامه توضیحات مختصری در مورد هر یک از این دو قالب برنامه نویسی WPF خواهم داد. نکته دیگری که در ویژوال استودیو 2008 قابل توجه است، این است که شما می توانید نسخه دات نت فریم ورک خود را انتخاب کنید، و برنامه خود را بر طبق آن نسخه پیاده سازی کنید. (به این قابلیت اصطلاحا (Multi targeting می گویند.

قالب : WPF Application

این مدل از برنامه نویسی WPF ، شباهت بسیار زیادی با مدل برنامه نویسی WinApp دارد. در عین حال نیز تفاوت های بسیاری با آن نیز دارد که مهمترین تفاوت بین آن ها، کنترلی است که به عنوان پدر تمامی کنترل های دیگر شناخته می شود. در برنامه نویسی WinApp تمامی کنترل های بایستی، بر روی آبجکتی از کلاسی به نام Form قرار بگیرند. در حالی که در WPF این کنترل، آبجکتی از کلاس Window می باشد. زمانی که یک پروژه WPF Applicationایجاد می کنید، یک آبجکت از کلاس Window ساخته می شود که به صورت پیش فرض نام آن Window1می باشد. هر کلاس Window دارای دو حالت قسمت مجزا می باشد. قسمتی مربوط به کد نویسی و

ایجاد منطق های برنامه شما، و قسمت دیگر مربوط به Design برنامه می باشد، که در این قسمت دستورات XAML را می توانید مشاهده کنید و اقدام به طراحی برنامه خود نمایید. شکل زیر نتیجه حاصل از ایجاد یک پروژه WPF Applicationرا نشان می دهد.



همانطور که مشاهده می شود، یک Window به وجود امده است. بخش های مختلف روی عکس مشخص شده است. تغییراتی در دو پنجره Toolbox و Properties به وجود امده است که با مشاهده آن، خودتان متوجه تغییرا تخواهید شد.

نکته ای که مهم است این است که در بیش از 90 درصد زمان کار با پروژه خودتان، به سراغ پنجره های Toolbox و Properties ***نخواهید رفت***) .در دات نت 2.0 تقریبا عکس این موضوع بود) این موضوع به این دلیل است که تقریبا تمامی کارها در پروژه شما با نوشتن دستورات و کد ها در قسمت XAML صورت می گیرد. از ایجاد آبجکت ها، تنظیم خواص، رویداد ها و .... (البته در بعضی موراد هم استفاده از پنجره Properties باعث صرفه جویی در وقت خواهد شد.(

قالب: WPF Browser Application

این قالب که یک جنبه جدیدی از برنامه نویسی را پیش روی شما قرار می دهد، شباهت زیادی به برنامه های تحت وب دارد. بزرگترین تفاوت آن با مدل WPF Application این است، که در این حالت کنترل مادر، به جای Window ، آبجکتی است که از کلاس Page ارث بری می کند. این نوع برنامه ها، مستقیما توسط مرورگرهای وب از جمله IE و Fire Foxقابل اجرا شدن هستند. البته محدودیت هایی در به کار گیری جنبه هایی از WPF در این مدل، وجود دارد. به عنوان مثال بسیاری از افکت های گرافیکی را نمی توان در این روش به کار برد.

توصیه :

همیشه در هنگام ایجاد پروژه ها، نسخه 3.5 از دات نت فریم ورک را انتخاب کنید. به دلیل اینکه WPF 3.5 از لحاظ کارایی نسبت به WPF 3.0 بهتر شده است.

نکته :

همچنان می توانید در برنامه های WPF از فرم های ویندوزی سابق نیز استفاده کنید.

کلاس: APP

کلاس دیگری که هر پروژه WPF حتما یک نمونه از آن را دارا می باشد، کلاس APP می باشد که از کلاس Applicationارث بری می کند. این کلاس نیز دارای بخشی کد به صورت XAML می باشد که در زیر مشاهده می کنید:

کد:

```
<Application x:Class="WpfApplication1.App"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
StartupUri="Window1.xaml">
<Application.Resources>

</Application.Resources>
</Application>
```

همانطور که می بینید، آبجکتی از کلاس Application ایجاد شده است که کلاس App را به عنوان کلاسی که کد های آن در آن جا قرار دارد، معرفی کرده است. دو آیتمی که در کد فوق جدید هستند عبارتند از خاصیت StartupUri و تگ. Resources

خاصیت StartupUri ، نام کلاسی را مشخص می کند که به عنوان اولین پنجره برنامه یا همان پنجره اصلی برنامه

نمایش داده خواهد شد.
تگ Resources که می تواند با خیلی از کنترل های به کار رود، منابع برنامه شما را مشخص می کند. چنانچه این تگ در کلاس Application و به صورتی که در کد فوق مشاهده می کنید، تعریف شود، منابعی که در آن تعریف می شوند به عنوان منابع کل پروژه می باشند که در همه جای پروژه قابل استفاده می باشند.

اگر بخواهیم، تناسبی بین این کلاس و کلاسی در دات نت فریم ورک 2.0 ایجاد کنیم، می توان گفت که کلاس Applicationدر WPF عملکردی مانند کلاس Program در برنامه های WinApp دارد. همچنین دستور StartupUri چیزی شبیه به دستور Application.Run در کلاس Program می باشد.

بخش سوم: چیدمان و طراحی کنترل ها (قسمت دوم(

چیدمان عناصر در: WPF

در بخش قبلی، با مقدمات محیط طراحی و کد نویسی ویژوال استودیو 2008 آشنا شدید. در این بخش به کنترل های کانتینر (کنترل هایی که می توانند کنترل های دیگر، شامل همه کنترل های ویژوال و نیز کنترل های کانتینر دیگر را در برگیرند) می پردازیم.

کنترل های کانتیر اساس برنامه نویسی در WPF محسوب می شوند. این کنترل ها، امکانت متعددی را در اختیار شما قرار می دهند که بتوانید کنترل های خود را به صورت صحیح بر روی فرم خود قراردهید.
در WPF کنترل های کانتینر متعددی وجود دارد که هر یک به نوعی امکانات خاصی را برای چیدمان کنترل های شما ایجاد می کنند. به عنوان مثال توسط کنترل کانتینری به نام StackPanel می توانید، کنترل های خود را به صورت پشته ای قراردهید. همچنین کنترلی به نام Grid به شما اجازه قرار دادن و تنظیم کنترل ها را در سلول هایی در سطر ها و ستون هایی که شما تعیین می کنید، را می دهد. اما قبل از وارد شدن به بحث کنترل های کانتینر و معرفی ان ها و خواص و امکاناتی که برای شما فراهم می کنند، بهتر است نگاهی به پایه و اساس قالب بندی یا طرح بندی (Layout) در WPF و تفاوت آن با نسخه های قبلی دات نت فریم ورک بیاندازیم.

فلسفه چیدمان و قالب بندی در: WPF

در دات نت فریم ورک 1 1.0) x.و 1.1) ، دو خاصیتی که در چیدمان عناصر بر روی فرم ها، موثر بودند، خواص Anchor و Dock بودند. توسط این دو خاصیت می توانستید، کنترل ها را بر روی فرم خود، چنان تنظیم کنید، که در صورت تغییر سایز فرم، کنترل ها نیز به تناسب ان تغییر سایز بدهند و یا محل قرار گیری آ نها به صورت پویا تغییر کنید. اما باز هم این خواص جواب گوی نیاز های شما به صورت کامل نبودند. به ویژوه زمانی که کنترل های خود را به صورت پویا و در زمان اجرای برنامه ایجاد می کردید، این مسئله بیشتر باعث عذاب و رنجش بود. به صورتی که گاها نیاز به کد نویسی های بسیاری برای چیدمان کنترل ها بر روی فرم بود.

دردات نت فریم ورک 2.0 عناصر دیگری اضافه شدند که می توانستند، چیدمان عناصر را تا حدی، کنترل نمایند. یکی از این کنترل ها، کنترل FlowLayoutPanel بود (است). امااین کنترل ها نیز قابلیت ها و کارایی بسیار خوبی را مهیا نمی کردند. ودلیل دیگر آن این بود که این کنترل ها به صورت یک افزونه وارد دات نت 2.0 شده بودن و در واقع جزء هسته اصلی فرم های ویندوزی نیستند و به واقع یک افزونه برای آن ها به شمار می آیند. مانند کنترل های بسیاری که شرکت های ثالث نوشته اند و می نویسند. علاوه بر این، اساس این کنترل ها نیز بر پایه مکان قرار

گرفتن کنترل ها می باشد که این خود نیز به نوعی محدودیت محسوب می شود.

تکنولوژی WPF سیستم جدیدی را از پایه برای شما فراهم می کند که به شما اجازه ایجاد برنامه هایی را می دهد که وابسته به سایز و یا رزولوشن صفحه نمایش نباشد. همانطور که قبلا نیز گفته شد، تعیین سایز و محل قرار گیری کنترل ها به صورت مشخص و ثابت، کارایی برنامه را به شدت کاهش می دهد.(البته در مواردی اجتناب ناپذیر است. به عنوان مثال زمانی که از Canvas استفاده می کنید، ناچار هستید که محل قرار گیری کنترل های روی آن را صراحتا تعیین کنید.)
راه حل WPF برای اینکه بتوان بر محل قرارگیری و اندازه کنترل ها نظارت کامل داشت، استفاده کردن از کنترل هایی است که بدین منظر ایجاد شده اند.

چند نکته اصلی و مهم در پشت مفهوم فلسفه چیدمان و قالب بندی در WPF وجود دارد که تیتر وار بیان می شوند:

الف (سایز عناصر نبایستی صراحتا تعیین گردد (تا جای که امکان پذیر باشد(

ب (محل قرار گیری عناصر نباید به صورت دستی نسبت به گوشه صفحه نمایش تعیین گردد(تا جایی که ممکن باشد(

ج (کنترل های کانتینر، می توانند به صورت داخلی قرار بگیرند.(هر کنترل کانتینر می تواند شامل 0، 1 و یا بیش از یک کنترل کانتینر دیگر باشد(

د(کل فضای موجود یک کنترل کانتینر بین تمامی کنترل های درونی آن (Children Elements) تقسیم بندی می شود. این تقسیم بندی بر اساس نیاز هر کنترل به فضایی که نیاز دارد تعیین می گردد و می تواند به صورت پویا تغییر کند.

کنترل های کانتینر(Container Controls)

همانطور که قبلا نیز اشاره شد، تمامی کنترل های قالب بندی WPF از کنترل پایه ای به نام Panel ارث بری می کنند. این کنترل نیز طی ارث بری هایی به آبجکت Dispatcher Object ختم می شود .
کنترل های اساسی کانتینر در WPF عبارتند از:

الف Stack Panel **(**
بCanvas **(**
جDockpanel **(**
دWrapPanel **(**
هUniformGrid **(**
یGrid **(**

که در زیر توضیح مختصری در مورد هر یک داده شده است. و در ادامه به صورت مفصل به بررسی هریک از این عناصر با ذکر مثال هایی خواهم پرداخت.

کنترل : StackPanel

همانطور که از نام آن مشخص است، این کنترل, عناصر را به صورت پشته ای مرتب می کند. به دو صورت افقی و عمودی می توانید کنترل ها را قرارد هید.

کنترل : Canvas ا

ین کنترل اجازه قرار گرفتن کنترل ها را در مکان مشخص و ثابتی می دهد. پس از قرار گرفتن عناصر بر وری این کنترل، مکان ان ها برای همیشه ثبت می ماند.

کنترل : DockPanel

این کنترل عملکردی شبیه به خاصیت Dock در کنترل های دات نت فریم ورک2.0 را دارد. با این کنترل می توانید، عناصر خود را نسبت به لبه های مختلف آن تنظیم نمایید.

کنترل : WrapPanel

این کنترل، عناصر را به صورت سطری و ستونی تا جایی که امکان داشته باشد، قرار می دهید. در حالت سطری، کنترل ها تا جایی که بتوانند در یک سطر قرار می گیرند. اگر فضای مورد نیاز کنترل ها از فضای موجود در یک سطر بیشتر باشد، بقیه کنترل ها به سطر بعدی منتقل می شوند. در حالت ستونی نیز عملی مشابه، ولی در مورد ستون ها انجام میگیرد .

کنترل: UniformGrid

این کنترل شبیه به کنترل Grid میباشد. با این تفاوت که در این کنترل، سایز تمامی سلول ها یکسان می باشد.

کنترل: Grid

این کنترل، از پرکاربرد ترین کنترل های کانتینر می باشد. این کنترل با ایجاد سطر ها و ستون هایی به شما امکان قرار دادن عناصر خود را در سلول مشخصی از ان می دهد. این کنترل
شبیه به کنترل TableLayoutPanel در دات نت فریم ورک 2.0 می باشد.

کنترل: StackPanel

این کنترل، عناصر داخل خودش را که در خاصیت Children این کنترل قرار گرفته اند را بر اساس جهتی که شما مشخص می کنید (افقی یا عمودی) به صورت پشته ای مرتب می کند.

نحوه تعریف StackPanel به صورت زیر می باشد:

کد:

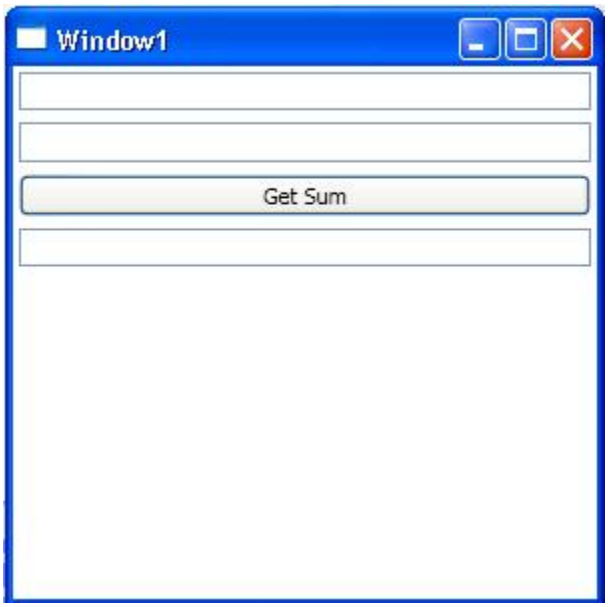
```
<StackPanel>
<!-- Some Controls Goes Here-->
</StackPanel>
```

به عنوان مثال کد زیر، سه کنترل TextBox و یک کنترل Button بر روی StackPanel قرار می دهد.

کد:

```
                                                <StackPanel>
                                    <TextBox Margin="3" Name="txtNum1"></TextBox>
                                    <TextBox Margin="3" Name="txtNum2"></TextBox>
                            <Button  Margin="3" Name="btnSum" Click="btnSum_Click">Get
                                                                            Sum</Button>
                                  <TextBox Margin="3" Name="txtResult"></TextBox>

                                                                        </StackPanel>
```

شکل حاصل از دستورات فوق، مشابه زیر خواهد بود:

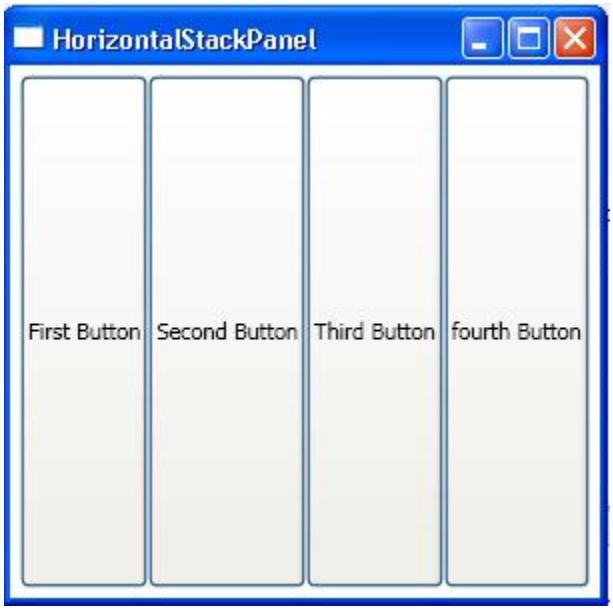


همانطور که اشاره شد، کنترل StackPanel قابلیت چیدن عناصر را به صورتی افقی نیز دارا می باشد. با به کار گیری خاصیت Orientation از این کنترل می توانید، نحوه قرار گیری عناصر را مشخص سازید.
این حاصیت دارای دو مقدار Horizontal و Vertical می باشد. که به ترتیب برای تراز کردن عناصر به صورت افقی و عمودی بر روی StackPanel به کار می رود.
به عنوان مثال در کدزیر، چهار دکمه به صورت افقی قرار گرفته اند:

کد:

```
<StackPanel Margin="5" Orientation="Horizontal" Button.Click="ButtonClick" >
                                                    <Button>First Button</Button>
                                                   <Button>Second Button</Button>
                                                    <Button>Third Button</Button>
                                                   <Button>fourth Button</Button>
                                                                    </StackPanel>
```

در این کد، خاصیت Orientation در تگ آغازین کنرل StackPanel بر روی Horizontal قرار گرفته است



نکته :
مقدار پیش فرض خاصیت Orientation برابر با Vertical می باشد. در واقع اگر خاسیت Orientation را برای StackPanelتنظیم نکنید، عناصر به صورت پشته عمودی قرار خواهد گرفت

هر کنترلی علاوه بر خواص مخصوصی به خودش دارای خواصی می باشد که تقریبا بین همه کنترل ها مشترک هستند. در واقع خواصی در WPF وجود دارد که اکثر کنترل های WPF ، ان خواص را شامل می شوند. این خواص در هر کنترلی عملکردی مشابه خواهد داشت. در بخش بعدی به تعدادی از این خواص اشاره خواهیم کرد.

ادامه کنترل) StackPanel نمونه کد های نوشته شده در این پست و پست قبلی در قالب یک پروژه در آخر همین تاپیک پیوست شده است(.

خواص تراز بندی :

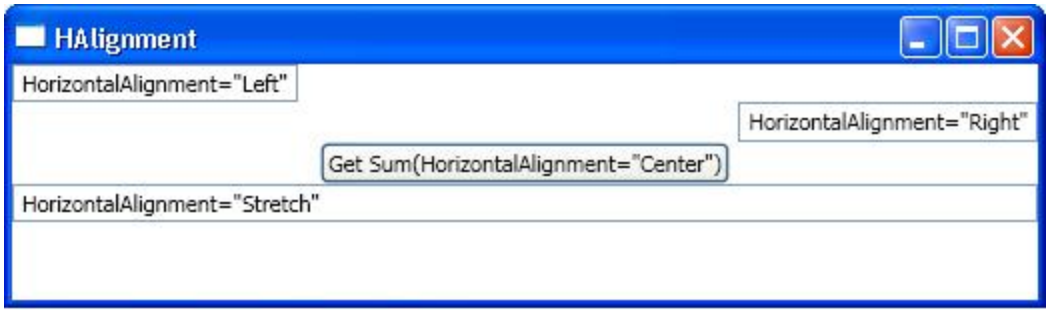
دو خاصیت HorizontalAlignment و VerticalAlignment که در موارد متعددی استفاده می گردند، محل قرار گیری افقی و عمودی کنترل را نسبت به کنترل کانتینر خودش مشخص می کند .

خاصیت: HorizontalAlignment

مقادیر خاصیت HorizontalAlignment عبارتند از:

Left : این مقدار، باعث می شود که کنترل مورد نظر از سمت چپ کنترل پدرش تراز شود.
Right : این مقدار، باعث می شود که کنترل مورد نظر از سمت زاست کنترل پدرش تراز شود.
Center : این مقدار، باعث می شود که کنترل مورد نظر در قسمت وسط کنترل پدرش تراز شود.
Stretch : این مقدار باعث می شود که کنترل تمامی عرض کنترل پدرش را پوشش دهد.

عکس زیر، موارد گفته شده را نشان می دهد:



کدی که برای برنامه فوق نوشته شده است::

کد:

```
<Window x:Class="StackPanel.HAlignment"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
Title="HAlignment" Height="300" Width="300">
<StackPanel>
<TextBox HorizontalAlignment="Left"
Name="txtNum1">HorizontalAlignment="Left"</TextBox>
<TextBox HorizontalAlignment="Right"
Name="txtNum2">HorizontalAlignment="Right"</TextBox>
```

```
                    <Button  HorizontalAlignment="Center" Name="btnSum"
        Click="btnSum_Click">Get Sum(HorizontalAlignment="Center")</Button>
        <TextBox   Name="txtResult">HorizontalAlignment="Stretch"</TextBox>
                                                                </StackPanel>
                                                                    </Window>
```

خاصیت: VerticalAlignment

این خاصیت دارای چهار مقدار زیر می باشد:
Top:که باعث می شود کنترل از سمت بالای کنترل پدر خویش تراز شود.
Bottom : که باعث می شود کنترل از سمت پایین کنترل پدر خویش تراز شود
Center:که باعث می شود کنترل دروسط کنترل پدر خویش تراز شود.
Stretch:که باعث می شود، کنترل از تمامی فضای موجود، استفاده کند.

نکته :

کنترل StackPanel ، به کنترل های فرزند خود به همان مقدار فضا که نیاز دارند، فضا اختصاص می دهد. به همین دلیل اگر دستورات مربوط خاصیت VerticalAlignment را با شکل فوق به کار ببرید، تاثیری در چیدمان کنترل ها نخواهد داشت.

خاصیت:Margin

این خاصیت، فاصله کنترل را از کنترل های اطراف خودش مشخص می کند . این خاصیت دارای چهار مقدار Left،Top، Bottomو Right می باشد.
نحوه مقدار دهی این خاصیت در اسناد XAML به صورت زیر می باشد:

کد:

```
<ElementName ... Margin="5,5,5,5"></ElementName>
```

با کد فوق، عنصری که برای آن خاصیت Margin مشخص شده است، از هر طرف به مقدار 5 واحد با کنترل های اطرافش فاصله خواهد دشت.

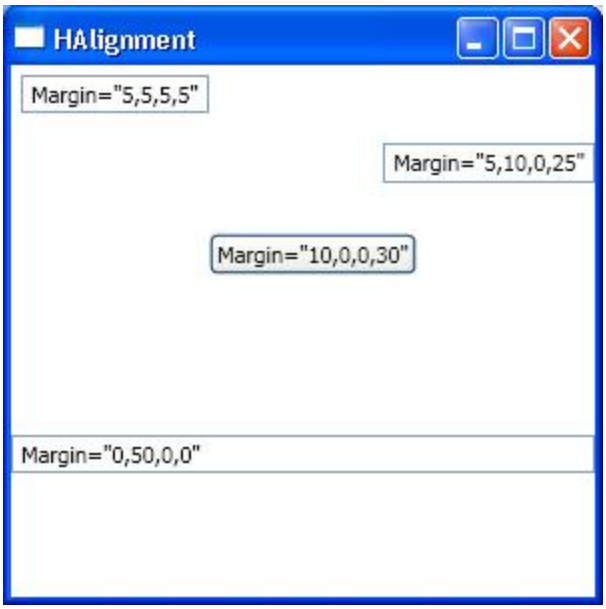
نکته :

اگر مقادیر فاصله ای که می خواهید قرار دهید برای هر چهار طرف یکسان باشد، کافی است به جای کد فوق از کد زیر استفاده کنید:
کد:

```
<ElementName ... Margin="5 "></ElementName>
```

دو قطعه کد فوق یکسان می باشند. در واقع می توانید تنها با قرار دادن یک، هر چهار مقدار این خاصیت را مقدار دهی کنید.

شکل زیر، نمونه ای از نحوه استفاده از این خاصیت را مشخص می کند:



کد مربوط به شکل فوق:
کد:

```
                                                    <StackPanel>
                                <TextBox  HorizontalAlignment="Left" Margin="5"
                                     Name="txtNum1">Margin="5,5,5,5"</TextBox>
                            <TextBox  HorizontalAlignment="Right" Margin="5,10,0,25"
                                 Name="txtNum2">Margin="5,10,0,25"</TextBox>
                        <Button   HorizontalAlignment="Center" Margin="10,0,0,30"
                Name="btnSum" Click="btnSum_Click">Margin="10,0,0,30"</Button>
                                   <TextBox  Name="txtResult" Margin="0,50,0,0"
                                                >Margin="0,50,0,0"</TextBox>
                                                    </StackPanel>
```

خواص سایز :

شش خاصیت زیر برای تنظیم سایز کنترل ها به کار می روند:

خاصیت : Width این خاصیت، عرض کنترل را به صورت صریح مشخص می کند .

خاصیت : Height این خاصیت به صورت صریح، مقدار ارتفاع کنترل را مشخص می کند.

خاصیت :MinHeight این خاصیت مینیمم ارتفاعی را که یک کنترل می تواند اختیار کند را مشخص می کند.

خاصیت :MinWidth این خاصیت مینیمم عرضی را که یک کنترل باید داشته باشد را مشخص می کند.

خاصیت :MaxHeight این خاصیت ماکزیمم ارتفاعی را که یک کنترل می تواند اختیار کند را مشخص می کند.

خاصیت :MaxWidth این خاصیت ماکزیمم عرضی را که یک کنترل باید داشته باشد را مشخص می کند.

نکته:
همانور که گفته شد، تا جایی که امکان پذیر است، نبایستی از خواص Width و Height برای مقدار دهی عرض و ارتفاع کنترل ها استفاده کرد. یکی از مواردی که این موضوع می تواند کارایی برنامه را پایین آورد، زمانی است که بخواهید برنامه خود را Localizable کنید.

کنترل Canvas : (نمونه برنامه را از پایان همین پست دانلود کنید)

این کنترل نیز یکی دیگر از کنترل های کانتینری می باشد که عناصر مختلف می توانند بر روی آن قرار بگیرند. از این کنترل به ندرت در برنامه ها استفاده می شود. به این دلیل که این کنترل، عناصر داخلی خود را بر مبنای مکان آن عنصر که به صورت صریح در خواص آن عنصر ذکر گردیده است، تراز بندی می کند. به همین دلیل در مواقعی که امکان تغییر سایز پنجرها و مقادیر عناصر در زمان اجرای برنامه باشد، استفاده از این کنترل، انتخاب مناسبی نمی تواند باشد.

قبلا اشاره شد که کنترل هایی که درون کنترل های کانتینر قرار می گیرند، بسته به نوع کنترل کانتینر، خواص جدیدی به مجوع خواص آن ها اضافه می شود. کنترل Canvas نیز از این امر مستثنا نیست .
چهارخاصیت **Left** ، **Top**، **Bottom** و **Right** ، به کنترل های فرزندی که درون کنترل Canvas قرار بگیرند، اضافه خواهد شد که توسط این خواص، محل کنترل فرزند بر روی کنترل Canvas تعیین می شود.

به نمونه کد زیر دقت کنید:

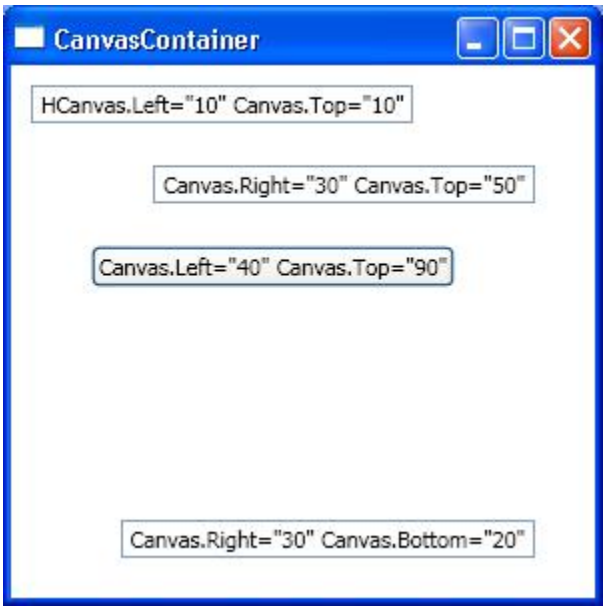
کد:

```
<Window x:Class="StackPanel.CanvasContainer"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
Title="CanvasContainer" Height="300" Width="300">
<Canvas>
<TextBox Canvas.Left="10" Canvas.Top="10"
Name="txtNum1">HCanvas.Left="10" Canvas.Top="10"</TextBox>
```

```
                                    <TextBox Canvas.Right="30" Canvas.Top="50"
                            Name="txtNum2">Canvas.Right="30" Canvas.Top="50"</TextBox>
                               <Button Canvas.Left="40" Canvas.Top="90"  Name="btnSum"
                          Click="btnSum_Click">Canvas.Left="40" Canvas.Top="90"</Button>
                      <TextBox  Canvas.Right="30"  Canvas.Bottom="20" Name="txtResult"
                                        >Canvas.Right="30"  Canvas.Bottom="20"</TextBox>
                                                                              </Canvas>
                                                                              </Window>
```

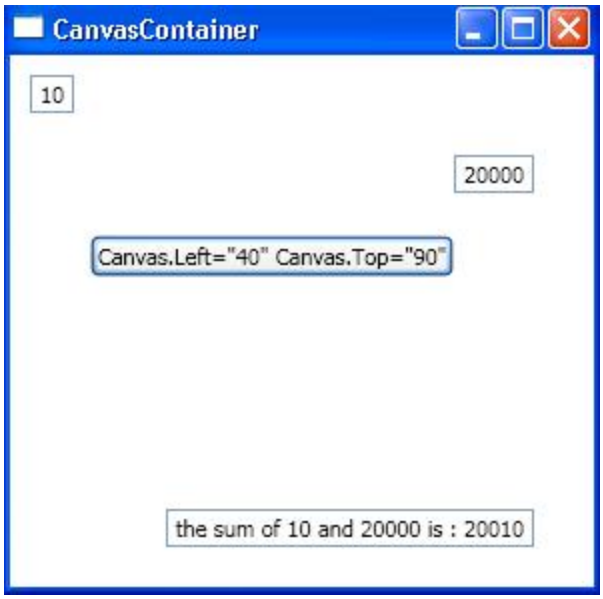
همانطور که می بینید، مکان هر کنترلی درون کنترل Canvas با خواص پیوست شده کLeft ، Top، Rightو Bottom مشخص شده است.

شکل نهایی حاصل از اجرای این کد به صورت زیر خواهد بود:



نکته:

دقت داشته باشید، که در این حالت، چون سایز کنترل ها پیش فرض بر روی Stretch نیست (بر خلاف زمانی که منترل ها بر روی StackPanel قرار داشتند)، با تغییر محتوای TextBox ها، سایز آن ها نیز تغییر پیدا می کند.



به عنوام مثال اگر کاربر در دو TextBox بالایی مقادیر 10 و 20000 را وارد کند، TextBoxها تغییر اندازه داده و به شکل زیر در خواهند آمد:

همچنین اگر مقادیر TextBox ها بیش از عرض فعلی آن ها باشد، عرض TextBox ها زیاد شده تا بتواند تمامی مقادیر را در خود جای دهد.
برای پیشگیری از تغییر سایز textbox ها، می توانید از خواص MaxWidth و MinWidth این عناصر استفاده کنید .
برای ثابت نگه داشتن طول textbox می توانید خواص MaxWidth و MinWidth را با مقادیر یکسان، مقدار دهی کنید .

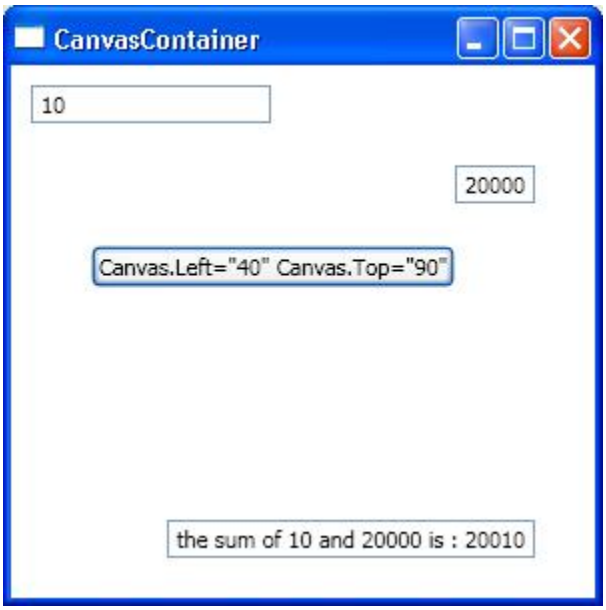
در کد زیر، که با تغیر در کد قبلی به وجود امده است، TextBoxبالایی، در اندازه 120 ثابت شده است. این به این معنی است که با تغییر محتویات داخل TextBox طول TextBox ثابت می ماند.

کد:
کد:

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

```
                <TextBox  Canvas.Left="10" Canvas.Top="10"   Name="txtNum1"
    MaxWidth="120" MinWidth="120">HCanvas.Left="10" Canvas.Top="10"</TextBox>
                                <TextBox   Canvas.Right="30" Canvas.Top="50"
                Name="txtNum2">Canvas.Right="30" Canvas.Top="50"</TextBox>
                   <Button Canvas.Left="40" Canvas.Top="90"  Name="btnSum"
             Click="btnSum_Click">Canvas.Left="40" Canvas.Top="90"</Button>
          <TextBox  Canvas.Right="30"  Canvas.Bottom="20" Name="txtResult"
                              >Canvas.Right="30"  Canvas.Bottom="20"</TextBox>
                                                                  </Canvas>
                                                                  </Window>
```

نتیجه اجرای حاصل از کد فوق به صورت زیر خواهد بود:



همانطور که در شکل فوق مشاهده می کنید، تکست باکس بالایی با تغییر مقدار آن به 10 ، تغییری در شول آن صورت نگرفته است.

خاصیت: ZIndex

توسط این خاصیت می توانید نحوه چیدمان عناصری را که بر روی هم قرار گرفته اند را مشخص کنید. هر کنترلی که مقدار **ZIndex** آن**بزگتر** باشد، **بر روی** کنترل هایی که مقدار ZIndex آن ها کوچکتر است قرار خواهد گرفت .
به نمونه کد زیر دقت کنید:

کد:

```
<Window x:Class="StackPanel.ZIndexProp"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
Title="ZIndexProp" Height="300" Width="300">
<Canvas>
<Button  Background="Green" Canvas.Left="60" Canvas.Top="24"
MinHeight="50" Canvas.ZIndex="0" >My ZIndex Value IS 0</Button>
<Button  Background="Red" Canvas.Left="147" Canvas.Top="60"
MinHeight="50" Canvas.ZIndex="1">My ZIndex Value IS 1</Button>
<Button  Background="Blue"  Canvas.Left="129" Canvas.Top="98"
MinHeight="50" Canvas.ZIndex="2" >My ZIndex Value IS 2</Button>
<Button Background="Yellow" Canvas.Left="30" Canvas.Top="60"
MinHeight="50" Canvas.ZIndex="3" >My ZIndex Value IS 3</Button>



</Canvas>
</Window>
```

در این کد که شامل یک کنترل Canvas و چهار کنترل Button بر روی آن می باشد، دکمه ها دارای خاصیت ZIndex می باشند. نتیجه اجرای کد فوق رادر شکل زیر مشاهده می کنید.

همانطور که در شکل فوق نیز مشخص است، دکمه ای که با رنگ زرد مشخص شده است، به دلیل اینکه دارای مقدار ZIndex با لاتری نسبت به دکمه های دیگر دارد، بر روی تمامی دکمه ها قرار گرفته است .
به دلیل مشابهی، دکمه سبز رنگ در زیر تمامی دکمه ها قرار گرفته است.
دکمه آبی رنگ دارای مقدار ZIndex برابر با 2 می باشد. به عمین دلیل، قسمتی از آن بالاتر از دکمه قرمز رنگ قرار گرفته است (چون دکمه قرمز رنگ مقدار ZIndex کتری نسبت به دکمه آبی رنگ دارد) و قسمتی از آن زیر دکمه زرد رنگ قرار گرفته است.(به این دلیل که دکمه زرد رنگ دارای خاصیت ZIndex بالاتری نسبت به دکمه آبی رنگ دارد(
فایل های ضمیمه

Canvas.rar (41.5 کیلوبایت, 717 دیدار)

کنترل:DockPanel

در دات نت 1 x.و 2.0 با خاصیتی به نام Dock آشنا شدید. این خاصیت برای هر کنترلی که تنظیم می شد (می شود)، با عث می شد (می شود) که کنترل مورد نظر بر اساس مقداری که برای خاصیت Dock آن تنظیم شد است، به یکی از گوشه های کنترل پدر خودش وصل شود. به عنوان مثال اگر کنترلی دارای مقدار Top برای خاصیت Dock باشد، واین کنترل مستقیما بر روی فرم قرار گرفته باشد، این کنترل به گوشه بالایی فرم متصل می شود. در نتیجه با تغییر عرض فرم، این کنترل به صورت اتوماتیک نیز تغییر سایز می دهد و عرض خودش را با عرض فرم مجددا تنظیم می کند.از این خاصیت در برنامه ها، زیاد استفاده می شود، به عنوان مثال کنترل Status Bar که معمولا در پایین فرم، Dockمی شود، ویا منوی برنامه که در گوشه بالایی فرم Dock می شود از این نمونه هستند.

کنترل DockPanel نیز عملکری مشابه با عملکر خاصیت Dock دارد. با این تفاوت که قدرت بسیار بالاتر و امکانات بسیار بیشتری را در اختیار شما قرار می دهد.
هر کنترلی که در کنترل DockPanel قرار بگیرد، خاصیتی به نام Dock به آن پیوست خواهد شد. این خاصیت دارای چهار مقدار**Left، Top ، Bottom**و **Right** می باشد. توسط این مقادیر می توانید کنترل های خود را در کنترل DockPanelتنظیم نمایید.

به نمونه کد زیر دقت کنید:

کد:

```
<Window x:Class="DockPanel.DockPanelContainer"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
Title="DockPanelContainer" Height="300" Width="300">
<DockPanel>
<Button DockPanel.Dock="Right">"Right"</Button>
<StackPanel Orientation="Horizontal" DockPanel.Dock="Top">
<TextBlock Margin="3">sample TextBlock</TextBlock>
<Button HorizontalAlignment="Right" Margin="3">button</Button>
```

```
                                                    </StackPanel>

                                                <Menu DockPanel.Dock="Bottom" >
                                                      <MenuItem Header="Item0">
                                                    <MenuItem Header="Item00">
                                            <MenuItem Header="Item000"></MenuItem>
                                                                    </MenuItem>
                                             <MenuItem Header="Item01"></MenuItem>
                                             <MenuItem Header="Item02"></MenuItem>
                                             <MenuItem Header="Item03"></MenuItem>
                                                                    </MenuItem>
                                                  <MenuItem Header="Item1">
                                                <MenuItem Header="Item10">
                                            <MenuItem Header="Item100"></MenuItem>
                                                                    </MenuItem>
                                             <MenuItem Header="Item11"></MenuItem>
                                             <MenuItem Header="Item12"></MenuItem>
                                             <MenuItem Header="Item13"></MenuItem>
                                                                    </MenuItem>
                                                                        </Menu>
                                                <TextBox TextWrapping="Wrap" MaxWidth="80"
        DockPanel.Dock="Left">DockPanel.Dock="Left"</TextBox>
      <TextBox TextAlignment="Center" VerticalAlignment="Center">Dock value
                                                                Is Fill</TextBox>
                                                             </DockPanel>
                                                                </Window>
```

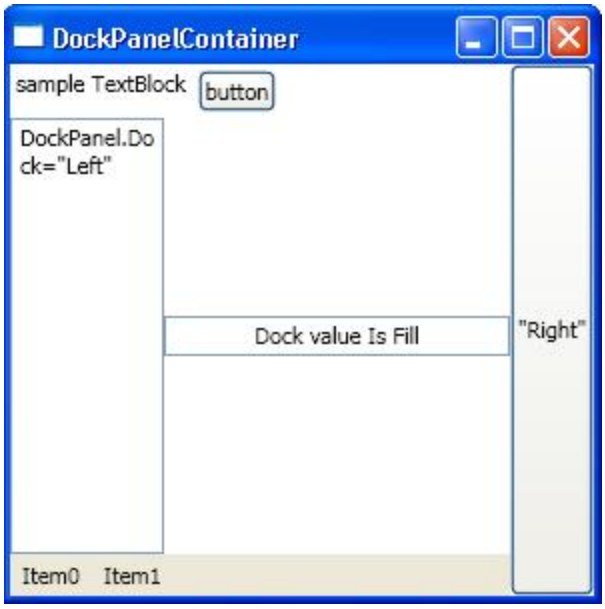
در بالاترین، سطح از این کنترل ها، کنترل DockPanel قرار گرفته است. پنج کنترل دیگر به عنوان فرزندان مستقیم کنترل DockPanel قرار گرفته اند که بعضی از این کنترل ها، خود نیز شامل چندین کنترل دیگر می باشند. تمامی کنترل های فرزندف دارای خاصیت پیوست شده Dock می باشند. به عنوان مثال قطعه کد زیر:

کد:

```
<Button DockPanel.Dock="Right">"Right"</Button>
```

دکمه ای را به عنوان فرزند کنترل DockPanel تعریف می کند که در سمت راست خودش قرار گرفته است.
دستور **DockPanel.Dock="Right"**باعث می شود که کنترل، به گوشه سمت راست کنترل DockPanel قرار گیرد.

نتیجه حاص از اجرای کد فوق در شکل زیر مشخص شده است:



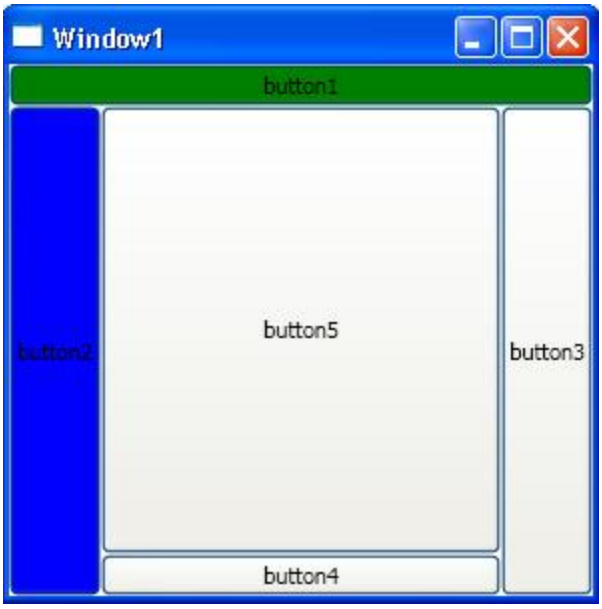
ترتیب در Dock کردن کنترل ها:

ترتیب تنظیم خاصیت Dock مربوط به کنترل ها در کنترل DockPanel مهم می باشد. به عنوان مثال دو قعطه کد زیر را نظر بگیرید:

کد:

```
<Window x:Class="DockPanel.Window1"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
Title="Window1" Height="300" Width="300">
<DockPanel>
<Button Background="Green" DockPanel.Dock="Top">button1</Button>
<Button Background="Blue" DockPanel.Dock="Left">button2</Button>
<Button DockPanel.Dock="Right">button3</Button>
<Button DockPanel.Dock="Bottom">button4</Button>
<Button >button5</Button>
</DockPanel>
</Window>
```

شکل حاصل از اجرای قطعه کد فوق مشابه زیر خواهد بود:

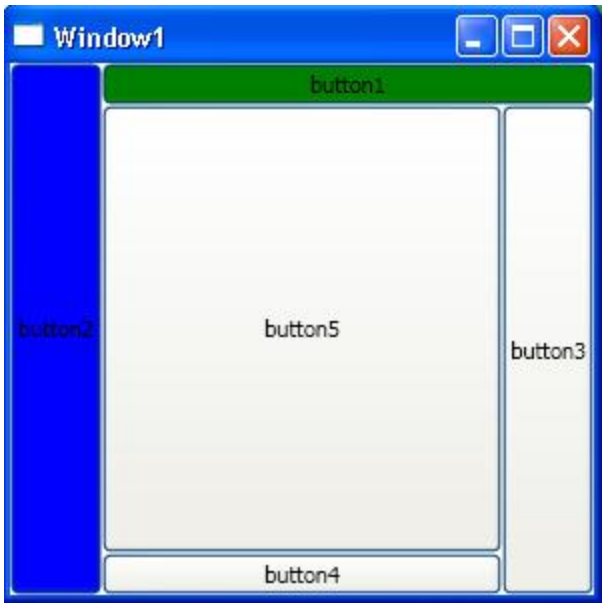


در کد فوق، دکمه ای که با رنگ سبز مشخص شده است، به دلیل اینکه قبل از دکمه آبی رنگ تعریف شده است، کل فضای گوشه بالایی را به خود اختصاص داده است. حال اگر جای تعریف این دو دکمه (button1 و button2) را در کد عوض کنیم، یعنی داشته باشیم:

کد:

```
                                        <Window x:Class="DockPanel.Window1"
            xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
                      xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
                                  Title="Window1" Height="300" Width="300">
                                                          <DockPanel>
          <Button Background="Blue" DockPanel.Dock="Left">button2</Button>
          <Button Background="Green" DockPanel.Dock="Top">button1</Button>

                                <Button DockPanel.Dock="Right">button3</Button>
                              <Button DockPanel.Dock="Bottom">button4</Button>
                                            <Button >button5</Button>
                                                       </DockPanel>
                                                          </Window>
```

نتیجه حاصل مشابه زیر خواهد بود:



مقدار Fill در خاصیت : Dock

خاصیت Dock از کنترل DockPanel دارای مقدار Fill نمی باشد .در واقع به آن نیاز ندارد. دلیل این امر به نحوه چیدمان کنترل ها در کنترل DockPanel مربوط می شود. کنترل DockPanel همواره سعی بر دادن کل فضای موجود به کنترل های فرزند خودش می باشد. به عنوان مثال اگر کنترل DockPanel فقط دارای یک کنترل Button باشد، این دکمه کل فضای موجود بر روی کنترل DockPanel را به خود اختصاص می دهد حتی اگر خاصیت Dock مربوط به کنترل را مثلا بر روی Left قرار دهید..(اگر مقادیر عرض و ارتفاع برای کنترل Button تعریف نشده باشد(

حال اگر دکمه دیگری با خاصیت Dock برابر با Right به کنترل DockPanel اضافه کنید، فضای موجود بر روی کنترل DockPanelبه دوقسمت تقسیم بندی می شوند. در این حالت مقدار فضایی که دکمه اول احتیاج دارد، در اختیارش قرار داده می شود و بقیه فضای موجود به دکمه دوم اختصاص داده می شود.

این تقسیم بندی برای کنترل های فرزند دیگر (در صورت وجود) نیز انجام می شود تا نهایتا کل فضای موجود بر روی کنترل DockPanel بین کلیه کنترل های فرزندش تقسیم شود.

خاصیت: LastChildFill

ذکر این نکته ضروری است که تمامی مطالب گفته شده در بخش قبلی (ترتیب در Dock کردن کنترل ها) زمانی درست می باشند که خاصیت LastChildFill از کنترل DockPanel بر روی True تنظیم گردد. اگر این خاصیت True باشد که مقدار پیش فرض آن نیز True می باشد، آخرین کنترلی که بر روی کنترل DockPanel قرار می گیرد، کل

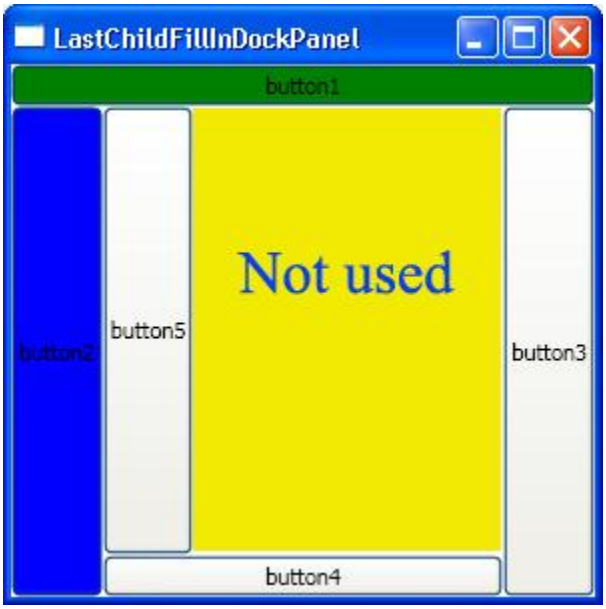
فضای باقی مانده از کنترل DockPanel را به خود اختصاص می دهد. در غیر این صورت همه کنترل ها به اندازه مقدار فضایی که نیاز داشته باشند از کنترل DockPanel گرفته و بقیه فضای باقی مانده (اگر موجد باشد) بدون استفاده باقی خواهد ماند.

به عنوان مثال اگر در کد فوق دستور **LastChildFill="False"** را اضافه کنیم:

کد:

```
                                        DockPanel LastChildFill="False">
        <Button Background="Green" DockPanel.Dock="Top">button1</Button>
        <Button Background="Blue" DockPanel.Dock="Left">button2</Button>
                            <Button DockPanel.Dock="Right">button3</Button>
                          <Button DockPanel.Dock="Bottom">button4</Button>
                                              <Button >button5</Button>
                                                                </DockPanel>
```

نتیجه اجرا به صورت شکل زیر خواهد بود:



در شکل فوق، آخرین کنترلی که به کنترل DockPanel اضافه شده است، button5می باشد. کنترل DockPanel به این دکمه به اندازه مورد نیازش (اندازه ای که بتواند متن داخل دکمه مشخص باشد) فضا می دهد و بقیه فضای موجود (کادر زرد رنگ) بدون استفاده باقی خواهد ماند.

فایل های ضمیمه

DockPanel.rar (49.2 کیلوبایت, 688 دیدار)

کنترل: WrapPanel

کنترل WrapPanel نیز یکی از کنترل های کانتینر می باشد. این کنترل در طراحی واسط کاربری شما نقش زیادی نمی تواند بازی کند. در واقع مواردی که از این کنترل می توان استفاده کرد محدود و در بعضی از کاربرد های خاص به کار می رود.

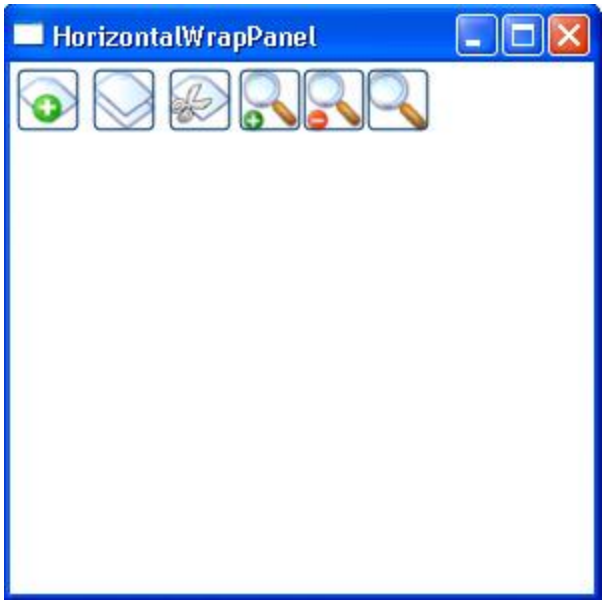
کنترل WrapPanel عناصر فرزند خود را به دوصورت می تواند تراز بندی کنید. این امر بستگی به خاصیت **Orientation** از این کنترل دارد. اگر این خاصیت بر روی **Horizontal** باشد (حالت پیش فرض horizontal می باشد)، عناصر به صورت سطری و در داخل اولین سطر از این کنترل قرار میگیرند. چنانچه مقدار فضای مورد نیاز برای کنترل های فرزند، بیش از فضای موجود بر روی یک سطر باشد، عناصر فرزند به صورت اتواتیک به سطر بعدی شیفت داده می شوند. این عمل آن قدر تکرار می گردد تا تمامی عناصر بر روی کنترل WrapPanel قرار داده شوند. به کد زیر توجه کنید:

کد:

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

<WrapPanel>
<Button Margin="3" MinWidth="32" MaxWidth="32" MinHeight="32">
<Button.Background>
<ImageBrush ImageSource="./Images/add-file.png"></ImageBrush>
</Button.Background>
</Button>
<Button Margin="3" MinWidth="32" MaxWidth="32" MinHeight="32">
<Button.Background>
<ImageBrush ImageSource="./Images/copy.png"></ImageBrush>
</Button.Background>
</Button>
<Button Margin="3" MinWidth="32" MaxWidth="32" MinHeight="32">
<Button.Background>
```

```
                <ImageBrush ImageSource="./Images/cut.png"></ImageBrush>
                                                    </Button.Background>
                                                        </Button>
            <Button Height="32" MaxWidth="32" MinHeight="32" MinWidth="32"
                                                                Width="32">
                                                     <Button.Background>
                            <ImageBrush ImageSource="./Images/zoom+.png" />
                                                    </Button.Background>
                                                        </Button>
            <Button Height="32" MaxWidth="32" MinHeight="32" MinWidth="32"
                                                                Width="32">
                                                      <Button.Background>
                            <ImageBrush ImageSource="./Images/zoom-.png" />
                                                    </Button.Background>
                                                        </Button>
            <Button Height="32" MaxWidth="32" MinHeight="32" MinWidth="32"
                                                                Width="32">
                                                     <Button.Background>
                          <ImageBrush ImageSource="./Images/search.png" />
                                                    </Button.Background>
                                                        </Button>
                                                    </WrapPanel>

                                                        </Window>
```

این قطعه کد مربوط به پنجره ای است که یک کنترل WrapPanel به عنوان کنترل کانتینر آن قرار داده شده است. درون این کنترل، شش دکمه قرار داده شده است که هر یک دارای خواص مربوط به خودش می باشد. نتیجه حاصل از کد فوق در شکل زیر نشان داده شده است:



همانطور که در شکل نیز مشخص است، عناصر روی کنترل WrapPanel به صورت سطری قرار گرفته اند. حال اگر در زمان اجرا، عرض پنجره فوق کم شود، بعضی از عناصر به سطر بعدی در کنترل WrapPanel شیفت داده می شوند:

حال اگر خاصیت **Orientation** این کنترل را بر روی **Vertical** تنظیم کنید، عناصر به صورت ستونی تراز می شوند، ابتدا ستون اول و چنانچه فضای کافی برای قرار گیری عناصر بر روی ستون اول موجود نباشد، بقیه عناصر به ستون

بعدی شیفت داده می شوند.

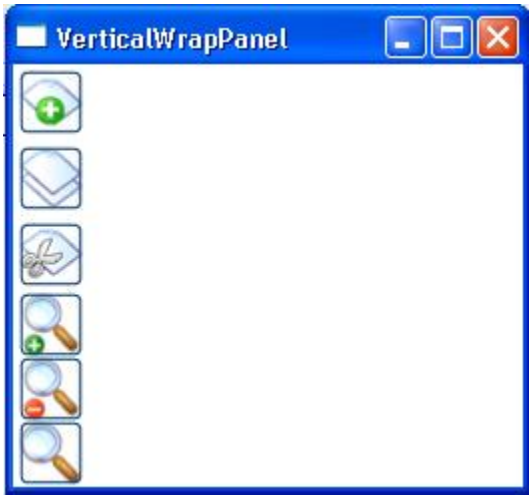
برای اعمال این تغییر کافیست تگ آغازین کنترل WrapPanel را به صورت زیر تغییر دهید:

کد:

```
                                <WrapPanel Orientation="Vertical">
            <Button Margin="3" MinWidth="32" MaxWidth="32" MinHeight="32">
                                        <Button.Background>
             <ImageBrush ImageSource="./Images/add-file.png"></ImageBrush>
                                        </Button.Background>
                                                </Button>
            <Button Margin="3" MinWidth="32" MaxWidth="32" MinHeight="32">
                                        <Button.Background>
                 <ImageBrush ImageSource="./Images/copy.png"></ImageBrush>
                                        </Button.Background>
                                                </Button>
            <Button Margin="3" MinWidth="32" MaxWidth="32" MinHeight="32">
                                        <Button.Background>
                  <ImageBrush ImageSource="./Images/cut.png"></ImageBrush>
                                        </Button.Background>
                                                </Button>
            <Button Height="32" MaxWidth="32" MinHeight="32" MinWidth="32"
                                                        Width="32">
                                        <Button.Background>
                        <ImageBrush ImageSource="./Images/zoom+.png" />
                                        </Button.Background>
                                                </Button>
            <Button Height="32" MaxWidth="32" MinHeight="32" MinWidth="32"
                                                        Width="32">
                                        <Button.Background>
                        <ImageBrush ImageSource="./Images/zoom-.png" />
                                        </Button.Background>
                                                </Button>
            <Button Height="32" MaxWidth="32" MinHeight="32" MinWidth="32"
                                                        Width="32">
                                        <Button.Background>
                      <ImageBrush ImageSource="./Images/search.png" />
                                        </Button.Background>
                                                </Button>
```

```
</WrapPanel>
```

دو شکل زیر نمایی از اجرای کد قبل را با تغییری که در تگ آغازین کنترل WrapPanel داده شده است را نشان می دهد.



یکی از کاربر هایی که از WrapPanel ها می تواند در آن بهره جست، ایجاد کنترل هایی مانند Tool strip می باشد) .البته ابزار های مجزایی برای toolbar ها نیز موجود است .(

نکته :

خاصیت دیگری که اکثر کنترل های WPF ، از جمله کنترل WrapPanel دارامی باشند، خاصیت **Flow Direction** می

باشد .
این خاصیت دارای دو مقدار **Left To Right** و **Right To Left** می باشد که مقدار پیش فرض آن Left To Right می باشد. این خاصیت جهت تراز بندی کنترل ها را نشان می دهد. بری برخی از زبان ها از جمله زبان فارسی، این خاصیت بسیار پر کاربرد می باشد.

به عنوان مثال می توانیدبا اضافه کردن دستور

کد:

```
FlowDirection="RightToLeft"
```

یک تگ آغازین کنترل WrapPanel ، ستون آغازین را برای قرار گیری کنترل ها را از سمت راست به چپ تعیین کنید.

موفق باشید
فایل های ضمیمه

WrapPanel.rar (112.1 کیلوبایت, 794 دیدار)

بخش سوم: چیدمان و طراحی کنترل ها (قسمت هشتم)

کنترل:UniformGrid

این کنترل ظاهری شبیه به عنصر Gird که در بخش بعدی توضیح داده خواهد شد) دارد. این کنترل به تعدادی سطر و ستون با اندازه های یکسان تقسیم بندی می شود. عناصر فرزند این کنترل می توانند در هریک از این سلول ها قرار بگیرند .
سلول های حاصل از ایجاد این کنترل، همگی دارای اندازه های یکسان می باشند.

نحوه استفاده از این کنترل به صورت زیر می باشد:
کد:

```
<UniformGrid Roes="5" Columns="5">
.
.
.
</UniFormGrid>
```

که شامل تعریف خود کنترل، تعداد سطر ها و تعداد ستون های آن می باشد.

این کنترل برای موارد خاصی به کار برده می شود و به ندرت در طراحی واسط های برنامه شما به کار برده می شود. به عنوان مثال شکل زیر یک جدول ضرب 5*5 را نشان می دهد.

کد:
کد:

```
<Window x:Class="UniformGrid.UniformGridContainer"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
Title="UniformGridContainer" Height="200" Width="200">
<Window.Resources>
<SolidColorBrush x:Key="btnBack" Color="Coral"></SolidColorBrush>
</Window.Resources>
<UniformGrid Rows="6" Columns="6">
<Button  Background="Red" >*</Button>
<Button  Background="{StaticResource btnBack}">1</Button>
<Button Background="{StaticResource btnBack}">2</Button>
<Button Background="{StaticResource btnBack}">3</Button>
<Button Background="{StaticResource btnBack}">4</Button>
```

```
                    <Button Background="{StaticResource btnBack}">5</Button>
                    <Button Background="{StaticResource btnBack}">1</Button>
                                                        <Button >1</Button>
                                                        <Button >2</Button>
                                                        <Button >3</Button>
                                                        <Button >4</Button>
                                                        <Button >5</Button>
                    <Button Background="{StaticResource btnBack}">2</Button>
                                                        <Button >2</Button>
                                                        <Button >4</Button>
                                                        <Button >6</Button>
                                                        <Button >8</Button>
                                                       <Button >10</Button>
                    <Button Background="{StaticResource btnBack}">3</Button>
                                                        <Button >3</Button>
                                                        <Button >6</Button>
                                                        <Button >9</Button>
                                                      <Button >12</Button>
                                                      <Button >15</Button>
                <Button Background="{StaticResource btnBack}" >4</Button>
                                                        <Button >4</Button>
                                                        <Button >8</Button>
                                                      <Button >12</Button>
                                                      <Button >16</Button>
                                                      <Button >20</Button>
                <Button  Background="{StaticResource btnBack}">5</Button>
                                                        <Button >5</Button>
                                                      <Button >10</Button>
                                                      <Button >15</Button>
                                                      <Button >20</Button>
                                                      <Button >25</Button>
                                                            </UniformGrid>
                                                                 </Window>
```

نکته:

در این کنترل، صراحتا نمی توانید مشخص کنید که کدام کنترل در چه سلولی قرار بگیرد. در واقع سلول هر کنترل بر اساس ترتیبی که آن کنترل در کنترل های فرزند کنترل UniformGrid دارد، مشخص می شود.

فایل های ضمیمه

UniformGrid.rar 37.2) کیلوبایت, 562 دیدار(

بخش سوم: چیدمان و طراحی کنترل ها (قسمت نهم(

سورس مربوط به کنترل گرید، در پایان مبحث گرید ارائه می گردد.

کنترل: Grid

کنترل گرید یکی از قوی ترین و پرکاربرد ترین کنترل های کانتینر می باشد. به طور حتم می توان گفت که در بیش از 90 درصد موارد، شما از این کنترل برای طراحی واسط کاربری برنامه های خود استفاده خواهید کرد. این کنترل دارای خواص زیادی می باشد که شما را در طراحی واسط های کاربری پیچیده یاری می کند.
همانطور که تاکنون نیز شاهد بوده اید، این کنترل به صورت پیش فرض، کنترل کانتینر اصلی در هر پنجره ای قرار داده شده است که در مثال های قبلی ما ر حسی نیاز آن را تغییر داده ایم و متناسب با نیاز خود آن را تعریف کرده ایم.
مهمترین خواص کنترل گرید، **خواص مربوط به تعاریف سطر و ستون آن می باشد .**با تعریف سطر ها وستون ها، می توانید مجموعه ای از سلول ها را در این کنترل به وجود آورید.
عناصری که بر روی این کنترل قرار میگیرند، هر یک می توانند در یک سلول و یا بر حسب تعریفی که شما برای آن مشخص می کنید، در دو و یا بیش از دو سلول قرار گرید .
تعریف کنترل گرید در ساده ترین حالت، (تنها دارای یک سطر و یک ستون) به صورت زیر می باشد.

کد:

```
<Grid>
<!--Controls goes here-->
</Grid>
```

تعریف فوق که در ابتدای ایجاد هر پنجره جدیدی،د رون کد XAML آن پنجره مشاهده می کنید، یک کنترل گرید دارای یک سطر و ستون برای شما ایجاد می کند. همانطور که قبلا نیز اشاره شد، اکثر کنترل ها در حالت پیش فرض، دارای خواص **HorizontalAlignment** و **VerticalAlignment**هستند که به صورت پیش فرض دارای مقدار Stretchمی باشند. با توجه به این نکته چنانچه شما یک دکمه مانند Button را به توسط دستور زیر:
کد:

```
<Button>sample Button</Button>
```

به کنترل گرید تعریف شده در کد قبل اضافه کنید، کل فضای گرید به کنترل Sample Button تعلق خواهد گرفت .

این موضوع به این دلیل است که کنترل گرید سعی در اختصاص دادن کل فضای موجود به کنترل های درونی آن می باشد و نیز کنترل Button همانطور که گفته شد، دارای خواص HorizontalAlignment و VerticalAlignment با مقدار

Stretchمی باشد که باعث می شود، از حداکثر فضای موجود بر روی کنترل والد خویش که به آن اختصاص داده شده است استفاده نماید. کد زیر همراه با شکل نشان دهنده مطالب گفته شده می باشد.

کد:

```
<Window x:Class="Grid.Window1"


xmlns="http://schemas.microsoft.com/winfx/2006/xaml

/presentation"


xmlns:x="http://schemas.microsoft.com/winfx/2006/xa

ml"
Title="Window1" Height="300" Width="300">
<Grid>
<Button>sample Button</Button>
</Grid>
</Window
```

همانطور که در شکل فوق مشاهده می شود، کنترل Button کل فضای موجد گرید و در نتیجه کل فضای پنجره را در اختیار گرفته است. (زیرا کنترل گرید نیز کل فضای پنجره را در اختیار گرفته است(

بخش سوم: چیدمان و طراحی کنترل ها (قسمت دهم(

ادامه کنترل گرید:

سطر ها وستون ها در کنترل گرید:

از شکل فوق، می توان نتیجه گرفت که کنترل گرید به آن صورتی که مطرح شد، دارای کارایی بالایی نمی باشد. زمانی این کنترل می تواند به طور موثر و کارا در تولید واسط های کاربری شما موثر واقع شود که بتوانید برای آن سطر ها و ستون های دلخواه خود را تعریف کنید و از آن ها به صورت موثر و دلخواه استفاده نمایید.
با تعریف سطر ها و ستون ها برای کنترل گرید، می توانید آن را به تعداد سلول دلخواه خود تقسیم کرده و کنترل های مورد نظر خود را درون آن سلول ها قرار دهید.
هر کنترلی می توانید درون یک یا چند سلول قرار بگیرد و هر سلول با استفاده از کنترل های کانتینر می تواند شامل یک یا چند کنترل باشد.
در قسمت قبلی و در زمانی که کنترل **UniformGrid** را توضیح می دادم، اشاره شکردم که در آن کنترل نمی توانید بر محل قرار گیری کنترل های فرزند، نظارت داشته باشید و سلول های دلخواه خود را به کنترل های دلخواه خود، نسبت دهید.
در واقع این کنترل UniformGrid است که برای شما تصمیم می گیرید و کنترل های شما را بر اساس ترتیب تعریف شده آ ن ها در سلول های خود قرار می دهد.
اما در کنترل Grid همه چیز در اختیار شما خواهد بود. می توانید مشخص کنید که کدام کنترل در چه سلولی قرار گیرد، نیز مشخص کنید که یک کنترل تا چند سطر و ستون را می تواند احاطه کند و امکانات دیگری که در قسمت های بعدی به آن ها اشاره خواهد شد.

قرار گیری کنترل ها در کنترل گرید:

برای قرار دادن کنترل های خود در ون یک کنترل گرید، دو مرحله زیر را بایستی انجام دهید:

-1مشخص کردن تعداد سطر ها و ستون های گرید :

توسط خواصی که برای این امر اختصاص داده شده است، می توانید تعداد سطر ها و ستون ها را در کنترل گرید خود مشخص کنید.

-2مشخص کردن سطر و ستون دلخواه برای کنترل های فرزند :

همانطور که قبلا نیز اشاره شد، هر کنترل فرزندی بر اساس نوع کنترل والد خویش، دارای خواص جدیدی می شود که آن ها را خواص پیوست شده نامیدیم. این امر برای کنترل هایی که کنترل گرید به عنوان کنترل والد آن ها می باشد نیز مستثنا نمی باشد. دو خاصیت **Row**و **Column** از جمله خواص پیوست شده به کنترل های فرزند، کنترل گرید می باشند که توسط آن ها می توانید محل قرار گیری کنترل فرزند را مشخص نمایید. به عنوا مثال، کنرلی که

دارای مقدار **Row = 2** و **Column = 3** می باشد، درون سطر دوم و ستون سوم از کنترل گرید قرار خواهد گرفت.

سطر ها در کنترل گرید:

کنترل گرید دارای خاصیتی به نام **RowDefinitions** می باشد که توسط آن می توانید تعداد سطر دلخواه برای کنترل گرید تعریف نمایید. به قطعه کد زیر توجه فرمایید:
کد:

```
<Grid>
<Grid.RowDefinitions>
<RowDefinition></RowDefinition>
<RowDefinition></RowDefinition>
<RowDefinition></RowDefinition>
</Grid.RowDefinitions>

</Grid>
```

در شکل فوق، سه سطر برای کنترل گرید تعریف شده است. تعاریف سطر ها بین دو تگ آغازین **<Grid.RowDefinitions>**، و تگ پایانی **</Grid.RowDefinitions>** قرار می گیرند .
هر سطر توسط دستور **<RowDefinition></RowDefinition>** مشخص می شود. در واقع هر دستور **<RowDefinition></RowDefinition>**بیانگر یک سطر به عنوان سطر های گرید می باشد. در نتیجه تعریف فوق، سه سطر برای کنترل گرید ایجاد می کند.

هر سطر در کنترل گرید دارای خواص بسیاری است که رفتار آن سطر را در مقابل کنترل هایی که درون آن سطر قرار خواهند گرفت مشخص می کند .

از میان این خواص، خواص**Height** ، **MaxHeight** و **MinHeight** از اهمیت بالاتری بر خوردار هستند. تعاریف مربوط به خواص**MaxHeight** و **MinHeight** در بخش های قبلی به تفضیل گفته شد .
خاصیت **Height** دارای سه مقدار مختلف می تواند باشد که کمی جلوتر در بخش "تعادل **در اندازه سطر ها و ستون ها** "توضیح داده خواهند شد.

ستون ها در کنترل گرید:

کنترل گرید دارای خاصیتی به نام **ColumnDefinitions** می باشد که توسط آن می توانید تعداد ستون های دلخواه برای کنترل گرید تعریف نمایید. به قطعه کد زیر توجه فرمایید:
کد:

```
<Grid>
<Grid.ColumnDefinitions>
<ColumnDefinition></ColumnDefinition>
<ColumnDefinition></ColumnDefinition>
<ColumnDefinition></ColumnDefinition>
</Grid.ColumnDefinitions>
```

```
</Grid>
```

در شکل فوق، سه ستون برای کنترل گرید تعریف شده است. تعاریف ستون ها بین دو تگ آغازین **<Grid.ColumnDefinitions>**، و تگ پایانی **</Grid.ColumnDefinitions>** قرار می گیرند .
هرستون توسط دستور **<ColumnDefinition></ColumnDefinition>** مشخص می شود. در واقع هر دستور **<ColumnDefinition></ColumnDefinition>**بیانگر یک ستون به عنوان ستون های گرید می باشد.
در نتیجه تعریف فوق، سه ستون برای کنترل گرید ایجاد می کند.
هر ستون در کنترل گرید دارای خواص بسیاری است که رفتار آن ستون را در مقابل کنترل هایی که درون آن ستون قرار خواهند گرفت مشخص می کند .
از میان این خواص، خواص**Width** ، **MaxWidth** و **MinWidth** از اهمیت بالاتری بر خوردار هستند. تعاریف مربوط به خواص **MaxWidth** و **MinWidth**در بخش های قبلی به تفضیل گفته شد .
خاصیت **Width** دارای سه مقدار مختلف می تواند باشد که کمی جلوتر در بخش "تعادل **در اندازه سطر ها و ستون ها** "به همراه تعریف مقادیر Height برای سطر ها توضیح داده خواهند شد.

ادامه کنترل گرید:

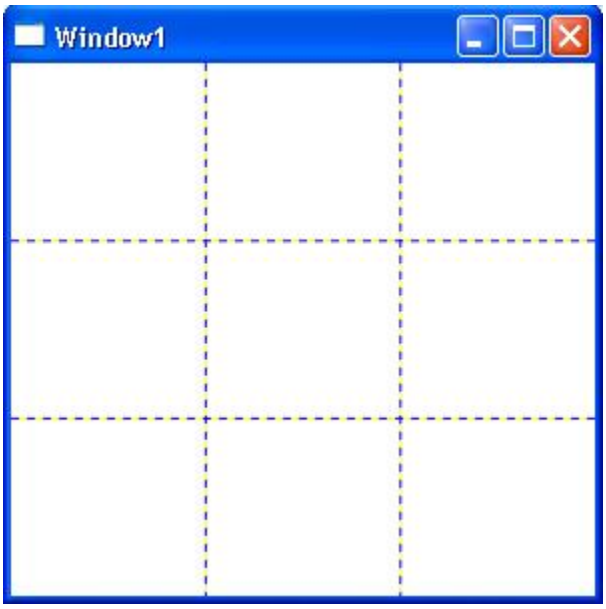
خاصیت: ShowGridLines

کنترل گرید دارای خاصیتی به نام **ShowGridLines** می باشد که مقدار پیش فرض آن **False** می باشد .
چنانچه این مقدار را به **True** تغییر دهید، می توانید خطوطی که سطر ها وستون های کنترل گرید را از یکدیگر جدا می کند را مشاهده نمایید. البته معمولا از این خاصیت استفاده چندانی نمی شود مگر در موارد خاصی که شما نیاز داشته باشید، خط واصل بین سطر ها وستون ها را به کاربر نهایی نشان دهید.
حال به قطعه کد زیر که ترکیبی از دو قطعه کد فوق است توجه فرمایید:
کد:

```
<Window x:Class="Grid.Window1"

xmlns="http :// schemas.microsoft.com/winfx/2006/xaml
/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xa
ml"
Title="Window1" Height="300" Width="300">
<Grid ShowGridLines="True">
<!--Begin Grid Row Definitions-->
<Grid.RowDefinitions>
<RowDefinition></RowDefinition> <!--
Frist Row-->
<RowDefinition></RowDefinition><!--
Second Row-->
```

```
                <RowDefinition></RowDefinition><!--
                                        Third Row-->
                                </Grid.RowDefinitions>
                <!--End of Grid Row Definitions-->

                <!--Begin Grid Column Definitions-->
                                <Grid.ColumnDefinitions>

            <ColumnDefinition></ColumnDefinition><!--First
                                                Column-->

            <ColumnDefinition></ColumnDefinition><!--Second
                                                Column-->

             <ColumnDefinition></ColumnDefinition><!--Third
                                                Column-->
                                </Grid.ColumnDefinitions>
                <!--End of Grid Column Definitions-->

                                                    </Grid>
                                                </Window>
```

قطعه کد فوق، سه سطر و سه ستون برای کنترل گرید تعریف می کند که در مجموع 9 سلول را برای این کنترل به وجود می آورند. نتیجه حاصل از قطعه کد فوق را در شکل زیر مشاهده می کنید:



تعادل در اندازه سطر ها و ستون ها:

همانطور که در شکل فوق مشخص است، فضای موجود بر روی کنترل گرید، به صورت مساوی بین تمامی سطر ها وستون های این کنترل تقسیم می شود. در زمان اجرای برنامه و با تغییر سایز پنجره و به تبع آن تغییر سایز گرید، این فضا همچنان به صورت اتوماتیک، به قسمت های مساوی بین سطر هاو ستون های کنترل گرید، تقسیم می شود.

در واقع در حالت عادی، کنترل گرید دارای رفتاری مشابه با کنترل **UniformGrid** می باشد که در بخش قبلی در رابطه با آن توضیحات مفصلی داده شد.

همانطور که اشاره شد، سطر ها دارای خاصیتی به نام **Height** و ستون ها دارای خاصیتی به نام **Width** در کنترل گرید می باشند. هر یک از این دو خاصیت، می توانند یکی از سه مقدار زیر را در بر گیرند که با توجه به آن مقدار فضای آن و رفتار آن با کنترل درونی خودش مشخص می شود.

-1سایز مطلق :

توسط این خاصیت می توانید، مقدار مشخص و ثابتی را به عنوان عرض ستون و یا ارتفاع سطر مشخص کنید. به عنوان مثال دستور

کد:

```
<RowDefinition Height="200"></RowDefinition>
```

کد:

سطری را با ارتفاع ثابت 200 تعریف می کند و دستور زیر:

کد:

```
<ColumnDefinition Width="200"></ColumnDefinition>
```

ستونی را با عرض ثابت 200 تعریف می کند.

-2سایز اتوماتیک:

این خاصیت، عرض ستون و ارتفاع سطر را به صورت اتوماتیک و بر اساس نیاز کنترل درونی خودش تنظیم میکند. به عنوان مثال اگر کنترلی با دارای عرض 200 و ارتفاع 300 باشد و درون سلولی که سطر و ستون آن با این روش مقدار دهی شده اند، قرارداشته باشد، سلول مورد نظر تغییر اندازه داده تا بتواند کل فضای مورد نیاز آن کنترل را تامین کنید. مقداری که باید به کار برده شود تا سطر و ستونی به عنوان سطر و ستون اتوماتیک معرفی گردد، کلمه Auto می باشد. به کد های زیر توجه کنید:

کد:

```
<RowDefinition Height="Auto"></RowDefinition>
<ColumnDefinition Width="Auto"></ColumnDefinition>
```

در شکل فوق سطر و ستونی با ارتفاع اتوماتیک تعریف شده اند.

-3سایز نسبی :

توسط این خاصیت، فضای موجود بین تعدای سطر و ستون تقسیم بندی می شود. در واقع این حالت، حالت پیش فرض برای عرض ستون ها و ارتفاع سطرها می باشد. توسط کاراکتر * می توانید از این خاصیت برای خواص Width

و Height استفاده نمایید.

به عنوان مثال، کد های زیر نتیجه ای یکسان با آن چیزی که در شکل قبلی ملاحظه کردید را خواهد داشت.

کد:

```
<Grid ShowGridLines="True">
<!--Begin Grid Row Definitions-->
<Grid.RowDefinitions>
<RowDefinition Height="*"></RowDefinition>
<!--Frist Row-->
<RowDefinition
Height="*"></RowDefinition><!--Second Row-->
<RowDefinition
Height="*"></RowDefinition><!--Third Row-->
</Grid.RowDefinitions>
<!--End of Grid Row Definitions-->

<!--Begin Grid Column Definitions-->
<Grid.ColumnDefinitions>
<ColumnDefinition
Width="*"></ColumnDefinition><!--First Column-->
<ColumnDefinition
Width="*"></ColumnDefinition><!--Second Column-->
<ColumnDefinition
Width="*"></ColumnDefinition><!--Third Column-->
</Grid.ColumnDefinitions>
<!--End of Grid Column Definitions-->

</Grid>
```

نکته قابل توجه ای که می توان در حالت سایز نسبلی به آن توجه داشت و به واقع دلیل اصلی نام گذاری آن نیز، همین نکته می باشد این است که شما می توانید سطری یا ستونی را از لحاظ ارتفاع و عرض، چندین برابر سطر و یا ستون دیگری تعریف کنید. به عنوان مثال سطری که خاصیت ارتفاع آن به صورت **Height = 2*** تعریف شده باشد، ارتفاعش دو برابر سطری است که ارتفاع آن به صورت **Height = *** تعریف شده باشد. به همین صورت سطری که ارتفاع آن به صورت **Height=0.25*** تعریف شده است، ارتفاعی به اندازه 1/8 سطری دارا که ارتفاع آن به صورت **Height =2*** تعریف شده است. این موضوع در مورد ستون ها و عرض آن ها نیز صدق است.

بخش سوم: چیدمان و طراحی کنترل ها (قسمت دوازدهم(

ادامه کنترل گرید

تعیین سلول برای کنترل ها ی فرزند در کنترل گرید:

قبلا اشاره شد که دلیل اصلی اینکه کنترل گرید از قویترین و پرکاربرد ترین کنترل های کانتینر می باشد این است که شما کنترل کاملی بر روی سطر ها وستون های گرید می توانید داشته باشید. در این قسمت به نحوه تخصیص یک سلول خاص در کنترل گرید را به یک کنترل مشخص نشان خواهم داد.

برای تخصیص سلول خاصی از کنترل گرید به یکی از کنترل های فرزند از خواص پیوست شده **Row** و **Column** استفاده خواهیم کرد. به نمونه کد زیر دقت کنید:

کد:

```
<Button Grid.Row="2" Grid.Column="2"
button3</Button>
```

در قعه کد فوق، دکمه تعریف شده در سطر و ستون سوم (سلول 9) در گرید مربوطه قرار خواهید گرفت.

نکته :

شماره سطر ها وستون ها در گرید از صفر شروع می شود. بنابرین مقدار 2 برای سطر و ستون معرف سطر و ستون سوم در گرید می باشد.

نکته:

چنانچه بخواهید کنترلی را در سلول اول از گرید قرار دهید، نیاز به تنظیم خواص **Row** و **Column** از گرید ندارید . درواقع مقدار پیش فرض این خواص، صفر می باشد که بیانگر سطر و ستون اول گرید می باشد.

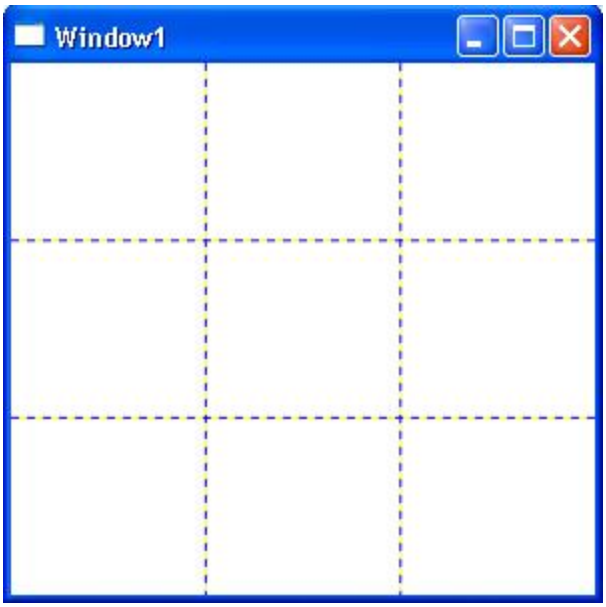
برای درک بهتر به نمونه کر زیر دقت کنید:

کد:

```
<Window x:Class="Grid.Window1"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml
/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xa
ml"
Title="Window1" Height="300" Width="300">
<Grid ShowGridLines="True">
<!--Begin Grid Row Definitions-->
<Grid.RowDefinitions>
<RowDefinition ></RowDefinition> <!--
Frist Row-->
<RowDefinition ></RowDefinition><!--
Second Row-->
<RowDefinition ></RowDefinition><!--
Third Row-->
</Grid.RowDefinitions>
<!--End of Grid Row Definitions-->
```

```
                    <!--Begin Grid Column Definitions-->
                                <Grid.ColumnDefinitions>
                                      <ColumnDefinition
                    ></ColumnDefinition><!--First Column-->
                                      <ColumnDefinition
                   ></ColumnDefinition><!--Second Column-->
                                      <ColumnDefinition
                    ></ColumnDefinition><!--Third Column-->
                                  </Grid.ColumnDefinitions>
                      <!--End of Grid Column Definitions-->
                    <!--Start Of Defining Child Controls-->
          <Button >button1</Button>  <!--the Grid cell
                              of this button is 0-->
                      <Button Grid.Row="1" Grid.Column="1"
    >button2</Button><!--the Grid cell of this button
                                                  is 5-->
                      <Button Grid.Row="2" Grid.Column="2"
    >button3</Button><!--the Grid cell of this button
                                                  is 9-->
                                                  </Grid>
                                                </Window>
```

هماطور که مشاهده می کنید، برای اولین دکمه خواص **Row** و **Column** تنظیم نشده اند، در نتیجه دکمه مذکور در اولین سطر و ستون از گرید قرار خواهد گرفت.
نتیجه اجرای کد فوق مشابه زیر خواهد بود:



بخش سوم: چیدمان و طراحی کنترل ها (قسمت سیزدهم)

پایان کنترل گرید (سورس مثال ها را در پایان همین پست می توانید دانلود کنید)

محدوده سطر و ستون ها در کنترل گرید:

در این قسمت دو خاصیت دیگر از خواص پیوست شده برای کنترل های درون کنترل گرید را مورد بررسی قرار خواهیم داد. این خواص نشان دهنده محدوده ای از تعداد سطر ها وستون هایی از گرید هستند که یک کنترل می تواند دربر گیرد. توسط این خواص می توانید کنترل بهتری بر روی عناصر و محل قرار گیری آن ها داشته باشید.

خاصیت: RowSpan

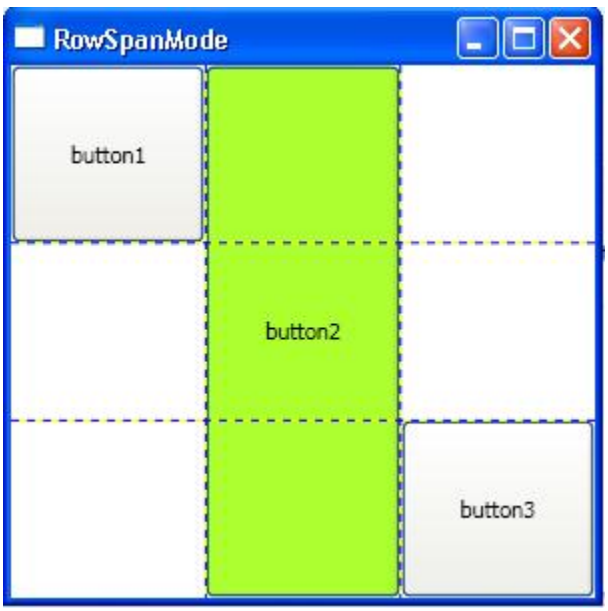
توسط این خاصیت می توانید مشخص کنید که کنترلی چند سطر از سطر های گرید را می تواند در بر گیرد. به عنوان مثال قطعه کد زیر:

کد:

```
<Grid ShowGridLines="True">
<!--Begin Grid Row Definitions-->
<Grid.RowDefinitions>
<RowDefinition ></RowDefinition>
<!--Frist Row-->
<RowDefinition ></RowDefinition>
<!--Second Row-->
<RowDefinition ></RowDefinition>
<!--Third Row-->
</Grid.RowDefinitions>
<!--End of Grid Row Definitions-->
<!--Begin Grid Column Definitions-->
<Grid.ColumnDefinitions>
```

```
                <ColumnDefinition ></ColumnDefinition>
                    <!--First Column-->
                <ColumnDefinition ></ColumnDefinition>
                    <!--Second Column-->
                <ColumnDefinition ></ColumnDefinition>
                    <!--Third Column-->
                </Grid.ColumnDefinitions>
            <!--End of Grid Column Definitions-->
        <!--Start Of Defining Child Controls-->
                <Button >button1</Button>
            <!--the Grid cell of this button is 0-->
        <Button Background="GreenYellow" Grid.Row="0"
Grid.Column="1" Grid.RowSpan="3" >button2</Button>

                <Button Grid.Row="2" Grid.Column="2"
                    >button3</Button>
            <!--the Grid cell of this button is 9-->
                </Grid>
```

در قطعه کد فوق، دکمه دوم دارای خاصیت **RowSpan** برابر با سه می باشد. همچنین دقت کنید که خاصیت **Row** آن از یک در مثال قبل به صفر در این مثال تغییر کرده است.
نتیجه حاصل از کد فوق را در شکل زیر مشاهده می کنید



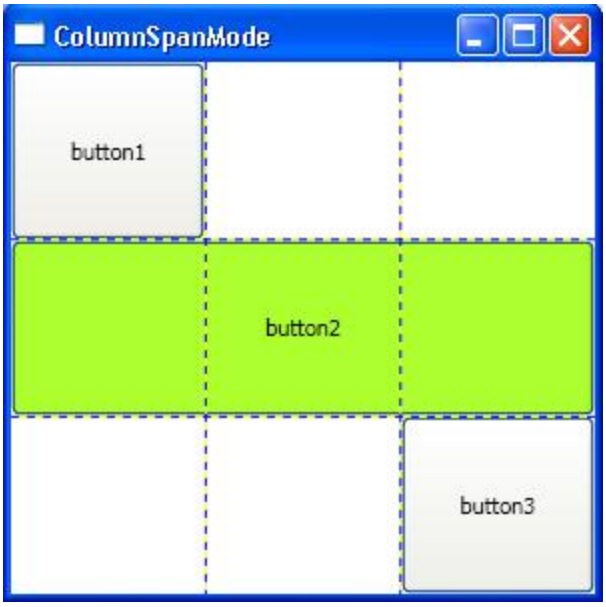
خاصیت: ColumnSpan

توسط این خاصیت می توانید مشخص کنید که کنترلی چند ستون از ستون های گرید را می تواند در بر گیرد. به عنوان مثال قطعه کد زیر:

کد:

```
<Grid ShowGridLines="True">
<!--Begin Grid Row Definitions-->
<Grid.RowDefinitions>
<RowDefinition ></RowDefinition>
<!--Frist Row-->
<RowDefinition ></RowDefinition>
<!--Second Row-->
<RowDefinition ></RowDefinition>
<!--Third Row-->
</Grid.RowDefinitions>
<!--End of Grid Row Definitions-->
<!--Begin Grid Column Definitions-->
<Grid.ColumnDefinitions>
<ColumnDefinition ></ColumnDefinition>
<!--First Column-->
<ColumnDefinition ></ColumnDefinition>
<!--Second Column-->
<ColumnDefinition ></ColumnDefinition>
<!--Third Column-->
</Grid.ColumnDefinitions>
<!--End of Grid Column Definitions-->
<!--Start Of Defining Child Controls-->
<Button >button1</Button>
<!--the Grid cell of this button is 0-->
<Button Background="GreenYellow" Grid.Row="1"
Grid.Column="0" Grid.ColumnSpan="3"
>button2</Button>

<Button Grid.Row="2" Grid.Column="2"
>button3</Button>
<!--the Grid cell of this button is 9-->
</Grid>
```

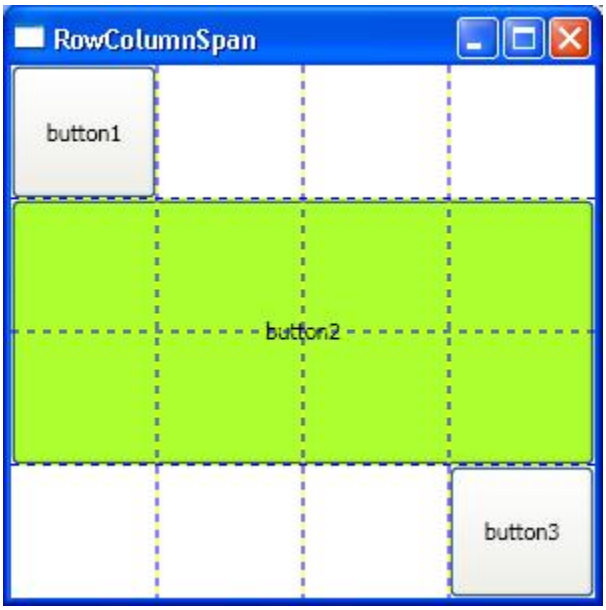
در قطعه کد فوق، دکمه دوم دارای خاصیت **ColumnSpan** برابر با سه می باشد. همچنین دقت کنید که خاصیت **Column** آن از یک در مثال قبل به صفر در این مثال تغییر کرده است.
نتیجه حاصل از کد فوق را در شکل زیر مشاهده می کنید:



نمونه کد زیر نحوه استفاده از خواص **RowSpan** و **ColumnSpan** و نتیجه آن را در شکل بعد از آن نشاند می دهد.
کد:

```
<Grid ShowGridLines="True">
<!--Begin Grid Row Definitions-->
<Grid.RowDefinitions>
<RowDefinition ></RowDefinition>
<!--Frist Row-->
<RowDefinition ></RowDefinition>
<!--Second Row-->
<RowDefinition ></RowDefinition>
<!--Third Row-->
<RowDefinition ></RowDefinition>
<!--Fourth Row-->
</Grid.RowDefinitions>
<!--End of Grid Row Definitions-->
<!--Begin Grid Column Definitions-->
<Grid.ColumnDefinitions>
<ColumnDefinition ></ColumnDefinition>
<!--First Column-->
<ColumnDefinition ></ColumnDefinition>
```

```
                                        <!--Second Column-->
                                <ColumnDefinition ></ColumnDefinition>
                                         <!--Third Column-->
                                <ColumnDefinition ></ColumnDefinition>
                                        <!--Fourth Column-->
                                    </Grid.ColumnDefinitions>
                                <!--End of Grid Column Definitions-->
                            <!--Start Of Defining Child Controls-->
                                    <Button >button1</Button>

                        <Button Background="GreenYellow" Grid.Row="1"
                                Grid.Column="0" Grid.ColumnSpan="4"
                                                Grid.RowSpan="2" >
                                            button2</Button>
                                <Button Grid.Row="3" Grid.Column="3"
                                                >button3</Button>

                                                                </Grid>
```



دقت کنید که در این نمونه، تعداد سطر ها وستون ها به چهار، افزایش پیدا کرده است .خاصیت **RowSpan** از دکمه دوم برابر با چهار و خاصیت **ColumnSpan** از آن برابر با دو تنظیم شده است.

پ و1:

کنترل گرید از مهمترین و پر کاربرد ترین کنترل های کانتینر در WPF می باشد. پس تمرکز زیادی بر آن

داشته باشید و سعی کنید که کاملا بر آن و نحوه استفاده از ان مسلط شوید

پ و2:
از بابت فاصله زیادی که بین پست ها افتاد؛ پوزش می طلبم. گرفتاری پیش آمده بود.. اگر باز هم تاخیر چند روزه داشتم، نگران نباشید و مطمئنا این تاپیک به جاهای خوبی خوهد رسید(انشاء الله(
ایام به کامتان
فایل های ضمیمه

Grid.rar (68.9 کیلوبایت, 691 دیدار(

بخش چهارم) Content Controls :کنترل های محتوا) (قسمت اول(

پیش گفتار:

در بخش قبلی با کنترل های کانتینر که اساس برنامه نویسی در WPF محسوب می شوند، آشنا شدید. توسط آن ها توانستید پنجره های خود را به قسمت های مختلف تقسیم کنید، و کنترل های خود را درون قسمت های مختلف قرار دهید و برنامه ها یی ایجاد کنید که وابسته به رزولوشن صفحه نمایش نبوده و در محدوده وسیعی از مانیتور ها با رزولوشن های مختلف قابل استفاده باشد.

کنترل های کانتینر، تنها امکانات WPF برای نگهداری کنترل های فرزند نیستند. تکنولوژی WPF مدل دیگری از مفهوم **محتوا** را ارائه می کند که در این بخش به آن ها خواهیم پرداخت.
در این مدل جدید، می توانید درون کنترل های ساده ای مانند Button ، کنترل های دیگری مانند TextBox ، Label، و یا ترکیبی از این کنترل ها را قرار دهید. همچنین قادر خواهید بود، اشکال برداری و کنترل های دیگر را درون این کنترل با مدیریت بسیار بالایی قرار دهید.

مفهوم Content در: WPF

در تکنولوژی WPF مفهوم عناصر کمی متفاوت با مفهوم کنترل می باشد. به هر چیزی که شما در پنجره های برنامه خود در WPF قرار می دهید، یک عنصر تلقی می شود، در حالی که کنترل ها عناصر ویژه ای هستند که می توانند نسبت به اعمال کاربر، عکس العمل نشان دهند. مثلا می توانند ورودی های کاربر را دریافت کنند، خروجی ها ی برنامه را به کاربر بفرستند و...

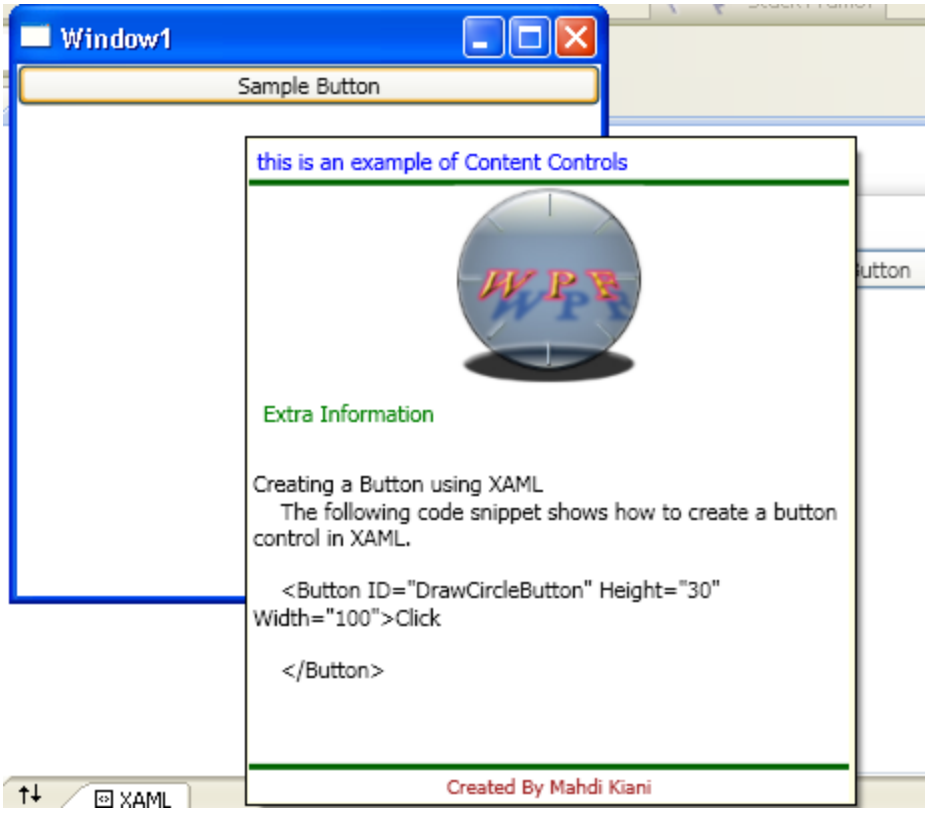
کنترل هایContent ، نمونه خاص دیگری از عناصر در تکنولوژی WPF می باشند که قابلیت دربرگیری و نمایش مقادیری را دارند که ما به آن ها محتوا می گوییم و به کنرل هایی که این خاصیت را شامل می شوند، کنترل های محتوا می گوییم. در واقع هر کنترل محتوا می توانند تنها یک عنصر داخلی را درون خود قرار دهند. به همین جهت، تفاوتی بین کنترل های محتوا و کنترل های کانتینر وجود دارد. کنترل های کانتیر می توانند، صفر، یک و یا چندین کنترل را به عنوان کنترل های داخلی و کنترل های فرزند خود در برگیرند.

نکته:

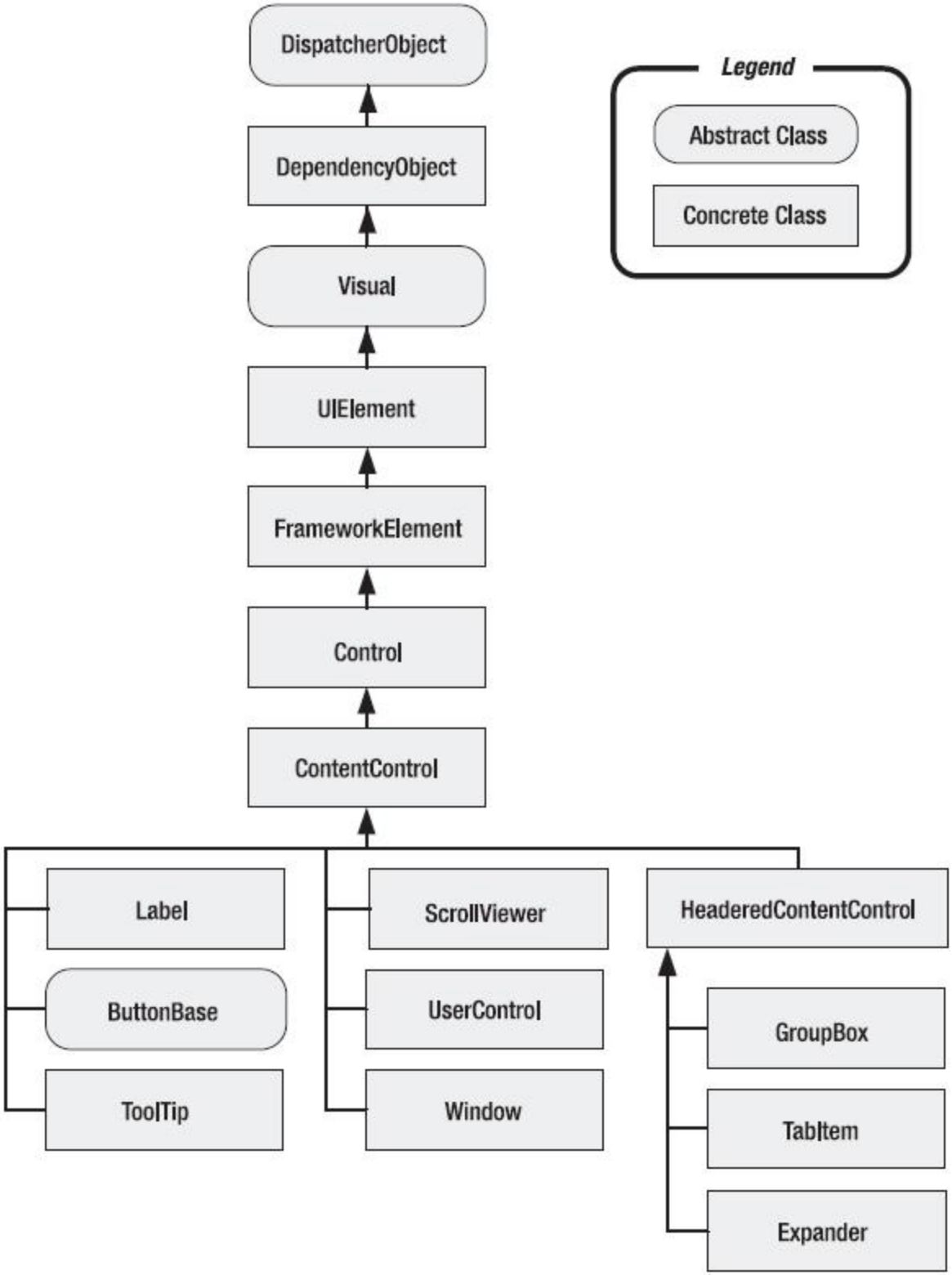
البته این گفته که کنترل های محتوا می توانند، تنها یک عنصر به عنوان عنصر داخلی خود بپذیرند، آن ها را محدود

نمی کند. در واقع شما می توانید، با به کارگیریی کنترل های کانتینر، عناصر و کنترل های بسیاری را درون یک کنترل محتوا نشان دهید.

به عنوان مثال، عکس زیر نمونه ای از یک عنصر ToolTip را نشاند می دهد. همانطور که می دانید ایجاد چنین عنصری در دات نت 2.0 نیاز به کد نویسی زیادی دارد. ضمن اینکه عکس زیر حالت ساده ای از نحوه استفاده از Contentمب باشد



همانطور که اشاره شد، کنترل های کانتینر از کلاس انتزاعی پایه ای به نام Panel ارث بری می کنند، کنترل های محتوا نیز از کلاس انتزاعی به نام ContentControl ارث بری می کنند. این موضوع در شکل زیر نشان داده شده است.


DispatcherObject
DependencyObject
Visual
UIElement
FrameworkElement
Control
ContentControl
Label
ButtonBase
ToolTip
ScrollViewer
UserControl
Window
HeaderedContentControl
GroupBox
TabItem
Expander
Legend
Abstract Class
Concrete Class

همانطور که در شکل فوق نشان داده شده است، بسیاری از کنترل های اولیه WPF مانندLabel ، Button، ToolTip و ...کنترل های محتوا می باشند. علاوه بر این کنترل ها، کنترل های محتوای دیگری وجود دارند که علاوه بر محتوایی بودن آن ها، خواص دیگری را نیز دارا می باشند .

به عنوان مثال کنترل **ScrollViewer** یکی از کنترل های محتوایی می باشد که قابلیت اسکرول را نیز دارا می باشد. یا کنترل های کاربری که می توانند گروهی از کنترل ها را جهت قابلیت استفاده مجدد درون خود جمع کنند.

به طور کلی، کنترل های محتوا به دو دسته زیر تقسیم می شوند:

الف (دسته اول، کنترل هایی هستند که به آن ها اشاره شد. این کنترل ها دارای بخشی به نام **Content** می باشند که می توانند یک عنصر را به عنوان محتوای خود درون خود نگهداری کنند. حال این عنصر می تواند یک رشته متنی باشد و یا می تواند مجموعه ای از کنترل های متفاوت بر روی یک کنترل کانتیر باشند.

ب (دسته دوم، کنترل هایی هستند که علاوه بر داشتن خاصیت کنترل های دسته اول، دارای بخش دیگری به نام **Header** می باشند که عملی مانند بخش Content را انجام می دهند. از این کنترل ها می توان به کنترل **TabItem** ، کنترل**GroupBox** ، **Expander** و ... اشاره کرد که در بخش های آتی با آن ها آشنا خواهید شد.

خاصیت: Content

هر کنترل محتوا، دارای خاصتی به نام **Content** می باشد. این خاصیت در برگیرنده عناصری خواهد بود که به عنوان محتوای کنترل مورد نظر شناخته می شوند.

به طور کلی خاصیت **Content** در کنترل های محتوا، می تواند شامل دو دسته کلی زیر از عناصر WPF باشد:

الف (دسته اول عناصری هستند که از کلاس **UIElement** ، ارث بری نمی کنند. در این موارد، متد **()ToString** فراخوانی شده تا متن آن عنصر را به عنوان محتوای عنصر مورد نظر قرار دهد.

ب (دسته دم عناصری که از کلاس UIElement ارث بریم می کنند. این عناصر در واقع تمامی عناصر و.یژوال در WPF را شامل می شوند. کمپایلر از متد **OnRender** این عناصر استفاده می کند تا قابل نمایش در کنترل محتوا با شند.

برای درک بهتر این موضوع به دو قطعه کد زیر دقت کنید:
کد:

کد:

```
<Button HorizontalAlignment="Center" VerticalAlignment="Center">this a simple text Content</Button>
```

کد فوق تنها یک رشته متنی را به عنوان محتوای عنصر Button در نظر گرفته است. این نوع از محتوا، از نوع دسته (الف) می باشد.
نتیجه اجرای این کد شبیه به شکل زیر خواهد بود:

در واقع می توان در این حالت، خاصیت Content را به خاصیت Text عناصر در دات نت فریم ورک 2.0 تشبیه کرد.
البته دقت داشته باشید که این مورد صرفا یک تشبیه می باشد.

حال به کد زیر دقت کنید:
کد:

کد:

```
<Button>
<Image Source="./Images/WPF.jpg"></Image>
</Button>
```

کد فوق یک عکس را به عنوان محتوای Button معرفی می کند. این نمونه از محتوا، از نوع دسته) **ب** (می باشد که با فراخوانی متد**OnRender**، عکس مورد نظر قابل نمایش در کنترل Button به عنوان محتوا گردیده است.
نتیجه حاصل از اجرای کد فوق به صورت زیر می باشد:

دو قطعه کد فوق از ساده ترین، حالت های خاصیت Content بودند. درواقع قدرت این خاصیت بسیار فراتر از قرار گیری یک عکس و یا یک رشته متنی درون آن می باشد .**با استفاده از کنترل های کانتینر می توانید کنترل های بسیاری را در یک Button و یا هر کنترل محتوای دیگر قرار دهید.**

به قطعه کد زیر دقت کنید:

کد:

کد:

```
<Window x:Class="ContentControlsInWPF.TextAndImageContent"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
Title="TextAndImageContent" Height="300" Width="300">
<Grid>
<Button VerticalAlignment="Center" HorizontalAlignment="Center">
<StackPanel>
<Image VerticalAlignment="Top" MaxHeight="75" MaxWidth="75" Stretch="Fill"
Source="./Images/WPF.jpg"></Image>
<TextBlock TextAlignment="Center" >Fill Boxes And then Click Ok</TextBlock>
<StackPanel Orientation="Horizontal">
<Label MinWidth="50">name</Label>
<TextBox MinWidth="120"></TextBox>
```

```
</StackPanel>
<StackPanel Orientation="Horizontal">
<Label MinWidth="50">Family</Label>
<TextBox MinWidth="120"></TextBox>
</StackPanel>
<Button>Ok</Button>
<Image VerticalAlignment="Top" MaxHeight="75" MaxWidth="75" Stretch="Fill"
Source="./Images/WPF.jpg"></Image>
</StackPanel>
</Button>
</Grid>
</Window>
```

در قطعه کد فوق، یک Button در کنترل کانتینر اصلی فرم یعنی گرید قرار گرفته است. در خاصیت Content کنترل Buttonیک کنترل StackPanel قرار گرفته ایت که خود شامل پنج کنترل فرزند به شرح زیر می باشد:
ابتدا یک عنصر Image ، در بالاترین قسمت کنترل StackPanel قرار گرفته است. سپس یک کنترل TextBlock با متن Fill Boxes And then Click Ok قرار گرفته است. پس از آن دو کنترل StackPanel داخلی به مجموعه کنترل های فرزند اضافه شده اند .
هر یک از این دو کنترل StackPanel داخلی خود شامل دو کنترل فرزند می باشند، که به صورت افقی در کنترل کانتینر خود قرار گرفته اند.
پس از دو کنترل StackPanel داخلی، یک کنترل Button و در نهایت یک عنصر Image دیگر قرار داده شده است.

نتیجه حاصل از اجرای کد فوق را در شکل زیر مشاهده می کنید:

همانطور که در شکل فوق نیز مشاهده می کنید، به راحتی می توان با بهره گیری از کنترل های کانتینر، به تعداد دلخواهی از کنترل های WPF را در یک کنترل محتوا قرار داد. از این خاصیت می توان در ساخت کنترل های سفارشی و کنترل های کاربری بهره بسیاری برد.

کنترل های محتوا با خواص ویژه:

همانطور که پیش تر نیز اشاره کردم، در **WPF** کنترل های بسیاری وجود دارد که علاوه بر قابلیت محتوایی بودن، دارای یک سری خواص کاربردی دیگری می باشند که می توانند محتویات خودشان را به خوبی مدیریت کنند.
تعدادی از این کنترل ها که در ادامه به آن ها خواهم پرداخت، عبارتند از:

کنترل: ScrollViewer

این کنترل امکاناتی را در اختیار شما قرار می دهد که بتوانید با آن ها، محتوبات این کنترل را مدیریت کنید. چنانچه محتویات داخلی این کنترل از مقدار فضای موجود بر روی ان بیشتر باشد، کنترل به صورت اتوماتیک Scroll پیدا کرده تا بتواند همه محتویات خود را به خوبی در بر گیرد.

کنترل: GroupBox

با این کنترل آشنا هستید، این کنترل برای گروه بندی تعدادی از کنترل ها به کار می رود، که نسبت به نسخه 2.0 خود دارابی امکانات بیستری می باشد،که در ادامه بیشتر با این کنترل آشنا خواهید شد.

کنترل : TabControl

نمونه این کنترل نیز در نسخه 2.0 دات نت فریم ورک وجود دارد. در اینجا قابلیت های بسیاری به این کنترل اضافه

شده است که در جای خود، توضیح داده خواهند شد.

کنترل : Expander

این کنترل، همواره یکی از کنترل هایی بوده است که برنامه نویسیان علاقه زیادی به استفاده از آن دارند. این کنترل در دات نت فریم ورک 2.0 و نسخه های قبل از آن وجود نداشت و برنامه نویسان یا اقدام به نوشتن این کنترل می کردند و یا از کنترل های نوشته شده توسط اشخاص ثالث استفاده می کرند. نمونه این کنترل در ویندوز بسیار استفاده شده است.

حال، در ادامه به توضیح هر یک از این کنترل ها با ذکر مثال هایی در مورد هر یک خواهیم پرداخت:

کنترل: ScrollViewer

همانطور که در بخش قبلی ملاحظه کردید، تعدادی کنترل را به عنوان محتوای کنترل Button قرار دادیم. حال اگر تعداد این کنترل ها، زیاد شود و کنترل Button فضای لازم را برای آن ها نداشته باشد، نمی تواند همه محتویات خود را به درستی نشان دهد.

در این مواقع از کنترل دیگری به نام کنترل **ScrollViewer** استفاده می کنیم. این کنترل می توان با هر عنصری به کار رود. زمانی که محتویات این کنترل از فضای موجود، بیشتر شود، این کنترل به صورت اتوماتیک ، محتویات خود را قابل پیمایش کرده و کاربر می تواند با اسکرول کردن، به تمامی محتویات درون کنترل مورد نظر دسترسی داشته باشد.
به قطعه کد زیر که اصلاح شده تکه کد قبل می باشد توجه کنید:

کد:

```
<Window x:Class="ContentControlsInWPF.ContentWithScrollViewerFeature"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
              xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
     Title="ContentWithScrollViewerFeature" Height="300" Width="300">
                                                              <Grid>
     <Button VerticalAlignment="Center" HorizontalAlignment="Center">
                                                        <ScrollViewer>
                                                          <StackPanel>
                          <Image VerticalAlignment="Top" MaxHeight="75"
      MaxWidth="75" Stretch="Fill"  Source="./Images/WPF.jpg"></Image>
               <TextBlock TextAlignment="Center" >Fill Boxes And then
                                                    Click Ok</TextBlock>
                                    <StackPanel Orientation="Horizontal">
                                        <Label MinWidth="50">name</Label>
                                      <TextBox MinWidth="120"></TextBox>
```

```
                                                    </StackPanel>
                                                <StackPanel Orientation="Horizontal">
                                                  <Label MinWidth="50">Family</Label>
                                                    <TextBox MinWidth="120"></TextBox>
                                                    </StackPanel>
                                                <StackPanel Orientation="Horizontal">
                                                       <Label MinWidth="50">Age</Label>
                                                    <TextBox MinWidth="120"></TextBox>
                                                    </StackPanel>
                                                <StackPanel Orientation="Horizontal">
                                                 <Label MinWidth="50">Country</Label>
                                                    <TextBox MinWidth="120"></TextBox>
                                                    </StackPanel>
                                                <StackPanel Orientation="Horizontal">
                                                     <Label MinWidth="50">City</Label>
                                                    <TextBox MinWidth="120"></TextBox>
                                                    </StackPanel>
                                                <StackPanel Orientation="Horizontal">
                                                    <Label MinWidth="50">State</Label>
                                                    <TextBox MinWidth="120"></TextBox>
                                                    </StackPanel>
                                                <StackPanel Orientation="Horizontal">
                                                 <Label MinWidth="50">P__Code</Label>
                                                    <TextBox MinWidth="120"></TextBox>
                                                    </StackPanel>
                                                <StackPanel Orientation="Horizontal">
                                                 <Label MinWidth="50">Address</Label>
                                                    <TextBox MinWidth="120"></TextBox>
                                                    </StackPanel>
                                                <StackPanel Orientation="Horizontal">
                                                    <Label MinWidth="50">Phone</Label>
                                                    <TextBox MinWidth="120"></TextBox>
                                                    </StackPanel>
                                                        <Button>Ok</Button>
                                    <Image VerticalAlignment="Top" MaxHeight="75"
                MaxWidth="75" Stretch="Fill"  Source="./Images/WPF.jpg"></Image>
                                                    </StackPanel>
                                                  </ScrollViewer>
```

```
</Button>

</Grid>
</Window>
```

این مثال، مانند مثال قبلی می باشد با این تفاوت که کنترل های بیشتری درون محتوای کنترل **Button** قرار گرفته اند. علاوه بر این، تمامی محتویات کنترل Button درون کنترل دیگری به نام **ScrollViewer** قرار گرفته است.
چنانچه فضای لازم برای نمایش تمامی محتویات کنترل Button موجود باشد، کنترل **ScrollViewer** غیر فعال می شود و زمانی که فضای لازم برای نمایش کنترل های درونی کنترل Button مجود نباشد، کنترل **ScrollViewer** به صورت اتوماتیک فعال شده تا بتواند تمامی محتویات کنترل Button را نمایش دهد.
این کنترل بسیار پر کاربرد می باشد. به این دلیل که علاوه بر اینکه غیر وابستگی برنامه شما به رزولوشن صفحه نمایش حفظ می گردد، باعث می شود که بتوانید تعداد بسیاری از کنترل ها را درون یک کنترل محتوایی قرار دهید .
نتیجه حاصل از اجرای مثال فوق را در شکل زیر مشاهده می کنید:

نکته:
توسط دو خاصیت **HorizontalScrollBarVisibility** و **VerticalScrollBarVisibility** می توانید کنترل بیشتری بر روی عملکرد کنترل **ScrollViewer**داشته باشید. هر یک از این خواص دارای چهار مقدار می باشند که در زیر به

توضیح هریک خواهیم پرداخت:

-1مقدار : Auto این مقدار باعث می شود که نوار های اسکرول (برای هر کدام که تنظیم شده باشد) در حالتی که غیر فعال هستند، نشان داده نشوند.

-2مقدار : Disabled این مقدار باعث می شود که نوار های اسکرول همیشه در حالت غیر فعال باشند.

-3مقدار : Hidden این مقدار باعث می شود که نوار های اسکرول (برای هر کدام که تنظیم شده باشد) مخفی باشند. در این حالت می توانید عمل اسکرول را با کلید های جهت نما انجام دهید و یا با کلید Tab بین کنترل ها حرکت کنید که در این صورت زمانی که به کنترلی می رسید که در محدوده فضایی کنترل مورد نظر (در مثال قبلی Button) نیست، عمل اسکرول به صورت اتوماتیک انجام می شود.

-4مقدار : Visible این گزینه که مقدار پیش فرض برای هر دوخاصیت نیز می باشد، در زمانی که اسکرول در حالت غیر فعال می باشد، همچنان به صورت Visible هستند و کاربر می تواند آن ها را مشاهده نماید.

ادامه کنترلScroll Viewer

کنترل اسکرول با برنامه نویسی:

همانطور که در بخش قبل دیدیدریال می توانستید با موس و یا با کنترل های جهت نما، کنترل اسکرول خود را مدیریت کنید.
در این بخش با توابعی آشنا خواهید شد که می توانید توسط آن ها کنترل ScrollViewer را در زمان اجرای برنامه و با کد نویسی مدیریت نمایید.

نکته :
این توابع هر کدام دارای دو نسخه می باشند، یکی برای اسکرول عمودی و دیگری برای اسکرول افقی. در اینجا توابع مربوط به اسکرول عمودی و عملکرد آن ها را خواهیم دید و در مورد توابع مروبط به اسکرول افقی تنها نام آن ها ذکر می گردند. چون عملکرد آن ها دقیقا مشابه عملکر نسخه خود برای اسکرول عمودی هستند.

متد : LineUp()

عملکر این متد مانند کلیک کردن بر روی علامت جهت نمای بالایی اسکرول عمودی می باشد. پس میزان جا بجایی که توسط این متد انجام می گیرید بستگی به میزان جابجایی توسط علامت جهت نمای بالایی دارد.
معادل این متد برای نوار اسکرول افقی ، متد LineLeft می باشد.

متد : LineDown()
عملکر این متد مانند کلیک کردن بر روی علامت جهت نمای پایینی اسکرول عمودی می باشد. پس میزان جا بجایی

که توسط این متد انجام می گیرید بستگی به میزان جابجایی توسط علامت جهت نمای بالایی دارد.
معادل این متد برای نوار اسکرول افقی ، متد LineRight می باشد

متد : PageUp

عملکر این متد مانند کلیک کردن بر روی فاصله بین علامت جهت نمای بالایی و دستگیره اسکرول در اسکرول عمودی می باشد .
معادل این متد برای نوار اسکرول افقی ، متد PageLeft می باشد

متد : PageDown عملکر این متد مانند کلیک کردن بر روی فاصله بین علامت جهت نمای پایینی و دستگیره اسکرول در اسکرول عمودی می باشد .
معادل این متد برای نوار اسکرول افقی ، متد **PageRight** می باشد

متد: ScrollToHome

این متد اسکرول را به پایین ترین قسمت ممکن انتقال می دهد.
معادل این متد برای نوار اسکرول افقی ، **ScrollToLeftEnd** می باشد.

متد : ScrollToEnd

این متد اسکرول را به بالاترین قسمت ممکن انتقال می دهد.
معادل این متد برای نوار اسکرول افقی ، متد ScrollToRightEnd می باشد.

متد : ScrollToVerticalOffset
این متد بر اساس مقداری که به عنوان آرگومان می گیرد عمل اسکرول را انجام می دهد.
معادل این متد برای نوار اسکرول افقی؛ متد **ScrollToHorizontalOffset** می باشد

قطعه کد زیر نحوه استفاده از این متد ها را نشان می دهد:

کد:

```
<Window x:Class="contentControls.ProgrammingScrollViewer"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
Title="ProgrammingScrollViewer" Height="300" Width="650">
<Grid>
<Grid.RowDefinitions>
<RowDefinition  Height="Auto"></RowDefinition>
```

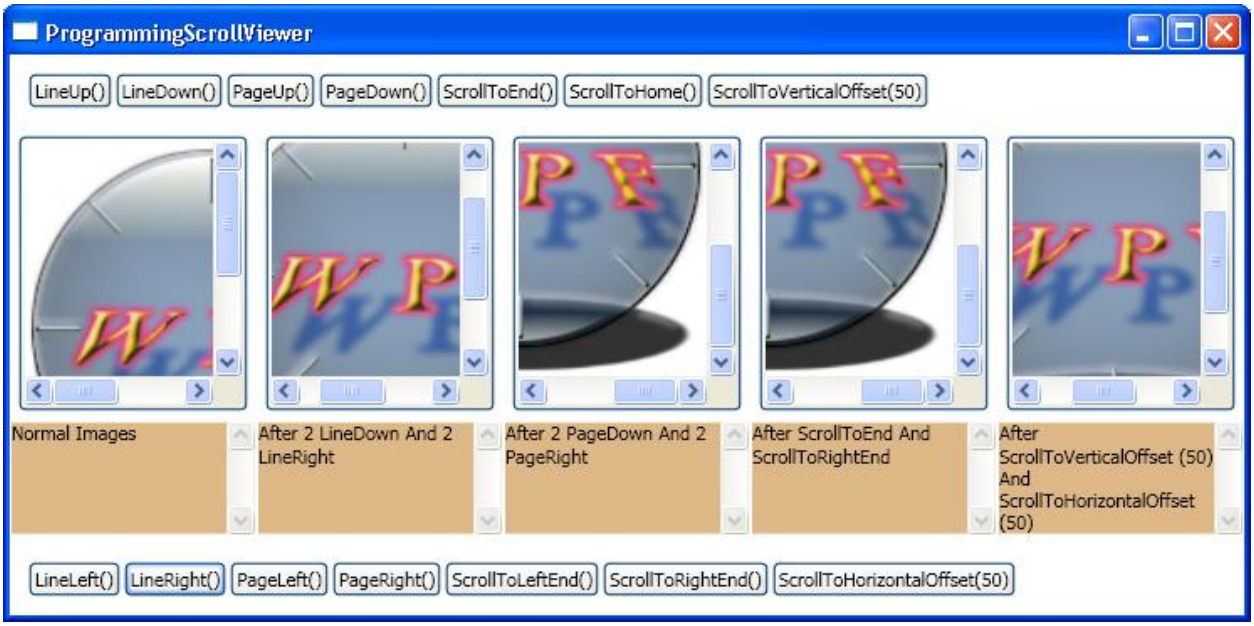
```
                                        <RowDefinition></RowDefinition>
                                <RowDefinition Height="Auto"></RowDefinition>
                                <RowDefinition Height="AUto"></RowDefinition>
                                                </Grid.RowDefinitions>

                                                    <Grid.ColumnDefinitions>
                                            <ColumnDefinition></ColumnDefinition>
                                            <ColumnDefinition></ColumnDefinition>
                                            <ColumnDefinition></ColumnDefinition>
                                            <ColumnDefinition></ColumnDefinition>
                                            <ColumnDefinition></ColumnDefinition>
                                                        </Grid.ColumnDefinitions>


        <WrapPanel Margin="10" Grid.ColumnSpan="5" Orientation="Horizontal">
                                                <Button Margin="1" Tag="LU"
                                  Click="ScrollButtons_Click">LineUp()</Button>
                                                <Button Margin="1" Tag="LD"
                                Click="ScrollButtons_Click">LineDown()</Button>
                                              <Button  Margin="1" Tag="PU"
                                  Click="ScrollButtons_Click">PageUp()</Button>
                                                <Button Margin="1" Tag="PD"
                                Click="ScrollButtons_Click">PageDown()</Button>
                                                <Button Margin="1" Tag="TE"
                             Click="ScrollButtons_Click">ScrollToEnd()</Button>
                                                <Button Margin="1" Tag="TH"
                           Click="ScrollButtons_Click">ScrollToHome()</Button>
                                             <Button  Margin="1" Tag="TOV"
                Click="ScrollButtons_Click">ScrollToVerticalOffset(50)</Button>
                                                                    </WrapPanel>

                                    <ScrollViewer Grid.Row="2" Margin="1,1,1,5">
                            <TextBlock TextWrapping="Wrap" Background="BurlyWood">

                                                                     Normal Images
                                                                       </TextBlock>
                                                                    </ScrollViewer>
```

```
                <ScrollViewer Grid.Row="2" Grid.Column="1" Margin="1,1,1,5">
                      <TextBlock TextWrapping="Wrap" Background="BurlyWood">

                                              After 2 LineDown And 2 LineRight
                                                                    </TextBlock>
                                                                </ScrollViewer>
                <ScrollViewer Grid.Row="2" Grid.Column="2" Margin="1,1,1,5">
                      <TextBlock TextWrapping="Wrap" Background="BurlyWood">

                                              After 2 PageDown And 2 PageRight
                                                                    </TextBlock>
                                                                </ScrollViewer>
                <ScrollViewer Grid.Row="2" Grid.Column="3" Margin="1,1,1,5">
                      <TextBlock TextWrapping="Wrap" Background="BurlyWood">

                                          After ScrollToEnd And ScrollToRightEnd
                                                                    </TextBlock>
                                                                </ScrollViewer>
                <ScrollViewer Grid.Row="2" Grid.Column="4" Margin="1,1,1,5">
                      <TextBlock TextWrapping="Wrap" Background="BurlyWood">

                                           After ScrollToVerticalOffset (50) And
                                                    ScrollToHorizontalOffset(50)
                                                                    </TextBlock>
                                                                </ScrollViewer>

                                    <Button Margin="5" Grid.Row="1" Grid.Column="0"
                               VerticalAlignment="Center" HorizontalAlignment="Center">
                                                 <ScrollViewer Name="btnScroll1"
    HorizontalScrollBarVisibility="Auto" VerticalScrollBarVisibility="Auto">

                                    <Image VerticalAlignment="Top"  Stretch="Fill"
                                                        Source="WPF.jpg"></Image>

                                                                </ScrollViewer>
                                                                      </Button>
```

```
                        <Button Margin="5"  Grid.Row="1" Grid.Column="1"
                VerticalAlignment="Center" HorizontalAlignment="Center">
                                        <ScrollViewer Name="btnScroll2"
    HorizontalScrollBarVisibility="Auto" VerticalScrollBarVisibility="Auto">

                            <Image VerticalAlignment="Top"  Stretch="Fill"
                                            Source="WPF.jpg"></Image>

                                                    </ScrollViewer>
                                                          </Button>
                        <Button Margin="5" Grid.Row="1" Grid.Column="2"
                VerticalAlignment="Center" HorizontalAlignment="Center">
                                        <ScrollViewer Name="btnScroll3"
    HorizontalScrollBarVisibility="Auto" VerticalScrollBarVisibility="Auto">

                            <Image VerticalAlignment="Top"  Stretch="Fill"
                                            Source="WPF.jpg"></Image>

                                                    </ScrollViewer>
                                                          </Button>

                        <Button Margin="5" Grid.Row="1" Grid.Column="3"
                VerticalAlignment="Center" HorizontalAlignment="Center">
                                        <ScrollViewer Name="btnScroll4"
    HorizontalScrollBarVisibility="Auto" VerticalScrollBarVisibility="Auto">

                            <Image VerticalAlignment="Top"  Stretch="Fill"
                                            Source="WPF.jpg"></Image>

                                                    </ScrollViewer>
                                                          </Button>

                        <Button Margin="5" Grid.Row="1" Grid.Column="4"
                VerticalAlignment="Center" HorizontalAlignment="Center">
                                        <ScrollViewer Name="btnScroll5"
    HorizontalScrollBarVisibility="Auto" VerticalScrollBarVisibility="Auto">
```

```
                    <Image VerticalAlignment="Top"  Stretch="Fill"
                                        Source="WPF.jpg"></Image>

                                                    </ScrollViewer>
                                                          </Button>
                <WrapPanel Margin="10" Grid.ColumnSpan="5" Grid.Row="3"
                                                Orientation="Horizontal">
                                              <Button Margin="1" Tag="LL"
                                    Click="ScrollButtons_Click">LineLeft()</Button>
                                              <Button Margin="1" Tag="LR"
                                   Click="ScrollButtons_Click">LineRight()</Button>
                                             <Button  Margin="1" Tag="PL"
                                    Click="ScrollButtons_Click">PageLeft()</Button>
                                              <Button Margin="1" Tag="PR"
                                   Click="ScrollButtons_Click">PageRight()</Button>
                                              <Button Margin="1" Tag="TL"
                                Click="ScrollButtons_Click">ScrollToLeftEnd()</Button>
                                              <Button Margin="1" Tag="TR"
                               Click="ScrollButtons_Click">ScrollToRightEnd()</Button>
                                           <Button  Margin="1" Tag="TOH"
                        Click="ScrollButtons_Click">ScrollToHorizontalOffset(50)</Button>
                                                                  </WrapPanel>
                                                               </Grid>
                                                            </Window>
```

در قطعه کد فوق، دکمه هایی برای کنترل متد ها برای اسکرول عمودی و نیز دکمه هایی برای کنترل متد ها در اسکرول افقی بر روی دو کنترل WrapPanel قرار گرفته اند. در این مثال نحوه به کارگیری کنترل های کانتینر جهت چیدمان عناصر WPF به خوبی مشخص است.



بخش چهارم) Content Controls :کنترل های محتوا) (قسمت پنجم(

کنترل : GroupBox

همانطور که گفته شد، بعضی از کنترل های محتوا دارای دو بخش برای قرار دادن کنترل ها به عنوان محتوای خود دارند. یکی از این بخش ها همانطور که در کنترل های قبل نیز دیدید، خاصیت Content می باشد. دیگر خاصیتی که می تواند به عنوان محتوا به کار برود و فقط تعدادی ازکنترل ها دارای این خاصیت می باشند، خاصیت **Header** می باشد. ه این نوع کنترل ها اصطلاحا کنترل های **HeaderedContentControls**می گویند.
کنترل هایی که از این دسته می باشند و در این بخش بررسی آن ها خواهیم پرداخت، عبارتند از کنترل **GroupBox** ، آبجکت **TabItem** از کنترل **TabControl** و کنترل **Expander** که کنترل جدیدی در ویژوال استودیو 2008 می باشد.

ساده ترین کنترل بین این سه کنترل، کنترل **GroupBox** می باشد. همانطور که از نام این کنترل مشخص است این کنترل برای گروه بندی دسته ای از کنترل ها به کار می رود. یکی از کاربرد های موثر این کنترل در زمان استفاده از عناصر **RadioButton** در برنامه می باشد .

از **RadioButton** ها که به دکمه های رادیویی نیز معروف می باشند، زمانی استفاده می شود که کاربر مجاز به انتخاب تنها یک آیتم از بین چندین آیتم مختلف باشد. به عنوان مثال در فرمی از برنامه کاربر می خواهد که جنسیت خود را مشخص کند، در این حالت از دو دکمه رادیویی ، یکی برای زن و دیگری برای آیتم مرد استفاده خواهیم کرد. حال اگر این دکمه ها در قالب یک گروه مشخص نشوند، کاربر می تواند هر دو این دکمه ها را انتخاب کند (به اصطلاح تیک بزند) که خلاف منطق برنامه نویسی می باشد. در این زمان این دو دکمه را در قالب یک گروه معرفی می کنیم. از این پس برنامه به صورت اتوماتیک اجازه انتخاب فقط یکی از این دو دکمه را در هر لحظه به کاربر می دهد.
به قعطه کد زیر توجه فرمایید:

کد:

```
<Grid>
<Grid.RowDefinitions>
<RowDefinition Height="Auto"></RowDefinition>
<RowDefinition Height="Auto"></RowDefinition>
</Grid.RowDefinitions>
<GroupBox Header="Group Box 1 " Margin="10" Padding="10"
VerticalAlignment="Top">
<StackPanel>
<RadioButton Margin="3" Name="rbtn1" IsChecked="True" >radio
button
1</RadioButton>
<RadioButton Margin="3" Name="rbtn2">radio button
2</RadioButton>
<RadioButton Margin="3" Name="rbtn3">radio button
3</RadioButton>
<RadioButton Margin="3" Name="rbtn4">radio button
4</RadioButton>
</StackPanel>
</GroupBox>
<GroupBox Grid.Row="1" Header="GroupBox 2 " Margin="10" Padding="10"
VerticalAlignment="Top">
<StackPanel>
<RadioButton Margin="3" IsChecked="True" Name="rbtn5" >radio
button
5</RadioButton>
<RadioButton Margin="3" Name="rbtn6">radio button
6</RadioButton>
<RadioButton Margin="3" Name="rbtn7">radio button
7</RadioButton>
<RadioButton Margin="3" Name="rbtn8">radio button
8</RadioButton>
</StackPanel>
</GroupBox>
</Grid>
```

در کد فوق، برای گرید اصلی پنجره دو سطر تعریف شده است. یکی از **GroupBox** ها در سط اول یا سطر شماره

0 و دیگری درون سطر شماره 1 از کنترل گرید قرار گرفته است. هر کنترل **GroupBox** دارای 4 دکمه رادیویی می باشد.همچنین در کنترل های GroupBox از کنترل StackPanel نیز برای چیدمان بهتر دکمه های رادیویی استفاده شده است.
همانطور که در شکل می بینید، دکمه های رادیویی در هر گروه از دکمه های رادیویی گروه دیگر کاملا مجزا می باشند و انتخاب شدن و تیک خوردن آن ها تاثیری بر یکدیگر ندارد. ولی در هر گروه تنها می توانید یک دکمه رادیویی را انتخاب نمایید.
نتیجه اجرای کد های فوق را در شکل زیر مشاهده می کنید:

خاصیت: Groupname

این خاصیت مربوط به کنترل **GroupBox** نمی باشد و مربوط به کنترل های رادیوی می باشد. ولی چون از این دکمه ها در این مثال استفاده کردیم، این خاصیت را در اینجا توضیح میدهم.
در بعضی از موارد ممکن است حالتی پیش آید که بخواهید کنترل ها را در GroupBox های مختلف ولی در قالب یک گروه سازماندهی کنید. در این حالت نیز کاربر بایستی در هر لحظه فقط انتخاب یکی از آیتم ها را داشته باشد. اگر چه این آیتم ها در **GroupBox** های مختلفی چیده شده باشند. در این حالت از خاصیت GroupName مربوط به دکمه های رادیوی استفاده می کنیم. حال به کد زیر که تغییر کرده کد فوق می باشد، دقت کنید:

کد:

```
                                                                <Grid>
                                                        <Grid.RowDefinitions>
                                                <RowDefinition Height="Auto"></RowDefinition>
                                                <RowDefinition Height="Auto"></RowDefinition>
                                                        </Grid.RowDefinitions>
                                    <GroupBox Header="Group Box 1 " Margin="10" Padding="10"
                                                        VerticalAlignment="Top">
                                                                <StackPanel>
                            <RadioButton  GroupName="sameGroup" Margin="3" Name="rbtn1"
                                                                IsChecked="True"
                                                        >radio button 1</RadioButton>
                                                <RadioButton GroupName="sameGroup" Margin="3"
                                                        Name="rbtn2">radio button
                                                                2</RadioButton>
                                                <RadioButton GroupName="sameGroup" Margin="3"
                                                        Name="rbtn3">radio button
```

```
                                                    3</RadioButton>
                                <RadioButton GroupName="sameGroup" Margin="3"
                                                Name="rbtn4">radio button
                                                    4</RadioButton>
                                                  </StackPanel>
                                                    </GroupBox>
        <GroupBox Grid.Row="1" Header="GroupBox 2 " Margin="10" Padding="10"
                                                  VerticalAlignment="Top">
                                                    <StackPanel>
                               <RadioButton GroupName="sameGroup"  Margin="3"
                                            IsChecked="True" Name="rbtn5"
                                             >radio button 5</RadioButton>
                               <RadioButton GroupName="sameGroup"  Margin="3"
                                                Name="rbtn6">radio button
                                                    6</RadioButton>
                                <RadioButton GroupName="sameGroup" Margin="3"
                                                Name="rbtn7">radio button
                                                    7</RadioButton>
                                <RadioButton GroupName="sameGroup" Margin="3"
                                                Name="rbtn8">radio button
                                                    8</RadioButton>
                                                  </StackPanel>
                                                    </GroupBox>
                                                        </Grid>
```

در کد فوق، خاصیت **GroupName** همه دکمه های رادیویی با یک اسم یکسان مشخص شده اند. این عمل با عث می شود که هر دو یا چند دکمه رادیویی که دارای مقدار یکسان در خاصیت GroupName می باشند به عنوان یک گروه
مشخص شوند. همانطور که در شکل زیر مشاهده می کنید

اگر چه دارای دو کنترل **GroupBox** جدا هستیم، و کنترل های رادیویی در دو **GroupBox** مختلف چیده شده اند، ولی چون خاصیت GroupName همه ان ها دارای یک مقدار مشخص می باشند، در هر لحظه تنها می توانید یکی از هشت دکمه رادیویی فوق را انتخاب نمایید.

نکته:

در به کارگیری خاصیت **GroupName** محتاط باشید !!!در به کار گیری خاصیت **GroupName** به این نکته دقت داشته باشید، که تنها ئکمه هایی که دارای مقدار یکسان برای این خاصیت باشند، به عنوان یک گروه تلقی می شوند. به عنوان مثال اگر تنها دو دکمه از هشت دکمه رادیویی بالا (یکی در **GroupBox1** و دیگری در(**GroupBox2** را با خاصیت **GroupName** داده شده، مقدار دهی کنید، تنها همین دو دکمه به عنوان یک گروه مشخص می
شوند و کاربر می تواند همزمان دو دکمه را در GroupBox1 و یا دو دکمه را در GroupBox2 انتخاب نماید.
**

هدر های پیشرفته در کنترل: GroupBox

همانطور که اشاره شد، کنترل **GroupBox** از ان دسته کنترل های محتوایی است که دارای یک بخش به نام **header** می باشد. این بخش که با خاصیت **Header** مشخص می شود، می توانند مانند خاصیت **Content** دارای یک عنصر باشد. حال این عنصر می تواند یک متن ساده باشد و یا می تواند یک کنترل کانتینر باشد که خود شامل چندین کنترل فرزند می باشد.

شکل ساده خاصیت **Header** در کنترل **GroupBox** همانی است که در مثال قبل دیدید. یعنی قرار دادن تنها یک متن ساده به عنوان هدر کنترل. GroupBox

اما در **WPF** می توانید با کمی به کار گیری ذوق و سلیقه و با به کار گیری کنترل های کانتینر که همانطور که گفتم از اساسی ترین مباحث **WPF**می باشند و در بخش قبلی به صورت کامل با آن ها آشنا شدید، کنترل های GroupBoxسفارشی زیبایی
بسازید و دیگر نیاز مند کنترل های نوشته شده توسط شرکت و یا اشخاص ثالث نباشید.
به عنوان مثال به نمونه کد زیر دقت کنید:

کد:

```
<Window x:Class="contentControls.HeaderdGroupBoxContent"
xmlns="http://schemas.microsoft.com/winfx/2...l/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
Title="HeaderdGroupBoxContent" Height="220" Width="350">
<Grid>
<Grid.ColumnDefinitions>
<ColumnDefinition Width="Auto"></ColumnDefinition>
<ColumnDefinition Width="Auto"></ColumnDefinition>
</Grid.ColumnDefinitions>
<GroupBox  Margin="10" Padding="10" VerticalAlignment="Top">
<GroupBox.Header>
<StackPanel Orientation="Horizontal">
```

```
                        <Image Source=".\Images\star.png" MaxHeight="20"
                                                    MaxWidth="20"
                                              Margin="0,0,2,0"></Image>
                     <TextBlock Margin="5,0,5,0"> ( Select Score )
                                                 <TextBlock.Foreground>
                                                  <LinearGradientBrush>
                                             <LinearGradientBrush.GradientStops>
                                          <GradientStop Color="Brown" Offset="0"/>
                                       <GradientStop Color="Bisque"  Offset=".5"/>
                                          <GradientStop Color="Black" Offset="1"/>
                                              </LinearGradientBrush.GradientStops>
                                                 </LinearGradientBrush>
                                               </TextBlock.Foreground>
                                                           </TextBlock>
                                                         </StackPanel>
                                                     </GroupBox.Header>
                                                           <StackPanel>
                    <RadioButton Margin="3" Name="rbtn1" IsChecked="True" >
                           <Image Source=".\Images\star.png" MaxHeight="16"
                                                 MaxWidth="16"></Image>
                                                        </RadioButton>
                                    <RadioButton Margin="3" Name="rbtn2">
                                    <StackPanel Orientation="Horizontal">
                          <Image Source=".\Images\star.png" MaxHeight="16"
                                                 MaxWidth="16"></Image>
                          <Image Source=".\Images\star.png" MaxHeight="16"
                                                 MaxWidth="16"></Image>
                                                          </StackPanel>
                                                        </RadioButton>
                                    <RadioButton Margin="3" Name="rbtn3">
                                    <StackPanel Orientation="Horizontal">
                          <Image Source=".\Images\star.png" MaxHeight="16"
                                                 MaxWidth="16"></Image>
                          <Image Source=".\Images\star.png" MaxHeight="16"
                                                 MaxWidth="16"></Image>
                          <Image Source=".\Images\star.png" MaxHeight="16"
                                                 MaxWidth="16"></Image>
                                                          </StackPanel>
```

```
                                                    </RadioButton>
                                <RadioButton Margin="3" Name="rbtn4">
                                <StackPanel Orientation="Horizontal">
                        <Image Source=".\Images\star.png" MaxHeight="16"
                                                MaxWidth="16"></Image>
                        <Image Source=".\Images\star.png" MaxHeight="16"
                                                MaxWidth="16"></Image>
                        <Image Source=".\Images\star.png" MaxHeight="16"
                                                MaxWidth="16"></Image>
                        <Image Source=".\Images\star.png" MaxHeight="16"
                                                MaxWidth="16"></Image>
                                                  </StackPanel>
                                                 </RadioButton>
                                <RadioButton Margin="3" Name="rbtn5">
                                <StackPanel Orientation="Horizontal">
                        <Image Source=".\Images\star.png" MaxHeight="16"
                                                MaxWidth="16"></Image>
                        <Image Source=".\Images\star.png" MaxHeight="16"
                                                MaxWidth="16"></Image>
                        <Image Source=".\Images\star.png" MaxHeight="16"
                                                MaxWidth="16"></Image>
                        <Image Source=".\Images\star.png" MaxHeight="16"
                                                MaxWidth="16"></Image>
                        <Image Source=".\Images\star.png" MaxHeight="16"
                                                MaxWidth="16"></Image>
                                                  </StackPanel>
                                                 </RadioButton>
                                                  </StackPanel>
                                                    </GroupBox>
                    <GroupBox Grid.Column="1"  Margin="10" Padding="10"
                                                VerticalAlignment="Top">
                                                      <GroupBox.Header>
                                <StackPanel Orientation="Horizontal">
                    <Image Source=".\Images\banner.png" MaxHeight="32"
                                                MaxWidth="32"></Image>
                        <TextBlock Margin="5,0,5,0"> ( Select Country )
                                                <TextBlock.Foreground>
                                                 <LinearGradientBrush>
```

```
                                        <LinearGradientBrush.GradientStops>
                                    <GradientStop Color="Brown"  Offset="0"/>
                                     <GradientStop Color="Gray" Offset=".5"/>
                                       <GradientStop Color="Red" Offset="1"/>
                                        </LinearGradientBrush.GradientStops>
                                                    </LinearGradientBrush>
                                                </TextBlock.Foreground>
                                                            </TextBlock>
                                                          </StackPanel>
                                                      </GroupBox.Header>
                                                            <StackPanel>
                            <RadioButton  Margin="3"  Name="rbtn6" IsChecked="True">
                                            <StackPanel Orientation="Horizontal">
                                    <Image Source=".\Images\Iran.ico" MaxWidth="20"
                                                                MaxHeight="20"/>
                                                <TextBlock>(Iran)</TextBlock>
                                                              </StackPanel>
                                                            </RadioButton>
                                         <RadioButton   Margin="3" Name="rbtn7">
                                            <StackPanel Orientation="Horizontal">
                                  <Image Source=".\Images\Albania.ico" MaxWidth="20"
                                                                MaxHeight="20"/>
                                                    <TextBlock>(Albani)</TextBlock>
                                                              </StackPanel>
                                                            </RadioButton>
                                            <RadioButton Margin="3" Name="rbtn8">
                                            <StackPanel Orientation="Horizontal">
                                <Image Source=".\Images\Hong-Kong.ico" MaxWidth="20"
                                                                MaxHeight="20"/>
                                                  <TextBlock>(Hong-Kong)</TextBlock>
                                                              </StackPanel>
                                                            </RadioButton>
                                            <RadioButton Margin="3" Name="rbtn9">
                                            <StackPanel Orientation="Horizontal">
                                    <Image Source=".\Images\Italy.ico" MaxWidth="20"
                                                                MaxHeight="20"/>
                                                    <TextBlock>(Italy)</TextBlock>
                                                              </StackPanel>
```

```
</RadioButton>
</StackPanel>
</GroupBox>
</Grid>
</Window>
```

قبل از توضیحات مربوط به کد فوق، نتیجه حاصل از اجرای آن را در شکل زیر با هم مشاهده می کنیم

همانطور که در شکل فوق مشاهده می کنید، دو کنترل **GroupBox** سفارشی برای امتیاز ها و کشور ها ایجاد شده است. همانطور که می بینید، هدر کنترل های GroupBox از حالت متنی خارج شده است.
در کد فوق، خاصیت **Header** هر یک **GroupBox** ها به صورت زیر تعریف شده است:

کد:

```
<GroupBox  . . . >
.
.
.
<GroupBox.Header>
.
.
.
</GroupBox.Header>
.
.
.
</GroupBox  >
```

دلیل اینکه خاصیت **Header** به صورت فوق تعریف شده است، این است که می خواهیم درون این خاصیت عکس و متن را در کنار یکدیگر قرار دهیم و یک هدر سفارشی برای آن ایجاد نماییم.
قبلا نیز اشاره کرده بودم که در زمانی که می خواهیم از مقادیر پیچیده و پیشرفته ای برای یک خاصیت استفاده کنیم بایستی آن ها را از حالت Attribute خارج کنیم و به شکل خود عناصر تعریف نماییم.
حال می توانیم درون خاصیت **Header** یک کنترل کانتینر (متناسب با نیاز) تعریف کنیم و کنترل های خود را درون ان قرار دهیم. به عنوان مثال، در مثال قبل، در خاصیت **Header** هر یک از **GroupBox** ها یک
عنصر **StackPanel** تعریف کرده ایم که خاصیت **Orientation** آن بر روی **Horizontal** تنظیم شده است.
سپس درون عنصر **StackPanel** مذکور، دو عنصر دیگر، یکی از نوع **Image** و دیگری از نوع TextBlock تعریف کرده ایم. عنصر **Image** وظیفه نگه داری عکسی را که در خاصیت Source آن مشخص
شده است را دارد و عنصر **TextBlock** متن مشخص شده درون ان را نشان می دهد.
علاوه بر این ، خاصیت **Content** مربوط به هر یک از دکمه های رادیویی نیز از حالت ساده تک متنی خارج شده و توسط کنترل های کانتیر به مقادیر پیچشده و پیشرفته نسبت داده شده اند. و بدین ترتیب کنترل
های **GroupBox** کاملا سفارشی
ایجاد شده اند.
دقت کنید که کد، فوق یکی از ساده ترین شکل ها برای تعریف عناصر سفارشی می باشد. در **WPF** امکانات بسیاری برای شما ایجاد شده اند که می توانید توسط آن ها کنترل های سفارشی زیبا و متناسب با پروژه های خود را ایجاد نمایید.
به عنوان مثال با تغییر کوچکی در کد فوق، و بدون تغییر دادن عکس های استفاده شده می توانید **GroupBox** سمت چپ را به شکلی در آورید که در زیر مشاهده می کنید:

همانطور که می بینید، کنترل **GroupBox** مربوط به **Score** ها، ظاهر بسیار زیباتری نسبت به نمونه قبلی خودش پیدا کرده است.

بخش چهارم) Content Controls :کنترل های محتوا) (قسمت ششم(

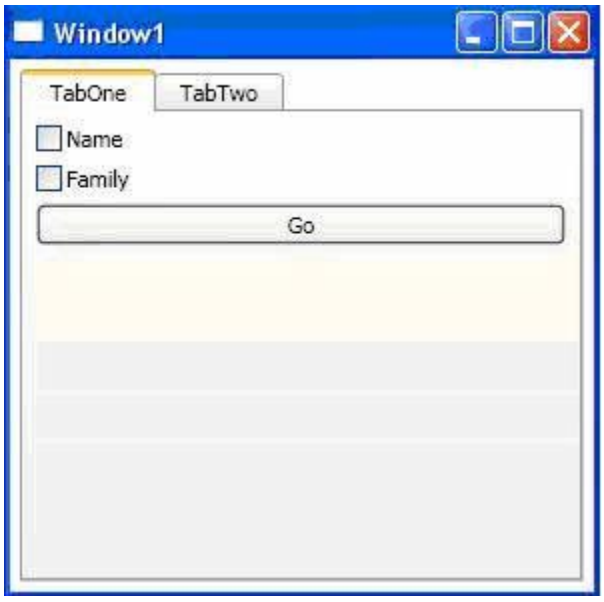
کنترل: TabControl

با نام این کنترل و عملکرد این کنترل در دات نتفریم ورک 2.0 آشنا شدید. در این جا قصد صحبت کردن در مورد این کنترل را ندارم، و بیشتر به شرح خاصیت **TabItem** از این کنترل خواهیم پرداخت. ولی به عنوان اشاره کوچکی در مورد ساخت این کنترل در **WPF** به کد زیر دقت کنید (کد نوشته شده توسط کاربر(*sajadlove*

کد:

```
<TabControl Margin="5">
<TabItem Header="TabOne">
<StackPanel>
<CheckBox Margin="3">Name</CheckBox>
<CheckBox Margin="3">Family</CheckBox>
<Button Margin="3">Go</Button>
</StackPanel>
</TabItem>
<TabItem Header="TabTwo">
<StackPanel>
<CheckBox Margin="3">SettingOne</CheckBox>
<CheckBox Margin="3">SettingTwo</CheckBox>
<Button Margin="3">Go</Button>
</StackPanel>
</TabItem>
</TabControl>
```

نتیجه حاصل از اجرای کد فوق (درون یک window را در شکل زیر مشاهده می کنید(



خاصیت: TabItem

خاصیت **TabItem** شبیه به **TabPage** در کنترل **TabControl** در دات نت فریم ورک 2.0 می باشد. در واقع هر **TabItem** یک صفحه را در کنترل tabControl مشخص می کند. عنصر **TabItem** نیزاز ان دسته از عناصری است که دارای خاصیت **header** می باشد. همانند **GroupBox**خاصیت **Header** از عنصر **TabItem** نیز می تواند، شامل یک متن ساده باشد و یا اینکه شامل یک محتوای پیچیده تشکیل شده از چندین عنصر باشد.

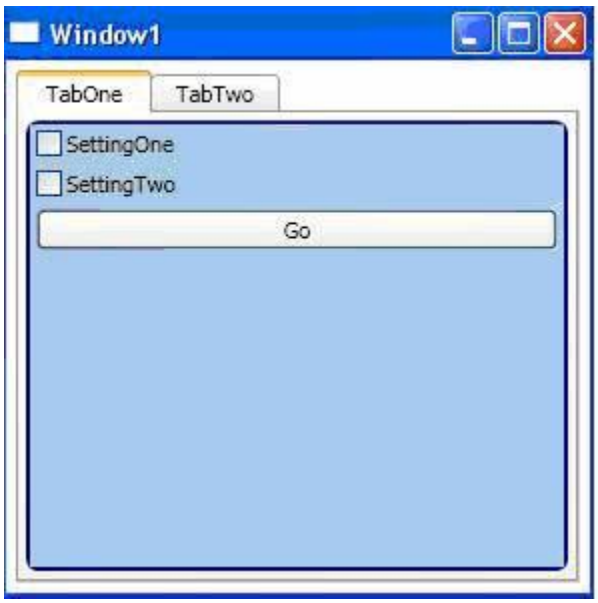
به کد زیر دقت کنید ...(نوشته شده توسط کاربر*sajadlove*)

کد:

```
<Grid>
<TabControl Margin="5">
<TabItem Header="TabOne">
<TabItem.Background>
<LinearGradientBrush>
<LinearGradientBrush.GradientStops>
<GradientStop Color="LightSteelBlue" Offset="0"/>
<GradientStop Color="LightBlue" Offset="1" />
</LinearGradientBrush.GradientStops>
</LinearGradientBrush>
</TabItem.Background>
<Border BorderBrush="DarkBlue" BorderThickness="2" CornerRadius="5">
<StackPanel>

<StackPanel.Background>

<LinearGradientBrush>
<LinearGradientBrush.GradientStops>
<GradientStop Color="LightSteelBlue" Offset="0"/>
<GradientStop Color="LightBlue" Offset="1"/>
</LinearGradientBrush.GradientStops>
</LinearGradientBrush>
</StackPanel.Background>
<CheckBox Margin="3">SettingOne</CheckBox>
<CheckBox Margin="3">SettingTwo</CheckBox>
<Button Margin="3">Go</Button>
</StackPanel>
</Border>
```

```
                </TabItem>
                <TabItem>
                <TabItem.Header>
                    <StackPanel>
                <TextBlock>TabTwo</TextBlock>
                    </StackPanel>
                </TabItem.Header>
                    <StackPanel>
            <CheckBox Margin="3">SettingOne</CheckBox>
            <CheckBox Margin="3">SettingTwo</CheckBox>
                <Button Margin="3">Go</Button>
                    </StackPanel>
                </TabItem>
            </TabControl>
        </Grid>
```

نتیجه اجرا:



حال به مثال جامع تر زیر که تر کیب، پست های قبلی با مطالب این پست می باشد نیز توجه فر مایید..
کد:

```
```

```
                                                    <TabControl Margin="10">
                                                <TabItem Header="Simpe Header">
            <TextBlock TextAlignment="Center" HorizontalAlignment="Center"
        VerticalAlignment="Center" FontSize="18">Without Content</TextBlock>
                                                                    </TabItem>
                                                                    <TabItem >
                                                            <TabItem.Header>
                                        <StackPanel Orientation="Horizontal">
            <Image Source=".\Images\star.png" MaxHeight="20" MaxWidth="20"
                                                    Margin="0,0,2,0"></Image>
                                <TextBlock Margin="5,0,5,0"> ( Score Tab Page )
                                                        <TextBlock.Foreground>
                                                         <LinearGradientBrush>
                                                <LinearGradientBrush.GradientStops>
                                            <GradientStop Color="Brown" Offset="0"/>
                                           <GradientStop Color="Green" Offset=".5"/>
                                            <GradientStop Color="Black" Offset="1"/>
                                                </LinearGradientBrush.GradientStops>
                                                                </LinearGradientBrush>
                                                               </TextBlock.Foreground>
                                                                          </TextBlock>
                                                                        </StackPanel>
                                                                </TabItem.Header>
                  <GroupBox Margin="10" Padding="10" VerticalAlignment="Top">

                                                            <GroupBox.Header>
                                        <StackPanel Orientation="Horizontal">
            <Image Source=".\Images\star.png" MaxHeight="20" MaxWidth="20"
                                                    Margin="0,0,2,0"></Image>
                                  <TextBlock Margin="5,0,5,0"> ( Select Score )
                                                        <TextBlock.Foreground>
                                                         <LinearGradientBrush>
                                                <LinearGradientBrush.GradientStops>
                                            <GradientStop Color="Brown" Offset="0"/>
                                          <GradientStop Color="Bisque" Offset=".5"/>
                                            <GradientStop Color="Black" Offset="1"/>
                                                </LinearGradientBrush.GradientStops>
                                                                </LinearGradientBrush>
```

```
                                                    </TextBlock.Foreground>
                                                              </TextBlock>
                                                            </StackPanel>
                                                          </GroupBox.Header>
                                                                <StackPanel>
                    <RadioButton Margin="3" Name="rbtn1" IsChecked="True" >
      <Image Source=".\Images\star.png" MaxHeight="16" MaxWidth="16"></Image>
                                                                </RadioButton>

                                          <RadioButton Margin="3" Name="rbtn2">
                                          <StackPanel Orientation="Horizontal">
      <Image Source=".\Images\star.png" MaxHeight="16" MaxWidth="16"></Image>
      <Image Source=".\Images\star.png" MaxHeight="16" MaxWidth="16"></Image>
                                                                <StackPanel>
                                                              </RadioButton>
                                          <RadioButton Margin="3" Name="rbtn3">
                                          <StackPanel Orientation="Horizontal">
      <Image Source=".\Images\star.png" MaxHeight="16" MaxWidth="16"></Image>
      <Image Source=".\Images\star.png" MaxHeight="16" MaxWidth="16"></Image>
      <Image Source=".\Images\star.png" MaxHeight="16" MaxWidth="16"></Image>
                                                                <StackPanel>
                                                              </RadioButton>
                                          <RadioButton Margin="3" Name="rbtn4">
                                          <StackPanel Orientation="Horizontal">
      <Image Source=".\Images\star.png" MaxHeight="16" MaxWidth="16"></Image>
      <Image Source=".\Images\star.png" MaxHeight="16" MaxWidth="16"></Image>
      <Image Source=".\Images\star.png" MaxHeight="16" MaxWidth="16"></Image>
      <Image Source=".\Images\star.png" MaxHeight="16" MaxWidth="16"></Image>
                                                                <StackPanel>
                                                              </RadioButton>
                                          <RadioButton Margin="3" Name="rbtn5">
                                          <StackPanel Orientation="Horizontal">
      <Image Source=".\Images\star.png" MaxHeight="16" MaxWidth="16"></Image>
      <Image Source=".\Images\star.png" MaxHeight="16" MaxWidth="16"></Image>
      <Image Source=".\Images\star.png" MaxHeight="16" MaxWidth="16"></Image>
      <Image Source=".\Images\star.png" MaxHeight="16" MaxWidth="16"></Image>
      <Image Source=".\Images\star.png" MaxHeight="16" MaxWidth="16"></Image>
                                                                <StackPanel>
```

```
                                                                    </RadioButton>
                                                                     </StackPanel>
                                                                       </GroupBox>
                                                                        </TabItem>
                                                                        <TabItem >
                                                                <TabItem.Header>
                                                        <StackPanel Orientation="Horizontal">
    <Image Source=".\Images\banner.png" MaxHeight="32" MaxWidth="32"></Image>
                                        <TextBlock Margin="5,0,5,0"> ( Country Page )
                                                                    <TextBlock.Foreground>
                                                                     <LinearGradientBrush>
                                                                  <LinearGradientBrush.GradientStops>
                                                               <GradientStop Color="Brown" Offset="0"/>
                                                               <GradientStop Color="Gray" Offset=".5"/>
                                                                 <GradientStop Color="Red" Offset="1"/>
                                                                   </LinearGradientBrush.GradientStops>
                                                                               </LinearGradientBrush>
                                                                            </TextBlock.Foreground>
                                                                                        </TextBlock>
                                                                                       </StackPanel>
                                                                                </TabItem.Header>
                                                <GroupBox Margin="10" Padding="10" VerticalAlignment="Top">
                                                                                <GroupBox.Header>
                                                                 <StackPanel Orientation="Horizontal">
    <Image Source=".\Images\banner.png" MaxHeight="32" MaxWidth="32"></Image>
                                                    <TextBlock Margin="5,0,5,0"> ( Select Country )
                                                                    <TextBlock.Foreground>
                                                                     <LinearGradientBrush>
                                                                  <LinearGradientBrush.GradientStops>
                                                               <GradientStop Color="Brown" Offset="0"/>
                                                               <GradientStop Color="Gray" Offset=".5"/>
                                                                 <GradientStop Color="Red" Offset="1"/>
                                                                   </LinearGradientBrush.GradientStops>
                                                                               </LinearGradientBrush>
                                                                            </TextBlock.Foreground>
                                                                                        </TextBlock>
                                                                                       </StackPanel>
                                                                                </GroupBox.Header>
```

```
                                                    <StackPanel>
                    <RadioButton Margin="3" Name="rbtn6" IsChecked="True">
                                        <StackPanel Orientation="Horizontal">
          <Image Source=".\Images\Iran.ico" MaxWidth="20" MaxHeight="20"/>
                                            <TextBlock>(Iran)</TextBlock>
                                                    </StackPanel>
                                                   </RadioButton>
                                  <RadioButton   Margin="3" Name="rbtn7">
                                    <StackPanel Orientation="Horizontal">
        <Image Source=".\Images\Albania.ico" MaxWidth="20" MaxHeight="20"/>
                                           <TextBlock>(Albani)</TextBlock>
                                                    </StackPanel>
                                                   </RadioButton>
                                    <RadioButton Margin="3" Name="rbtn8">
                                    <StackPanel Orientation="Horizontal">
      <Image Source=".\Images\Hong-Kong.ico" MaxWidth="20" MaxHeight="20"/>
                                       <TextBlock>(Hong-Kong)</TextBlock>
                                                    </StackPanel>
                                                   </RadioButton>
                                    <RadioButton Margin="3" Name="rbtn9">
                                    <StackPanel Orientation="Horizontal">
         <Image Source=".\Images\Italy.ico" MaxWidth="20" MaxHeight="20"/>
                                           <TextBlock>(Italy)</TextBlock>
                                                    </StackPanel>
                                                   </RadioButton>
                                                    </StackPanel>
                                                      </GroupBox>
                                                       </TabItem>
                                                       <TabItem >
                                                  <TabItem.Header>
                                                        <StackPanel>
           <Image Source=".\Images\star.png" MaxHeight="20" MaxWidth="20"
                                                 Margin="0,0,2,0"></Image>
                                    <TextBlock Margin="5,0,5,0"> ( Score )
                                                  <TextBlock.Foreground>
                                                   <LinearGradientBrush>
                                      <LinearGradientBrush.GradientStops>
                                   <GradientStop Color="Brown" Offset="0"/>
```

```
                                    <GradientStop Color="Bisque" Offset=".5"/>
                                      <GradientStop Color="Black" Offset="1"/>
                                          </LinearGradientBrush.GradientStops>
                                                      </LinearGradientBrush>
                                                    </TextBlock.Foreground>
                                                                  </TextBlock>
    <Image Source=".\Images\banner.png" MaxHeight="32" MaxWidth="32"></Image>
                                      <TextBlock Margin="5,0,5,0"> ( Country )
                                                    <TextBlock.Foreground>
                                                     <LinearGradientBrush>
                                           <LinearGradientBrush.GradientStops>
                                      <GradientStop Color="Brown" Offset="0"/>
                                      <GradientStop Color="Gray" Offset=".5"/>
                                        <GradientStop Color="Red" Offset="1"/>
                                          </LinearGradientBrush.GradientStops>
                                                      </LinearGradientBrush>
                                                    </TextBlock.Foreground>
                                                                  </TextBlock>
                                                                </StackPanel>
                                                              </TabItem.Header>
                                                                        <Grid>
                                                   <Grid.ColumnDefinitions>
                              <ColumnDefinition Width="Auto"></ColumnDefinition>
                              <ColumnDefinition Width="Auto"></ColumnDefinition>
                                                  </Grid.ColumnDefinitions>
                  <GroupBox Margin="10" Padding="10" VerticalAlignment="Top">

                                                       <GroupBox.Header>
                                     <StackPanel Orientation="Horizontal">
              <Image Source=".\Images\star.png" MaxHeight="20" MaxWidth="20"
                                                 Margin="0,0,2,0"></Image>
                                <TextBlock Margin="5,0,5,0"> ( Select Score )
                                                       <TextBlock.Foreground>
                                                        <LinearGradientBrush>
                                           <LinearGradientBrush.GradientStops>
                                      <GradientStop Color="Brown" Offset="0"/>
                                    <GradientStop Color="Bisque" Offset=".5"/>
                                      <GradientStop Color="Black" Offset="1"/>
```

```
                                        </LinearGradientBrush.GradientStops>
                                                    </LinearGradientBrush>
                                                </TextBlock.Foreground>
                                                            </TextBlock>
                                                        </StackPanel>
                                                    </GroupBox.Header>
                                                        <StackPanel>
                    <RadioButton Margin="3" Name="rbtn10" IsChecked="True" >
      <Image Source=".\Images\star.png" MaxHeight="16" MaxWidth="16"></Image>
                                                                </RadioButton>

                                        <RadioButton Margin="3" Name="rbtn11">
                                         <StackPanel Orientation="Horizontal">
      <Image Source=".\Images\star.png" MaxHeight="16" MaxWidth="16"></Image>
      <Image Source=".\Images\star.png" MaxHeight="16" MaxWidth="16"></Image>
                                                            </StackPanel>
                                                        </RadioButton>
                                        <RadioButton Margin="3" Name="rbtn12">
                                         <StackPanel Orientation="Horizontal">
      <Image Source=".\Images\star.png" MaxHeight="16" MaxWidth="16"></Image>
      <Image Source=".\Images\star.png" MaxHeight="16" MaxWidth="16"></Image>
      <Image Source=".\Images\star.png" MaxHeight="16" MaxWidth="16"></Image>
                                                            </StackPanel>
                                                        </RadioButton>
                                        <RadioButton Margin="3" Name="rbtn13">
                                         <StackPanel Orientation="Horizontal">
      <Image Source=".\Images\star.png" MaxHeight="16" MaxWidth="16"></Image>
      <Image Source=".\Images\star.png" MaxHeight="16" MaxWidth="16"></Image>
      <Image Source=".\Images\star.png" MaxHeight="16" MaxWidth="16"></Image>
      <Image Source=".\Images\star.png" MaxHeight="16" MaxWidth="16"></Image>
                                                            </StackPanel>
                                                        </RadioButton>
                                        <RadioButton Margin="3" Name="rbtn14">
                                         <StackPanel Orientation="Horizontal">
      <Image Source=".\Images\star.png" MaxHeight="16" MaxWidth="16"></Image>
      <Image Source=".\Images\star.png" MaxHeight="16" MaxWidth="16"></Image>
      <Image Source=".\Images\star.png" MaxHeight="16" MaxWidth="16"></Image>
      <Image Source=".\Images\star.png" MaxHeight="16" MaxWidth="16"></Image>
```

```
      <Image Source=".\Images\star.png" MaxHeight="16" MaxWidth="16"></Image>
                                                                    </StackPanel>
                                                                  </RadioButton>
                                                                   </StackPanel>
                                                                     </GroupBox>
  <GroupBox Grid.Column="1" Margin="10" Padding="10" VerticalAlignment="Top">
                                                                 <GroupBox.Header>
                                              <StackPanel Orientation="Horizontal">
    <Image Source=".\Images\banner.png" MaxHeight="32" MaxWidth="32"></Image>
                                 <TextBlock Margin="5,0,5,0"> ( Select Country )
                                                          <TextBlock.Foreground>
                                                           <LinearGradientBrush>
                                                <LinearGradientBrush.GradientStops>
                                            <GradientStop Color="Brown" Offset="0"/>
                                            <GradientStop Color="Gray" Offset=".5"/>
                                              <GradientStop Color="Red" Offset="1"/>
                                                </LinearGradientBrush.GradientStops>
                                                                  </LinearGradientBrush>
                                                               </TextBlock.Foreground>
                                                                        </TextBlock>
                                                                    </StackPanel>
                                                                </GroupBox.Header>
                                                                        <StackPanel>
                          <RadioButton Margin="3" Name="rbtn15" IsChecked="True">
                                              <StackPanel Orientation="Horizontal">
              <Image Source=".\Images\Iran.ico" MaxWidth="20" MaxHeight="20"/>
                                                 <TextBlock>(Iran)</TextBlock>
                                                                    </StackPanel>
                                                                  </RadioButton>
                                       <RadioButton   Margin="3" Name="rbtn16">
                                                 <StackPanel Orientation="Horizontal">
           <Image Source=".\Images\Albania.ico" MaxWidth="20" MaxHeight="20"/>
                                                 <TextBlock>(Albani)</TextBlock>
                                                                    </StackPanel>
                                                                  </RadioButton>
                                                <RadioButton Margin="3" Name="rbtn17">
                                                 <StackPanel Orientation="Horizontal">
          <Image Source=".\Images\Hong-Kong.ico" MaxWidth="20" MaxHeight="20"/>
```

```
                                        <TextBlock>(Hong-Kong)</TextBlock>
                                    </StackPanel>
                                </RadioButton>
                            <RadioButton Margin="3" Name="rbtn18">
                                <StackPanel Orientation="Horizontal">
        <Image Source=".\Images\Italy.ico" MaxWidth="20" MaxHeight="20"/>
                                        <TextBlock>(Italy)</TextBlock>
                                    </StackPanel>
                                </RadioButton>
                            </StackPanel>
                        </GroupBox>
                    </Grid>
                </TabItem>



            </TabControl>
```

حال، نگاهی به نتیجه اجرای مثال فوق خواهیم انداخت و توضیحات مربوطه را در هر بخش خواهید دید. مثال فوق که کد کاملی می باشد،(می توانید آن را در یک پنجره و در داخل عنصر **Grid** کپی کنید و اجرا نمایید) یک پنجره که دارای یک عنصر **TabControl** می باشد را نشان می دهد. عنصر **TabControl** دارای چهار **Page** یا جهار **TabItem** می باشد که به ترتیب در شکل های زیر مشاهده می کنید:

صفحه اول:

اولین عنصر **TabItem** دارای یک متن ساده در **header** خود می باشد و نیز یک عنصر **TextBlock** در خاصیت **Content** خود می باشد. این ساده ترین شکل از یک عنصر **TabItem** می تواند باشد.

صفحه دوم:

این صفحه دارای یک کنترل **StackPanel** با خاصیت **Orientaion** با مقدار **Horizontal** می باشد. درون این کنترل یک عکس و یک متن درون عنصر TextBlock قرار گرفته است. مقدار خاصیت **Content** این عنصر همان **GroupBox** سمت چپ مثال قبل می باشد که یک محتوای پیچیده و پیشرفته را برای شما ایجاد می کند.

صفحه سوم:

این صفحه دارای یک کنترل **StackPanel** با خاصیت **Orientaion** با مقدار **Horizontal** می باشد. درون این کنترل یک عکس و یک متن درون عنصر TextBlock قرار گرفته است. مقدار خاصیت **Content** این عنصر همان GroupBox سمت راست مثال قبل می باشد که یک محتوای پیچیده و پیشرفته را برای شما ایجاد می کند.

صفحه چهارم:

این صفح در **header** خود کمی متفاوت تر از **header** ها صفحات قبلی می باشد. اولین تفاوت اینکه در این صفحه و در قسمت هدر ان چهار عنصر مجزا قرار گرفته است. دو عنصر برای نگه داری عکس ها و دو عنصر برای نگهداری متن های مروبوطه. دو مین تفاوت ان این است که کنترل **StackPanel** موجود در هدر این صفحه به صورت نرمال می باشد. یا در واقع خاصیت Orientation آن تنظیم نشده است که باعث می شود به صورت پیش فرض Vertical در

نظر گرفته شود. خاصیت **Content** این صفحه نیز، با **GroupBox** های مثال قبلی مقدار دهی شده است.
مثال فوق، نشان دهنده این است که شما می توانید با بهری گیری از عناصر کانتینر کنترل های کاملا سفارشی خود را ایجاد نمایید.

کنترل : Expander

این کنترل بدون شک یکی از کنترل های پر کاربردی است که برنامه نویسان همیشه به دنبال آن جهت استفاده در برنامه های خود بوده اند. از زمان ظهور دات نت، این کنترل به صورت پیش فرض در ویژوال استودیو وجود نداشت و برنامه نویسان مجبور بودند یا خود اقدام به ایجاد چنین کنترلی نمایند و یا به سمت کنترل های نوشته شده توسط اشخاص و شرکت های ثالث روی آورند .
ماکروسافت در ویژوال استودیو 2008 و در تکنولوژی WPF ، این کنترل را برای استفاده کاربران قرار داد. این کنترل دارای دو بخش می باشد، بخش هدر و یک بخش محتوا. بخش هدر شامل یک دکمه که غالبا به صورت فلش است، می باشد. کاربر با کلیک کردن بر روی این دکمه می تواند بخش Content را مخفی کند و یا آن را از حالت مخفی خارج نماید .

بخش Content این کنترل نیز مانند همه عناصر WPF می تواند شامل هر عنصر دیگری باشد.
نحوه استفاده از این کنترل بسیار ساده می باشد. با استفاده از قطعه کد زیر می توانید یک کنترل Expander ایجاد نمایید:
کد:

```
<Expander Header="sample text header">sample text content</Expander>
```

قطعه کد فوق ساده ترین حالت تعریف یک کنترل Expander می باشد که شامل یک متن ساده به عنوان هدر و یک متن ساده دیگر به عنوان محتوا می باشد. چنانچه بخواهید از هدر و محتواهای پیچیده تر و پیشرفته تر و سفارشی خود در کنترل Expander استفاده کنید، بایستی کنترل Expander را به صورت زیر تعریف کنید:
کد:

```
<Expander >
<Expander.Header>
.
.
.
</Expander.Header>
<Expander.Content>
.
```

```
.
.
</Expander.Content>
</Expander>
```

در این حالت میتوانید با قرار دادن یک کنترل کانتیر در بخش مورد نظر، اقدام به ایجاد هدر و محتواهای مورد نظر خود کنید.
جهت در ک بیشتر به مثال زیر توجه کنید:

کد:

```
<Window x:Class="contentControls.ExpnaderControl"
xmlns="http://schemas.microsoft.com/winfx/2...l/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
Title="ExpnaderControl" Height="400" Width="350">
<Grid>
<Grid.RowDefinitions>
<RowDefinition ></RowDefinition>
<RowDefinition ></RowDefinition>
</Grid.RowDefinitions>
<Grid.ColumnDefinitions>
<ColumnDefinition ></ColumnDefinition>
<ColumnDefinition></ColumnDefinition>
</Grid.ColumnDefinitions>

<!-- Define Expanders-->
<Expander Header="simple expander" BorderBrush="Green"
BorderThickness="2" HorizontalContentAlignment="Center"
VerticalContentAlignment="Center">sample content</Expander>

<Expander  Margin="5" Grid.Row="0" Grid.Column="1" IsExpanded="True"
HorizontalContentAlignment="Left" BorderBrush="Brown" BorderThickness="3">
<Expander.Header>
<StackPanel Orientation="Horizontal">
<TextBlock Padding="3">Country GroupBox</TextBlock>
<Image Source=".\Images\banner.png" MaxWidth="20"
MaxHeight="20"></Image>
</StackPanel>
```

```
                                                    </Expander.Header>
                                                    <Expander.Content>
                <GroupBox  Margin="10" Padding="10" VerticalAlignment="Top">

                                                        <StackPanel>
                                    <RadioButton  Margin="3"  Name="rbtn6"
                                                        IsChecked="True">
                                     <StackPanel Orientation="Horizontal">
                                         <Image Source=".\Images\Iran.ico"
                                            MaxWidth="20" MaxHeight="20"/>
                                             <TextBlock>(Iran)</TextBlock>
                                                    </StackPanel>
                                                   </RadioButton>
                                  <RadioButton   Margin="3" Name="rbtn7">
                                    <StackPanel Orientation="Horizontal">
                                     <Image Source=".\Images\Albania.ico"
                                           MaxWidth="20" MaxHeight="20"/>
                                          <TextBlock>(Albani)</TextBlock>
                                                    </StackPanel>
                                                   </RadioButton>
                                    <RadioButton Margin="3" Name="rbtn8">
                                    <StackPanel Orientation="Horizontal">
                                   <Image Source=".\Images\Hong-Kong.ico"
                                           MaxWidth="20" MaxHeight="20"/>
                                       <TextBlock>(Hong-Kong)</TextBlock>
                                                    </StackPanel>
                                                   </RadioButton>
                                    <RadioButton Margin="3" Name="rbtn9">
                                    <StackPanel Orientation="Horizontal">
                                       <Image Source=".\Images\Italy.ico"
                                           MaxWidth="20" MaxHeight="20"/>
                                           <TextBlock>(Italy)</TextBlock>
                                                    </StackPanel>
                                                   </RadioButton>
                                                     </StackPanel>
                                                       </GroupBox>
                                                  </Expander.Content>
                                                          </Expander>
```

```
        <Expander IsExpanded="True" Grid.Row="1" Header="Gradient Expander"
                                    BorderBrush="Red" BorderThickness="2">
                                                  <Expander.Background>
                                                  <RadialGradientBrush>
                                      <RadialGradientBrush.GradientStops>
                                  <GradientStop Color="Green" Offset="0"/>
                                  <GradientStop Color="Wheat" Offset="1"/>
                                      </RadialGradientBrush.GradientStops>
                                                  </RadialGradientBrush>

                                                  </Expander.Background>
                                                      <Expander.Content>
            <GroupBox BorderBrush="Brown" BorderThickness="3" Margin="10"
                                    Padding="10" VerticalAlignment="Top">

                                                            <StackPanel>
                        <RadioButton Margin="3" Name="rbtn1" IsChecked="True"
                                                                        >
                             <Image Source=".\Images\star.png" MaxHeight="16"
                                                  MaxWidth="16"></Image>
                                                          </RadioButton>
                                    <RadioButton Margin="3" Name="rbtn2">
                                    <StackPanel Orientation="Horizontal">
                                        <Image Source=".\Images\star.png"
                                    MaxHeight="16" MaxWidth="16"></Image>
                                        <Image Source=".\Images\star.png"
                                    MaxHeight="16" MaxWidth="16"></Image>
                                                           </StackPanel>
                                                          </RadioButton>
                                    <RadioButton Margin="3" Name="rbtn3">
                                    <StackPanel Orientation="Horizontal">
                                        <Image Source=".\Images\star.png"
                                    MaxHeight="16" MaxWidth="16"></Image>
                                        <Image Source=".\Images\star.png"
                                    MaxHeight="16" MaxWidth="16"></Image>
                                        <Image Source=".\Images\star.png"
                                    MaxHeight="16" MaxWidth="16"></Image>
                                                           </StackPanel>
```

```
                                                    </RadioButton>
                                                <RadioButton Margin="3" Name="rbtn4">
                                                <StackPanel Orientation="Horizontal">
                                                    <Image Source=".\Images\star.png"
                                                MaxHeight="16" MaxWidth="16"></Image>
                                                    <Image Source=".\Images\star.png"
                                                MaxHeight="16" MaxWidth="16"></Image>
                                                    <Image Source=".\Images\star.png"
                                                MaxHeight="16" MaxWidth="16"></Image>
                                                    <Image Source=".\Images\star.png"
                                                MaxHeight="16" MaxWidth="16"></Image>
                                                    <Image Source=".\Images\star.png"
                                                MaxHeight="16" MaxWidth="16"></Image>
                                                     </StackPanel>
                                                    </RadioButton>
                                                <RadioButton Margin="3" Name="rbtn5">
                                                <StackPanel Orientation="Horizontal">
                                                    <Image Source=".\Images\star.png"
                                                MaxHeight="16" MaxWidth="16"></Image>
                                                    <Image Source=".\Images\star.png"
                                                MaxHeight="16" MaxWidth="16"></Image>
                                                    <Image Source=".\Images\star.png"
                                                MaxHeight="16" MaxWidth="16"></Image>
                                                    <Image Source=".\Images\star.png"
                                                MaxHeight="16" MaxWidth="16"></Image>
                                                    <Image Source=".\Images\star.png"
                                                MaxHeight="16" MaxWidth="16"></Image>
                                                     </StackPanel>
                                                    </RadioButton>
                                                     </StackPanel>
                                                       </GroupBox>
                                                  </Expander.Content>
                                                       </Expander>
                                <Expander BorderBrush="CadetBlue" BorderThickness="3"
    IsExpanded="True" Grid.Row="1" Grid.Column="1" Header="Larg Content text">
                                                    <ScrollViewer>
                                               <TextBlock TextWrapping="Wrap">

                                                    From WikiPedia
                                    he Windows Presentation Foundation (or WPF),
```

formerly code-named Avalon, is the graphical subsystem feature of the .NET Framework 3.0 (formerly called WinFX)[1] and is directly related to XAML.[2] It is pre-installed in Windows Vista,[3] the latest version of the Microsoft Windows operating system. WPF is also available for installation on Windows XP SP2 and Windows Server 2003. It provides a consistent programming model for building applications and provides a clear separation between the UI and the business logic. A WPF application can be deployed on the desktop or hosted in a web browser. It also enables rich control, design, and development of the visual aspects of Windows programs. It aims to unify a host of application services: user interface, 2D and 3D drawing, fixed and adaptive documents, advanced typography, vector graphics, raster graphics, animation, data binding, audio and video. Although WinForms will continue to be widely used, WPF is now the preferred choice for developing line of business applications, especially since the release of the .NET Framework 3.5, Visual Studio 2008, and Expression Blend[4].

Microsoft Silverlight is a web-based subset of WPF. During development it was named WPF/E, which stood for "Windows Presentation Foundation/Everywhere". The Silverlight subset enables Flash-like web and mobile applications with the same code as Windows .NET applications. 3D features are not supported, but XPS and vector-based drawing are included.

The architecture of Windows Presentation Foundation spans across both managed code and native code components; however, the public API exposed is only available via managed code. While the majority of WPF is in managed code, the composition engine which renders the WPF applications is a native component. It is named as Media Integration Layer (MIL) and resides as milcore.dll. It interfaces directly with DirectX and provides basic support for 2D and 3D surfaces , timer-controlled manipulation of contents of a surface with a view to exposing animation constructs at a higher

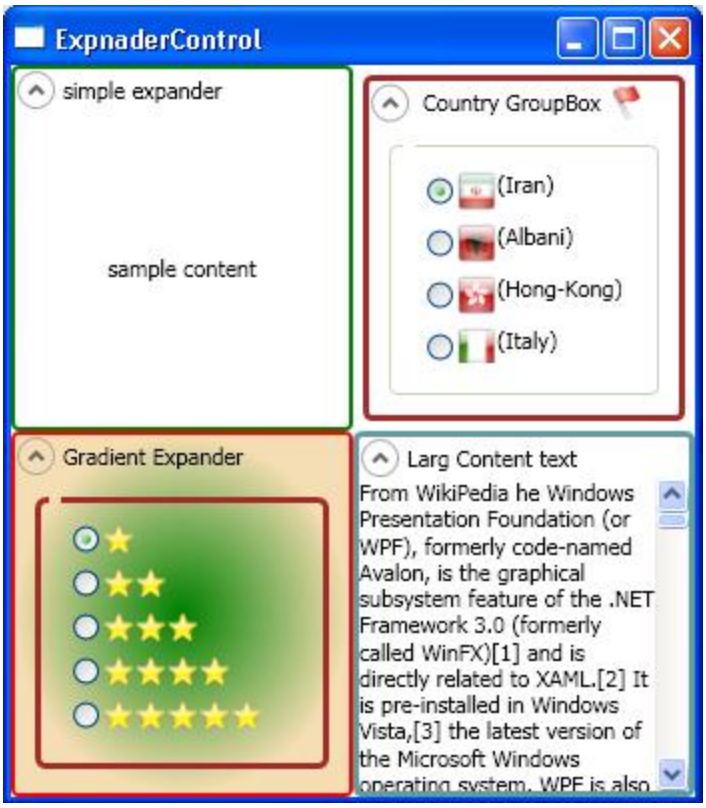
level, and compositing the individual elements of a WPF application into a final 3D "scene" that represents the UI of the application and rendering it to the screen.[5][6] The media codecs are also implemented in unmanaged code, and are shipped as windowscodecs.dll.[5] In the managed world, PresentationCore (presentationcore.dll) provides a managed wrapper for MIL as well as implements the core services for WPF,[5] including a property system that is aware of the dependencies between the setters and consumers of the property, a message dispatching system by means of a Dispatcher object to implement a specialized event system and services which can implement a layout system such as measurement for UI elements.[6] PresentationFramework (presentationframework.dll) implements the end-user presentational features, including layouts, time-dependent, story-board based animations, and data binding.[6] WPF exposes a property system for objects which inherit from DependencyObject, that is aware of the dependencies between the consumers of the property, and can trigger actions based on changes in properties. Properties can be either hard coded values or expressions, which are specific expressions that evaluate to a result. In the initial release, however, the set of expressions supported is closed.[6] The value of the properties can be inherited from parent objects as well. WPF properties support change notifications, which invoke bound behaviors whenever some property of some element is changed. Custom behaviors can be used to propagate a property change notification across a set of WPF objects. This is used by the layout system to trigger a recalculation of the layout on property-changes, thus exposing a declarative programming style for WPF, whereby almost everything, from setting colors and positions to animating elements can be achieved by setting properties.[6] This allows WPF applications to be written in XAML, which is a declarative mark-up language, by binding the keywords and attributes directly to WPF classes and properties. The UI elements of an WPF application is maintained as a class of Visual objects. Visual objects provide a managed interface to a composition tree which

is maintained by Media Integration Layer (MIL). Each element of WPF creates and adds one or more composition nodes to the tree. The composition nodes contain rendering instructions, such as clipping and transformation instructions, along with other visual attributes. Thus the entire application is represented as a collection of composition nodes, which are stored in a buffer in the system memory. Periodically, MIL walks the tree and executes the rendering instructions in each node, thus compositing each element on to a DirectX surface, which is then rendered on screen. MIL uses the painter's algorithm, where all the components are rendered from back of the screen to the front, which allows complex effects like transparencies to be easily achieved. This rendering process is hardware accelerated using the GPU.[6] The composition tree is cached by MIL, creating a retained mode graphics, so that any changes to the composition tree needs only to be incrementally communicated to MIL. This also frees the applications of managing repainting the screen, MIL can do that itself as it has all the information necessary. Animations can be implemented as time-triggered changes to the composition tree. On the user visible side, animations are specified declaratively, by setting some animation effect to some element via a property and specifying the duration. The code-behind updates the specific nodes of the tree, via Visual objects, to represent both the intermediate states at specified time intervals as well as the final state of the element. MIL will render the changes to the element automatically.

All WPF applications start with two threads: one for managing the UI and another background thread for handling rendering and repainting.[7] Rendering and repainting is managed by WPF itself, without any developer intervention. The UI thread houses the Dispatcher (via an instance of DispatcherObject), which maintains a queue of UI operations that need to be performed

```
                    (as a tree of Visual objects), sorted by priority. UI
      events, including changing a property that affects the layout, and user
                 interaction events raised are queued up in the dispatcher,
                       which invokes the handlers for the events. Microsoft
  recommends that the event handlers only update the properties to reflect new
                     content for application responsiveness; the new content
                    be generated or retrieved in a background thread.[7] The
          render thread picks up a copy of the visual tree and walks the tree
               calculating which components will be visible and renders them
                     to Direct3D surfaces. The render thread also caches the
  visual tree, so only changes to the tree need to be communicated, which will
                    result in updating only the changed pixels. WPF supports
                       an extensible layout model. Layout is divided into two
 phases: Measure and Arrange. The Measure phase recursively calls all elements
                  and determine the size they will take. In the Arrange phase,
                       the child elements are recursively arranged by their
     parents, invoking the layout algorithm of the layout module in use.[6][8]
                                                                  </TextBlock>
                                                              </ScrollViewer>
                                                                  </Expander>
                                                                      </Grid>
                                                                  </Window>
```

کد فوق، چهار کنترل Expander ایجاد می کند. هر کنترل Expander دارای هدر و محتوای خاص خود می باشد.
نتیجه اجرای کد فوق را در شکل زیر مشاهده می کنید:



همانطور که مشاهده می کنید، هر یک از کنترل های Expander در شکل فوق به نحو دلخواهی ساز ماندهی شده اند و دارای مقادیر هدر و محتوای خود می باشند.

خاصیت: IsExpanded

این خاصیت حالت کنترل Expander را نشان می دهد. به طور کلی کنترل Expander دارای دو حالت می باشد:
حالت اول: زمانی که مقدار خاصیت IsExpanded را برابر با False قرار می دهید. در این حالت، بخش محتوای کنترل Expander به صورت مخفی در خواهد آمد.
حالت دوم: این حالت عکس حالت اول می باشد. در این حالت بخش محتوای کنترل Expander در حالت نمایش می باشد.
نکته: مقدار پیش فرض خاصیت IsExpanded برابر با False می باشد.
**
نحوه استفاده از این خاصیت در کد XAML به صورت زیر می باشد:

کد:

```
<Expander . . . IsExpanded="True"> . . . </Expander>
```

مشخص کردن جهت Expand در کنترل: Expander

کنترل Expander دارای خاصیتی به نام ExpanderDirection می باشد که جهت باز و بسته شدن

کنترل Expander را مشخص می کند. این خاصیت دارای چهار مقدارDown، Up ، Left و Right می باشد که مقدار پیش فرض آن برابرا با Down می باشد.
نحوه به کار بردن این خاصیت در کد XAML به صورت زیر می باشد:
کد:

```
<Expander ExpandDirection="Up" . . .>
.
.
.
.
</Expander>
```

شکل زیر، نتیجه استفاده از هر یک از چهار مقدار خاصیت ExpanderDirection را نشان می دهد:

خاصیت FlowDirection

این خاصیت دارای دو مقدار LeftToRight و RightToLeft می باشد که مقدار LeftToRight حالت پیش فرض آن می باشد. این خاصیت نحوه چیدمان کنترل Epander را نشان می دهد. مقدار RightToLeft برای زبان هایی همچون فارسی، عربی و ... که چیدمان الفبای آن ها از سمت راست می باشد، مناسب می باشد.

با سلام
بخش اول مطالب آموزشی (مقدمه ای بر (WPF را در قالب فایل PDF آماده شد..

بخش های بعدی نیز به تدریج آماده خواهند شد...
موفق باشید...
فایل های ضمیمه

IntroductionToWPF.rar (434.0 کیلوبایت, 1418 دیدار)

خواص: CLR

هر برنامه نویس دات نتی، با مفهومی به نام خواص یا همان Property ها آشنا می باشد .
مفاهیمی مانند:
کد:

```
private int age;
public int Age
{
get
{
return age;
}
set
{
age = value;
}
}
```

به این گونه خواص، اصطلاحا خواص CLR می گویند . (CLR Property) .در این قسمت، قصد بیان این موضوع را که این خواص چه هستند و چگونه پیاده سازی می شوند و چرا از آن ها استفاده می کنیم، را ندارم. برای مطالعه خواص CLR به لینک زیر مراجعه کنید.
http://msdn.microsoft.com/en-us/library/x9fsa0sw.aspx
هدف از بحث فوق، یک یاداوری در مورد اینگونه خواص، مقایسه آن ها با خواص جدید در تکنولوژی WPF ، جهت تشریح خواص WPF که به Dependency Propertyمعروف هستند، می باشد.

خواص وابسته (Dependency properties) :

تکنولوژی WPF سرویس های جدیدی را برای گسترش خواص CLR در اختیار شما، قرار می دهد. این سرویس ها باعث می شود که بتوانید از این خواص در موارد زیادی از جمله متحرک سازی ها، استایل ها، قالب ها و عملیات بایند کردن و ... استفاده کنید. به چنین خواصی اصطلاحا Dependency Property می گویند.

خواص وابسته و خواص: CLR

در ساده ترین حالت، می توانید خواص وابسته را به مانند خواص CLR در نظر بگیرید. این وضوع را از ابتدای مباحث

آموزشی تا بدین جا نیز، با آن برخورد داشته اید. به عنوان مثال، در هنگام استفاده از خواص WPF احتمالا به این نکته توجه کرده اید که روش استفاده از این خواص نیز مانند خواص CLR می باشند.
به عنوان مثال دو قطعه کد زیر مشابه هم می باشند:
کد:

```
Textbox1.Text = "save";
Textbox1.Text = "save";
```

با این تفاوت که در کد اول، خاصیت Text یک خاصیت CLR می باشد ولی در کد دوم خاصیت Text یک Dependency Properrtyمی باشد. اما واقعیت این است که این خواص و قدرت آن ها بسیار فراتر از این موضوع می باشد.

این گونه خواص می توانند مقادیر خود را بر مبنای ورودی های دیگر محاسبه نمایند. این ورودی ها هر چیزی می توانند باشند. به عنوان مثال در بحث Trigger ها، خواهید دید که چگونه بر مبنای تغیر یک خاصیت، خواص دیگر می توانند به صورت اتوماتیک مقادیر خود را تنظیم کرده و آن رفتاری را از خود بروز دهند که شما انتظار دارید.
علاوه بر این، اینگونه خواص، سیستم جدیدی را در اعتبار سنجی مقادیر به شما ارائه می دهند.

خواص وابسته :

شما می توانید، با پیاده سازی خواص CLR به عنوان Dependency Property ،در سیستمWPF Properties ، قابلیت های بسیاری از جمله ارث بری، انیمیشن، پشتیبانی از استایل ها، مقید سازی داده ها و ... به کار برید .

Dependency propertyها ، خواصی هستند که در سیستم WPF Properties توسط متد Register ، ثبت شده باشند. هر Dependency property توسط کلاسی پیاده سازی می شود که در آن کلاس فیلدی با امضای public staticاز نوع Dependency Property وجود دارد که به آن شناسه آن خاصیت می گویند.
در نام گذاری شناسه ها اینگونه عمل میشود که پس از نام خاصیت، کلمه Property در ادامه آن ذکر می گردد. به عنوان مثال شناسه خاصیتBackground Property ، backgroundمی باشد.
این شناسه، اطلاعاتی راجع به خاصیت خود به هنگام ثبت شدن را درون خود نگه داری می کند .
دقت داشته باشید که Dependency property ها تنها توسط نوع های Dependency object قابل استفاده می باشند .
اما همانطور که قبلا اشاره کردم، کلاس Dependency object در بالای هرم ساختاری کلاس ها در معماری wpf قرار دارد،. به همین دلیل بسیاری از کلاس های اصلی این سیستم می توانند از Dependency property ها پشتیبانی کنند.

چرا و چه زمانی از Dependency property ها استفاده کنیم؟

زمانی که درون کلاسی که از کلاس Dependency object ارث بری می کند، حاصیتی را تعریف می کنید، می توانید این خاصیت را به صورت Dependency پیاده سازی نمایید. اما سوال این است که آیا همیشه و در همه حال بایستی از این نوع خواص استفاده کرد؟ یقینا جواب خیر می باشد. در بسیاری از مواقع پیاده سازی یک خاصیت CLR با یک فیلد خصوصی کافی است.
اما با این حال، ممکن است در بعضی مواقع نیاز به تعریف و پیاده سازی خواص یه شیوه Dependency property را داشته باشید. در ادامه چند دلیل برای تعریف این خواص آورده می شود:

زمانی که می خواهید، خاصیت شما قابل مقدار دهی در استایل ها (Styles) و قالب ها (Templates) باشد.

زمانی که می خواهید، خاصیت شما قابل استفاده در مقید سازی داده ها (Data Bindings) باشد.

زمانی که می خواهید، خاصیت شما به صورت اتوماتیک بتواند مقدار خودش را از مقدار
والد خود ارث بری کند.

زمانی که می خواهید از خاصیت خود در متحرک سازی استفاده نمایید. در واقع نیاز دارید
که مقدار خاصیت شما در یک پروسیه انیمیشنی قابل کنترل باشد.

اگر چه در سناریو های بالا، شما نیاز به خواص Dependency دارید، ولی در بسیاری از موارد می توانید به جای نوشتن خاصیت های جدید، با دوباره باز نویسی کردن متا داده های خواص Dependency موجود، به هدف خود دست یابید.

تعریف و پیاده سازی خواص: Dependency

تعریف و پیاده سازی خواص Dependency دارای چهار مرحله زیر می باشد.دقت داشته باشید که این چهار مرحله، مراحل ترتیبی نیستند. علاوه بر این، این مراحل قابل اقدام در یکدیگز نیز می باشند.

تعریف متا داده برای خاصیت. این مرحله اختیاری می باشد. می تواند در تعریف خاصیت مقدار null داشته باشد.

تعریف یک فیلد با امضای readonly public static به عنوان شناسه خاصیت . این فیلد بایستی از نوع Dependency propertyباشد.

ثبت کردن خاصیت، در سیستم wpf توسط متد.Register

ایجاد یک پوشاننده (wrapper) برای خاصیت، که همنام با نام خاصیت می باشد.

مثال:
فرض کنید کلاسی به نام MyControl ایجاد کرده اید. این کلاس از کلاس UserControl ارث بری می کند. می خواهیم خاصیتی به نام CornerRadius برای این کنترل تعریف کنیم.

کد:

```
Public class MyControl :UserControl
{
}
```

ابندا شناسه خاصیت را تعریف می کنیم.

کد:

```
public static readonly DependencyProperty
CornerRadiusProperty;
```

همانطور که مشاهده می کنید، شناسه به صورت CornerRadiusProperty نام گذاری شده است که CornerRadius نام خاصیت می باشد.

در مرحله بعدی می بایستی خاصیت خود را ثبت کنیم. برای این منظوراز متد Register استفاده می کنیم.

کد:

```
DependencyProperty.Register("CornerRadius", typeof
(CornerRadius), typeof(MyControl), new
FrameworkPropertyMetadata (0));
```

همانطور که مشاده می کنید از متد Register از کلاس Dependency Property که یک کلاس مهر شده (درای امضای (seald می باشد، استفاده شده است.

آرگومان اول این متد، نام خاصیت را مشخص می کند. در اینجا نام خاصیت، CornerRadiusمی باشد.

آرگومان دوم نوع خاصیت را مشخص می کند. نوع خاصیت همانطور که مشاهده می کنید، از نوع CornerRadius در نظر گرفته شده است.

نکته:
CornerRadiusیک ساختار (Structure) در فضای نام System.Windows می باشد که اینترفیس ژنریک IEquatable را پیاده سازی میکند.

آرگومان سوم، مالک (owner)خاصیت را مشخص می کند. که در این مثال، کلاس MyControlمی باشد.

آرگومان چهارم، متا داده ای است که برای این خاصیت در نظر گرفته شده است. مقدار صفر درون آن، معرف مقدار پیش فرض خاصیت، می باشد. موارد دیگری را نیز می توانید در تعریف متا داده های یک خاصیت، لحاظ کنید که در ادامه توضیح داده می شوند.

علاوه بر این، همانطور که پیش تر گفته شد، تعریف متا داده برای یک حاصیت اختیاری می باشد ولی تنظیم آن اغلب موارد دارای مزایایی است که در ادامه خواهید دید.
قبل از بررسی آرگومان FrameworkPropertyMetadata به صورت عمیق تر، به آرگومان دیگری که می توان به متد Registerپاس داد، می پردازیم .

ValidateValueCallback:

آرگومان دیگری که به متد Register می توان پاس داد، نمونه ای از Delegat ای به نام ValidateValueCallback می باشد. این Delegate دارای تعریفی مانند زیر می باشد:

کد:

```
public delegate bool ValidateValueCallback(object value);
```

از این Delegate می توان جهت اعتبار سنجی مقدار خاصیت، که توسط کاربر نهایی (end user)مقدار دهی شده است، استفاده کرد. همانطور که در امضای این Delegate مشخص است، نوع برگشتی متد هایی که این Delegate می تواند، مرجعی برای آن ها باشد، از نوع bool می باشد. در واقع چنانچه مقدار خاصیت، دارای اعتبار باشد، مقدار بازگشتی true و در غیر این صورت مقدار بازگشتی false می باشد پس می توان تعریف متد Register را به شکل زیر کامل تر کرد:

کد:

```
DependencyProperty.Register("CornerRadius", typeof
(CornerRadius), typeof(MyControl), new
FrameworkPropertyMetadata(0), new ValidateValueCallback
(CornerRadiusValidateCallBack));
```

که متد CornerRadiusValidateCallBack به صورت زیر می باشد:

کد:

```
static bool CornerRadiusValidateCallBack(object value)
{
// write your validastion code here
//
//
// if (value is Valid)
//    return true;
//  return false;
}
```

مرحله آخر پوشانیدن خاصیت می باشد.

Property Wrapper :

در این مر حله بایستی خاصیت شما به شکل یک خاصیت CLR پوشانیده شود تا قابل استفاده باشد. به این خاصیت اصطلاحا property wrapper می گویند.

نحوه تعریف یک wrapper برای خواص Dependency به صورت زیر می باشد:

کد:

```
public CornerRadius CornerRadius
{
get
{
return (CornerRadius)GetValue(CornerRadiusProperty);
}
set
{
SetValue(CornerRadiusProperty, value);
}
}
```

متد های GetValue و SetValue توسط کلاس Dependency Object مورد استفاده قرار می گیرند تا خاصیت شما را مقدار دهی کنند. به همین دلیل، نبایستی در Property Wrapper ها، منطقی را پیاده سازی کنید چون ممکن است که نا دیده گرفته شوند.