آموزش WPF مقدماتی تا پیشرفته

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 : WPFو 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 در WPF

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 در XAML

قسمت هفتم :خواص و رویداد ها در XAML

قسمت هشتم (آخر :)رویداد ها در XAML

بخش سوم : چیدمان و طراحی کنترل ها

قسمت اول : مقدمه

قسمت دوم :چیدمان عناصر در WPF

قسمت سوم : کنترل StackPanel

قسمت چهارم :ادامه کنترل StackPanel

قسمت پنجم: کنترل Canvas

قسمت ششم :کنترل DockPanel

قسمت هفتم :کنترل WrapPanel

قسمت هشتم :کنترل UniformGrid

قسمت نهم :کنترل Grid

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 Indigo نیز 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 پیچیده 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.

بخش اول : مقدمه ای بر WPF قسمت دوم

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 WPF .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 Applicationها 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 WPF برایتان 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 WPFخواهم 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.

بخش اول : مقدمه ای بر WPF قسمت سوم

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 Page می 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 WPF خواهید آموخت. توسط این مدل برنامه نویسی، می توانید اسمبلی های ایجاد شده را در مرورگر وب نظیر Internet Explorer بدون 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

1لبته 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.

بخش اول : مقدمه ای بر WPF (قسمت چهارم)

User32، GDI، GDI+ و DirectX

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. WPF 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 مفاهیم Texture، Gradient و ... تطبیق پیدا می کند، درارای سرعت بالاتری نسبت GDIو 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.

WPF به عنوان یک API سطح بالا

همناطور که پیشتر توضیح داده شد، WPF قادر 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 چدیمان 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، WPF ممکن 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 کامپوننت 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 کامپوننت 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 کامپوننت 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 کامپوننت 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 Based می 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)

بخش اول : مقدمه ای بر WPF (قسمت پنجم)

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 DPI 120 استفاده می کنند. در صورتی که مونیتور های CRT معمولا از تراکم DPI 96 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 96/1 (1 تقسیم بر 96) هر اینچ می باشد. در واقع می توان گفت هر DIU در صفحه نمایشی با تراکم پیکسل استاندارد یعنی DPI 96 ، 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 DPI 96 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؟)

بخش اول : مقدمه ای بر WPF (قسمت آخر)

معماری : WPF

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.dll و 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.

Direct3D در 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 WPFApplication دارا 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. همچنین Panel، کلاس پایه برای تمامی کنترل های Container از جمله Grid (که مهمترین آن ها و پر کاربرد ترین آن ها می باشد)، StackPanel، Canvas و ... 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 Direct3D نیاز 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 Cross نیز 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.

Direct3D : نیز یک API سطح پایین است که شامل تمامی گرافیک های رندر شده در WPF می باشد.

ساختار سلسله مراتبی آبجکت ها در WPF

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# 2008 نوشته 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.

بخش دوم : زبان XAML (قسمت اول)

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، XAML مخفف عبارت EXtensible Application Markup Language می باشد. زبان XAML که یک زبان توصیفی می باشد، بر پایه قواعد XML می 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 Intellisence بسیار 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 WPF چه 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.

بخش دوم : زبان XAML (قسمت دوم)

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.

XAML در 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.

XAML در WF :

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

XAMl در SilverLight : نسخه ای دیگر از WPF وحود 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:

XAML کاربرد 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.

بخش دوم : زبان XAML (قسمت سوم)

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 BAML درواقع 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 Load شدن، 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 که در این مورد در بخش های آتی بیشتر خواهید دانست.

خش دوم : زبان XAML (قسمت چهارم)

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 ههآ 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 Button را 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.

بخش دوم : زبان XAML (قسمت پنجم)

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 WPF به 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 Container است. 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.

بخش دوم : زبان XAML (قسمت ششم)

فضای نام ها در XAML

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 مربوطه 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 MathClasses و در اسمبلیMyCustomClasses قرار 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 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 WPF وجد 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 XAML را 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 localizable کردن، 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، Width و 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 پیشترفته 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 BackGround و 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، TextBox در سطر و ستون دوم کنترل گرید، قرار خواهد گرفت.

کد:

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

...

...

<TextBox/>
```

بخش دوم : زبان XAML (قسمت آخر)

رویداد ها در XAML:

خوشبختانه، XAML تمهیدات 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 های XAML، هوشمند بودن آن می باشد. به عنوان مثال در هنگام تعریف رویداد، XAML به 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 Enter بر 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 فبل 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.

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

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 Application می 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 Application ایجاد می کنید، یک آبجکت از کلاس Window ساخته می شود که به صورت پیش فرض نام آن Window1 می 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 Application را 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 Fox قابل 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 Application ارث 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 Application در 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 1.0.1) 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 داست، 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

ب) Canvas

ج) Dockpanel

د) WrapPanel

ه) UniformGrid

ی) Grid

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 :۱

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.

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

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 StackPanel تنظیم 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.

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

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:

مقادریر 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 مروبط 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،Bottom و 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.

http://hoghoogh.persiangig.com/document/StackPanel.rar

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

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 ی Left، Top، Right و 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، TextBox ها 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، TextBox بالایی، 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)

http://hoghoogh.persiangig.com/document/Canvas.rar

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

کنترل DockPanel:

در دات نت x.1 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، Dock می 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 DockPanel تنظیم 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 DockPanel به 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، button5 می 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.

http://hoghoogh.persiangig.com/document/DockPanel.rar

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

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.

http://hoghoogh.persiangig.com/document/WrapPanel.rar