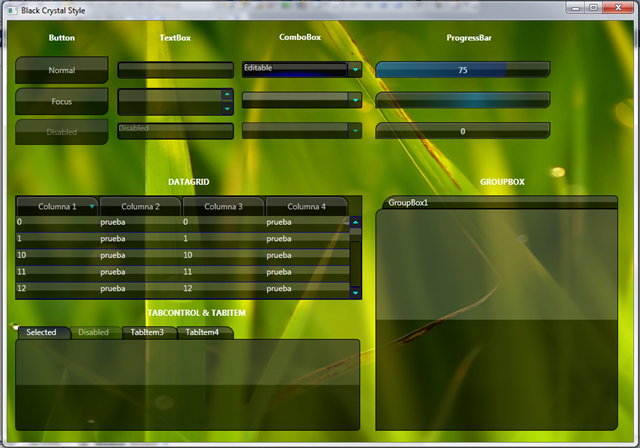
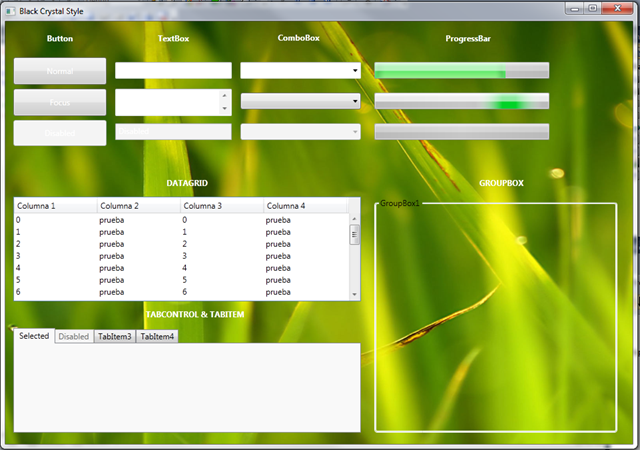
[](http://www.google.es/url?sa=i&rct=j&q=&esrc=s&source=images&cd=&cad=rja&uact=8&ved=0CAcQjRxqFQoTCI_BmNKhxcgCFQnaGgodzd4Cyg&url=http://geeks.ms/wikis/wpf/wpf-personalizar-nuestra-aplicaci-243-n-con-styles-y-control-templates-i.aspx&psig=AFQjCNEI1Y27oGiEJP2_Ou1e6_QYmnqzCw&ust=1445025493203278)

A continuación puedes ver el origen desde el que vamos a partir:



Como podéis ver el cambio es muy grande, es mucho mas atractivo a la vista, con lo que los usuarios se sentirán más cómodos y la aplicación ganará una apariencia exclusiva y única. Así que, manos a la obra, pero primero, vamos a repasar algunos conceptos básicos de Styles y ControlTemplates en WPF.

**Un poco de teoría**

WPF nos brinda varias formas de personalizar nuestros controles:

* Contenido Enriquecido
* Styles
* Data Templates
* Control Templates

Abajo veremos los Styles y los Control Templates.

**Styles**

Un objeto Style es una colección de valores que representan propiedades para el control indicado. Podemos asignarle automáticamente a todos los controles de un mismo tipo, por ejemplo un Button, ciertas propiedades como el color de fondo, tipo de letra, etc… de esta forma todos nuestros controles de este tipo obtendrán estas características:

<Style TargetType="{x:Type Button}">

<Setter Property="Background" Value="Red"></Setter>

<Setter Property="Foreground" Value="White"></Setter>

</Style>

También podemos establecer un nombre al estilo y aplicarlo manualmente a los controles que nos interese usando la propiedad Style de cada uno:

<Style x:Key="EstiloPropio" TargetType="{x:Type Button}">

<Setter Property="Background" Value="Red"></Setter>

<Setter Property="Foreground" Value="White"></Setter>

</Style>

<Button Style="{StaticResource EstiloPropio}"></Button>

\*\*\*Con Style se refiere al estilo de Fondo(Background) o Foregroun,…

**Control Templates**

Muchos controles en WPF utilizan plantillas para definir su estructura y apariencia. De esta forma su funcionalidad permanece intacta aunque cambiemos su apariencia, y nos da un grado de personalización sobre el aspecto del control sin precedentes. Esto nos permite elegir un control por su funcionalidad para llevar a cabo una tarea, sin tener en cuenta que su apariencia no se ajuste a lo que deseamos, pues esta puede ser redefinida totalmente. Este es un ejemplo sencillo de un ControlTemplate para un botón:

<ControlTemplate TargetType="{x:Type Button}">

<Border Name="fondoboton" BorderBrush="red" BorderThickness="5"

Background="Yellow">

<ContentPresenter Name="contenido"

Content="{Binding Path=Content,

RelativeSource={RelativeSource TemplatedParent}}">

</ContentPresenter>

</Border>

</ControlTemplate>

Y el resultado de esta plantilla si ponemos un botón en nuestro proyecto sería este:

[boton](http://lh4.ggpht.com/_7YCdMP8inuc/TAJ6b1MulrI/AAAAAAAAADg/MmfavknlVXY/s1600-h/boton%5b3%5d.png)

Como puedes ver, el control a perdido todo su aspecto, que ha sido reemplazado por el que hemos indicado, pero conservando todo su comportamiento y funcionalidad de botón.

**Definir el estilo de la aplicación**

Un paso muy importante a la hora de definir el estilo de la aplicación es que esta mantenga una coherencia visual a través de las diferentes pantallas de la misma, para que el usuario tenga una sensación de familiaridad con la aplicación aunque se enfrente a pantallas nuevas.

Otro aspecto muy importante es la sencillez de implantación, en una aplicación con cientos de controles y decenas de pantallas, no podemos ir control por control y pantalla por pantalla ajustando parámetros visuales, debe realizarse de forma automática, para esto WPF nos facilita los ResourceDictionary, archivos Xaml donde podemos incluir todo nuestro código de estilado e incluirlos de forma global en toda la aplicación, con lo que, cualquier cambio que hagamos en el estilo o plantilla de un control se aplicará automáticamente a todos los controles de la aplicación.

Lo primero que debemos realizar es añadir en la aplicación un nuevo elemento de tipo ResourceDictionary, en este caso el nombre es BlackCrystal.xaml, este archivo solo tiene vista xaml y al incluirlo debería aparecer algo así:

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

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

</ResourceDictionary>

Una vez que tengamos el ResourceDictionary agregado, es momento de indicarle a nuestra aplicación que haga uso de el, esto lo podemos hacer editando el archivo Application.xaml para decirle la ruta de un nuevo diccionario de recursos que queremos que use, debería quedar así:

<Application x:Class="Application"

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

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

StartupUri="MainWindow.xaml">

<Application.Resources>

<ResourceDictionary Source="BlackCrystal.xaml"></ResourceDictionary>

</Application.Resources>

</Application>

Simplemente añadimos un nuevo nodo Application.Resources y dentro del mismo creamos un nuevo objeto ResourceDictionary, indicándole la Ruta (Source) desde la que cargar este diccionario. Con esto, cualquier estilo o plantilla que creemos en nuestro ResourceDictionary estará inmediatamente disponible en toda la aplicación.

**Elementos Comunes**

Algo importante a la hora de diseñar el look & feel de la aplicación es dotarla con una apariencia uniforme, mismos colores, mismas geometrías y mismas animaciones, si definiésemos directamente estos parámetros en cada uno de los estilos y plantillas convertiría el mantenimiento de los mismos en un infierno, teniendo que recorrer decenas de sitios para cambiar un solo color. Para solucionar esto el primer trabajo que debemos hacer es definir en el resource dictionary todos los colores y animaciones que queramos usar.El tema de ejemplo creado, Black Crystal, usa en total 9 degradados distintos y 2 animaciones que se aplican a todos los controles, estos degradados son de tipo LinearGradient y RadialGradient y las animaciones son DoubleAnimations.

Aquí podéis ver un ejemplo de Degradado Lineal, este es usado para aplicar al fondo de las ventanas un degradado en tonos de grises:

<LinearGradientBrush x:Key="DarkBackground" StartPoint="0,0" EndPoint="1,1">

<GradientStop Offset="0" Color="#FF333344"></GradientStop>

<GradientStop Offset="1" Color="#FF666677"></GradientStop>

</LinearGradientBrush>

El código xaml es muy sencillo, en la primera línea definimos el tipo de objeto (LinearGradientBrush) le asignamos una Key para poder usar el recurso que estamos creando desde cualquier parte e indicamos el Punto de inicio y de Final, estos puntos se especifican con una pareja de números que van de 0 a 1, indicando 0,0 la esquina superior izquierda y 1,1 la esquina inferior derecha, el primer número de cada par indica el punto en el eje X y el segundo número indica el punto en el eje Y. de esta forma 0,0 a 1,1 indica que el degradado recorrerá la diagonal desde la esquina superior izquierda a la esquina inferior derecha, un valor de 0,0 a 1,0 indicaría que el degradado empezaría en la esquina superior izquierda y acabaría en la esquina superior derecha, con lo que tendríamos un aspecto de degradado de izquierda a derecha horizontal, un valor de 0,0 a 0,1 crearía un efecto de degradado vertical.

Los objetos gradientStop indican que color se usara en cada parte del degradado, un degradado puede tener tantos GradientStop como queramos, la propiedad Offset indica la posición del color dentro del degradado, siendo 0 el inicio y 1 el final, podemos usar cualquier valor entre estos números tal como .2, .35 o .98 y por ultimo Color indica el color a usar en ese punto del degradado, puede ser un color ya establecido por el sistema como White, Purple o Black o una combinación de colores ARGB (A = Alpha (transparencia), R = Rojo, G = Verde y B = Azul)

El Objeto RadialGradientBrush es muy parecido al LinearGradientBrush y se compone básicamente de los mismos objetos:

<RadialGradientBrush x:Key="GlowFX" GradientOrigin=".5,1" Center=".5,1">

<GradientStop Offset="0" Color="#990000FF"></GradientStop>

<GradientStop Offset=".5" Color="#660000DD"></GradientStop>

<GradientStop Offset="1" Color="#33000000"></GradientStop>

</RadialGradientBrush>

El mayor cambio aquí es que ya no tenemos las propiedades StartPoint y EndPoint, en vez de estas tenemos GradientOrigin y Center. GradientOrigin define el punto desde el que se inicia el degradado, y Center define el punto que será el centro del circulo externo que definirá el degradado, en este ejemplo tienen valores idénticos  pero podrían ser distintos sin ningún problema.

Para aplicar estos recursos a una propiedad de un control basta con enlazar esa propiedad a un recurso estático y especificar el nombre de nuestro degradado, de esta forma:

<Button Content="Hola"

Background="{StaticResource GlowFX}">

</Button>

Por último tenemos las animaciones, en este caso la usamos para dar un efecto de activación al pasar el ratón sobre los controles o al recibir el foco ciertos controles usando el tabulador:

<Storyboard x:Key="GlowOut">

<DoubleAnimation x:Name="AnimGlowOut" BeginTime="00:00:00"

Storyboard.TargetName="GlowRectangle"

Duration="00:00:00.250" From="1" To="0"

Storyboard.TargetProperty="Opacity">

</DoubleAnimation>

</Storyboard>

E primer Lugar definimos un objeto StoryBoard y le damos un nombre para poder invocarlo desde otros objetos. Este Storyboard contiene nuestra animación, en WPF existen muchos tipos distintos de animación, la primera parte del nombre indica el tipo de valor que es capaz de animar, en este caso es un valor de tipo Double, pero podemos encontrar BooleanAnimation, CharAnimation, ColorAnimation y muchos más tipos. Las propiedades que definimos son muy simples, name aplica un nombre a la animación, BeginTime indica el momento después de invocar el storyboard en el que esta animación se iniciará, Duration indica el tiempo total que tomara la animación para ejecutarse, la propiedad atada StoryBoard.TargetName indica el objeto sobre el que se aplicará la animación y StoryBoard.TargetProperty indica la propiedad de este objeto indicado que vamos a animar, por último From y To indican el valor inicial y el valor final aplicado a esta animación. En este caso esta animación se aplicará a un objeto llamado GlowRectangle, y modificará gradualmente su propiedad Opacity en el transcurso de 250 milisegundos para llevarla desde 1 (totalmente opaco) a 0 (totalmente transparente).

La aplicación de las animaciones se realiza a través de los disparadores de eventos (Event Triggers) de un control:

<Button Name="GlowRectangle" Content="Hola">

<Button.Triggers>

<EventTrigger RoutedEvent="MouseLeave">

<EventTrigger.Actions>

<BeginStoryboard Name="{StaticResource GlowOut}">

</BeginStoryboard>

</EventTrigger.Actions>

</EventTrigger>

</Button.Triggers>

</Button>

A continuación teneis un listado de todos los Degradados y animaciones que usaremos en este proyecto, para que podáis empezar a jugar con ellas y comprender su uso, en la segunda parte de este artículo entraremos directamente a ensuciarnos las manos creando los estilos de los controles.

Espero que os sea útil y os guste! Un gran saludo, nos vemos pronto.

<!-- DARK BACKGROUND -->

<LinearGradientBrush x:Key="DarkBackground" StartPoint="0,0" EndPoint="1,1">

<GradientStop Offset="0" Color="#FF333344"></GradientStop>

<GradientStop Offset="1" Color="#FF666677"></GradientStop>

</LinearGradientBrush>

<!-- GLASS EFFECT -->

<LinearGradientBrush x:Key="GlassFX" StartPoint=".5,0" EndPoint=".5,.5">

<GradientStop Offset="1" Color="#33DDDDDD"></GradientStop>

<GradientStop Offset="1" Color="#33000000"></GradientStop>

</LinearGradientBrush>

<LinearGradientBrush x:Key="GlassFXDisabled" StartPoint=".5,0" EndPoint=".5,.5">

<GradientStop Offset="1" Color="#33BBBBBB"></GradientStop>

<GradientStop Offset="1" Color="#11000000"></GradientStop>

</LinearGradientBrush>

<!-- GLOW EFFECT -->

<RadialGradientBrush x:Key="GlowFX" GradientOrigin=".5,1" Center=".5,1">

<GradientStop Offset="0" Color="#990000FF"></GradientStop>

<GradientStop Offset=".5" Color="#660000DD"></GradientStop>

<GradientStop Offset="1" Color="#33000000"></GradientStop>

</RadialGradientBrush>

<RadialGradientBrush x:Key="GlowFXPressed" GradientOrigin=".5,1" Center=".5,1">

<GradientStop Offset="0" Color="#660000CC"></GradientStop>

<GradientStop Offset="1.2" Color="#33FFFFFF"></GradientStop>

</RadialGradientBrush>

<LinearGradientBrush x:Key="GlowFXRowPressed">

<GradientStop Offset="0" Color="#660000FF"></GradientStop>

<GradientStop Offset=".7" Color="#660000AA"></GradientStop>

<GradientStop Offset="1" Color="#66000000"></GradientStop>

</LinearGradientBrush>

<LinearGradientBrush x:Key="GlowFXProgress">

<GradientStop Offset="0" Color="#660099FF"></GradientStop>

<GradientStop Offset=".99" Color="#660022AA"></GradientStop>

<GradientStop Offset="1" Color="#00000000"></GradientStop>

</LinearGradientBrush>

<LinearGradientBrush x:Key="GlowFXProgressAnimated" MappingMode="RelativeToBoundingBox">

<GradientStop Offset="0" Color="#00000000"></GradientStop>

<GradientStop Offset=".50" Color="#660099FF"></GradientStop>

<GradientStop Offset="1" Color="#00000000"></GradientStop>

</LinearGradientBrush>

<LinearGradientBrush x:Key="GlowFXTabSelected" StartPoint=".5,1" EndPoint=".5,0">

<GradientStop Offset="0" Color="#33DDDDDD"></GradientStop>

<GradientStop Offset="1" Color="#332222FF"></GradientStop>

</LinearGradientBrush>

<!-- GLOW ANIMATION -->

<Storyboard x:Key="GlowOut">

<DoubleAnimation x:Name="AnimGlowOut" BeginTime="00:00:00"

Storyboard.TargetName="GlowRectangle"

Duration="00:00:00.250" From="1" To="0"

Storyboard.TargetProperty="Opacity">

</DoubleAnimation>

</Storyboard>

<Storyboard x:Key="GlowIn">

<DoubleAnimation x:Name="AnimGlow" BeginTime="00:00:00"

Storyboard.TargetName="GlowRectangle"

Duration="00:00:00.250" From="0" To="1"

Storyboard.TargetProperty="Opacity">

</DoubleAnimation>

</Storyboard>