Práctica 3

Entorno de programación Visual C#.

Interfaz HMI SCADA

Material: PC y Visual Studio 2013

Duración: 2 horas

Lugar: Laboratorios de prácticas (Laboratorio de Redes-Hardware)

La herramienta de desarrollo que utilizaremos para el desarrollo de las prácticas de la asignatura será el entorno de programación de Visual C, Microsoft Visual Studio 2013, el cual permite la creación, compilación y ejecución de programas escritos en lenguaje Visual C#; tanto en modo consola como en modo gráfico. Se recuerda al alumnado que, para la mejor comprensión y asimilación de los conocimientos y habilidades desarrollados durante la práctica, deberá haberse estudiado previamente el material docente disponible.

Introducción:

Como hemos visto durante el trascurso de las clases de teoría, una interfaz hombre/máquina o HMI (del inglés *Human Machine Interface*) nos permite presentar gráficamente el estado de un proceso industrial, al usuario que controla el proceso, es decir, estas interfaces son una ventana al proceso.

Nuestra interfaz HMI a desarrollar será el software de monitorización y control; deberemos interpretar la información que recibimos del propio proceso y visualizarla correctamente. Como comentamos en clase, los sistemas de visualización se rigen en España por el Real Decreto 488/97 (Normativa), en lo que a formato, tamaño, colores a emplear y otras características se refiere. Por lo que, trataremos de plasmar dicha normativa en la/s interfaz/interfaces que diseñaremos durante el trascurso de esta práctica.

Como también comentamos en clase, la obtención de los datos por el sistema de visualización SCADA se realiza por medio de algún tipo de red de comunicaciones para, posteriormente, interpretarla y mostrarla adecuadamente por pantalla. Esta práctica será complementada por la Práctica 4, por lo que en esta Práctica 3 simularemos la recepción de información y actualización de los datos mediante un tipo especial de variables, el *timer*.

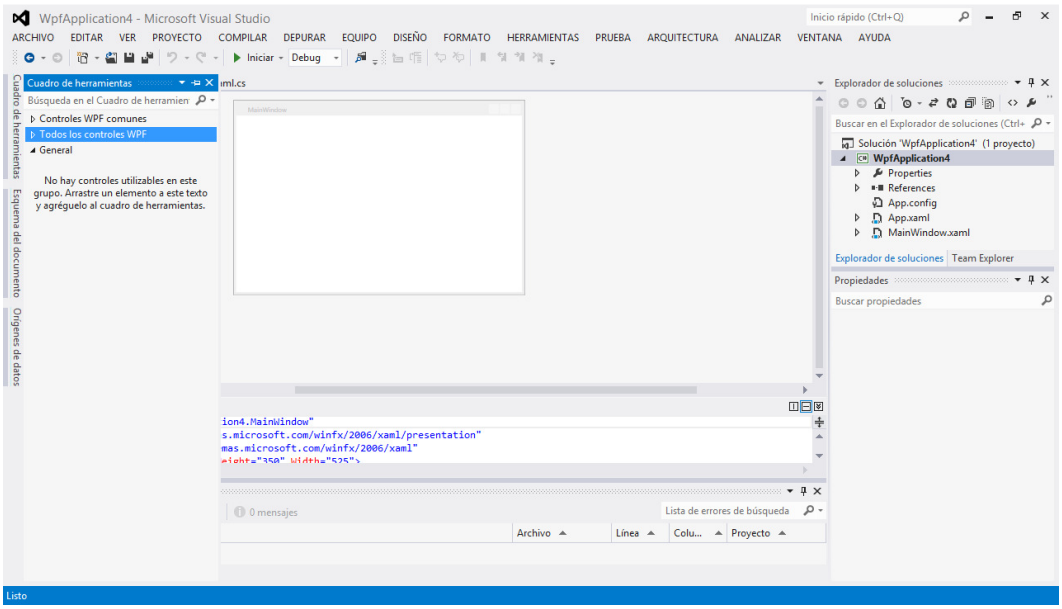
Desarrollo de la práctica:

1. Adición de archivos con componentes visuales SCADA.

En esta primera parte de la práctica procederemos a agregar, al entorno Visual Studio, dos archivos de componentes gráficos para sistemas SCADA, archivos con extensión XAML. Para ello, y debido a que se trata de archivos que contienen instrucciones para la presentación gráfica, procederemos de la siguiente forma, con el fin de agregarlos a nuestro entorno de la forma correcta.

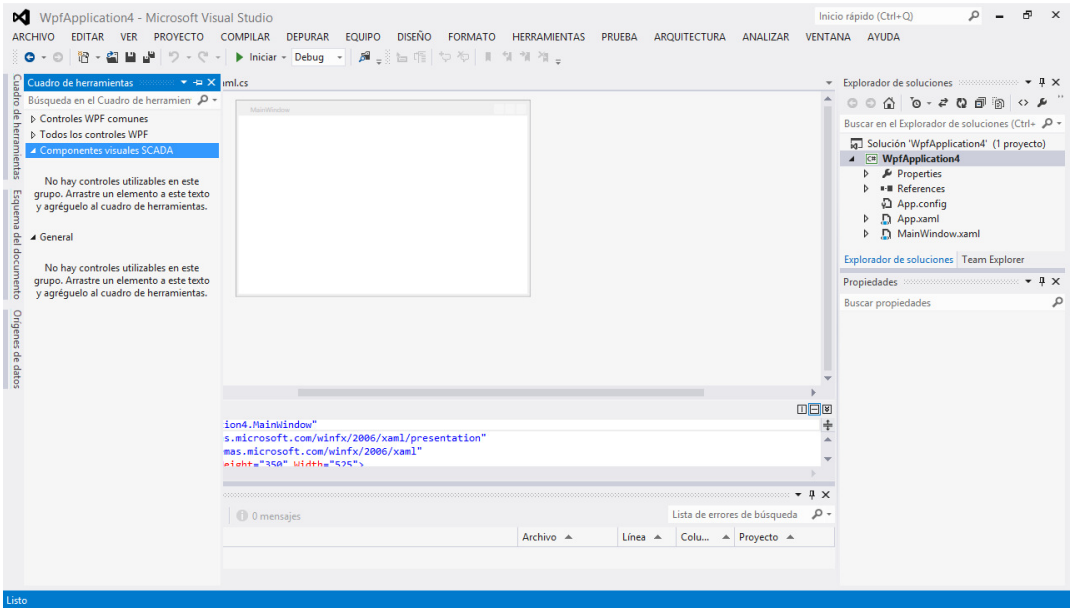
1) Igual que en la Práctica 2, crearemos un proyecto visual nuevo (*Aplicación WPF*). Consulta la Práctica 2.

2) Dentro del *Cuadro de Herramientas*, en donde se muestran los componentes gráficos disponibles, pulsaremos para cerrar el listado de los mismos, es decir, en la pestaña “Todos los controles WPF”. De esta forma veremos:



Es decir, hemos cerrado la vista de componentes y tenemos solamente un listado de los mismos agrupados por tipos.

3) Sobre dicho *Cuadro de herramientas* plegado, haremos click con el botón derecho, y seleccionaremos la opción de *Agregar pestaña*. Tras ello, le daremos un nombre, por ejemplo: *Componentes Visuales SCADA*. Esta será la pestaña en la que ubicaremos lo componentes específicamente diseñados para la interfaz HMI SCADA, y que nos permitirá localizarlos de manera independiente al resto de los componentes. Es conveniente que, cuando se agregan componentes de otros desarrolladores, se proceda de esta manera; con el fin de localizar los nuevos componentes rápidamente.

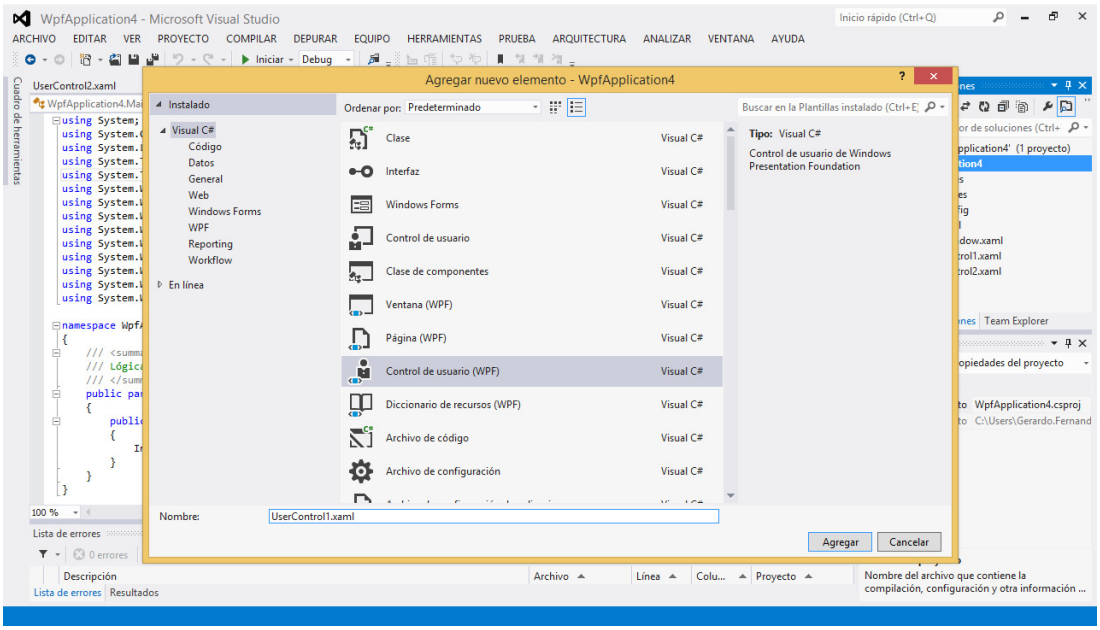


Como se muestra en la imagen anterior dicha pestaña aparecerá vacía, pues aún no hemos agregado ningún elemento a ella.

4) Descargaremos el archivo comprimido, denominado *archivos_práctica_3.rar*, disponible a través de Campus Virtual, en donde encontraremos los siguientes archivos: *bargraph_horizontal.xaml* y *cosphi.xaml*. Dichos archivos contienen las instrucciones necesarias para generar los elementos visuales que vamos a agregar al entorno Visual Studio. Lo primero de todo, son archivos gratuitos proporcionados por COPDATA (http://www.copadata.com/en/downloads/wpf-elements.html), por lo que debe ser revisada la documentación de copyright sobre ellos cuando se empleen para desarrollos comerciales. Recordad que, estos archivos, serán necesarios para cada proyecto en el que hagamos uso de ellos.

Una vez tengáis dichos archivos los ubicaremos en un directorio conocido, pues accederemos a ellos para tomar parte de la información contenida en ellos más adelante.

5) A continuación, pasaremos a preparar el proyecto recién creado para recibir los componentes que nos acabamos de descargar. Para ello, pulsaremos el botón derecho del ratón sobre el título del proyecto en el Explorador de soluciones y ejecutaremos la secuencia Agregar, Control de usuario. Automáticamente, se abrirá una ventana como la que se nos muestra a continuación:



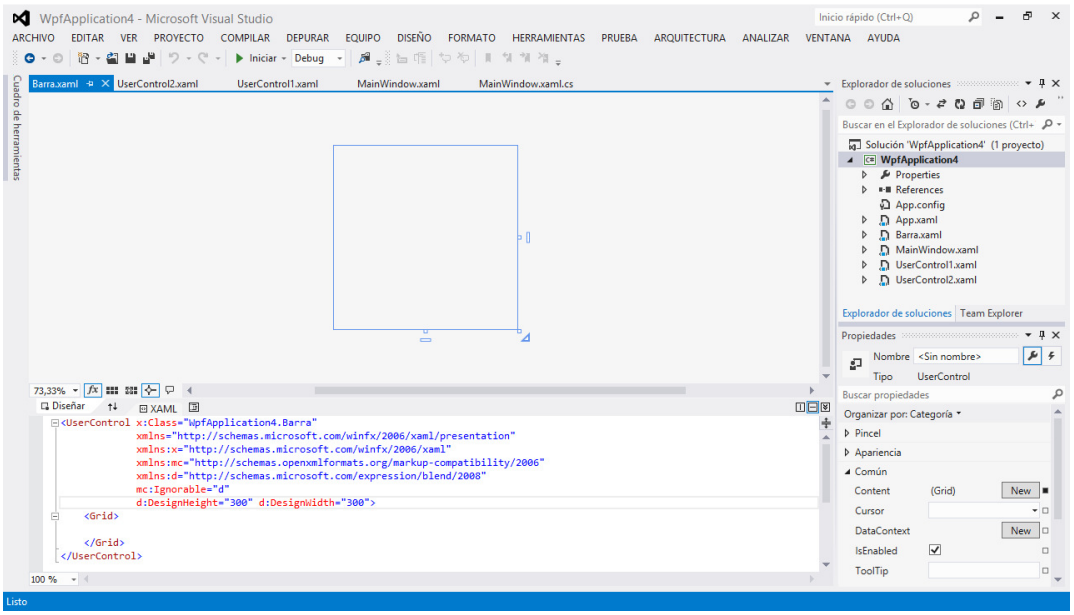
En dicha pantalla ya aparece seleccionada la opción que debemos utilizar, de las múltiples opciones que tiene, además de venir indicado el nombre que le dará al componente que vamos a agregar; podemos dejarlo así o podemos cambiar el nombre (se recomienda cambiarlo por un nombre intuitivo). Como vamos a agregar un componente de tipo “barra de progreso”, lo denominaremos barra en este ejemplo. *Muy importante, la extensión del archivo XAML no debe ser cambiada, solamente el nombre*. Finalmente, pulsaremos agregar.

Veremos que aparece, en el explorador de soluciones, un nuevo archivo denominado *Barra.xaml*, así como su vista de diseño, la cual es cargada automáticamente por el entorno (evidentemente vacía, no tenemos nada aún en ella). También podemos ver, en la pestaña de código XAML que nos aparece el siguiente código:

```
<UserControl x:Class="WpfApplication4.Barra"
             xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
             xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
             xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
             xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
             mc:Ignorable="d"
             d:DesignHeight="300" d:DesignWidth="300">
    <Grid>

    </Grid>
</UserControl>
```

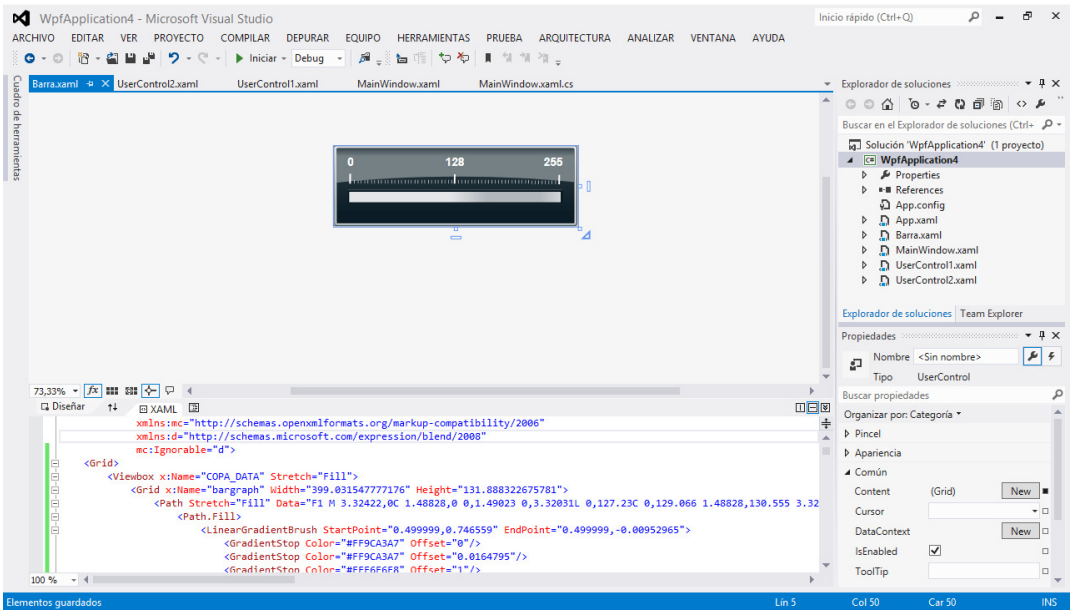
En la pantalla principal, debemos ver algo similar a lo que se muestra a continuación:

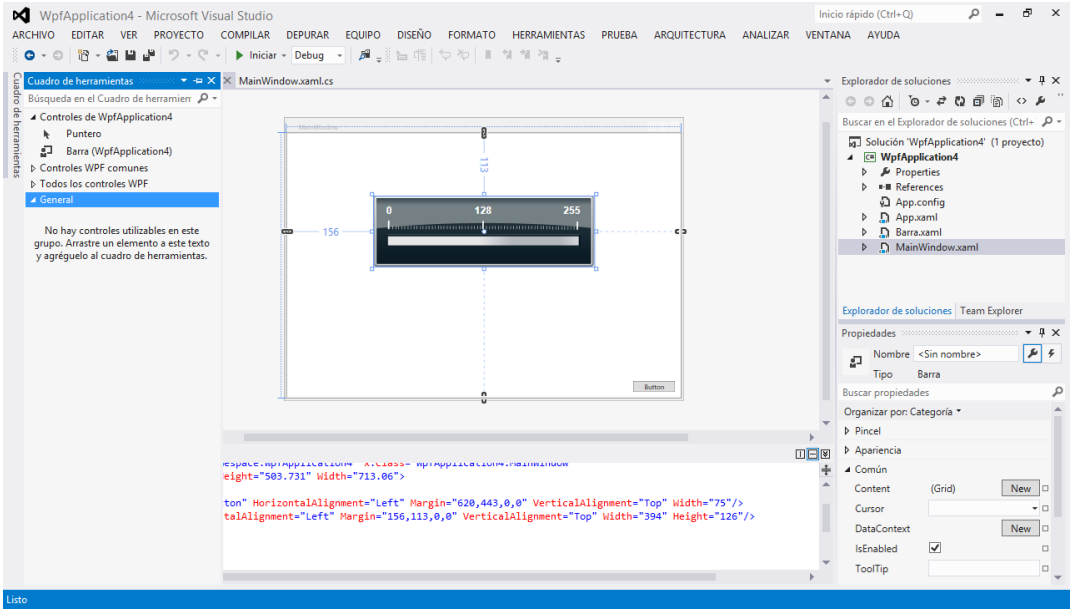


5) Una vez creada la plantilla contenedora del componente que vamos a agregar, el procedimiento para agregar el nuevo componente a nuestro entorno consiste en:

1. Eliminar la instrucción *d:DesignHeight="300" d:DesignWidth="300"*, que marca el tamaño por defecto del componente. Los componentes que agreguemos tendrán su propio tamaño. **Nota**. No borréis el signo “>”.
2. Abrir el fichero denominado *bargraph_horizontal.xaml* mediante un editor de textos.
3. Copiar el código comprendido entre las marcas *<Viewbox …>* y *</Viewbox>*, ambas marcas incluidas, es decir prácticamente todo el código del archivo.
4. Pegar dicho código entre las marcas *<Grid>* y *</Grid>* en la pantalla del visor de código *XAML*.
5. Guardar la modificación que acabáis de realizar.

Aunque el componente que acabamos de incluir ya es visible, aún no puede emplearse como un componente más, de la misma forma que vimos los componentes de la Práctica 2. Para poder emplearlo debemos ejecutar el proyecto creado, lo que permitirá compilar el componente y que éste aparezca en la pestaña del *Cuadro de herramientas* correspondiente a los Componentes de este proyecto en particular.





Como vemos en la imagen anterior, ya tenemos nuestro componente en la ventana principal de la vista de diseño agregado, simplemente arrastrando sobre la misma, igual que con un componente de tipo botón.

Nota. La explicación que se acaba de ver tiene implícitos muchas consideraciones y significados, que no se verán ni se justificarán, debido a que excede los requisitos establecidos para la asignatura (no veremos cómo crear componentes gráficos desde cero, solamente veremos algún ejemplo muy sencillo). Con todo, el estudiante interesado puede ponerse en

contacto con los profesores de la asignatura, para obtener información adicional sobre cómo funciona realmente la adicción de componentes del tipo WPF.

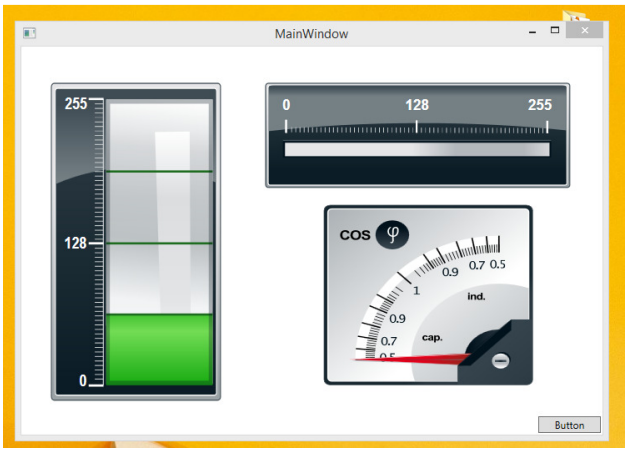
6) Ya tenemos prácticamente los componentes listos para ser utilizados en nuestro proyecto, ¿qué más necesitamos? Recordad que necesitamos dar un nombre o identificador a los componentes, igual que cuando trabajábamos con un botón o un campo de texto. Por ejemplo, al componente barra horizontal, lo nombraremos, *barra_horizonal_1*, o cualquier otro nombre que nos permita recordarlo fácilmente.

Muy importante: Este paso es necesario en todos los proyectos que realices con estos componentes. Recuerda que no debes eliminar los ficheros que el entorno crea dentro de la carpeta donde se ubica el proyecto, pues podría ocasionar que el proyecto no funcionase.

Ejercicios:

1) Repetir el procedimiento anterior para agregar los componentes incluidos en los ficheros *bargraph_vertical* y *cosphi.xaml*.

2) Crear una interfaz similar a la que se muestra en la pantalla siguiente.



2. Acceso e interacción con componentes visuales SCADA.

Los componente están agregados al entorno, podremos acceder a ellos en Visual C# mediante su nombre y la propiedad/método que queramos emplear, de la misma forma que hemos accedido a los componentes que el entorno tiene incluidos por defecto (ejemplos vistos en clase de teoría y prácticas). También podemos acceder a él mediante sus propiedades, si pulsamos con el botón izquierdo sobre él en la vista de diseño. Son muchas las propiedades que podemos modificar, pero no veremos todas.

Por ejemplo, una de las propiedades que podemos fijar en el componente barra horizontal es su valor. Por ejemplo, pulsando dos veces sobre el mismo se generará automáticamente la función que se ejecuta cuando el componente se carga en la ventana cuando iniciamos el programa, es decir, no aparece una función asociada al evento del doble click, como nos aparecía en el componente botón, sino un evento de carga (load) que ejecutará esta función cuando iniciemos el programa; de forma similar al *constructor* (visto en clase). Esto sucede porque cada componente tiene un evento asociado por defecto, dependiendo de la utilidad o finalidad del mismo. Vamos a asignar el valor de 30, mediante la instrucción *barra_horizontal_1.slider.Value=30;*. Ejecuta el programa y verás que, automáticamente, el entorno visual actualiza la representación gráfica del componente. Se trata de un componente dinámico.

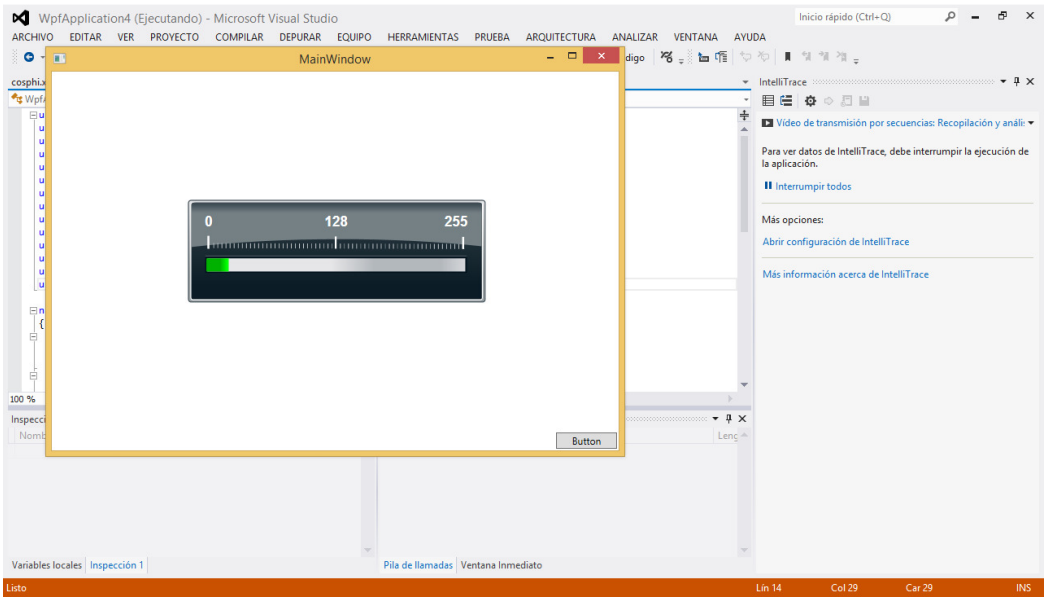
```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;

namespace WpfApplication4
{
    /// <summary>
    /// Lógica de interacción para MainWindow.xaml
    /// </summary>
    public partial class MainWindow : Window
    {
        public MainWindow()
        {
            InitializeComponent();
        }

        private void barra_horizontal_1_Loaded(object sender, RoutedEventArgs e)
        {
            barra_horizontal_1.slider.Value=30;
        }
    }
}
```

Además de poder modificar el valor que muestra este componente, podemos modificar los valores que se muestran en el indicador superior (actualmente, cero, 128 y 255):

```
//modificación de los valores de la etiqueta superior
barra_horizontal_1.min.Content = -1;
barra_horizontal_1.mid.Content = 0;
barra_horizontal_1.max.Content = 1;
```



Ejercicio:

3) Diseña una interfaz con una barra horizontal y asigna el valor de 128 a la misma, bien interactuando con el programa mediante un botón o cuando se carga el componente. ¿Qué sucede cuando ejecutas el programa?

Lo que sucede, en el ejercicio anterior que hemos propuesto, es un efecto secundario del diseño de componentes gráficos; éste ha sido diseñado teniendo en cuenta unos rangos y valores físicos para los límites representables en pantalla, lo cual choca con el ejercicio anterior propuesto. No es ningún problema, suele suceder con más de un componente.

Para resolver el problema anterior, normalizaremos los valores que queramos representar atendiendo tanto al mínimo como al máximo valor representable, así como al mínimo incremento que podemos hacer para mostrar un resultado apreciable. En este caso en concreto, el componente solamente permite valores enteros en el rango [0,347].

Ejercicio:

4) Repite el ejercicio anterior y asigna correctamente el valor de 128 a la misma, vista la explicación anterior.

Nota. No veremos todas las propiedades de cada uno de los componentes SCADA que utilizaremos (básicamente son propiedades relativas a su aspecto: color, forma, etc., aunque puedes probar a modificar alguno de ellos), solamente será necesario saber cómo se accede a ellos para modificar su valor y, por ende, el valor que muestran por pantalla, así como el

acceso a otros elementos representativos como las etiquetas. No modificaremos los componentes ni modificaremos aspectos de su funcionamiento interno.

Ejercicios:

5) Modifica el programa anterior para que, cada vez que se pulse un botón, el valor del componente modifique su valor sumando al valor actual 10 unidades.

6) Modifica el programa anterior para que, cuando se pulse un botón se incremente el valor del componente visual *barra horizontal* en una unidad cada segundo, hasta que se alcance el valor 128 (el valor inicial será de 30). **Nota**. En Visual C# podemos utilizar un tipo especial de variable denomina *timer* para la ejecución temporizada de líneas de código. Pasamos a mostrar un ejemplo, a continuación. **Muy importante**. Cada instrucción de las que se muestra a continuación deberá ser añadida al programa donde corresponda.

```
//Incluimos, en la zona superior del código, la librería que nos permitirá
//hacer uso de los timers que definamos
using System.Windows.Threading;


//creamos una variable de tipo timer, en la zona de declaración de
//variables globales, tal y como se muestra a continuación
DispatcherTimer mi_timer = new DispatcherTimer();

//incluiremos el código de configuración del timer donde sea necesario,
//por ejemplo, al pulsar un botón, si queremos que el timer que acabamos
//de declarar se ponga en marcha en ese preciso instante

//primero, definimos el intervalo de tiempo en el que el timer se
//ejecutará, en este caso cada segundo. Los valores entre paréntesis
//representan los días, horas, minutos, segundos y milisegundos.
mi_timer.Interval = new TimeSpan(0, 0, 0, 1, 0);

//segundo, definimos qué queremos hacer cuando el timer se active, lo cual
//se hace indicándolo con la instrucción. Entre paréntesis se indica el
//nombre de la función que queremos que se ejecute. Evidentemente,
//tendremos que definir la función.
mi_timer.Tick += new EventHandler(ejecutar_cada_tick);

//en tercer lugar, definimos la función, cuyos parámetros deben seguir el
//formato que se indica. Esta es la función que se ejecutara cada tick del
//timer definido (evento disparado o evento temporizado)
private void ejecutar_cada_tick(object sender, EventArgs e){
  //código a ejecutar en el intervalo definido
}

//por último, las instrucciones que permiten iniciar el timer definido y
detenerlo son, respectivamente.

mi_timer.Start();

mi_timer.Stop();
```

Ejercicios:

7) Añade un nuevo *timer* al ejemplo anterior, cuyo código se ejecutará cada segundo después de pulsar un botón. Dicho *timer* deberá actualizar, con un valor aleatorio comprendido entre 0 y 100, el valor mostrado por el componente *bargraph_horizontal.xaml*. **Nota**. Consulta los apuntes de clase para repasar cómo realizar la declaración de variables de tipo aleatorio.

8) Diseña una interfaz con un componente de tipo *bargraph_horizontal.xaml* y otro de tipo y *cosphi.xaml*. Añade un botón y prueba a modificar sus valores mediante componentes *TextBox*, es decir, lee el valor del *TextBox* y representa gráficamente el valor del mismo. La representación de valores en el componente *bargraph_horizontal.xaml* se encuentra en el mismo rango que en el componente *bargraph_horizontal.xaml*. ¿Cuál será el rango para el componente *cosphi.xaml*? **Nota**. Para poder acceder al valor visualizado en este último componente debes acceder a su propiedad *arrow_angle*, una vez que le hayas dado un nombre o identificador al mismo.

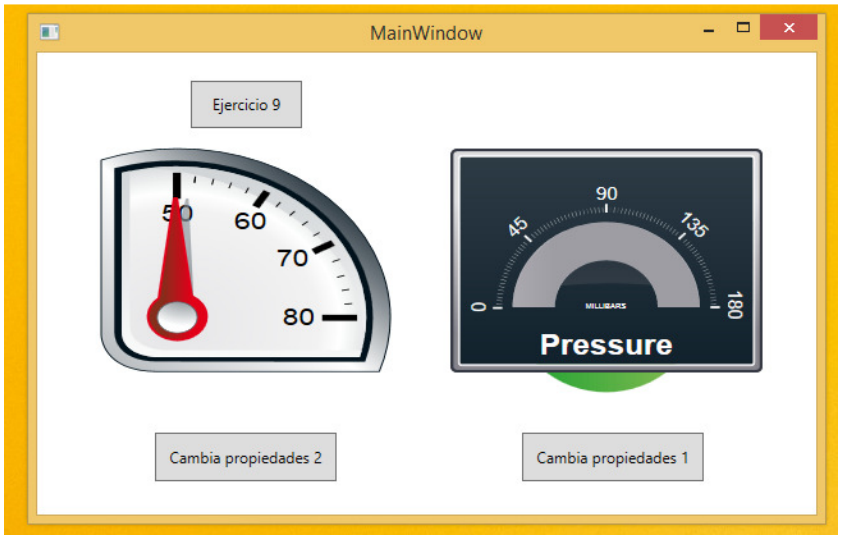
9) Diseña una interfaz como la que se muestra a continuación, empleando los componentes, *indicating_instrument.xaml* y *speedometer.xaml*. Define un intervalo mínimo de 20 y máximo de 30 para el rango de valores aleatorios generados para actualizar el componente speedometer y activa, mediante un botón, la obtención de un nuevo valor aleatorio con cada pulsación. **Nota**. A continuación, se muestra una tabla con las propiedades de los componentes a las que podéis acceder para modificar las mismas (recordar asignar un identificador al componente una vez agregado a la ventana de diseño para poder acceder a sus propiedades):

	indicating_instrument	
	Tipo	Propiedad
Texto inferior	String	Headline.Text
Texto de unidades	String	Headline.Text
Valores indicador (1 a 5)	String	value1.Content
Marcador	Double	indicator_angle.Angle

	speedometer	
	Tipo	Propiedad
Valores indicador (1 a 4)	String	value1.Content
Marcador	Double	arrow_angle.Angle

10) Añade un botón para actualizar todas las propiedades mostradas en las tablas anteriores para los componentes del ejercicio 9.

Ejemplo de interfaz visual para los ejercicios 9 y 10:



6. Componentes visuales agregados: creación de componentes.

Los componentes visuales agregados enriquecerán la aplicaciones industriales que desarrollemos (realmente, lo que estamos desarrollando es la interfaz de control), es más, existen multitud de componentes distribuidos por fabricantes de hardware/software, para el desarrollo de aplicaciones HMI SCADA. Además de los componentes que hemos visto en la primera parte de la práctica, iremos viendo nuevos componentes conforme vayan transcurriendo las prácticas de la asignatura, incluso podemos diseñar nosotros mismos nuevos componentes.

A continuación, mediante un ejemplo, diseñaremos un componente denominado *reloj digital*, con el fin de mostrar la hora en nuestras aplicaciones. Básicamente, seguiremos los pasos que hemos visto hasta ahora, solo que detallaremos algo más cuando veamos cómo implementar el reloj.

Ejemplo:

Vamos a implementar un reloj digital, diseñando el componente de usuario *reloj.xaml*. Para ello, añadiremos dicho componente en un proyecto y crearemos un *timer* que, activado cada segundo, nos actualice el componente del reloj. Para que este componente muestre la hora correctamente, el campo *Value* debe tener un formato especial, denominado *TimeSpan*, un

tipo especial de variable que permite almacenar valores de tiempo en formato horas, minutos y segundos (también admite fechas, pero esto lo veremos más adelante).

Primero, agregaremos un control de usuario, siguiendo las instrucciones que hemos visto durante el desarrollo de esta práctica. Le daremos el nombre *reloj.xaml*. Acto seguido, añadiremos el siguiente código entre las marcas *<Viewbox …>* y *</Viewbox>*, marcas incluidas:

```
<Viewbox Stretch="Fill">
<TextBlock Text ="hh:mm:ss" TextAlignment="Center" Height="120"
Width="350"  FontFamily="Consolas" FontSize="80" Name="tbk_clock">
  <TextBlock.Foreground>
     <LinearGradientBrush EndPoint="0.5,1" StartPoint="0.5,0">
         <GradientStop Color="Black" Offset="0.52"/>
         <GradientStop Color="Black"/>
     </LinearGradientBrush>
   </TextBlock.Foreground>
   <TextBlock.Background>
       <LinearGradientBrush EndPoint="0.5,1" StartPoint="0.5,0">
          <GradientStop Color="White" Offset="0"/>
          <GradientStop Color="White" Offset="1"/>
       </LinearGradientBrush>
   </TextBlock.Background>
</TextBlock>
</Viewbox>
```

Podéis probar asignando diferentes valores de *Color* (Blue, Green, etc.), para ver cómo se comporta la representación del componente en pantalla, o con el tipo de letra *FontFamily*, así como con el resto de campos (solamente haremos este tipo de pruebas con este componente y a modo de ejemplo, pues es necesario disponer de un manual de referencia técnica para conocer todas las combinaciones y cambios posibles).

Una vez hecho esto, dispondremos del control creado al igual que con el resto de controles que hemos añadido en esta práctica, por lo que podremos agregarlo a nuestra ventana principal de diseño. Recodad que, debéis ejecutar el programa antes de que este control aparezca en el cuadro de herramientas.

Por último, deberemos definir un *Timer* de la forma que hemos visto en esta misma práctica para que, cada segundo, se proceda automáticamente a actualizar la hora en el evento que se dispara según dicho *Timer* (repasa la página 10 de está práctica). En Visual C# tenemos una instrucción que nos permite obtener directamente un dato *TimeSpan* para actualizar el valor del reloj y es la instrucción *DateTime.Now*. Simplemente, el código siguiente nos permitirá actualizar el reloj digital:

```
private void myDispatcherTimer_Tick(object sender, EventArgs e){
  tbk_clock.tbk_clock.Text = DateTime.Now.ToString("hh:mm:ss");
}
```

El aspecto final, tomando como punto de partida la interfaz visual de los ejercicios 9 y 10, podría ser:

Ejercicios:

11) ¿Qué deberías añadir en el programa realizado en el ejemplo anterior para que, además, el reloj se encuentre inicializado con la hora correcta cuando ejecutamos el programa? Dependiendo de cómo completes el ejemplo anterior igual no es necesario añadir nada.

12) Investiga el objeto de Visual C# *TimeSpan*, de manera que puedas fijar la hora que decidas mediante la modificación de la instrucción que inicializa el reloj cuando éste es actualizado. **Nota**. Consulta la ayuda de Microsoft Visual Studio disponible. Ten en cuenta que admite múltiples formatos la creación de marcas *TimeSpan*, elije aquella que solamente muestra la hora, los minutos y los segundos, o no funcionará correctamente cuando actualices el campo *Value* del componente *DigitalClock* (*Instantiating a TimeSpan Value* en http://msdn.microsoft.com/en-us/library/system.timespan%28v=vs.110%29.aspx). También tenéis ejemplos en el enlace http://www.csharp-examples.net/string-format-datetime/. Hay que seguir un procedimiento similar a la creación del objeto *random*, descrito en el ejercicio 5 del Tema 3.

13) Descarga del espacio virtual de la asignatura, dentro del campus virtual, el proyecto disponible bajo el nombre de *Proyecto Práctica 3* (también puedes utilizar el que hayas desarrollado en las clases teóricas, en relación al ejercicio del Tema 3). Sobre él realizaremos una serie de modificaciones y ampliaciones. Se recomienda leer por completo el enunciado, antes de proceder a su resolución y consultar el ejemplo de apariencia mostrado al final del enunciado.

Los cambios y/o ampliaciones a realizar son:

1. Sustituye la etiqueta que muestra el reloj con la hora del sistema por un componente *reloj.xaml* y modifica el código para que se actualice la hora correctamente. La fecha deberá seguir apareciendo mediante un componente *Label*, siguiendo las indicaciones vistas en el trascurso de las clases teóricas (instrucción *DateTime.Now.ToString("dd/MM/20yy")*).
2. Mediante un *timer* y, en intervalos de 5 segundos, simularemos la recepción de información relativa al funcionamiento del motor (revoluciones por minuto y temperatura). Ésta deberá mostrarse mediante componentes *speedometer* y *horizontal_bar* respectivamente, de manera correcta y adecuada, además de mostrarse en el registro de eventos y alarmas un evento informativo al efecto (recepción de información y valores). La temperatura podrá oscilar entre los 0 y los 120 grados y las revoluciones por minuto de 0 a 9.000 (valores aleatorios en dichos rangos), aunque se asume que las revoluciones por minuto poseen un factor de escalado de x100 (indicar mediante una etiqueta, componente *Label*), para que el componente solamente muestre en su escala valores de cero a 45. Esta información comenzará a recibirse cuando el motor se encuentre en estado encendido. **Nota**. Para realizar este apartado deberás convertir el valor entero generado aleatoriamente para la temperatura, en el rango correcto de visualización (repasa la primera parte de la práctica, ejercicio 4). En cuanto al valor de r.p.m., el enunciado del ejercicio ha sido ajustado para que no haya que realizar conversiones en el *speedometer*, pues la visualización de valores está basada en un ángulo de entre 0 y 90 grados.
3. Corrige el punto 2 para que, los valores aleatorios generados estén entre los 50 y los 60 grados para la temperatura, y entre las 20 y 30 mil revoluciones para las r.p.m.; ambos valores incluidos.
4. Corrige el punto 2 para que, el valor de la temperatura y las r.p.m. pasen a cero cuando el motor es apagado. **Nota**. Esto en un sistema real no ocurre exactamente así, debido a los tiempos de enfriamiento y parada, respectivamente.
5. Modifica el comportamiento del programa para que:
    a. Si la temperatura del motor supera los 70 grados centígrados el color del texto que aparece en el componente *RichTextBox* (información de eventos y alarmas) muestre el mensaje en color rojo. **Nota**. Recuerda cambiar el interlineado a 1px, dentro de las propiedades del componente *RichTextBox*. Ejemplo:

```
//Cambio del color del texto mostrado en un componente richTextBox
//explicado en clase de teoría
TextRange tr = new TextRange(textbox.Document.ContentEnd,
textbox.Document.ContentEnd);
tr.Text = DateTime.Now.ToString("hh:mm:ss") + ": r.p.m.: " + rpm + ",
temperatura: " + temperatura;
tr.ApplyPropertyValue(TextElement.ForegroundProperty, Brushes.Red);
```

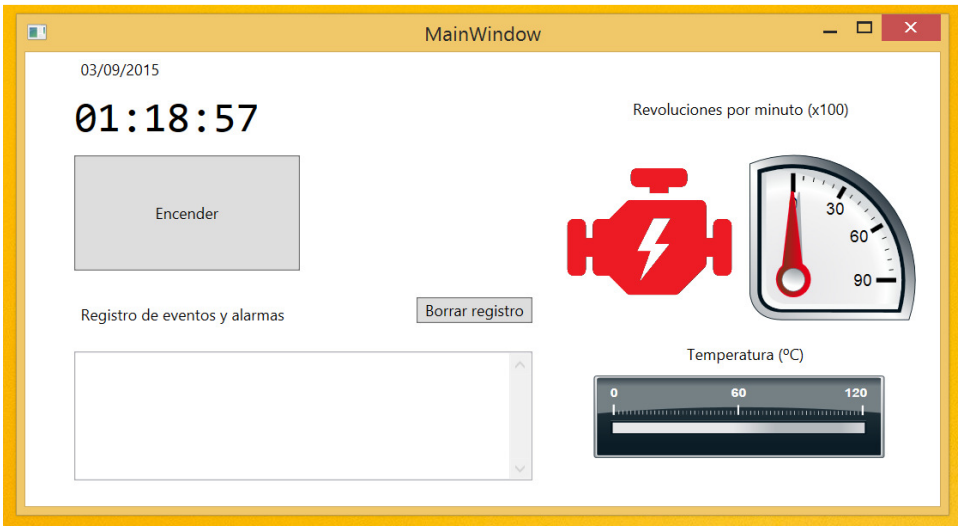
Existen algunos colores predefinidos, como se muestra en la instrucción anterior (rojo, verde, azul, amarillo, etc., pero en inglés), aunque también puedes establecer el color en base a los componentes RGB (Red Green Blue) del mismo. Puedes investigar las propiedades y métodos de *Color*. Recuerda que esta propiedad está disponible para los componentes *RichTextBox*.

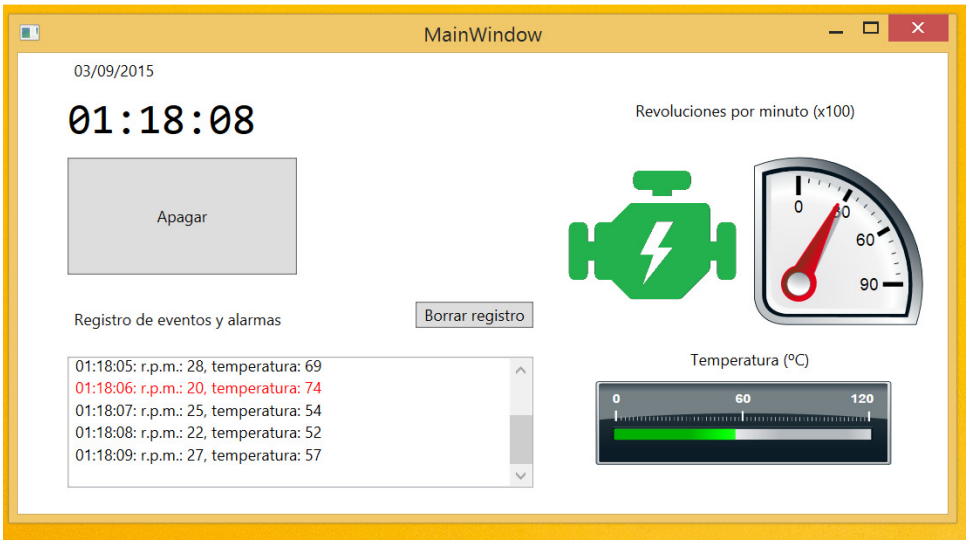
b. Si la temperatura del motor supera los 80 grados el programa deberá apagar automáticamente el motor e indicarlo en el registro de eventos. En este caso no se pondrán a cero los valores de temperatura y de rpm indicados en sus respectivos displays.

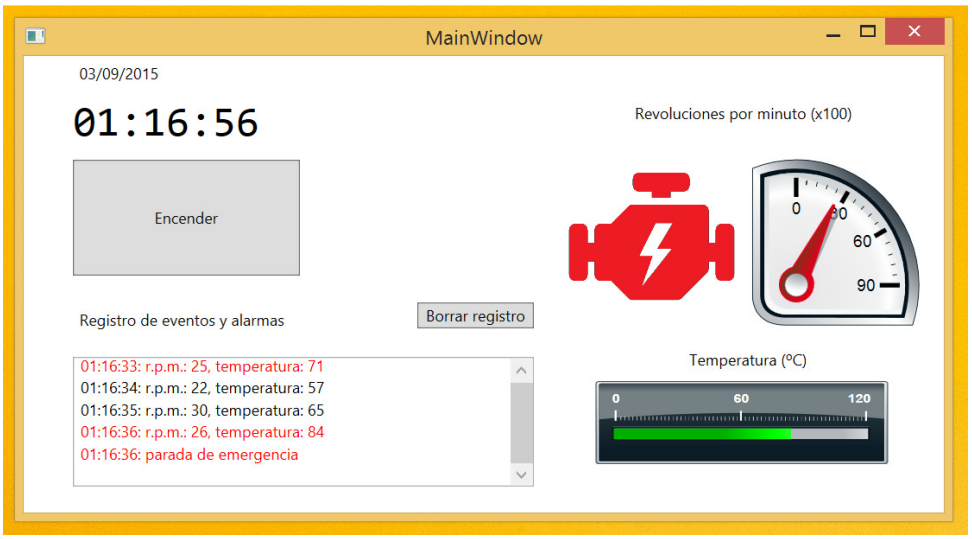
c. Corrige el punto 2 para que, los valores aleatorios generados para la temperatura estén entre los 50 y los 85 grados.

6. Debes cuidar el aspecto final de la interfaz, para que el tamaño de los componentes sea proporcionado, coherente y claro.

El aspecto visual deberá ser similar al siguiente, imágenes en estado apagado, en funcionamiento y con alarma de temperatura, respectivamente:







7. Lecturas aconsejadas y tutoriales

Lecturas aconsejadas para ampliar los conocimientos de Visual C#:

- http://www.wpf-tutorial.com/
- http://www.wpftutorial.net/