



# GRAPHER

Interfaces Graficas de Usuario - Practica Final  
Curso 2020-21

**JAVIER MATEOS DEL AMO**  
**DNI: 70832988A**

# Índice

<b>Manual de usuario</b>	<b>3</b>
Proyectos (Projects)	3
Introducción	3
Creación de un proyecto	3
Cerrar un proyecto	3
Gráficas (Graphs)	4
Introducción	4
Creación de gráficas	4
Ocultación de gráficas	5
Reordenación de gráficas	5
Borrado de gráficas	6
Edición de gráficas	6
Generación de puntos automática (Polinomio)	7
Otros	8
<b>Manual del programador</b>	<b>9</b>
Introducción	9
Views (Vistas)	9
ViewModel	10
Modelo	11
Helpers	12
Validation	12
Converters	13
Styles	13
Images & Fonts	13
<b>Librerías usadas</b>	<b>14</b>
<b>Referencias</b>	<b>15</b>

# Manual de usuario

## Proyectos (Projects)

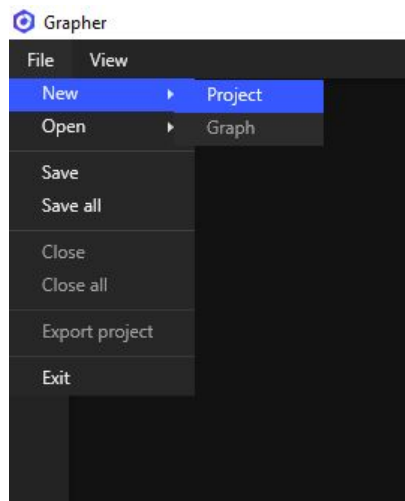
### Introducción

La aplicación consta de un sistema de proyectos. Estos, permiten la agrupación de varias gráficas cada una con sus ajustes independientes.

Pueden ser creados, cerrados y exportados como imágenes (PNG o JPEG). Estos se muestran en la barra de pestañas superior de la aplicación.

### Creación de un proyecto

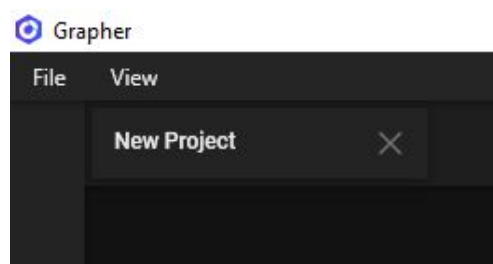
Para crear un proyecto se ha de navegar al menú superior y seguir la siguiente secuencia: **File > New > Project**



### Cerrar un proyecto

Para cerrar un proyecto, se debe localizar su correspondiente pestaña y presionar el botón **X** que aparece a la derecha de su nombre.

Alternativamente, se puede cerrar un proyecto acudiendo al menú superior y siguiendo la siguiente secuencia: **File > Close**, o, si se desea cerrar todos los abiertos, **File > Close all**.



# Gráficas (Graphs)

## Introducción

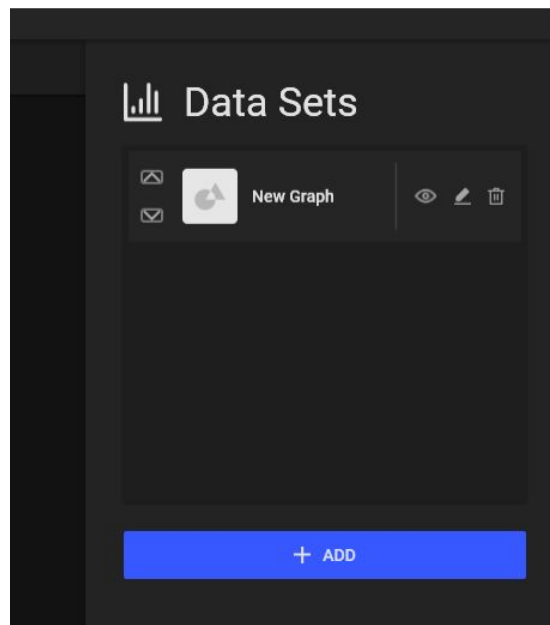
Cada proyecto puede almacenar cero o más gráficos. Cada gráfico contiene una serie de propiedades: Nombre, tipo de visualización y generación, visibilidad, opciones visuales del gráfico y datos del gráfico.

Los gráficos pueden ser creados, modificados, reordenados, ocultos y eliminados. Estos se muestran en el panel lateral de la aplicación.

## Creación de gráficas

Para la creación de un gráfico debemos de haber creado y seleccionado un proyecto antes. Si no, el botón estará bloqueado.

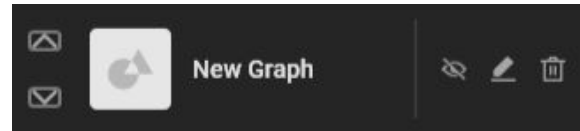
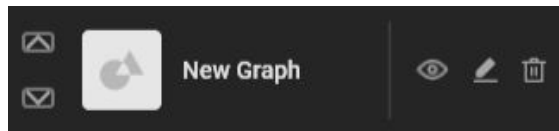
Después, debemos navegar a la parte inferior del panel lateral, donde se encuentra el botón 'Add'. Una vez pulsemos el botón, se añadirá un gráfico con los ajustes y nombres por defecto



Alternativamente, estos pueden ser también creados desde el menú superior, siguiendo la siguiente secuencia: **File > New > Graph**

## Ocultación de gráficas

Para ocultar un gráfico, se ha de presionar el botón con forma de *ojo* del mismo. Este cambiara de forma a un *ojo tachado* si este está oculto

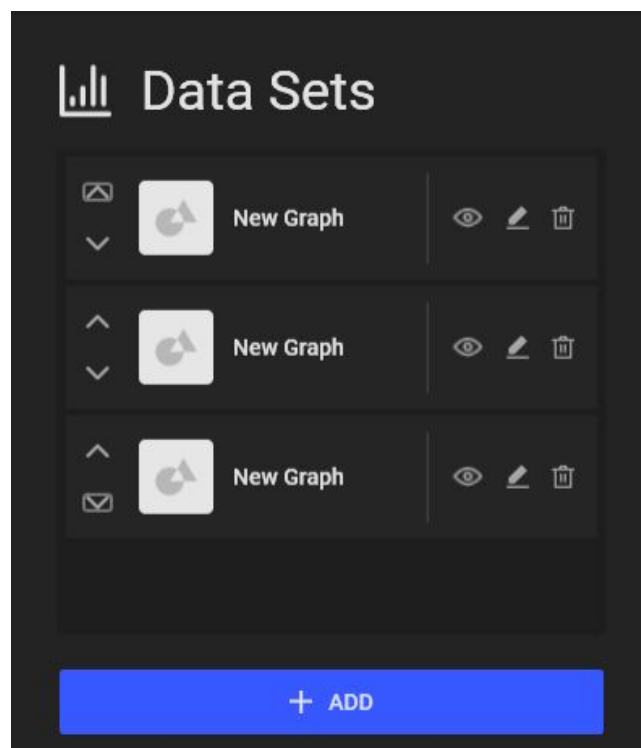


## Reordenación de gráficas

Para reordenar un gráfico, se ha de pulsar en los botones *flecha* que aparece a la izquierda de su imagen.

Si alguno de estos aparecen con reborde gris, significa que el gráfico no puede ser reordenado en esa dirección por estar en la última posición posible para esa acción.

El orden de los gráficos determina el orden de dibujado, siendo el de más arriba el que es dibujado por encima de todos sus inferiores.

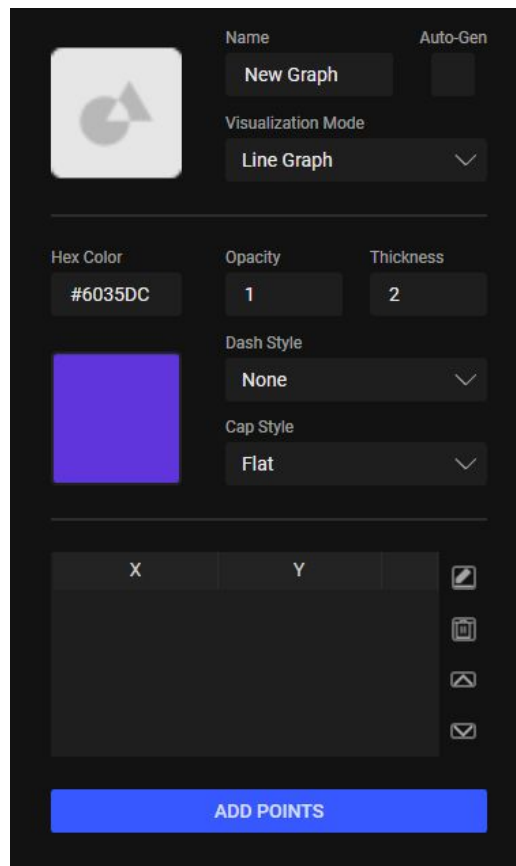


## Borrado de gráficas

Para borrar un gráfico, se ha de pulsar en el botón *cubo de basura* que aparece a la derecha de su casilla. El borrado no solicita confirmación. Sus datos y ajustes no se guardarán.

## Edición de gráficas

Para la edición de una gráfica se ha de pulsar el botón *lápiz* de su casilla. Este abrirá la siguiente ventana:



Los cambios en esta ventana, si son correctos, se actualizan instantáneamente.

A continuación, se citan cada una de los ajustes y su funcionalidad:

- **Name:** Nombre de la gráfica
- **Auto-Gen:** Si se activa, la gráfica pasará a generar sus puntos de manera automática a través de una expresión polinómica. Si está desactivado, se deben introducir de forma manual
- **Visualization Mode:** Modo de visualización de la gráfica: de líneas o de barras
- **Hex Color:** Color en Hexadecimal de la gráfica. Este campo solo acepta formatos válidos. Si no se introdujese uno válido, no se aplicarán los cambios.

- **Opacity:** Opacidad de la gráfica. Este campo solo acepta valores *Double* entre 0 y 1. Si no se introdujese uno válido, no se aplicarán los cambios.
- **Thickness:** Grosor de la línea o barra. Este campo solo acepta valores *Double*. Si no se introdujese uno válido, no se aplicarán los cambios.
- **Dash Style:** Estilo de la línea o barra
- **Cap Style:** Estilo de la terminación de la línea o barra
- **Cuadro de color:** Visualización del color y opacidad seleccionados
- **Tabla de valores:** Tabla de valores de puntos con coordenadas X e Y.

Se puede realizar selección múltiple o individual para la utilización de los botones a su derecha. Estos, en orden descendente, son: Editar punto/s, eliminar punto/s, mover punto/s hacia arriba y mover punto/s hacia abajo.

Si el modo 'Auto-Gen' se encuentra activo, esta tabla se desactiva a la interacción del usuario

- **Botón Add Points:** Visible cuando el 'Auto-Gen' se encuentra desactivado. Se abre una nueva ventana para la introducción de puntos de forma manual (Valor X y Valor Y). Solo se aceptan valores 'Double' válidos
- **Botón Edit Polynomial Expression:** Visible cuando el 'Auto-Gen' se encuentra activado. Se abre una nueva ventana para la introducción de la expresión polinómica.

## Generación de puntos automática (Polinomio)

Al presionar el botón 'Edit Polynomial Expression', se abrirá la siguiente ventana:

The image shows a software window titled "Polynomial Expression". On the left, there is a large text input field for the polynomial expression. Below it, there are two input fields labeled "X Min Value" and "X Max Value", both containing the number "0". Below those is a "Poly Grade" input field containing the number "2". On the right side of the window, there is a table with three rows, each representing a grade and its corresponding value:

Grade 0	0
Grade 1	0
Grade 2	0

Los cambios en esta ventana, si son correctos, se actualizan instantáneamente.

A continuación, se citan cada una de los ajustes y su funcionalidad:

- **Polynomial Expression:** Campo de sólo lectura que muestra el polinomio resultante tras los ajustes.
- **X Min Value y X Max Value:** Valores mínimo y máximo respectivamente con los que se sustituirá en el polinomio. Solo se aceptan valores 'Integer' válidos.
- **Poly Grade:** Grado del polinomio. Solo se aceptan valores 'Integer' válidos.
- **Tabla lateral:** Tabla encargada de representar cada monomio del polinomio. Los datos de cada monomio solo pueden ser valores del tipo 'Integer'. Si se quiere omitir un monomio, basta con dejar su valor a 0.

## Otros

- Si existen gráficas representadas en la pantalla, seleccionar una porción rectangular manteniendo el click izquierdo del ratón, causará que los puntos fuera del área sean purgados.
- El panel lateral se puede intercambiar de posición acudiendo al menú superior y siguiendo la siguiente secuencia: **View > Swap Panels**.
- Se puede salir del programa pulsando la X de la ventana o acudiendo al menú superior y siguiendo la siguiente secuencia: **File > Exit**.



# Manual del programador

## Introducción

La práctica está construida entorno al patrón de diseño MVVM (Model, View, ViewModel) **(Ver diagrama de clases al final del documento)**.

Se ha seguido la estrategia de una clase *ViewModel* por cada clase *View*, y clases auxiliares, de datos y enumeraciones en el modelo. Adicionalmente, se crean una serie de carpetas que almacenan los *Converters*, *Validators*, estilos (*Styles*), fuentes de texto (*Fonts*) e Imágenes (*Images*).

## Views (Vistas)

La aplicación consta de una vista principal (*MainWindow*) y otras cuatro secundarias (*EditGraphWindow*, *ManualAddWindow*, *AutoAddWindow* y *EditPointWindow*).

Todas estas se comunican con su respectivo *ViewModel* a través de *Binding* y *Command*. Se utilizan los *Bindings* para recuperar datos (*Binding OneWay*) o modificar datos (*Binding TwoWay/OneWayToSource*) de su correspondiente *ViewModel*. Se establece el *DataContext* de cada vista en el constructor de la misma (*DataContext = mainVM*, por ejemplo)

Los *Commands* (Comandos) se utilizan para la ejecución de funciones en el *ViewModel* tras alguna interacción con la vista. Por ej: Click de un botón o reescalado de la ventana.

La única excepción al *Binding* de la información es el *Canvas*. Tras una extensiva búsqueda de información en diversas páginas, se llega a la conclusión de que no es posible vincular una colección de formas (*Shapes*) o de ninguno de sus derivados mediante *Binding* a un *Canvas*, por las propiedades del mismo y de las formas que contiene, las cuales no escuchan a los eventos *CollectionChanged*. Como alternativa, se pasa el *canvas* de la ventana principal al constructor del *ViewModel* de la misma.

La única excepción a la implementación de *Commands* en los eventos de las *vistas* es en el caso de que se necesiten los parámetros que reciben los eventos (*EventArgs*) para realizar su correcta función. En el caso de *MainWindow*, los eventos de *Mouse*, los cuales necesitan recuperar la posición del mismo de *EventArgs*, o el *TabItem* actualmente seleccionado, que necesita establecer *e.Handled = true* para que no replique la llamada del evento a capas superiores.

A continuación, se introduce cada vista y sus métodos principales mediante una breve descripción:

- **MainWindow:** Ventana principal. Se abre con la ejecución de la aplicación. Se encarga de mostrar los proyectos creados y las gráficas y su representación. Sirve como enlace para abrir cualquier otra ventana.
  - projectsTab SelectionChanged(): Actualiza el Canvas si el proyecto seleccionado (SelectionChanged) cambia
  - Eventos "Mouse": Se manejan los distintos eventos del ratón (Clic y Move) para la selección del área rectangular a purgar
- **EditGraphWindow:** Ventana secundaria. Permite la edición de una gráfica y sus datos. Modo no modal.
- **ManualAddWindow:** Ventana secundaria. Permite añadir un nuevo punto de forma manual. Modo modal
- **AutoAddWindow:** Ventana secundaria. Permite modificar la expresión polinómica para la generación automática de puntos. Modo no modal.
- **EditPointWindow:** Ventana secundaria. Permite editar un punto existente de forma manual. Modo modal.

## ViewModel

Cada ventana de la aplicación dispone de su propio *ViewModel*. La ventana *MainWindow* crea su *ViewModel* en su constructor. En el resto de ventanas, se crea el *ViewModel* de la ventana que se desea abrir desde el *ViewModel* actual, y el nuevo *ViewModel* se encarga de crear la vista (*View*) para ese nuevo *ViewModel*, estableciendo el `DataContext` de la ventana a `this` (A su *ViewModel*).

A continuación, se introduce cada *ViewModel* y sus métodos y propiedades principales mediante una breve descripción:

- **MainViewModel:** *ViewModel* de la venta *MainWindow*. Proporciona la implementación para cada uno de los comandos usados en la vista. Cada comando, consta de dos métodos: `Execute` y `CanExecute`. El primero es el que realiza la acción. El segundo, es el que se encarga de confirmar o no si la acción puede ser realizada, antes de ejecutarse. También, durante la ejecución del programa, este se encargará de forma automática y regular de llamar a esas funciones para comprobar su estado.

También, nos suscribimos a todos los eventos de `INotifyPropertyChanged` o `CollectionChanged` que ocurran en los distintos subniveles del modelo: Proyecto, grafica, puntos, etc, para, una vez sean activados, poder llamar a la función `RefreshCanvas()`, encargada de refrescar la representación del canvas.

Posteriormente se definen las siguientes funciones:

- **TrimCanvas:** Función encargada de recortar los puntos de la gráfica los cuales se encuentren fuera del área seleccionada al ejecutarse el evento de soltar el botón izquierdo del ratón. Llamada por eventos.
- **PolyCalc:** Función encargada de llamar al cálculo de los puntos a representar para cada gráfica de tipo polinómica. Llamada por eventos.
- **RefreshCanvas:** Función encargada de refrescar el canvas dibujando los datos almacenados. Llamada por eventos.
- **EditGraphViewModel:** Aloja los comandos propios de su vista e invoca a los *ViewModel* *ManualAddViewModel*, *AutoAddViewModel* y *EditPointViewModel* una vez se ejecuten sus respectivos comandos.
- **ManualAddViewModel:** Aloja los comandos propios de su vista. *DataContext* de *ManualAddWindow*
- **AutoAddViewModel:** Se suscribe a los eventos de cambio en las colecciones relacionadas con Polinomios para mostrar el polinomio en forma de string por pantalla. *DataContext* de *AutoAddWindow*.
- **EditPointViewModel:** Aloja los comandos propios de su vista. *DataContext* de *EditPointWindow*
- **RelayCommand:** Clase que implementa la interfaz *ICommand* la cual sirve como plantilla genérica para todos los comandos de la aplicación. Sin esto, habría que crear una clase nueva por cada comando.

Este concepto, si bien no oficial, es una práctica bastante común en aplicaciones WPF MVVM. Originalmente implementado en la librería *MVVMLight*. Código extraído de [ver ref.]

## Modelo

La aplicación consta de las siguientes clases:

- **CanvasData:** Almacena los datos del Canvas como: *XMin*, *XMax*, *YMin* e *YMax* representadas y los puntos de comienzo de la selección rectangular y final de la selección rectangular.
- **Project:** Colección de objetos *Graph*
- **Graph:** Contiene los atributos de la gráfica (Nombre, tipo, color, etc), una colección de puntos (*Point2D*) y un objeto de tipo expresión polinómica (*PolynomialExpression*)

- **PolynomialExpression:** Alberga una colección de monomios (MonomialMember), el rango de valores de la X (XMin y XMax) y su grado. Este, se auto subscribe a el cambio en sus propiedades, y actúa cuando la propiedad Grade cambia, ajustando los elementos de la colección de monomios en consecuencia
- **MonomialMember:** Alberga el valor del monomio y su grado.
- **Point2D:** Alberga las coordenadas X e Y de cada punto;
- **GraphType y DashType:** Enumeraciones que almacenan los posibles tipos de gráfico y estilo de línea, respectivamente.

## Helpers

A lo largo de la aplicación se han usado distintas clases helper, las cuales no almacenan datos, sino que aportan funcionalidad. Son las siguientes:

- **CanvasTranslator:** Proporciona los métodos estáticos de conversión entre coordenadas reales y coordenadas de pantalla.
- **MathCalc:** Proporciona el método estático encargado de calcular los puntos para un polinomio dado.
- **EnumBindingSourceExtension y EnumDescriptionTypeConverter:** Clases encargadas de proporcionar funcionalidad para que los enums anteriormente citados pueden ser mostrados a través de Bindings en la parte visual de la aplicación.

## Validation

Clases que implementan la interfaz ValidationRule. Estas proporcionan una manera de comprobar que los valores introducidos en campos cuya entrada está directamente vinculada con los datos a través de Bindings cumplen una serie de requisitos. Se implementan los siguientes:

- **StringToDouble:** La cadena puede ser formateada a una variable de tipo Double
- **StringToInt:** La cadena puede ser formateada a una variable de tipo Int
- **StringToHexColor:** La cadena cumple el formato de un color hexadecimal (#XXXXXX)

## Converters

Clases que implementan la interfaz `IValueConverter`. Estas proporcionan una manera de transformar los valores bien a guardar o bien recuperados de la capa de datos mediante Bindings. Se implementan los siguientes:

- **GraphColorConverter:** Convierte un String con formato de color hexadecimal en un objeto de la clase `SolidColorBrush`.
- **GraphTypeConverter:** Convierte el tipo de una gráfica en una imagen correspondiente para su representación en los distintos menús.
- **GraphVisibilityIconConverter:** Convierte la visibilidad de una gráfica en una imagen correspondiente para su representación en los distintos menús.
- **InverseBooleanConverter:** Convierte un valor boolean en su opuesto

## Styles

En la carpeta `Styles` se almacenan los diccionarios de recursos (`ResourceDictionary`) usados en la aplicación para dar una apariencia personalizada a los controles.

Existen algunos genéricos, como `Colors`, `Typography` o `Fonts`, y otros específicos para cada control, como `TextBox` o `ListView`. Todos los diccionarios específicos usan recursos de los genéricos.

Todos los diccionarios son importados en el `App.xaml` usando la etiqueta XAML `<ResourceDictionary.MergedDictionaries>`.


## Images & Fonts

Carpetas donde se almacenan los distintos recursos gráficos. Las fuentes usadas en la aplicación, icono de la aplicación y distintos logos se encuentran almacenados ahí.

# Librerías usadas

En la aplicación se han usado las siguientes librerías:

- **PropertyChanged.Fody (v3.3.1) + Fody (Dependencia) (v6.3.0):**
  - Función: Esta librería trata con la gestión de INotifyPropertyChanged. Permite que las propiedades autoimplementadas automáticamente invoquen al evento PropertyChanged sin necesidad de extender el setter para que este invoque al evento manualmente.
  - Ventajas: Permite ahorrar tiempo a la hora de desarrollar la práctica pero sin restar importancia a la correcta utilización de INotifyPropertyChanged.
  - URL: Nuget: <https://www.nuget.org/packages/PropertyChanged.Fody/>



The image shows two side-by-side code snippets comparing the state of a C# class before and after applying the PropertyChanged.Fody library.

**Left Snippet (Without Library):** Shows a class `Amigo` implementing `INotifyPropertyChanged`. It has a `Nombre` property. The `set` method manually calls `OnPropertyChanged("Nombre")`. The `OnPropertyChanged` method is implemented to call `PropertyChanged` with a new `EventArgs` object.

**Right Snippet (With Library):** Shows the same class `Amigo` after applying the library. The `set` method is simplified, and the `OnPropertyChanged` method is no longer needed. A new class `Person` is shown, which also implements `INotifyPropertyChanged` and uses the library's `InternalEventArgsCache` for automatic property change notifications.

- **Microsoft.Xaml.Behaviors.Wpf (v1.1.31):**
  - Antiguo nombre: System.Windows.Interactivity (Deprecated)
  - Función: Esta librería proporciona marcado extra de XAML para tener la posibilidad de adjuntar comandos a eventos los cuales no los permiten de forma nativa, como el cambio de tamaño de una ventana.
  - Ventajas: Permite ceñirse más al patrón MVVM con el mínimo código en el archivo code-behind
  - URL: Nuget: <https://www.nuget.org/packages/Microsoft.Xaml.Behaviors.Wpf/>

# Referencias

Material Design Dropshadows in WPF:

<http://web.archive.org/web/20170722144651/http://marcangers.com:80/material-design-shadows-in-wpf/>

Recommended WPF icon size:

<https://stackoverflow.com/questions/12901212/taskbar-ugly-icon-in-wpf-application>

ListBox and DataBinding:

<https://www.wpf-tutorial.com/list-controls/listbox-control/>

Button Styling:

<https://docs.microsoft.com/es-es/dotnet/desktop/wpf/controls/button-styles-and-templates>

Menu Styling

<https://docs.microsoft.com/es-es/dotnet/desktop/wpf/controls/menu-styles-and-templates>

TabControl Styling:

<https://docs.microsoft.com/es-es/dotnet/desktop/wpf/controls/tabcontrol-styles-and-templates>

Dynamic TabControl Items:

<https://stackoverflow.com/questions/4058003/wpf-tabcontrol-datatemplates>

WPF/MVVM Commands:

<https://www.wpf-tutorial.com/commands/using-commands/>

Relay Commands:

<https://stackoverflow.com/questions/48527651/full-implementation-of-relay-command-can-it-be-applied-to-all-cases>

IValueConverter:

<https://www.wpf-tutorial.com/data-binding/value-conversion-with-ivalueconverter/>

Bind Enum to ComboBox:

<https://brianlagunas.com/a-better-way-to-data-bind-enums-in-wpf/>

Data Validation with Bindings:

<https://blog.magnusmontin.net/2013/08/26/data-validation-in-wpf/>

Data Validation with Bindigs:

<https://docs.microsoft.com/en-us/dotnet/desktop/wpf/data/how-to-implement-binding-validation>

ListView Delete on Keypress:

<https://stackoverflow.com/questions/9280625/delete-listview-selecteditem-on-keypres>

Commands and UI Events:

<https://stackoverflow.com/questions/4897775/wpf-binding-ui-events-to-commands-in-viewmodel>

(New Microsoft Library):

<https://stackoverflow.com/questions/8360209/how-to-add-system-windows-interactivity-to-project/56240223>

Save Canvas as image:

<https://stackoverflow.com/questions/8881865/saving-a-wpf-canvas-as-an-image>

Save Dialog:

<https://www.wpf-tutorial.com/dialogs/the-savefiledialog/>

