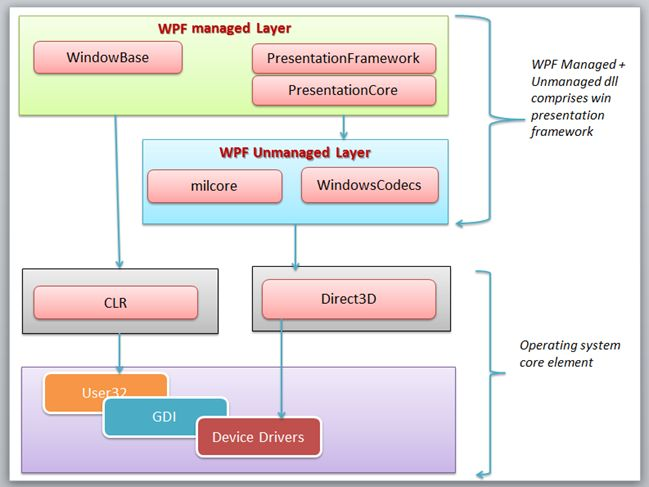
# **WPF**

It is a technology introduced by Microsoft in .NET Framework 3.5.

## **Architecture:**



### **Managed Module:**

**PresentationFramework**.**dll**: This section contains high-level features like application windows, panels, styles controls, layouts, content and so on that helps us to build our application.

**PresentationCore**.**dll**: This is a low-level API exposed by WPF providing features for 2D, 3D, geometry and so on.

**WindowsBase**.**dll**: It holds the more basic elements that are capable to be reused outside the WPF environment like Dispatcher objects and Dependency objects.

### **Un Managed Layer**

This section is unmanaged code because it acts as a bridge between WPF managed and the DirectX / User32 unmanaged API.

**milCore**.**dll**: It is called the Media Integration Layer (MIL) and resides in milCore.dll. The purpose of the milCore is to interface directly with DirectX and provide basic support for 2D and 3D surface.

**WindowsCodecs**.**dll**: WindowsCodecs support in WPF applications like image processing, image displaying and scaling and so on.

**Core operating System Layer (Kernel)**  
  
This layer has OS core components like User32, GDI, Device Drivers, Graphic cards and so on. These components are used by the application to access low-level APIs.

**DirectX**: DirectX is the low-level API through which WPF renders all graphics. DirectX talks with drivers and renders the content.

**User32**: User32 actually manages memory and process separation. It is the primary core API that every application uses. User32 decides which element will be placed where on the screen.

**GDI**: GDI stands for Graphic Device Interface. GDI provides an expanded set of graphics primitives and a number of improvements in rendering quality.

**CLR**: WPF leverages the full .NET Framework and executes on the Common Language Runtime (CLR).

**Device** **Drivers**: Device Drivers are used by the applications to access low-level APIs.

## **Why do we use WPF?**



1. Vector based rendering and resolution Independence.

WPF uses vector graphics instead of bitmaps. Due to this we get the feature of resolution independence.

Question 1: My question here is how WPF manages the resolution independent property?

The basic unit of measurement in the WPF graphics system is the device independent pixel, which is 1/96th of an inch, regardless of actual screen resolution, and provides the foundation for resolution-independent and device-independent rendering. Each device-independent pixel automatically scales to match the dots-per-inch (dpi) setting of the system it renders on.

In WPF we Implement resolution Independency by understanding the layout control. There is mainly 2 keywords in it.

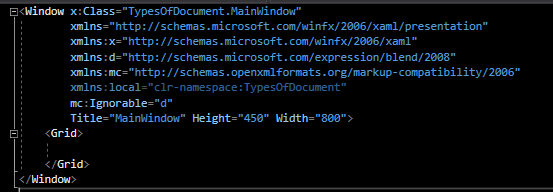
1. Auto : Takes only control space.

2. \* : Takes all the available space.

1. Hardware acceleration

WPF uses DirectX to obtain GPU accelerated rendering. This GPU is the chip that powers your video card. So, By using hardware on your video card that is specifically designed for rendering graphics instead of software functions run by the CPU, the performance of your application's graphics and user interface is increased, or accelerated.

1. Declarative UI with XAML



We will learn more about XAML in later part of this ebook.

1. Multimedia support
2. Types of Document supported in WPF

There are 2 types of documents supported in WPF:

1. Fixed Document – Static ready to print document.
2. Flow Document – Dynamic and formatting can be done.

