Introducción a GDI+, el motor gráfico de .NET Framework (Un nuevo modelo de gestión gráfica para una nueva generación de aplicaciones)

Luis Miguel Blanco Ancos

Si cualquiera de nuestros lectores ha tenido que generar formas gráficas en alguna ocasión utilizando el clásico API de Windows, habrá sufrido sus complejidades en la creación y gestión de figuras e imágenes.

Cierto es también que a todo se acostumbra uno, y tras unas primeras “peleas” con las funciones del API, progresivamente nos adaptaremos a este modo de trabajo.

Sin embargo, y como a nadie amarga un dulce, la llegada de la tecnología .NET ha venido acompañada de GDI+ (Graphics Device Interface), que como su nombre deja entrever, se trata de la nueva generación del API gráfico de Windows, pero esta vez adaptado a la plataforma .NET Framework.

GDI+ aúna potencia y sencillez, cualidades apreciadas por todo programador, gracias a una estupenda jerarquía de clases, bien organizada en un conjunto de espacios de nombres, que facilita al desarrollador su trabajo a la hora de localizar y utilizar los tipos de la plataforma que deben intervenir en las operaciones gráficas de sus aplicaciones.

Todo cambio en el sentido en que estamos hablando, lleva implícito una nueva forma de hacer las cosas; en lo que respecta a los gráficos bajo .NET en general, y en cómo afecta al programador de Visual Basic en particular, se traduce en que ciertos elementos de los formularios, como métodos y controles, que en versiones anteriores de este lenguaje utilizábamos para crear determinadas formas gráficas, en VB.NET han desaparecido. Este hecho, sin embargo, no constituye una desventaja sino un gran beneficio, expliquémonos:

En VB6, cuando creábamos una figura en un formulario con el control Shape, disponíamos de un número limitado de figuras para generar, e igualmente de efectos para aplicar a dicha figura. Además, tampoco era factible utilizar este control al imprimir, ya que está ligado al formulario, y es sólo en este tipo de objeto en el que podemos utilizarlo.

Con GDI+ podemos generar una figura sobre un formulario y la impresora, por poner dos ejemplos de dispositivos diferentes, utilizando básicamente el mismo conjunto de clases. Ello se debe a que GDI+ es un interfaz de programación independiente del dispositivo sobre el que se crea el gráfico, por lo que resuelve internamente las complejidades del dispositivo sobre el que vayamos a generar nuestro gráfico, proporcionando al programador un modo transparente de trabajo.

De todo lo anterior se deduce, que lo que aprendamos sobre GDI+ para un lenguaje y contexto de trabajo determinado, lo podremos aplicar en otro contexto diferente y también distinto lenguaje, siempre y cuando el lenguaje a utilizar pertenezca a la plataforma .NET, ya que GDI+ es una jerarquía de clases de .NET Framework, por lo que no está ligado a ningún lenguaje concreto, sino al entorno de ejecución de .NET.

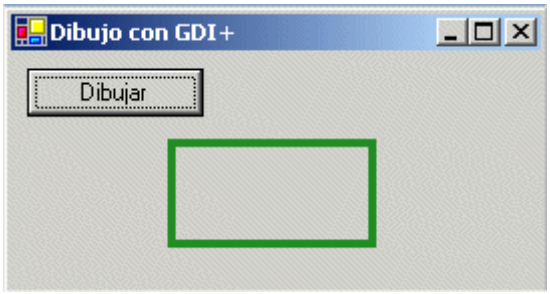
Vamos a dibujar

Y ya sin más preámbulos, pasemos directamente a crear nuestro primer dibujo utilizando GDI+.

Para comenzar, escribiremos el siguiente código fuente, que dibujará un rectángulo en el formulario. A continuación comentaremos los principales aspectos de este código.

```
Imports System.Drawing
' ....
' ....
' evento Click de un botón:

' coordenadas
Dim pntUbicacion As New Point(80, 45)

' tamaño
Dim szMedidas As New Size(100, 50)

' zona para el dibujo
Dim recArea As New Rectangle(pntUbicacion, szMedidas)

' color de la figura
Dim oColor As Color = Color.ForestGreen

' pincel
Dim penPincel As New Pen(oColor, 4)

' superficie del formulario para dibujar
Dim gphSuperfDibujo As Graphics = Me.CreateGraphics()

' dibujar un rectángulo
gphSuperfDibujo.DrawRectangle(penPincel, recArea)

' liberar recursos
gphSuperfDibujo.Dispose()
```

Si ponemos este código en un botón del formulario, el resultado tras su ejecución sería similar a la siguiente imagen.



Diseccionando el dibujo pieza a pieza

Para aquellos lectores que provengan de lenguajes OOP, o bien de VB6, pero que de este último se hayan acostumbrado progresivamente a utilizar las técnicas de codificación orientadas a objeto del lenguaje, la interpretación del anterior fuente creo que será relativamente sencilla para ellos.

En cualquier caso, y para que todo quede claro punto por punto, vamos a ir describiendo el proceso que hemos llevado a cabo, hasta la consecución de nuestro rectángulo en el formulario.

El espacio de nombres para las clases gráficas

En el comienzo o cabecera del editor de código debemos importar el espacio de nombres System.Drawing, el cual alberga las clases, enumeraciones, y demás tipos de la plataforma que necesitaremos para la manipulación de gráficos.

La ubicación del gráfico

En primer paso que vamos a dar para crear nuestra figura en el formulario, consiste en indicar las coordenadas en que dicha figura será mostrada.

Esto lo conseguiremos mediante la clase Point, a la que deberemos pasar en su constructor los valores que representan los puntos x e y en el plano de dibujo del formulario.

```
Dim pntUbicacion As New Point(80, 45)
```

El tamaño del gráfico

Al igual que establecemos una posición, también debemos hacer lo propio respecto a las dimensiones de la figura. Para ello nos valdremos de la clase Size, pasando a su constructor los valores que representan la anchura y altura respectivamente.

```
Dim szMedidas As New Size(100, 50)
```

El área de dibujo

Para dibujar una figura en un formulario, primeramente debemos delimitar un área en la superficie del formulario, dentro de la cual realizaremos nuestro dibujo.

Con figuras simples, esta región consiste en una zona rectangular dentro de nuestro formulario, que definiremos mediante un objeto de la clase Rectangle, pasando a su constructor los objetos Point y Size creados anteriormente, que servirán para decirle al rectángulo cuál es su posición y tamaño.

```
Dim recArea As New Rectangle(pntUbicacion, szMedidas)
```

El pincel y la paleta de colores

Ya que nos hemos convertido en artistas digitales, en lugar del pincel y la paleta tradicionales, en nuestro caso utilizaremos clases como material de dibujo (una de las ventajas es que no nos mancharemos de pintura).

Para la paleta de colores disponemos de la estructura Color, que define en su lista de miembros los colores que podemos utilizar al dibujar una figura, como vemos en la siguiente imagen.

En cuanto al pincel, se encuentra representado por la clase Pen, a cuyo constructor tenemos que pasarle un tipo Color, y un valor numérico que establece el grosor del trazo con el que vamos a dibujar.

```
Dim penPincel As New Pen(oColor, 4)
```

El lienzo. La unión de todos los materiales

Llegados a este punto tenemos todo lo preciso para dar rienda suelta a nuestra actividad ciberartística. ¿Qué nos falta?. Pues de igual modo que un pintor clásico necesita un lienzo sobre el que pintar, nosotros también precisamos este elemento clave en todo proceso de dibujo.

En nuestro caso, el lienzo será la superficie del formulario. Sin embargo, no podemos tomar directamente el formulario y empezar a pintar en él. “¿Y por qué no?”, nos

preguntaremos. Pues porque en primer lugar tenemos que seleccionar sobre qué tipo de lienzo queremos pintar.

Para ayudar a comprender mejor este concepto, volvamos al ejemplo del pintor tradicional. Supongamos que nuestro pintor ya tiene todo lo necesario en cuanto a color, pincel, motivo del dibujo, etc. A partir de aquí, puede pintar en un lienzo de tela, una pared (si es un artista del graffiti), una valla publicitaria, etc. Como vemos, hay muchas superficies que pueden representar la base o lienzo de su dibujo.

En nuestro caso ocurre algo similar, con los mismo materiales, nuestra superficie de dibujo puede ser un formulario, la impresora, etc., por lo que en primer lugar, tenemos que seleccionar el tipo de superficie, y luego pintar sobre ella.

La superficie sobre la que dibujaremos, recibe en Windows el nombre técnico de "contexto de dispositivo gráfico", y en la plataforma .NET, está representada por la clase Graphics, por lo que necesitamos un objeto de este tipo para poder crear nuestro gráfico. Más en concreto, lo que precisamos es el objeto Graphics que representa la superficie del formulario, y que obtendremos mediante el método Form.CreateGraphics.

```
Dim gphSuperfDibujo As Graphics = Me.CreateGraphics()
```

Una vez obtenido este objeto, a través de los métodos que comienzan por el nombre Draw, indicaremos qué figura queremos pintar. La siguiente tabla muestra alguno de los métodos disponibles.

Método	**Tipo de figura**
DrawRectangle	Rectángulo
DrawEllipse	Elipse
DrawCurve	Curva
DrawLine	Linea
DrawPolygon	Polígono
DrawString	Cadena de texto en modo gráfico

Para una referencia más extensa, consulte el lector la documentación de .NET Framework.

En el ejemplo anterior hemos dibujado un rectángulo mediante el método Graphics.DrawRectangle, pasando como parámetros los objetos Pen y Rectangle, que utilizará el objeto Graphics como tipo de pincel y área de dibujo respectivamente.

```
gphSuperfDibujo.DrawRectangle(penPincel, recArea)
```

Las operaciones gráficas en Windows hacen un consumo intensivo de recursos del sistema, por lo que es altamente recomendable que una vez finalizada la creación de nuestra figura, liberemos los recursos que están siendo utilizados llamando al método Graphics.Dispose.

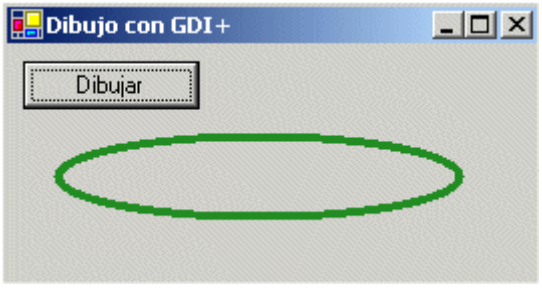
Simplificando – resumiendo la operación de dibujo

En el ejemplo anterior, hemos creado por separado cada uno de los objetos integrantes de nuestra operación de dibujo. Si bien esta sería la mejor técnica de trabajo a efectos didácticos, una aproximación mucho más cercana a una codificación real, sería la instanciación en línea de algunos objetos en el momento de ser pasados como parámetros a los métodos de otros objetos. Con ello ahorraremos líneas de código, consiguiendo el mismo resultado y rendimiento.

Para apreciar esta diferencia, el siguiente código fuente muestra la misma operación de dibujo que el ejemplo anterior, pero aplicando esta técnica. El único cambio en este caso, reside en que vamos a dibujar una elipse en lugar de un rectángulo.

```
' superficie del formulario para dibujar
Dim gphSuperfDibujo As Graphics = Me.CreateGraphics()

' dibujar un rectángulo
gphSuperfDibujo.DrawEllipse(New Pen(Color.ForestGreen, 4), _
   New Rectangle(New Point(25, 45), New Size(200, 40)))

' liberar recursos
gphSuperfDibujo.Dispose()
```

El resultado lo apreciamos en la siguiente imagen.



La clase Rectangle dispone de un constructor en el que podemos pasar directamente los valores numéricos correspondientes a la posición y tamaño, sin necesidad de crear un objeto Point ni Size, como muestra la siguiente línea de código.

```
New Rectangle(25, 45, 200, 40)
```

Respecto al color, aunque el medio más sencillo de seleccionarlo es eligiendo uno de los nombres de color de la estructura, también podemos usar el método FromArgb, en el que pasamos los valores numéricos correspondientes a los parámetros alpha (transparencia), rojo, verde y azul respectivamente; o también el método FromName, en el que pasamos como parámetro una cadena con el nombre del color a usar, como vemos en los siguientes ejemplos.

```
Dim oColor As Color
oColor = Color.FromArgb(150, 60, 120, 170)
oColor = Color.FromName("Brown")
```

Estilos de trazo

Por defecto, al dibujar con un objeto Pen, se utiliza una línea continua para trazar la figura. Sin embargo, mediante la propiedad DashStyle, podemos variar el estilo de línea, consiguiendo que esté compuesta por guiones, puntos, etc.

Las propiedades StartCap y EndCap, por otro lado, permiten que el comienzo y final de línea sea diferente de una línea estándar, pudiendo aplicar un efecto de recuadro, redondo, romboide, etc. Como comprobará el lector si escribe este código de ejemplo, es muy sencillo asignar estas propiedades, ya que al tratarse de enumeraciones, el editor de código, gracias al Intellisense, nos muestra la lista con los valores admisibles.

En el siguiente fuente dibujamos una línea con algunas de las variaciones que hemos comentado.

```
' crear pincel y aplicarle estilo de línea
Dim penPincel As New Pen(Color.DarkCyan, 7)
penPincel.DashStyle = Drawing.Drawing2D.DashStyle.Dash
penPincel.StartCap = Drawing.Drawing2D.LineCap.DiamondAnchor
penPincel.EndCap = Drawing.Drawing2D.LineCap.RoundAnchor

' obtener superficie del formulario y dibujar una línea
Dim gphSuperfDibujo As Graphics = Me.CreateGraphics()
gphSuperfDibujo.DrawLine(penPincel, 25, 50, 200, 50)
gphSuperfDibujo.Dispose()
```

El resultado lo vemos a continuación.

Polígonos y curvas

Para dibujar una figura de este tipo, al ser más compleja de generar, no podemos especificar un rectángulo como área de dibujo, sino que tenemos que utilizar un conjunto de objetos Point contenidos en un array, que usaremos para establecer las coordenadas por donde pasará el trazo del pincel al dibujar la figura. A continuación se muestra un ejemplo de polígono.

```
' array de objetos Point con las coordenadas del polígono
```

```
Dim pntPuntos(4) As Point
pntPuntos(0) = New Point(25, 70)
pntPuntos(1) = New Point(50, 40)
pntPuntos(2) = New Point(75, 70)
pntPuntos(3) = New Point(60, 90)
pntPuntos(4) = New Point(40, 90)

Dim gphSuperfDibujo As Graphics = Me.CreateGraphics()
gphSuperfDibujo.DrawPolygon(New Pen(Color.DarkMagenta, 3), _
   pntPuntos)

gphSuperfDibujo.Dispose()
```

En la siguiente imagen vemos el gráfico generado.

Para crear una curva, el proceso es igual, pero empleando el método Graphics.DrawCurve, como vemos en el siguiente código.

```
Dim oGraphics As Graphics = Me.CreateGraphics()

oGraphics.DrawCurve(New Pen(Color.Brown, 4), _
   New Point() {New Point(20, 75), New Point(50, 15), New Point(100, 60)})

oGraphics.Dispose()
```

La siguiente imagen muestra la curva obtenida.

Borrado de los elementos gráficos

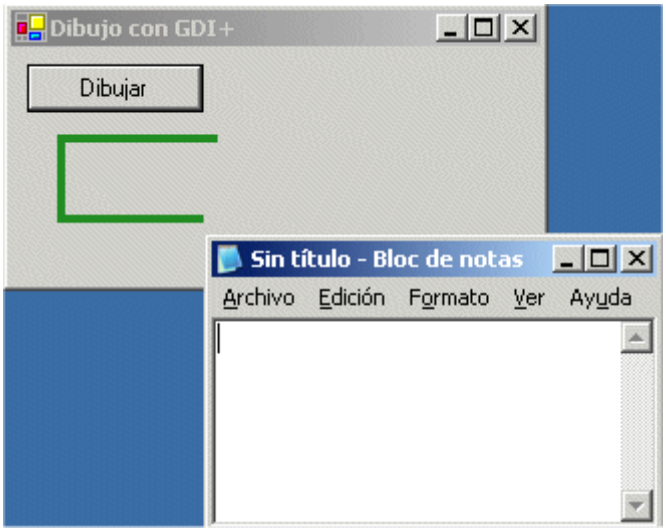
Después de haber realizado varias pruebas, a buen seguro que nuestro formulario está repleto de figuras, por lo que posiblemente nos interesará en un momento dado, limpiar su superficie sin tener que finalizar y ejecutar nuevamente el proyecto. Para

ello, lo que debemos hacer es llamar al método Form.Invalidate, que borra las figuras que hayamos dibujado en la ventana.

Me.Invalidate()

El evento Paint. Refrescando zonas ocultas del formulario

Cuando el total o parte del gráfico que hemos dibujado en el formulario queda tapado por otra ventana, el área oculta se dice que ha quedado invalidada. Esto significa que la región del gráfico que se ha tapado no es mantenida en memoria por el formulario para una posterior visualización, de manera que cuando volvamos a mostrar dicha zona de la ventana, la porción del gráfico habrá sido borrada. La siguiente imagen muestra esta situación.



Este comportamiento permite al sistema operativo evitar un excesivo consumo en recursos gráficos, pero deja en manos del programador la responsabilidad de ocuparse de que tales elementos se muestren en el formulario.

Lo que debemos hacer ante esta circunstancia, es un refresco o repintado del gráfico en el formulario. Seguidamente explicamos cómo llevar a cabo esta operación.

Cuando una región del formulario que había estado oculta vuelve a mostrarse, se produce en ese formulario el evento Paint, de esta forma sabemos que el objeto está solicitando que se vuelvan a dibujar aquellos elementos gráficos que podrían haber quedado borrados.

En nuestro ejemplo por lo tanto, vamos a escribir el procedimiento manipulador para este evento, y trasladaremos al mismo el código que teníamos en el botón del formulario, como vemos en el siguiente código fuente.

Private Sub Form1_Paint(ByVal sender As Object, ByVal e As System.Windows.Forms.PaintEventArgs) Handles MyBase.Paint

```
    Dim gphSuperfDibujo As Graphics = Me.CreateGraphics()

    gphSuperfDibujo.DrawRectangle(New Pen(Color.ForestGreen, 4), _
       New Rectangle(New Point(25, 45), New Size(200, 40)))

    gphSuperfDibujo.Dispose()

End Sub
```

Una vez escrito este código, volveremos a ejecutar el programa, comprobando esta vez cómo el gráfico del formulario se mantiene gracias a que se vuelve a pintar cada vez que una de sus zonas queda oculta, por muy pequeña que sea.

Como consecuencia añadida, ya no necesitaremos el botón del formulario para efectuar el dibujo, puesto que gracias al evento Paint, el gráfico será constantemente pintado.

Y terminamos

Bien, pues esto ha sido todo, en este artículo hemos hecho una pequeña introducción a GDI+, el API de programación gráfica que acompaña a la plataforma .NET Framework, describiendo algunas de sus principales características. También hemos descrito los principales objetos implicados en una operación de dibujo, y efectuado unas pruebas demostrativas de la potencia y facilidad de programación de este conjunto de clases. Invitamos al lector a que como ejercicio, experimente con otros métodos de los objetos comentados, para que vaya adquiriendo soltura en la gestión de gráficos con .NET. Un saludo a todos.