Ediciones ENI

# C# 7

## Desarrolle aplicaciones Windows con Visual Studio 2017

Colección Expert IT

Extracto del Libro



# Capítulo 11
# Creación de controles de usuario

## 1. Introducción

El desarrollo de aplicaciones se basa principalmente en los controles, que proporcionan las distintas funcionalidades en forma visual y permiten al usuario interactuar con ellos. Todos estos controles provienen, de una manera más o menos directa, de la clase base `System.Windows.Forms.Control`. Visual Studio ofrece la integración de controles de terceros, añadiéndolos a la caja de herramientas. Pero si la necesidad es muy específica, es posible crear sus propios controles.

La clase base de los controles, `Control`, proporciona las funcionalidades básicas que son necesarias, fundamentalmente para la entrada de datos del usuario a través del teclado y el ratón. Esto implica propiedades, métodos y eventos comunes a todos los controles. Sin embargo, esta clase base no proporciona la lógica de visualización del control.

Existen tres modos de creación del control:

– Los controles personalizados.
– La herencia de controles.
– Los controles de usuario.

La creación de controles sigue el principio de reutilización del código. La lógica se crea en un único sitio y se puede usar varias veces. La ventaja es muy importante en términos de mantenimiento de la aplicación, ya que para cambiar el comportamiento de este control, solo será necesario modificar un archivo.

## 2. Los controles personalizados

Estos controles ofrecen las mayores posibilidades de personalización, tanto a nivel gráfico como lógico. Un control personalizado hereda directamente de la clase `Control`. Por lo tanto, es necesario escribir toda la lógica de visualización lo que, según el resultado esperado, puede resultar una fase muy larga y complicada. El desarrollador también debe definir los métodos, propiedades y eventos.

La clase base `Control` expone el evento `Paint`. Es el que se produce cuando se genera el control e implica la ejecución del controlador del evento por defecto `OnPaint`. Este método recibe un único argumento de tipo `PaintEventArgs`, que contiene la información necesaria en la superficie de dibujo del control. El tipo `PaintEventArgs` tiene dos propiedades, `Graphics`, del tipo `System.Drawing.Graphics` y `ClipRectangle`, del tipo `System.Drawing.Rectangle`. Para añadir la lógica de diseño al control hay que sobrecargar el método `OnPaint` y añadirle el código de diseño:

```
protected override void OnPaint
                        (System.Windows.Forms.PaintEventArgs e)
{
    // Código de diseño del control
}
```

La propiedad `Graphics` del objeto `PaintEventArgs` representa la superficie del control, mientras que la propiedad `ClipRectangle` representa su área de dibujo. Durante la primera representación del control, la propiedad `ClipRectangle` representa los límites del control. Estos límites se pueden modificar, por ejemplo, si un control por debajo oculta una parte, de tal manera que sea necesario volver a dibujar el control. La parte `ClipRectangle` representara el área a modificar.

Cree una carpeta **Controles** en la raíz del proyecto y añada una nueva clase llamada **CustomControl** definida de la siguiente manera:

```
using System.Drawing;

namespace SelfMailer.Controls
{
    public class CustomControl: System.Windows.Forms.Control
    {
        protected override void OnPaint
                        (System.Windows.Forms.PaintEventArgs e)
        {
            Rectangle R = new Rectangle(0, 0,
                              this.Size.Width, this.Size.Height);
            e.Graphics.FillRectangle(Brushes.Green, R);
        }
    }
}
```

En el constructor del formulario **MailServerSettings**, añada el código de instanciación del control personalizado:

```
Controls.CustomControl C = new Controls.CustomControl();
C.Localization = new System.Drawing.Point(0, 0);
C.Size = this.Size;
this.Controls.Add(C);
```

Este código instancia un nuevo control de tipo `CustomControl`, lo ubica en la parte superior izquierda y ajusta su tamaño al del formulario. Para terminar, el control se añade a la colección de controles del formulario.

Ejecute la aplicación ([F5]) y abra el formulario de los argumentos del servidor de mail para ver que el control, que representa un sencillo rectángulo verde, rellena el formulario con un color de fondo.

**Observación**

*Las posibilidades de dibujo con GDI+ se abordarán más adelante en este libro, en la sección El diseño con GDI+, del capítulo Para llegar más lejos.*

Para terminar, basta con añadir los miembros necesarios para la lógica del control.

## 3. La herencia de controles

Si el objetivo es extender las funcionalidades de un control existente, ya sea un control del Framework .NET o de un editor de terceros, la manera más rápida es heredar de este control. El nuevo control tiene de esta manera todos los miembros y la representación visual de su clase padre. Solo hay que añadir la lógica de procesamiento. De la misma manera que los controles personalizados, es posible sobrecargar el método `OnPaint` para modificar el aspecto visual del control.

Si una aplicación tiene varios formularios que necesitan un email como campo de entrada, es mejor crear un control heredando de la clase `TextBox`, implementar la lógica de validación y después añadir este control a los formularios de manera que no sea necesario repetir el código de validación de cada uno de ellos.

La creación de un control heredado se hace de la misma manera que un control personalizado, creando una clase que va a heredar del control que tiene el comportamiento base requerido. Cree la clase `EmailTextBox` en la carpeta **Controles** y herede de la clase `TextBox`:

```
public class EmailTextBox: System.Windows.Forms.TextBox
{
}
```

Añada una sobrecarga del método `OnValidating` para realizar las comprobaciones de formato y añada el código del método `FromEmail_Validating` del formulario **MailServerSettings**:

```
protected override void
          OnValidating(System.ComponentModel.CancelEventArgs e)
{
    base.OnValidating(e);
    string pattern = @"^([a-zA-Z0-9_\-\.]+)@((\[[0-9]{1,3}\." +
                     @"[0-9]{1,3}\.[0-9]{1,3}\.)|" +
                     @"(([a-zA-Z0-9\-]+\.)+))" +
                     @"([a-zA-Z]{2,4}|[0-9]{1,3})(\]?)$";
    Regex reg = new Regex(pattern);
    if (!reg.IsMatch(this.Text))
    {
         this.BackColor = Color.Bisque;
         e.Cancel = true;
    }
```

```
        else
            this.BackColor = this.PreviousBackColor;
    }
```

Es preciso hacer cambios puesto que ya no se accede a la propiedad `Text` del control a partir del formulario. Por tanto, hay que sustituir:

```
this.FromEmail.Text
```

por un acceso directo:

```
this.Text
```

La primera instrucción:

```
base.OnValidating(e);
```

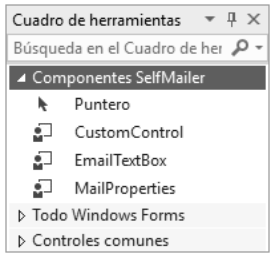
permite llamar al método de validación de la clase base. De esta manera, existe una primera validación en la clase base y después hay una validación del control por parte del código adicional.

El último aspecto destacable es que el componente **ErrorProvider** ya no está al mismo nivel. Para indicar al usuario que el campo es inválido, en lugar de mostrar un icono, se modifica el color de fondo del control. La propiedad `PreviousBackColor`, que ha sido en principio inicializada con el color de fondo de base en el constructor, permite conservarlo para reasignarlo al control en caso de que la validación tenga éxito:

```
protected Color PreviousBackColor { get; set; }

public EmailTextBox()
{
    this.PreviousBackColor = this.BackColor;
}
```

El controlador del evento `FromEmail_Validating` es, en este caso, irrelevante, ya que la validación se realiza dentro del control. Además, puede eliminar el control de tipo `TextBox` para la entrada de datos del email de la persona que envía el correo electrónico y sustituirlo por un control de tipo `EmailTextBox`, que Visual Studio ha añadido en la caja de herramientas dentro del grupo **Componentes SelfMailer** (**SelfMailer** representa el nombre del proyecto).

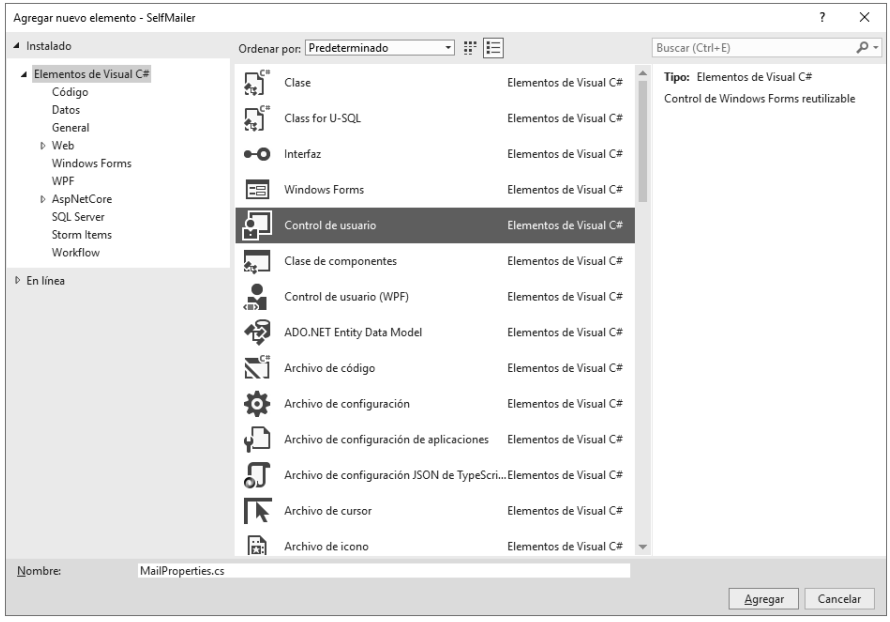



Ejecute la aplicación ([F5]) para probar la funcionalidad.

## 4. Los controles de usuario

El objetivo de un control de usuario es agrupar de manera lógica los controles para obtener una entidad reutilizable. La creación se hace añadiendo al proyecto un **Control de usuario** desde la ventana de adición de un nuevo elemento.

Añada un control de usuario llamado **MailProperties** en la carpeta **Elementos de Visual C#** del proyecto:



Ediciones ENI

# ASP.NET en C# con Visual Studio 2017

## Diseño y desarrollo de aplicaciones Web

Colección Expert IT

Extracto del Libro



# Capítulo 4
# Los sitios web MVC

## 1. El enfoque MVC

Una vez pasada la época en la que se discutía la estructura de una aplicación web, el universo Java ha popularizado el uso de frameworks tales como Struts o Spring. Éste, y Struts en primer lugar, han sentado las bases de una separación de responsabilidades entre los distintos niveles de una aplicación web. Es cierto que las primeras tecnologías web no invitaban a los programadores a organizar sus aplicaciones; el mantenimiento se vuelve muy delicado, al tiempo que el rendimiento es ridículo.

### 1.1 El patrón de diseño MVC

La expresión MVC se refiere a un enfoque de diseño generalizado, o patrón de diseño. El objetivo consiste en no reinventar la rueda con cada aplicación. Como veremos, el MVC es un patrón bastante simple. No utilizarlo supone, realmente, dirigirse hacia una aplicación complicada y, por tanto, mal hecha, lo que nos recuerda al pasado tal y como veíamos antes.

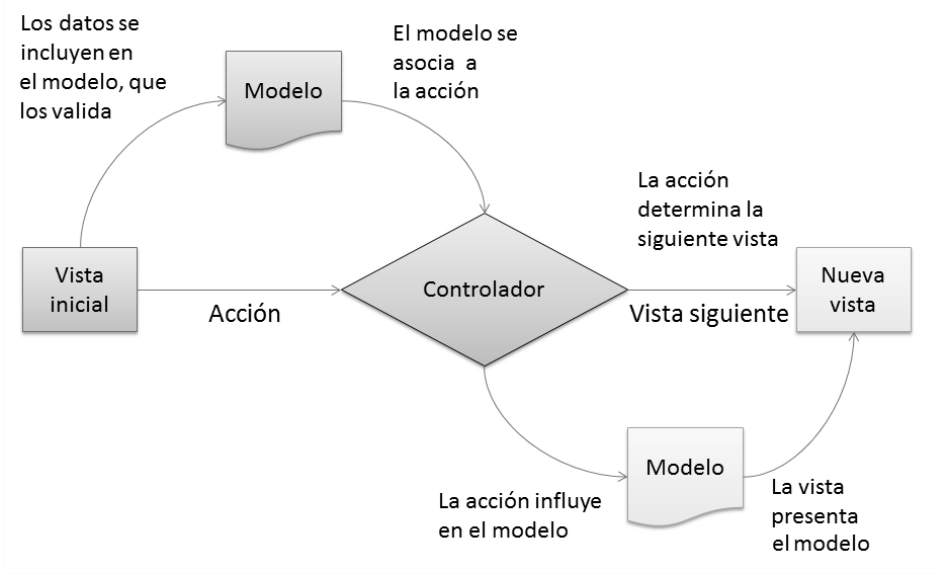
Cada letra del acrónimo MVC se corresponde con un rol bien definido; el modelo, la vista y el controlador.

El modelo es un objeto "de negocio" que agrupa sus datos, su comportamiento (métodos) y sus reglas de validación. No contiene, por lo general, ninguna lógica técnica (presentación, navegación). Es posible atribuirle aspectos (inyección de servicios tales como persistencia de archivos o SQL, transacciones, seguridad...). En los enfoques menos completos, el objeto de negocio se asocia con una clase de servicios que sirve de interfaz (API).

La vista se encarga de restituir el modelo en el seno de una interfaz gráfica (web, en nuestro caso), y permite al usuario interactuar con el modelo.

El controlador define las reglas de navegación (también llamada la cinemática). El paso de una vista a otra se realiza mediante acciones dirigidas por un controlador. El modelo se interroga, o enriquece, para condicionar el desarrollo de acciones.

La siguiente ilustración describe la secuencia de interacciones entre estos objetos:



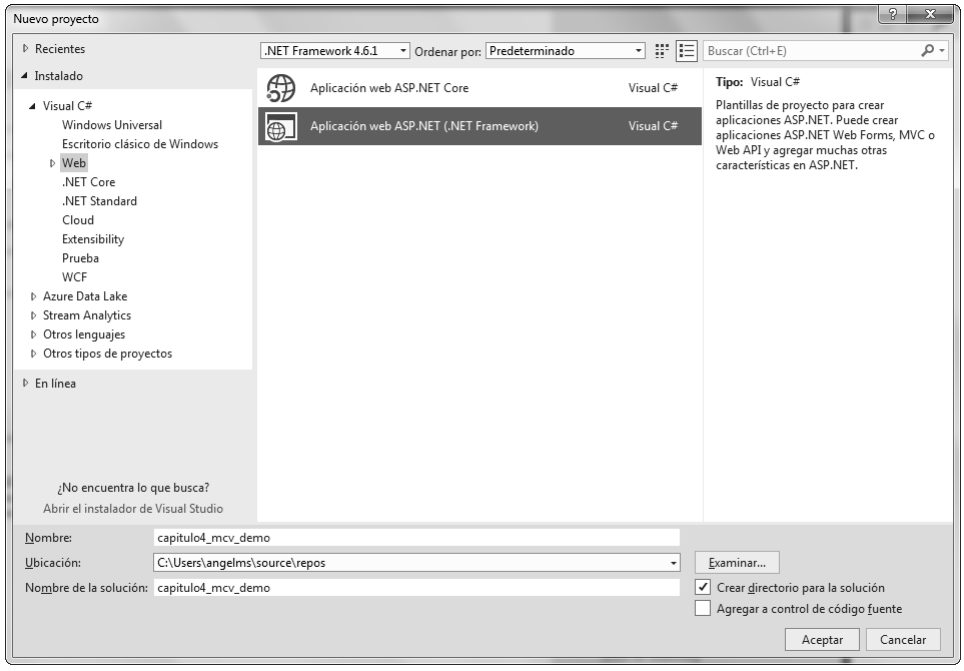
## 1.2 Evolución de MVC

El enfoque MVC 2 es, principalmente, una evolución del framework; consiste en utilizar únicamente un controlador para varias acciones. Esta evolución reduce considerablemente el esfuerzo en cuanto a programación y a configuración. Por suerte, el framework ASP.NET soporta, en lo sucesivo, el nivel MVC 2 mediante ASP.NET MVC 3/4/5.

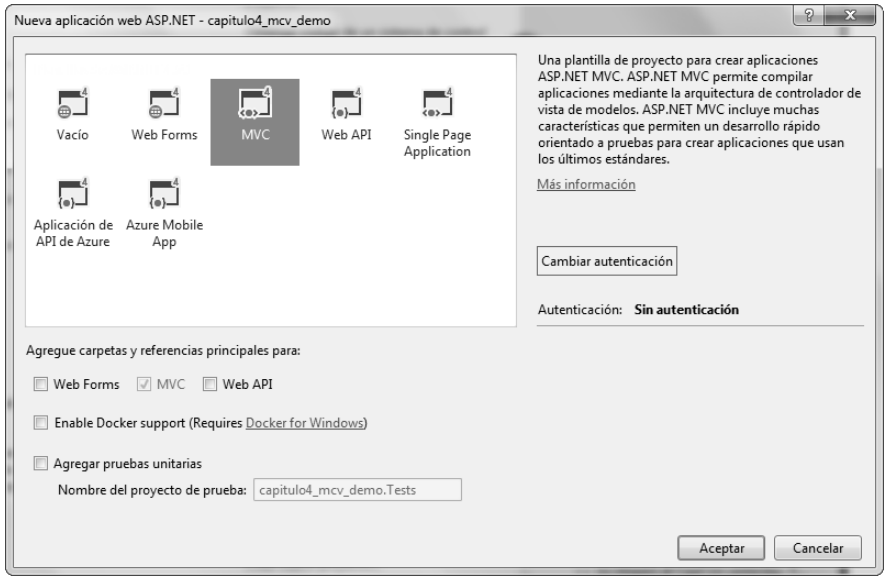
# 2. Los sitios ASP.NET MVC

## 2.1 Creación de un sitio

La creación de un sitio web MVC se realiza mediante la opción **Nuevo proyecto**:



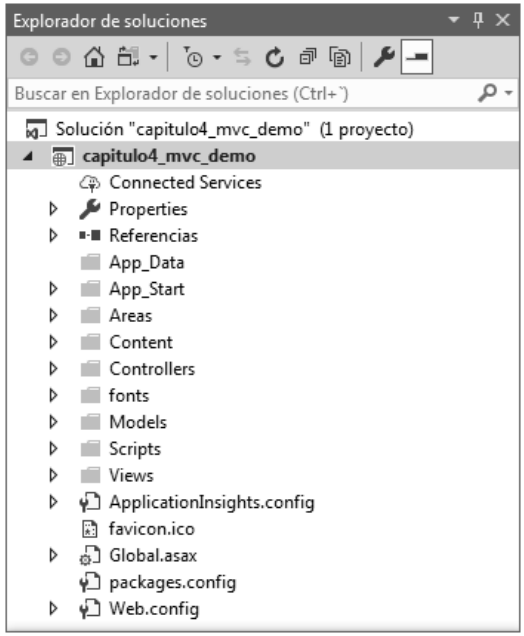
A continuación, seleccionaremos la plantilla `MVC`:




Como la aplicación MVC requiere el uso de clases que no están en el código subyacente (como con los Web Forms), la plantilla Visual Studio está disponible únicamente para un proyecto web, y no para un sitio web.

## 2.2 Organización de carpetas

La solución del proyecto web contiene muchas más carpetas que un proyecto Web Forms.




Estas carpetas tienen, como objetivo, guiar al programador:

| | |
|---|---|
| App_Start | Instrucciones de configuración que se ejecutan durante el arranque del sitio. |
| Content | Contiene las hojas de estilo CSS y demás recursos compartidos. |
| Controllers | Agrupa los controladores destinados a procesar las acciones. |
| fonts | Fuentes de tipos de letra que se descargará el navegador. |
| Models | Agrupa los modelos que son entidades de negocio. |
| Scripts | Conjunto de módulos JavaScript, jQuery y AJAX. |
| Views | Vista .cshtml. |

Las vistas son páginas web, aunque no tienen código subyacente (como veremos a continuación). Se agrupan, en principio, en carpetas llamadas zonas, las cuales se corresponden con controladores. Esta regla no tiene ningún carácter obligatorio desde un punto de vista técnico, aunque es más sencillo utilizar el framework si nos ceñimos a ella.