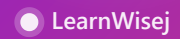


Reglas de validación integradas y el extensor Validation

Guía de la lección · Guía del laboratorio / examen



Guía de la lección

El extensor Validation de Wisej.NET: la propiedad extendida ValidationRules en tiempo de diseño, SetValidationRules en tiempo de ejecución, las reglas Required, Email, Integer, Decimal, Currency, Telephone y Regex, un orden de reglas que se detiene en el primer fallo y los destinos de visualización ErrorProvider o Label. Qué significa esto para un desarrollador que conoce C# básico y empieza a desarrollar formularios con Wisej.NET, y cómo leer este módulo.



Enfoque de producción

El extensor Validation de Wisej.NET: la propiedad extendida ValidationRules en tiempo de diseño, SetValidationRules en tiempo de ejecución, las reglas Required, Email, Integer, Decimal, Currency, Telephone y Regex, un orden de reglas que se detiene en el primer fallo y los destinos de visualización ErrorProvider o Label.

1. El problema en producción

Al final del Módulo 2 el formulario de alta de ValidationClinic funciona, pero fíjate en cuánto código ha hecho falta: un método auxiliar para el nombre, otro para el correo electrónico, otro para el teléfono, y cada uno repite las mismas comprobaciones de «¿está vacío?, ¿tiene la forma correcta?». Añade la edad, el límite de crédito y un código de cliente, y el formulario crece con otro puñado de métodos casi idénticos. Las comprobaciones no están mal; simplemente se vuelven a escribir en cada campo y en cada formulario. Este módulo sustituye esa repetición por el extensor **Validation**, con el que describes cómo es un campo válido y Wisej.NET ejecuta las comprobaciones por ti.

2. Conceptos clave

Concepto	Por qué importa
extensor <code>Validation</code>	Un componente no visual que añades a un formulario; se engancha a los eventos <code>Validating</code> y <code>Validated</code> de cada control que tiene reglas, así que ya no escribes un controlador por campo.
propiedad extendida	Una propiedad que un componente añade a otros controles; en cuanto el extensor está en el formulario, cada control compatible muestra una entrada <code>ValidationRules</code> en la ventana Propiedades.
<code>SetValidationRules</code>	El equivalente en código del diseñador: <code>validation.SetValidationRules(txtEmail, new ValidationRule[] { ... })</code> asigna a un control su lista de reglas en tiempo de ejecución.
reglas integradas	Comprobaciones ya preparadas para entradas habituales: <code>RequiredValidationRule</code> , <code>EmailValidationRule</code> , <code>IntegerValidationRule</code> , <code>DecimalValidationRule</code> , <code>CurrencyValidationRule</code> , <code>TelephoneValidationRule</code> y <code>RegexValidationRule</code> .
orden de las reglas	Las reglas se ejecutan en el orden en que las enumeras; la primera que falla detiene las demás y cancela el evento <code>Validating</code> del control.
destino de visualización	Dónde aparece el <code>InvalidMessage</code> de la regla que falla: en el propio control, mediante un icono de <code>ErrorProvider</code> o como texto en una Label, todos accesibles a través de <code>validation.ErrorProvider</code> .
método auxiliar de configuración de reglas	Un pequeño método que construye los arrays de reglas de los campos habituales, para que un segundo formulario obtenga las mismas comprobaciones y la misma redacción con solo llamarlo.

3. Desarrollo de la lección

Un componente, dos formas de configurarlo

El extensor `Validation` es un componente, como un temporizador o el `ErrorProvider` del Módulo 2: no tiene superficie visible propia. Arrástralo desde el Cuadro de herramientas al formulario de alta en el diseñador de Visual Studio y aparecerá en la bandeja de componentes, debajo del formulario. A partir de ese momento, cada control compatible del formulario gana una nueva entrada en la ventana Propiedades llamada `ValidationRules`. Selecciona `txtEmail`, abre ese editor de colecciones y añade las reglas que necesita el campo. El diseñador las escribe en el archivo del diseñador del formulario, así que cualquiera que abra el formulario puede ver qué campos son obligatorios sin leer ni un solo controlador de eventos.

Lo mismo se puede hacer en código, que es lo que usa el laboratorio. Creas el componente, diriges su `ContainerControl` al formulario y llamas a `SetValidationRules` una vez por control, normalmente desde un método `ConfigureValidationRules` llamado en el constructor del formulario. Ambos caminos producen el mismo resultado: el extensor engancha los eventos de validación del control y ejecuta las reglas cuando el usuario sale del campo, justo el momento que el Módulo 1 te enseñó a gestionar a mano.

```

errorProvider.ContainerControl = this;
validation.ContainerControl = this;
validation.ErrorProvider = errorProvider;

validation.SetValidationRules(txtEmail, new ValidationRule[]
{
    new RequiredValidationRule { InvalidMessage = "Email is required.", Trim = true },
    new EmailValidationRule { InvalidMessage = "Enter a valid email address." }
});
validation.SetValidationRules(txtAge, new ValidationRule[]
{
    new RequiredValidationRule { InvalidMessage = "Age is required." },
    new IntegerValidationRule { InvalidMessage = "Age must be a whole number." }
});

```

Las siete reglas integradas

Cada regla responde a una pregunta sobre el texto de un control. `RequiredValidationRule` pregunta «¿hay algo aquí?» y tiene tres propiedades útiles: `Required` activa la comprobación, `Trim` elimina primero los espacios para que un campo que solo contiene espacios siga contando como vacío, y `MinimumLength` exige al menos un número determinado de caracteres. `EmailValidationRule` comprueba que el valor parece una dirección de correo electrónico. `IntegerValidationRule` acepta números enteros, como una edad, mientras que `DecimalValidationRule` acepta números con parte decimal. `CurrencyValidationRule` comprueba un importe, como el límite de crédito, y puede dar formato al valor una vez aceptado. `TelephoneValidationRule` comprueba un número de teléfono con una máscara, y `RegexValidationRule` compara el valor con una expresión regular, que es como ValidationClinic impone un código de cliente formado por las letras CUS, un guion y cuatro dígitos.

Cada regla lleva un `InvalidMessage`. Ese texto es lo que lee el usuario, así que escríbelo como texto del producto que diga qué hacer a continuación: «Age must be a whole number.» en lugar de «Integer parse failed».

El orden de las reglas y el primer fallo

Las reglas de un control se ejecutan de la primera a la última. En cuanto una falla, se omiten las restantes y se cancela el evento `Validating` del control, lo que tiene el mismo efecto que establecer tú mismo `e.Cancel = true` como en el Módulo 1. Solo se muestra un mensaje: el de la primera regla que ha fallado.

Por eso el orden es una decisión de diseño. Coloca `RequiredValidationRule` en primer lugar en cada campo obligatorio. Deja vacío el cuadro del correo electrónico y el usuario lee «Email is required.», que es el problema real. Si la regla Email fuera primero, el mismo cuadro vacío diría «Enter a valid email address.», lo cual es técnicamente cierto y mucho menos útil. En el laboratorio lo demostrarás con un `txtEmail` vacío: Required falla, Email nunca se ejecuta y el foco se comporta según el modo `AutoValidate` que elegiste antes. Los campos opcionales, como el límite de crédito, simplemente empiezan por su regla de formato.

Elegir dónde aparece el mensaje

Sin un destino de visualización, una regla que falla escribe su mensaje en el propio control como tooltip de valor no válido. Asigna un `ErrorProvider` a `validation.ErrorProvider` y el extensor llamará a `SetError` sobre él, lo que te da el icono y el tooltip del Módulo 2 sin ninguna llamada a `SetError` en tu propio código. Una Label también puede ser el destino: muestra el mensaje como texto y se oculta cuando no hay nada que decir, lo que encaja con un formulario corto o con un campo que necesita texto visible en lugar de un icono. Cualquier clase que implemente `IErrorProvider` puede desempeñar este papel; el Módulo 7 construye una que alimenta un panel de resumen.

El extensor también desencadena sus propios eventos `Validating` y `Validated` para cada control que tiene reglas. Eso te da un lugar central para actualizar `validationSummaryLabel`, de modo que el resumen visible del Módulo 2 se mantiene sincronizado con los iconos aunque ya no escribas tú las comprobaciones.

Configuraciones de reglas reutilizables

El correo electrónico, la edad y el teléfono aparecen en más de una pantalla de ValidationClinic. En lugar de copiar los arrays, colócalos en un pequeño método auxiliar, por ejemplo una clase estática `ClinicRules` con métodos como `RequiredEmail()` y `RequiredAge()` que devuelven cada uno un `ValidationRule[]` nuevo. El formulario de alta y cualquier segundo formulario llaman a los mismos métodos, así que el orden y la redacción de cada mensaje se deciden una sola vez.

Limita el método auxiliar a la forma de los campos. Las reglas que comparan dos campos, consultan algo en una base de datos o expresan una política de negocio no pertenecen al extensor, porque solo se ejecutarían cuando un control pierde el foco. El Módulo 4 las traslada a métodos de validación, a un servicio validador y a clases `ValidationRule` personalizadas.



Regla de las capas de validación

Nombra la capa antes de escribir la comprobación. La validación de campo protege la edición y se ejecuta cuando el usuario sale de un control; la validación del modelo protege los datos y ejecuta las mismas reglas desde el formulario, la cuadrícula y cualquier servicio; la validación al guardar protege la base de datos y vuelve a ejecutar todas las capas antes de escribir nada. Muestra cada error cerca de su campo y en un resumen visible, con un lenguaje claro que diga qué hacer a continuación, y mantén los errores que el usuario puede corregir separados de los fallos del sistema.



Objetivos de aprendizaje

Al final de este módulo serás capaz de:

- ✓ Usar el extensor Validation de Wisej.NET en tiempo de diseño y en tiempo de ejecución.
- ✓ Aplicar las reglas Required, Email, Integer, Decimal, Currency, Telephone y Regex.
- ✓ Explicar el orden de las reglas y por qué la validación se detiene en el primer fallo.
- ✓ Asignar un ErrorProvider o una Label para mostrar los mensajes de validación.
- ✓ Crear configuraciones de reglas reutilizables para formularios habituales.

Guía del laboratorio / examen

Qué vas a construir en el laboratorio práctico, el enfoque sugerido y los entregables requeridos.



Laboratorio práctico

Sustituye los controladores de campo escritos a mano en ValidationClinic por el extensor Validation. Añade un componente Validation y un ErrorProvider al formulario de alta, establece ContainerControl y validation.ErrorProvider, y llama a SetValidationRules en un método ConfigureValidationRules: Required y después Email en txtEmail, Required y después Integer en txtAge, Required y después Telephone en txtPhone, Currency en txtCreditLimit y una regla Regex para el código de cliente. Coloca Required en primer lugar en todos los casos para que gane el mensaje más útil, engancha el evento Validating del extensor para actualizar el resumen y traslada los arrays de reglas a un pequeño método auxiliar reutilizable para que otro formulario pueda aplicar la misma configuración.

Punto de partida

Abres ValidationClinic tal como lo dejó el Módulo 2: el formulario de alta tiene `txtName`, `txtEmail`, `txtPhone` y `txtAge`, un `ErrorProvider`, los métodos auxiliares `ValidateName`, `ValidateEmail` y `ValidatePhone` y un `validationSummaryLabel`. Añade para este laboratorio los TextBox `txtCreditLimit` y `txtCustomerCode` si todavía no están en el formulario.

Enfoque sugerido

- ✓ Añade un componente `Validation` al formulario de alta en el diseñador, selecciona `txtEmail` y añade una regla Required y una regla Email mediante el editor de la colección `ValidationRules`; después ejecuta el formulario una vez para ver cómo trabaja el extensor antes de escribir código.
- ✓ Elimina esas reglas del diseñador y escribe `ConfigureValidationRules`, llamado desde el constructor: establece `errorProvider.ContainerControl`, `validation.ContainerControl` y `validation.ErrorProvider`, y después llama a `SetValidationRules` para el correo electrónico, la edad, el teléfono, el límite de crédito y el código de cliente.
- ✓ Asigna a cada campo obligatorio `RequiredValidationRule` como primera regla, con `Trim = true` donde los espacios no deban contar, y escribe cada `InvalidMessage` como una frase que indique al usuario qué debe escribir.
- ✓ Gestiona los eventos `Validating` y `Validated` del extensor en un único lugar y reconstruye `validationSummaryLabel` a partir de los errores actuales, ocultándola cuando no quede ninguno.
- ✓ Traslada los arrays de reglas a un método auxiliar estático `ClinicRules` cuyos métodos devuelvan un `ValidationRule[]` nuevo, llámalo desde el formulario de alta y prueba un correo electrónico vacío, uno mal formado, uno válido, una edad no numérica y un código de cliente incorrecto.

Entregables requeridos

- ✓ Componente Validation conectado al formulario y a un ErrorProvider como destino de visualización
- ✓ Método ConfigureValidationRules que llama a SetValidationRules para el correo electrónico, la edad, el teléfono, el límite de crédito y el código de cliente
- ✓ Regla Required colocada en primer lugar en cada campo obligatorio, con un texto InvalidMessage claro
- ✓ Controlador Validating del extensor que mantiene el resumen de validación sincronizado con las reglas
- ✓ Método auxiliar de configuración de reglas reutilizable al que puede llamar un segundo formulario

Preguntas de repaso

- ✓ Cambias el orden para que la regla Email se ejecute antes que Required en `txtEmail`. Describe exactamente qué ve un usuario después de dejar el campo vacío y por qué ese mensaje es menos útil.
- ✓ Tanto los métodos auxiliares del Módulo 2 como el extensor terminan con un icono junto al campo. ¿Qué has dejado de escribir a mano y qué has tenido que seguir decidiendo tú?
- ✓ Un compañero quiere añadir «la fecha de fin debe ser posterior a la fecha de inicio» a `ClinicRules`. ¿Por qué el extensor no es el lugar adecuado para esa comprobación y cuándo dejaría de ejecutarse?



Regla de las capas de validación

Nombra la capa antes de escribir la comprobación. La validación de campo protege la edición y se ejecuta cuando el usuario sale de un control; la validación del modelo protege los datos y ejecuta las mismas reglas desde el formulario, la cuadrícula y cualquier servicio; la validación al guardar protege la base de datos y vuelve a ejecutar todas las capas antes de escribir nada. Muestra cada error cerca de su campo y en un resumen visible, con un lenguaje claro que diga qué hacer a continuación, y mantén los errores que el usuario puede corregir separados de los fallos del sistema.