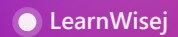


Editores, botones, validación y feedback

Guía de la lección · Guía del laboratorio / examen



Guía de la lección

Elegir el editor que corresponde al valor, un modelo de comandos para los botones, validación con `ErrorProvider` que ayuda en lugar de bloquear, y feedback no bloqueante con `Toast` y `AlertBox`. Qué significa esto para un desarrollador de Wisej.NET y cómo leer este módulo.



Enfoque de producción

Elegir el editor que corresponde al valor, un modelo de comandos para los botones, validación con `ErrorProvider` que ayuda en lugar de bloquear, y feedback no bloqueante con `Toast` y `AlertBox`.

1. El problema en producción

Los editores son el primer lugar donde una biblioteca de controles se convierte en un sistema de negocio. Un formulario que parece correcto pero acepta entradas ambiguas o no válidas genera tickets de soporte y scripts de limpieza mucho después de haberse publicado. Los editores de Wisej.NET siguen el conocido modelo de componentes de .NET pero se ejecutan en el navegador, así que una misma regla de negocio puede aplicarse en varias capas: propiedades del editor, extenders de validación, manejadores de eventos, enlace de datos, validación del servicio y restricciones de la base de datos. La función de la capa de interfaz es evitar errores evidentes y explicarlos con claridad; nunca debe ser la única capa que protege la aplicación.

2. Conceptos clave

Concepto	Por qué importa
editor adecuado al valor	El editor cuyo tipo ya restringe la entrada: <code>DateTimePicker</code> para fechas, <code>NumericUpDown</code> para números acotados, <code>ComboBox</code> para valores de una lista, de modo que el servidor nunca analiza texto ambiguo.
modelo de comandos	Una separación deliberada entre comandos principales (Save, Approve, Search, Send, Export), comandos secundarios (ToolBar, MenuBar, SplitButton, menú contextual) y comandos destructivos que necesitan confirmación o una ubicación cuidadosa.
ErrorProvider	El extender que asocia un mensaje concreto y accionable al control que está mal, en lugar de un genérico «Valor no válido» en otro lugar de la página.
feedback bloqueante frente a no bloqueante	<code>MessageBox</code> interrumpe para pedir una decisión o una confirmación crítica; <code>AlertBox</code> y <code>Toast</code> informan del estado sin detener el flujo de trabajo.
validadores con nombre	Métodos como <code>ValidateEmail</code> o <code>ValidateCreditLimit</code> que contienen una regla cada uno, para que la regla se pueda probar y explicar en lugar de quedar enterrada en un manejador de clic.
estado ocupado	Deshabilitar Save y mostrar el progreso mientras se ejecuta una operación, lo que evita envíos duplicados y forma parte de la fiabilidad, no de la decoración.
UserControl de edición	Una composición de etiquetas, editores, un <code>ErrorProvider</code> y un método de validación que devuelve un objeto de resultado, para que el resto de la aplicación nunca sepa cómo se comprueba el campo de email.

3. Desarrollo de la lección

Adapta el editor al valor

La primera decisión en cualquier pantalla de entrada es qué editor representa cada valor: elige el control cuyo tipo ya excluye las entradas imposibles. `TextBox` es para texto normal. Activa `Multiline` solo cuando el usuario necesita párrafos, no porque un campo resulte ser ancho, y usa `Watermark` como ejemplo o indicación, nunca como sustituto de una etiqueta. Cuando la forma de la entrada es predecible, como un código, un patrón tipo teléfono o un número de referencia, `MaskedTextBox` impone esa forma mientras el usuario escribe. Cuando analizar y formatear un valor con tipo es la esencia del campo, `TypedTextBox` hace la conversión por ti.

Las fechas y los números merecen sus propios editores. `DateTimePicker` elimina la ambigüedad del análisis y valida la selección de una fecha de forma mucho más natural que el texto libre. `NumericUpDown`, `DomainUpDown` y `TimeUpDown` expresan valores acotados con un mínimo, un máximo y un incremento. `ComboBox` es la elección adecuada cuando un valor procede de una lista; su `DropDownStyle` decide si el usuario también puede escribir un valor que no está en la lista. Es una mala elección cuando el usuario necesita buscar o explorar un conjunto de datos muy grande: existe para una selección compacta de una lista pequeña o mediana.

En Operations Console, el `CustomerEditor` aplica exactamente esta correspondencia: el nombre y el email son controles `TextBox`, el estado y el tipo de cliente son controles `ComboBox` con elementos fijos, la fecha de inicio es un `DateTimePicker` y el límite de crédito es un `NumericUpDown` con un rango. Con las listas importa un hábito: mantén separados el texto mostrado y la clave almacenada, para que «Active» en pantalla nunca se convierta en el valor que guardas.

Botones como comandos y controles de feedback

Los botones no son simples rectángulos en los que hacer clic; una pantalla de negocio necesita un modelo de comandos. Los comandos principales (Save, Approve, Search, Send, Export) son controles `Button` normales colocados donde la vista los espera. Los comandos secundarios van en una `ToolBar`, un `MenuBar`, un `SplitButton` o un menú contextual. Un `SplitButton` encaja cuando un comando tiene una acción por defecto y acciones alternativas, como Save junto a Save and Close. Los comandos destructivos necesitan confirmación, un estado deshabilitado o una ubicación clara. `CheckBox` y `RadioButton` son controles de estado, no comandos: van donde el usuario toma una decisión, nunca donde se dispara una acción.

Cada control de feedback tiene una función. `ToolTip` y `HelpTip` ofrecen orientación sobre lo que espera un campo; no son canales de error. `ErrorProvider` marca el control que tiene un error de validación y, en los controles enlazados a datos, debe asociarse al contenedor correcto o sus iconos aparecerán en el lugar equivocado. `MessageBox` se reserva para una decisión bloqueante o una confirmación crítica. `AlertBox` y `Toast` son para mensajes de estado que no deben interrumpir el flujo de trabajo y, como un `Toast` puede reutilizarse y actualizarse mientras está visible, también sirve como feedback de progreso de un guardado largo, junto a una `ProgressBar` cuando se conoce el progreso real.

Validación que ayuda sin bloquear

La validación debe ser visible, concreta y recuperable, y debe llegar lo bastante pronto para ayudar sin estorbar mientras se escribe. El flujo limpio tiene seis pasos. Primero, establece las propiedades del editor (`MaxLength`, `Minimum`, `Maximum`, elementos fijos) para que no se puedan introducir valores imposibles. Segundo, usa un análisis específico de la entrada para los valores con tipo. Tercero, en la validación del campo, dentro del manejador `Validating` del control, establece o borra el mensaje de `ErrorProvider` de ese control. Cuarto, al guardar, ejecuta la validación a nivel de formulario y después la del servicio. Quinto, si algo falla, mantén al usuario en la pantalla, pon el foco en el primer control no válido y muestra errores concretos. Sexto, cuando el guardado tenga éxito, muestra una confirmación no bloqueante y actualiza el área de estado.

El propio mensaje debe decir qué corregir: «Introduce un límite de crédito entre 0 y 250.000» en lugar de «Valor no válido». Evita meter toda la validación en los manejadores de clic; hace que la pantalla sea difícil de probar y difícil de explicar para un compañero o un asistente de IA. Coloca cada regla junto a su control o en un validador con nombre como `ValidateRequiredName`, `ValidateEmail` y `ValidateCreditLimit`, y deja que Save llame a esos métodos. Mantén explícita la ruta de Save (validar, guardar, refrescar el estado, notificar) y deshabilita los comandos mientras se ejecuta para que un doble clic no pueda enviar dos veces.

```
public EditorResult ValidateContent()
{
    var result = new EditorResult();
    this.errorProvider.SetError(this.txtEmail, "");
    this.errorProvider.SetError(this.numCreditLimit, "");
    if (!this.txtEmail.Text.Contains("@"))
        result.Add(this.txtEmail, "Enter an email address such as name@company.com.");
    if (this.numCreditLimit.Value < 0 || this.numCreditLimit.Value > 250000)
        result.Add(this.numCreditLimit, "Enter a credit limit between 0 and 250,000.");
    foreach (var error in result.Errors)
        this.errorProvider.SetError(error.Control, error.Message);
    result.FirstInvalid?.Focus();
    return result;
}
```

El UserControl CustomerEditor

Este es el punto en el que un conjunto de controles se convierte en un componente de nivel de aplicación. El `CustomerEditor` del laboratorio es un `UserControl` que contiene sus etiquetas, sus editores, un `ErrorProvider`, los botones Save, Reset y Validate y un método de validación que devuelve un objeto de resultado claro: qué controles no son válidos y qué mensajes se han mostrado. La página del shell que lo aloja solo pregunta «¿es válido?» y «dame el cliente»; nunca sabe cómo funciona la regla del email. El método se llama `ValidateContent` para que no pueda confundirse con los miembros de validación que ya expone un control contenedor.

Las reglas de diseño de las pantallas de entrada se derivan de esa frontera y son la lista de comprobación con la que evalúa el instructor:

- Cada campo editable tiene una etiqueta visible o un equivalente accesible.
- El editor corresponde al tipo de valor; las fechas y los números no se analizan desde controles `TextBox` normales sin una razón de peso.
- Los mensajes de validación son accionables, `ErrorProvider` transmite los errores de campo y `Toast` o `AlertBox` transmiten el feedback del flujo de trabajo.
- Los comandos se deshabilitan mientras se ejecuta una operación, y la ruta de Save es explícita: validar, guardar, refrescar el estado, notificar.



Errores comunes

Usar `MessageBox` para cada mensaje de estado menor. Guardar un estado parcial no válido porque la validación solo se ejecutó en los controles visibles. Esconder la validación dentro de manejadores de eventos anónimos. Leer el valor de un `ComboBox` sin distinguir el texto mostrado de la clave almacenada. Olvidar que los botones deshabilitados y el feedback de ocupado forman parte de la fiabilidad, no de la decoración.



Regla de la biblioteca de controles

Elige el control nativo más sencillo que exprese la tarea del usuario antes de escribir código personalizado. Los controles nativos ya te dan soporte en tiempo de diseño, temas, eventos y estado en el servidor; recurre después a un extender o a un UserControl, y construye un widget JavaScript solo cuando ningún control nativo encaje.



Objetivos de aprendizaje

Al terminar este módulo serás capaz de:

- ✓ Seleccionar el editor correcto para entradas de texto, numéricas, de fecha/hora, con máscara y de selección.
- ✓ Usar de forma adecuada Button, SplitButton, CheckBox, RadioButton, ToolTip, HelpTip, ErrorProvider, AlertBox, MessageBox, Toast y ProgressBar.
- ✓ Implementar una validación lo bastante inmediata para ayudar al usuario, pero no tan agresiva que bloquee la entrada normal de datos.
- ✓ Crear un UserControl de edición reutilizable para datos de clientes o contactos.

Guía del laboratorio / examen

Lo que construirás en el laboratorio práctico, el enfoque sugerido y los entregables requeridos.



Laboratorio práctico

Construye un UserControl CustomerEditor para Operations Console con campos etiquetados para nombre, email, estado, fecha de inicio, límite de crédito y tipo de cliente, usando el editor adecuado para cada valor en lugar de un TextBox para todo. Añade mensajes de ErrorProvider para campos obligatorios, formato del email, rango del límite de crédito y fechas no válidas; comandos Save, Reset y Validate con comportamiento deshabilitado y ocupado durante el guardado; y un Toast o MessageBox cuando termine la validación o el guardado.

Punto de partida

Abre la solución `OperationsConsole` del Módulo 1: la página del shell con sus áreas de navegación, contenido, comandos y estado ya funciona, y la página provisional de Editors está vacía. Mantén el trabajo de este módulo en su propia carpeta y namespace `Editors` para que el proyecto final pueda reutilizar el control sin cambios.

Enfoque sugerido

- ✓ Esboza los seis campos y anota el editor de cada uno antes de abrir el diseñador: `TextBox` para el nombre y el email, `ComboBox` para el estado y el tipo de cliente, `DateTimePicker` para la fecha de inicio y `NumericUpDown` para el límite de crédito.
- ✓ Añade un UserControl `CustomerEditor`, coloca las etiquetas y los editores con nombres claros (`txtName`, `cboStatus`, `dtpStartDate`, `numCreditLimit`), establece `Minimum`, `Maximum`, `MaxLength` y elementos fijos para que no se puedan escribir valores imposibles, y da a cada editor un `ToolTip` que describa lo que espera.
- ✓ Añade un `ErrorProvider` y validadores con nombre (`ValidateRequiredName`, `ValidateEmail`, `ValidateCreditLimit`, `ValidateStartDate`) que establezcan o borren un mensaje concreto; llama al validador correspondiente desde el evento `Validating` de cada editor y a todos ellos desde un método `ValidateContent` que devuelva un objeto de resultado.
- ✓ Añade los botones Save, Reset y Validate; haz que Save ejecute `ValidateContent`, después la llamada al servicio y luego una actualización del estado, y deshabilita los tres comandos y muestra feedback de ocupado mientras dura el guardado.
- ✓ Muestra un `Toast` o un `MessageBox` cuando termine la validación o el guardado, actualiza el área de estado del shell y aloja el control terminado en la página Editors.

Entregables requeridos

- ✓ UserControl CustomerEditor con campos etiquetados para nombre, email, estado, fecha de inicio, límite de crédito y tipo de cliente
- ✓ Editores adecuados: TextBox, DateTimePicker, NumericUpDown y ComboBox en lugar de texto libre en todas partes

- ✓ Mensajes de ErrorProvider para campos obligatorios, formato del email, rango del límite de crédito y fechas no válidas
- ✓ Comandos Save, Reset y Validate con comportamiento deshabilitado y ocupado durante el guardado
- ✓ Feedback con Toast o AlertBox cuando termine la validación o el guardado

Preguntas de repaso

- ✓ ¿Qué errores de validación ya no pueden producirse en absoluto gracias al editor que elegiste, y cuáles siguen necesitando un mensaje de ErrorProvider?
- ✓ Si el usuario hace clic dos veces en Save en un segundo, ¿qué hace tu pantalla y en qué parte del código está garantizado?
- ✓ ¿Qué sabe la página del shell sobre cómo se valida el campo de email, y qué tendría que cambiar si la regla se trasladara a un servicio?



Regla de la biblioteca de controles

Elige el control nativo más sencillo que exprese la tarea del usuario antes de escribir código personalizado. Los controles nativos ya te dan soporte en tiempo de diseño, temas, eventos y estado en el servidor; recurre después a un extender o a un UserControl, y construye un widget JavaScript solo cuando ningún control nativo encaje.