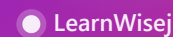


Validazione programmatica e feedback con ErrorProvider

Guida della lezione · Guida al lab / esame



Guida della lezione

Un solo ErrorProvider per form, SetLastError e la rimozione degli errori controllo per controllo, InvalidMessage per un feedback semplice e locale, un piccolo tipo di risultato FieldValidation che tiene le regole fuori dalla UI e un riepilogo di validazione visibile, così che nessun errore dipenda dal passare il mouse su un'icona. Che cosa significa per uno sviluppatore che conosce le basi di C# e sta iniziando a sviluppare form con Wisej.NET — e come affrontare questo modulo.



Focus di produzione

Un solo ErrorProvider per form, SetLastError e la rimozione degli errori controllo per controllo, InvalidMessage per un feedback semplice e locale, un piccolo tipo di risultato FieldValidation che tiene le regole fuori dalla UI e un riepilogo di validazione visibile, così che nessun errore dipenda dal passare il mouse su un'icona.

1. Il problema in produzione

Il Modulo 1 ha dato al form di accettazione clienti di ValidationClinic due gestori `Validating`, e per due campi funziona. Un vero form di contatto ha un nome, un'email, un numero di telefono, un codice cliente e altro ancora, e se ogni campo riceve un proprio gestore scritto a mano con un proprio modo di mostrare i messaggi, il form diventa presto incoerente: un campo mostra un'icona, un altro non mostra nulla, un terzo lascia a video un vecchio errore anche dopo che l'utente l'ha corretto. Questo modulo sostituisce questa deriva con un unico schema: piccoli metodi helper decidono che cosa non va, un solo metodo mostra il risultato e un riepilogo visibile dice a ogni utente che cosa correggere senza costringerlo a passare il mouse su qualcosa.

2. Concetti chiave

Concetto	Perché è importante
<code>ErrorProvider</code>	Un componente extender che disegna un'icona di errore accanto a qualsiasi controllo a cui assegni un messaggio e mostra quel messaggio come tooltip; una sola istanza serve l'intero form.
<code>ContainerControl</code>	Il form o lo user control a cui appartiene il provider; impostarlo su <code>this</code> indica al provider su quale superficie disegnare le sue icone.
<code>setError(control, message)</code>	Mostra <code>message</code> accanto a <code>control</code> ; chiamarlo con una stringa vuota o con <code>null</code> rimuove l'errore solo per quel controllo.
<code>GetError</code> , <code>HasErrors</code> , <code>Clear()</code>	Leggono il messaggio attuale di un controllo, chiedono se sul form resta qualche errore, oppure rimuovono tutti gli errori in un colpo solo quando l'intero form viene reimpostato.
<code>InvalidMessage</code>	Una proprietà dei controlli editor che supportano l'interfaccia <code>IValidation</code> ; è il controllo stesso a mostrare il messaggio come tooltip di invalidità, senza alcun provider separato.
Record <code>FieldValidation</code>	Un minuscolo tipo di risultato che contiene il controllo, il nome del campo e un messaggio, dove un messaggio vuoto significa valido; permette alle regole di restituire risposte invece di modificare lo schermo.
<code>validationSummaryLabel</code>	Una semplice Label che elenca in testo ogni problema attuale, visibile solo finché esistono errori, così che il feedback non dipenda mai dal vedere un'icona o dal passarci sopra il mouse.

3. Il filo della lezione

Mostrare e rimuovere gli errori di campo con ErrorProvider

Crea un solo `ErrorProvider` per il form, trascinandolo dalla Toolbox sul designer oppure come campo creato nel codice, e imposta il suo `ContainerControl` sul form. Due impostazioni facoltative lo rendono più tranquillo per gli utenti: `BlinkStyle = ErrorBlinkStyle.NeverBlink` impedisce all'icona di lampeggiare, e `SetIconAlignment(txtEmail, ErrorIconAlignment.MiddleRight)` insieme a `SetIconPadding(txtEmail, 4)` colloca l'icona subito a destra del campo, dove l'occhio se l'aspetta.

Mostrare un errore richiede una sola chiamata: `errorProvider.SetError(txtName, "Il nome è obbligatorio.")`. Ricorda che Wisej.NET la esegue sul server; il browser riceve semplicemente l'icona e il tooltip aggiornati. Per rimuoverlo si usa la stessa chiamata con una stringa vuota. Questo dettaglio conta più di quanto sembri. Quando l'utente corregge il nome, rimuovi l'errore su `txtName` e su nient'altro. `Clear()` rimuove tutti gli errori del form, quindi riservalo ai momenti in cui l'intero form riparte davvero da capo: il caricamento di un altro contatto, il reset del form o l'avvio di un passaggio di validazione completo che riapplica subito tutti gli errori attuali. `GetError(control)` ti dice che cosa mostra in quel momento un singolo controllo, e `HasErrors` è un rapido controllo finale prima che Save prosegua.

InvalidMessage o SetError?

Wisej.NET ti offre un secondo modo, più leggero, per dare feedback. I controlli editor che implementano `IValidation` hanno una proprietà `InvalidMessage`. Impostala dentro il gestore `Validating` del controllo quando il valore è errato, riportala a una stringa vuota quando il valore è corretto, e il controllo mostrerà il messaggio come proprio tooltip di invalidità. Nessun provider, nessun posizionamento dell'icona, nulla di condiviso.

Questo rende `InvalidMessage` adatto a un form molto semplice o a un singolo campo il cui feedback deve restare locale. `SetError` è la scelta migliore quando un form ha più campi, perché allora tutti gli errori hanno lo stesso aspetto, si possono rileggere con `GetError` e si possono contare con `HasErrors`. Nel lab li usi entrambi di proposito: il campo del codice cliente usa `InvalidMessage`, tutti gli altri campi passano dall' `ErrorProvider`, e scrivi una breve nota che confronta ciò che vede l'utente.

Una label di riepilogo che tutti possono leggere

Un'icona di errore è piccola, è fatta solo di un colore e di una forma, e il suo testo compare solo quando il puntatore si ferma sopra. Chi usa la tastiera, chi usa un touch screen o chi ha una vista ridotta può non notarla affatto. Per questo ogni form di questo corso mostra i suoi errori anche come testo semplice. La versione più semplice è una Label chiamata `validationSummaryLabel` posizionata sopra il pulsante Save, con una riga per problema, per esempio *Email: Inserisci un indirizzo email valido*.

Due regole mantengono affidabile il riepilogo. È visibile solo finché esiste almeno un errore, così nessun riquadro rosso vuoto resta a video. Ed è ricostruito a partire dagli stessi risultati che impostano le icone, così icone e riepilogo non possono mai essere in disaccordo. Un riepilogo visibile rende anche il form più facile da testare: puoi verificare il testo della label invece di passare il mouse sulle icone.

Un solo metodo helper per più campi

Il passo chiave di questo modulo è separare il decidere dal mostrare. Ogni regola diventa un piccolo metodo, `ValidateName`, `ValidateEmail` o `ValidatePhone`, che legge un valore e restituisce un record `FieldValidation`. Non chiama mai `SetError` e non tocca mai una Label. Un unico metodo `Apply` riceve poi tutti i risultati e svolge in un solo punto il lavoro di visualizzazione.

```
public sealed record FieldValidation(Control Control, string FieldName, string Message)
{
    public bool IsValid => string.IsNullOrEmpty(Message);
}

private bool Apply(params FieldValidation[] results)
{
    foreach (var result in results)
        errorProvider.SetError(result.Control, result.Message);
    validationSummaryLabel.Text = string.Join("\n", results
        .Where(r => !r.IsValid).Select(r => $"{r.FieldName}: {r.Message}"));
    validationSummaryLabel.Visible = results.Any(r => !r.IsValid);
    return results.All(r => r.IsValid);
}
```

Nota che `Apply` chiama `SetError` per ogni risultato, valido o no. Un risultato valido ha un messaggio vuoto, quindi passarlo rimuove l'icona di quel controllo. È esattamente così che correggere un campo rimuove solo l'errore di quel campo: gli altri risultati portano ancora i loro messaggi, quindi le loro icone restano. Il pulsante Save chiama `Apply(ValidateName(), ValidateEmail(), ValidatePhone())` ed esce subito se riceve `false`.

Feedback sul campo e feedback sulle regole di business

Non tutti i messaggi vanno accanto a un campo. Il *feedback sul campo* riguarda il valore di un singolo controllo: è vuoto, non contiene la @, il numero di telefono è troppo corto. L'utente può correggerlo senza lasciare quel controllo, quindi va accanto al controllo e nel riepilogo. Il *feedback sulle regole di business* riguarda il significato dei dati: esiste già un contatto con questo nome, oppure due date non sono compatibili. Queste regole spesso hanno bisogno di altri campi o di dati salvati, e di solito vengono eseguite al Save invece che quando l'utente esce da un campo con Tab.

Tieni separati i due tipi nel codice e nella formulazione. Gli helper dei campi restano piccoli e veloci. Una regola di business può comunque indicare il controllo più pertinente con `SetError`, ma il suo messaggio spiega la regola in un linguaggio semplice e compare nel riepilogo, così l'utente capisce perché Save si è fermato. Nessuno dei due è un'eccezione: un'email errata è un errore previsto con un normale messaggio di prodotto, mentre una connessione al database persa è un errore di sistema gestito a parte. I moduli successivi spostano le regole di business in un servizio riutilizzabile; questo modulo crea l'abitudine.



La regola dei livelli di validazione

Dai un nome al livello prima di scrivere la verifica. La validazione dei campi protegge la modifica e viene eseguita quando l'utente lascia un controllo; la validazione del modello protegge i dati ed esegue le stesse regole dal form, dalla griglia e da qualsiasi servizio; la validazione al salvataggio protegge il database ed esegue di nuovo tutti i livelli prima che venga scritto qualsiasi dato. Mostra ogni errore vicino al suo campo e in un riepilogo visibile, con un linguaggio semplice che dica che cosa fare, e tieni gli errori correggibili dall'utente separati dagli errori di sistema.



Obiettivi di apprendimento

Alla fine di questo modulo sarai in grado di:

- ✓ Usare `ErrorProvider` per mostrare e rimuovere gli errori a livello di campo.
- ✓ Capire la differenza tra `InvalidMessage` ed `ErrorProvider.SetError`.
- ✓ Creare una label di riepilogo di validazione per l'accessibilità.
- ✓ Validare più campi da un unico metodo helper.
- ✓ Separare il feedback sul campo dal feedback sulle regole di business.

Guida al lab / esame

Che cosa costruirai nel lab pratico, l'approccio suggerito e i deliverable richiesti.



Lab pratico

Trasforma il form di contatto di `ValidationClinic` in un form con validazione programmatica. Crea un solo `ErrorProvider` per il form con `ContainerControl` impostato, `NeverBlink` e l'icona allineata a destra, scrivi un record `FieldValidation` più gli helper `ValidateName`, `ValidateEmail` e `ValidatePhone` che restituiscono un messaggio invece di toccare la UI, e un metodo `Apply` che chiama `SetError` per ogni risultato e compila una `validationSummaryLabel` visibile solo finché esistono errori. Usa `InvalidMessage` sul campo del codice cliente per confrontare i due stili di feedback, e assicurati che correggere un campo rimuova solo l'errore di quel campo.

Punto di partenza

Apri il progetto `ValidationClinic` che hai costruito nel Modulo 1 oppure, se inizi da qui, la versione del Modulo 1 dal repository pubblico di esempi. Il form di accettazione clienti ha già `txtName`, `txtEmail`, un pulsante `Save` e un pulsante `Cancel`, con i gestori `Validating` e la protezione `ValidateChildren()` che hai scritto la volta scorsa; in questo lab aggiungi `txtPhone`, `txtCustomerCode` e una `Label` di riepilogo, e sposti le verifiche in metodi helper.

Approccio suggerito

- ✓ Aggiungi un `ErrorProvider` al form e, in un metodo `SetupValidation` chiamato dal costruttore, imposta `ContainerControl = this`, `BlinkStyle = ErrorBlinkStyle.NeverBlink` e `SetIconAlignment(..., ErrorIconAlignment.MiddleRight)` con un piccolo `SetIconPadding` per ogni campo.
- ✓ Dichiarare il record `FieldValidation`, poi scrivi `ValidateName`, `ValidateEmail` e `ValidatePhone` in modo che ciascuno elimini gli spazi dal valore, lo verifichi e restituisca un record con un messaggio in linguaggio semplice, oppure con un messaggio vuoto quando il valore va bene. Nessuno di essi può chiamare `SetError`.
- ✓ Posiziona sopra i pulsanti una `Label` chiamata `validationSummaryLabel` con `Visible = false`, e scrivi `Apply(params FieldValidation[] results)` in modo che imposti o rimuova l'errore di ogni controllo, compili la label con una riga per problema, mostri la label solo finché esistono problemi e restituisca se tutto è valido.
- ✓ Chiama `Apply(ValidateName(), ValidateEmail(), ValidatePhone())` all'inizio di `btnSave_Click` ed esci quando restituisce `false`. Fai in modo che il gestore `Validating` di ogni campo imposti o rimuova solo l'errore del proprio controllo in base al risultato del suo helper e, finché il riepilogo è visibile dopo un `Save` fallito, esegua di nuovo l'`Apply` completo così che l'elenco si accorci man mano che i campi vengono corretti.
- ✓ In `txtCustomerCode_Validating` usa `InvalidMessage` invece del provider, poi esegui il form con diversi valori non validi, correggi un campo alla volta e scrivi una breve nota su come i due stili di feedback differiscono per l'utente.

Deliverable richiesti

- ✓ ErrorProvider a livello di form con ContainerControl, BlinkStyle e allineamento dell'icona configurati
- ✓ Record FieldValidation e metodi helper ValidateName, ValidateEmail e ValidatePhone
- ✓ Metodo Apply che imposta o rimuove l'errore di ogni controllo e restituisce se tutti i campi sono validi
- ✓ validationSummaryLabel che elenca ogni problema attuale ed è nascosta quando non ce ne sono
- ✓ Un campo che usa InvalidMessage invece di ErrorProvider, con una nota che confronta i due approcci

Domande di revisione per lo studente

- ✓ Il gestore `txtName_Validating` di un collega chiama `errorProvider.Clear()` non appena il nome diventa valido. Che cosa va storto per un utente che ha anche digitato un'email non valida, e che cosa dovrebbe chiamare invece il gestore?
- ✓ Il campo del codice cliente mostra il suo messaggio tramite `InvalidMessage`. Che cosa dovrebbe sapere la label di riepilogo su quel campo per elencarlo anch'esso, e che cosa ti dice questo su quando un provider condiviso è la scelta migliore?
- ✓ Un utente digita un'email ben formata che appartiene già a un altro contatto. È feedback sul campo o feedback sulle regole di business, quando dovrebbe essere eseguita la verifica e dove dovrebbe comparire il messaggio?



La regola dei livelli di validazione

Dai un nome al livello prima di scrivere la verifica. La validazione dei campi protegge la modifica e viene eseguita quando l'utente lascia un controllo; la validazione del modello protegge i dati ed esegue le stesse regole dal form, dalla griglia e da qualsiasi servizio; la validazione al salvataggio protegge il database ed esegue di nuovo tutti i livelli prima che venga scritto qualsiasi dato. Mostra ogni errore vicino al suo campo e in un riepilogo visibile, con un linguaggio semplice che dica che cosa fare, e tieni gli errori correggibili dall'utente separati dagli errori di sistema.