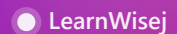


Editoren, Buttons, Validierung und Feedback

Lektionsleitfaden · Lab- und Prüfungsleitfaden



Lektionsleitfaden

Den Editor wählen, der zum Wert passt, ein Befehlsmodell für Buttons, ErrorProvider-Validierung, die hilft statt blockiert, und nicht blockierendes Feedback mit Toast und AlertBox. Was das für Wisej.NET-Entwickler bedeutet – und wie Sie dieses Modul lesen.



Fokus: Produktivbetrieb

Den Editor wählen, der zum Wert passt, ein Befehlsmodell für Buttons, ErrorProvider-Validierung, die hilft statt blockiert, und nicht blockierendes Feedback mit Toast und AlertBox.

1. Das Problem im Produktivbetrieb

Bei den Editoren wird aus einer Control-Bibliothek zum ersten Mal ein Geschäftssystem. Ein Formular, das korrekt aussieht, aber mehrdeutige oder ungültige Eingaben annimmt, erzeugt noch lange nach der Auslieferung Support-Tickets und Bereinigungsskripte. Editoren in Wisej.NET folgen dem vertrauten .NET-Komponentenmodell, laufen aber im Browser, sodass sich eine Geschäftsregel auf mehreren Ebenen durchsetzen lässt: Editor-Properties, Validierungs-Extender, Event-Handler, Data Binding, Service-Validierung und Datenbank-Constraints. Aufgabe der UI-Schicht ist es, offensichtliche Fehler zu verhindern und Fehler klar zu erklären; sie darf niemals die einzige Schicht sein, die die Anwendung schützt.

2. Zentrale Konzepte

Konzept	Warum es wichtig ist
zum Wert passender Editor	Der Editor, dessen Typ die Eingabe bereits einschränkt: <code>DateTimePicker</code> für Datumswerte, <code>NumericUpDown</code> für begrenzte Zahlen, <code>ComboBox</code> für Listenwerte, sodass der Server nie mehrdeutigen Text parsen muss.
Befehlsmodell	Eine bewusste Aufteilung in primäre Befehle (Save, Approve, Search, Send, Export), sekundäre Befehle (ToolBar, MenuBar, SplitButton, Kontextmenü) und destruktive Befehle, die eine Bestätigung oder eine sorgfältige Platzierung brauchen.
ErrorProvider	Der Extender, der eine konkrete, umsetzbare Meldung an das fehlerhafte Control heftet, statt irgendwo auf der Seite ein allgemeines „Invalid value“ anzuzeigen.
blockierendes vs. nicht blockierendes Feedback	<code>MessageBox</code> unterbricht für eine Entscheidung oder eine kritische Bestätigung; <code>AlertBox</code> und <code>Toast</code> melden einen Status, ohne den Arbeitsablauf anzuhalten.
benannte Validatoren	Methoden wie <code>ValidateEmail</code> oder <code>ValidateCreditLimit</code> , die jeweils eine Regel enthalten, sodass die Regel testbar und erklärbar ist, statt in einem Click-Handler zu verschwinden.
Beschäftigt-Zustand	Save deaktivieren und den Fortschritt anzeigen, solange ein Vorgang läuft; das verhindert doppeltes Absenden und gehört zur Zuverlässigkeit, nicht zur Dekoration.
Editor-UserControl	Eine Komposition aus Labels, Editoren, einem <code>ErrorProvider</code> und einer Validierungsmethode, die ein Ergebnisobjekt zurückgibt, sodass der Rest der Anwendung nie erfährt, wie das E-Mail-Feld geprüft wird.

3. Der rote Faden der Lektion

Den Editor zum Wert passend wählen

Die erste Entscheidung auf jedem Eingabebildschirm lautet, welcher Editor welchen Wert darstellt: Wählen Sie das Control, dessen Typ unmögliche Eingaben bereits ausschließt. `TextBox` ist für gewöhnlichen Text gedacht. Setzen Sie `Multiline` nur, wenn der Benutzer Absätze braucht, nicht weil ein Feld zufällig breit ist, und verwenden Sie `Watermark` als Beispiel oder Eingabeaufforderung, niemals als Ersatz für ein Label. Ist die Form der Eingabe vorhersehbar, etwa bei einem Code, einem telefonnummernähnlichen Muster oder einer Referenznummer, erzwingt `MaskedTextBox` diese Form schon während der Eingabe. Wenn das Parsen und Formatieren eines typisierten Werts der Kern des Feldes ist, übernimmt `TypedTextBox` die Umwandlung für Sie.

Datums- und Zahlenwerte verdienen eigene Editoren. `DateTimePicker` beseitigt Mehrdeutigkeiten beim Parsen und validiert eine Datumsauswahl weit natürlicher als Freitext. `NumericUpDown`, `DomainUpDown` und `TimeUpDown` bilden eingeschränkte Werte mit Minimum, Maximum und Schrittweite ab. `ComboBox` ist die richtige Wahl, wenn ein Wert aus einer Liste stammt; ihr `DropDownStyle` entscheidet, ob der Benutzer auch einen Wert eintippen darf, der nicht in der Liste steht. Eine schlechte Wahl ist sie, wenn der Benutzer einen sehr großen Datenbestand durchsuchen oder durchblättern muss: Sie ist für die kompakte Auswahl aus einer kleinen oder mittelgroßen Liste gedacht.

In der Operations Console setzt der `CustomerEditor` genau diese Zuordnung um: Name und E-Mail sind `TextBox` -Controls, Status und Kundentyp sind `ComboBox` -Controls mit festen Einträgen, das Startdatum ist ein `DateTimePicker` und das Kreditlimit ein `NumericUpDown` mit Wertebereich. Bei Listen zählt eine Gewohnheit: Halten Sie Anzeigetext und gespeicherten Schlüssel getrennt, damit aus „Active“ auf dem Bildschirm nie der Wert wird, den Sie speichern.

Buttons als Befehle und die Feedback-Controls

Buttons sind nicht bloß anklickbare Rechtecke; ein Geschäftsbildschirm braucht ein Befehlsmodell. Primäre Befehle (Save, Approve, Search, Send, Export) sind einfache `Button` -Controls, platziert dort, wo das Auge sie erwartet. Sekundäre Befehle gehören in eine `ToolBar`, eine `MenuBar`, einen `SplitButton` oder ein Kontextmenü. Ein `SplitButton` passt, wenn ein Befehl eine Standardaktion und alternative Aktionen hat, etwa Save neben „Save and Close“. Destruktive Befehle brauchen eine Bestätigung, einen deaktivierten Zustand oder eine eindeutige Platzierung. `CheckBox` und `RadioButton` sind Zustands-Controls, keine Befehle: Sie gehören dorthin, wo der Benutzer eine Auswahl trifft, niemals dorthin, wo eine Aktion ausgelöst wird.

Jedes Feedback-Control hat genau eine Rolle. `ToolTip` und `HelpTip` geben Hinweise darauf, was ein Feld erwartet; sie sind keine Fehlerkanäle. `ErrorProvider` markiert das Control mit einem Validierungsfehler, und bei datengebundenen Controls muss er dem richtigen Container zugeordnet sein, sonst erscheinen seine Icons an der falschen Stelle. `MessageBox` ist blockierenden Entscheidungen und kritischen Bestätigungen vorbehalten. `AlertBox` und `Toast` sind für Statusmeldungen gedacht, die den Arbeitsablauf nicht unterbrechen sollen, und weil sich ein `Toast` wiederverwenden und aktualisieren lässt, während er sichtbar ist, dient er zugleich als Fortschrittsanzeige für einen lang laufenden Speichervorgang, neben einer `ProgressBar`, wenn der tatsächliche Fortschritt bekannt ist.

Validierung, die hilft, ohne zu blockieren

Validierung sollte sichtbar, konkret und behebbar sein, und sie sollte früh genug kommen, um zu helfen, ohne beim Tippen im Weg zu stehen. Der saubere Ablauf hat sechs Schritte. Erstens: Setzen Sie Editor-Properties (`MaxLength`, `Minimum`, `Maximum`, feste Einträge), damit sich unmögliche Werte gar nicht erst eingeben lassen. Zweitens: Verwenden Sie für typisierte Werte ein eingabespezifisches Parsen. Drittens: Setzen oder löschen Sie bei der Feldvalidierung im `Validating` -Handler des Controls die `ErrorProvider` -Meldung für genau dieses Control. Viertens: Führen Sie beim Speichern die Validierung auf Formularebene und anschließend auf Service-Ebene aus. Fünftens: Wenn etwas fehlschlägt, bleibt der Benutzer auf dem Bildschirm, das erste ungültige Control erhält den Fokus, und konkrete Fehler werden angezeigt. Sechstens: Wenn das Speichern gelingt, zeigen Sie eine nicht blockierende Bestätigung und aktualisieren den Statusbereich.

Die Meldung selbst sollte sagen, was zu korrigieren ist: „Enter a credit limit between 0 and 250,000“ statt „Invalid value“. Vermeiden Sie es, die gesamte Validierung in Click-Handler zu packen; das macht den Bildschirm schwer testbar und für Kollegen oder einen KI-Assistenten schwer erklärbar. Legen Sie jede Regel nahe an ihr Control oder in einen benannten Validator wie `ValidateRequiredName`, `ValidateEmail` und `ValidateCreditLimit`, und lassen Sie Save diese Methoden aufrufen. Halten Sie den Save-Pfad explizit – validieren, speichern, Zustand aktualisieren, benachrichtigen – und deaktivieren Sie die Befehle, solange er läuft, damit ein Doppelklick nicht zweimal absendet.

```

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;
}

```

Das CustomerEditor-UserControl

An dieser Stelle wird aus einer Sammlung von Controls eine Komponente auf Anwendungsebene. Der **CustomerEditor** aus dem Lab ist ein **UserControl**, dem seine Labels, seine Editoren, ein **ErrorProvider**, die Buttons Save, Reset und Validate sowie eine Validierungsmethode gehören, die ein klares Ergebnisobjekt zurückgibt: welche Controls ungültig sind und welche Meldungen angezeigt wurden. Die Shell-Seite, die ihn enthält, fragt nur „Ist das gültig?“ und „Gib mir den Kunden“; sie erfährt nie, wie die E-Mail-Regel funktioniert. Die Methode heißt **ValidateContent**, damit sie nicht mit den Validierungs-Membren verwechselt wird, die ein Container-Control bereits bereitstellt.

Die Gestaltungsregeln für Eingabebildschirme folgen aus dieser Grenze und bilden die Checkliste, nach der die Kursleitung bewertet:

- Jedes bearbeitbare Feld hat ein sichtbares Label oder eine barrierefreie Entsprechung.
- Der Editor passt zum Werttyp; Datums- und Zahlenwerte werden nicht ohne triftigen Grund aus gewöhnlichen **TextBox**-Controls geparkt.
- Validierungsmeldungen sind umsetzbar, **ErrorProvider** transportiert Feldfehler und **Toast** oder **AlertBox** die Rückmeldungen zum Arbeitsablauf.
- Befehle sind deaktiviert, solange ein Vorgang läuft, und der Save-Pfad ist explizit: validieren, speichern, Zustand aktualisieren, benachrichtigen.



Typische Fehler

MessageBox für jede kleine Statusmeldung verwenden. Einen teilweise ungültigen Zustand speichern, weil die Validierung nur für die sichtbaren Controls lief. Validierung in anonymen Event-Handlern verstecken. Den Wert einer **ComboBox** lesen, ohne Anzeigetext und gespeicherten Schlüssel zu unterscheiden. Vergessen, dass deaktivierte Buttons und Beschäftigt-Anzeigen zur Zuverlässigkeit gehören, nicht zur Dekoration.



Gewohnheit für die Control-Bibliothek

Wählen Sie das einfachste native Control, das die Aufgabe des Benutzers abbildet, bevor Sie eigenen Code schreiben. Native Controls bringen Designer-Unterstützung, Theming, Events und Serverzustand bereits mit; greifen Sie als Nächstes zu einem Extender oder einem UserControl und bauen Sie ein JavaScript-Widget erst, wenn kein natives Control passt.



Lernziele

Am Ende dieses Moduls können Sie:

- ✓ den richtigen Editor für Text-, Zahlen-, Datums-/Zeit-, maskierte und Auswahleingaben wählen.
- ✓ Button, SplitButton, CheckBox, RadioButton, ToolTip, HelpTip, ErrorProvider, AlertBox, MessageBox, Toast und ProgressBar angemessen einsetzen.
- ✓ eine Validierung umsetzen, die früh genug greift, um dem Benutzer zu helfen, aber nicht so aggressiv ist, dass sie die normale Dateneingabe blockiert.
- ✓ ein wiederverwendbares Editor-UserControl für Kunden- und Kontaktdaten erstellen.

Lab- und Prüfungsleitfaden

Was Sie im praktischen Lab bauen, das vorgeschlagene Vorgehen und die geforderten Ergebnisse.



Praktisches Lab

Erstellen Sie für die Operations Console ein CustomerEditor-UserControl mit beschrifteten Feldern für Name, E-Mail, Status, Startdatum, Kreditlimit und Kundentyp, und verwenden Sie für jeden Wert den passenden Editor statt überall eine TextBox. Ergänzen Sie ErrorProvider-Meldungen für Pflichtfelder, E-Mail-Format, Kreditlimit-Bereich und ungültige Datumsangaben; die Befehle Save, Reset und Validate mit deaktiviertem und „beschäftigtem“ Zustand während des Speicherns; und eine Toast- oder MessageBox-Meldung, wenn Validierung oder Speichern abgeschlossen sind.

Ausgangspunkt

Öffnen Sie die Projektmappe `OperationsConsole` aus Modul 1: Die Shell-Seite mit Navigations-, Inhalts-, Befehls- und Statusbereich läuft bereits, und die Platzhalterseite Editors ist leer. Legen Sie die Arbeit dieses Moduls in einen eigenen Ordner und Namespace `Editors`, damit das Abschlussprojekt das Control unverändert wiederverwenden kann.

Vorgeschlagenes Vorgehen

- ✓ Skizzieren Sie die sechs Felder und notieren Sie für jedes den Editor, bevor Sie den Designer öffnen: `TextBox` für Name und E-Mail, `ComboBox` für Status und Kundentyp, `DateTimePicker` für das Startdatum, `NumericUpDown` für das Kreditlimit.
- ✓ Fügen Sie ein UserControl `CustomerEditor` hinzu, platzieren Sie Labels und Editoren mit klaren Namen (`txtName`, `cboStatus`, `dtpStartDate`, `numCreditLimit`), setzen Sie `Minimum`, `Maximum`, `MaxLength` und feste Einträge, damit sich unmögliche Werte nicht eingeben lassen, und geben Sie jedem Editor einen `ToolTip`, der beschreibt, was erwartet wird.
- ✓ Fügen Sie einen `ErrorProvider` und benannte Validatoren hinzu (`ValidateRequiredName`, `ValidateEmail`, `ValidateCreditLimit`, `ValidateStartDate`), die eine konkrete Meldung setzen oder löschen; rufen Sie den passenden Validator aus dem `Validating`-Event jedes Editors auf und alle zusammen aus einer Methode `ValidateContent`, die ein Ergebnisobjekt zurückgibt.
- ✓ Fügen Sie die Buttons Save, Reset und Validate hinzu; lassen Sie Save zuerst `ValidateContent`, dann den Serviceaufruf und schließlich eine Aktualisierung des Zustands ausführen, und deaktivieren Sie während des Speicherns alle drei Befehle mit einer Beschäftigt-Anzeige.
- ✓ Zeigen Sie einen `Toast` oder eine `MessageBox`, wenn Validierung oder Speichern abgeschlossen sind, aktualisieren Sie den Statusbereich der Shell und platzieren Sie das fertige Control auf der Seite Editors.

Geforderte Ergebnisse

- ✓ CustomerEditor-UserControl mit beschrifteten Feldern für Name, E-Mail, Status, Startdatum, Kreditlimit und Kundentyp

- ✓ Passende Editoren: TextBox, DateTimePicker, NumericUpDown und ComboBox statt überall Freitext
- ✓ ErrorProvider-Meldungen für Pflichtfelder, E-Mail-Format, Kreditlimit-Bereich und ungültige Datumsangaben
- ✓ Befehle Save, Reset und Validate mit deaktiviertem und „beschäftigtem“ Zustand während des Speicherns
- ✓ Toast- oder MessageBox-Feedback, wenn Validierung oder Speichern abgeschlossen sind

Wiederholungsfragen

- ✓ Welche Validierungsfehler können dank des gewählten Editors gar nicht mehr auftreten, und welche brauchen weiterhin eine ErrorProvider-Meldung?
- ✓ Was macht Ihr Bildschirm, wenn der Benutzer innerhalb einer Sekunde zweimal auf Save klickt, und wo im Code ist das sichergestellt?
- ✓ Was weiß die Shell-Seite darüber, wie das E-Mail-Feld validiert wird, und was müsste sich ändern, wenn die Regel in einen Service wandert?



Gewohnheit für die Control-Bibliothek

Wählen Sie das einfachste native Control, das die Aufgabe des Benutzers abbildet, bevor Sie eigenen Code schreiben. Native Controls bringen Designer-Unterstützung, Theming, Events und Serverzustand bereits mit; greifen Sie als Nächstes zu einem Extender oder einem UserControl und bauen Sie ein JavaScript-Widget erst, wenn kein natives Control passt.