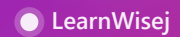


Integrierte Validierungsregeln und der Validation-Extender

Lektionsleitfaden · Lab- und Prüfungsleitfaden



Lektionsleitfaden

Der Validation-Extender von Wisej.NET: die erweiterte Eigenschaft ValidationRules zur Entwurfszeit, SetValidationRules zur Laufzeit, die Regeln Required, Email, Integer, Decimal, Currency, Telephone und Regex, eine Regelreihenfolge, die beim ersten Fehler stoppt, sowie ErrorProvider oder Label als Anzeigeziel. Was das für Entwickler bedeutet, die grundlegendes C# beherrschen und mit der Formularentwicklung in Wisej.NET beginnen – und wie Sie dieses Modul am besten durcharbeiten.



Fokus: Produktivbetrieb

Der Validation-Extender von Wisej.NET: die erweiterte Eigenschaft ValidationRules zur Entwurfszeit, SetValidationRules zur Laufzeit, die Regeln Required, Email, Integer, Decimal, Currency, Telephone und Regex, eine Regelreihenfolge, die beim ersten Fehler stoppt, sowie ErrorProvider oder Label als Anzeigeziel.

1. Das Problem im Produktivbetrieb

Am Ende von Modul 2 funktioniert das Aufnahmeformular von ValidationClinic, aber sehen Sie sich an, wie viel Code dafür nötig war: eine Hilfsmethode für den Namen, eine für die E-Mail-Adresse, eine für die Telefonnummer, und jede wiederholt dieselben Prüfungen „Ist es leer, hat es die richtige Form?“. Kommen Alter, Kreditlimit und Kundennummer hinzu, wächst das Formular um eine weitere Handvoll fast identischer Methoden. Die Prüfungen sind nicht falsch; sie werden nur für jedes Feld und jedes Formular erneut abgetippt. Dieses Modul ersetzt diese Wiederholung durch den **Validation**-Extender: Sie beschreiben, wie ein gültiges Feld aussieht, und Wisej.NET führt die Prüfungen für Sie aus.

2. Zentrale Konzepte

Konzept	Warum es wichtig ist
<code>Validation</code> -Extender	Eine nicht sichtbare Komponente, die Sie einem Formular hinzufügen; sie hängt sich an die Events <code>Validating</code> und <code>Validated</code> jedes Controls, das Regeln hat, sodass Sie keinen Handler pro Feld mehr schreiben.
erweiterte Eigenschaft	Eine Eigenschaft, die eine Komponente anderen Controls hinzufügt; sobald der Extender auf dem Formular liegt, zeigt jedes compatible Control im Eigenschaftfenster einen Eintrag <code>ValidationRules</code> .
<code>SetValidationRules</code>	Das Gegenstück zum Designer im Code: <code>validation.SetValidationRules(txtEmail, new ValidationRule[] { ... })</code> gibt einem Control zur Laufzeit seine Regelliste.
integrierte Regeln	Fertige Prüfungen für gängige Eingaben: <code>RequiredValidationRule</code> , <code>EmailValidationRule</code> , <code>IntegerValidationRule</code> , <code>DecimalValidationRule</code> , <code>CurrencyValidationRule</code> , <code>TelephoneValidationRule</code> und <code>RegexValidationRule</code> .
Regelreihenfolge	Regeln laufen in der Reihenfolge, in der Sie sie auflisten; die erste, die fehlschlägt, stoppt die übrigen und bricht das <code>Validating</code> -Event des Controls ab.
Anzeigeziel	Wo die <code>InvalidMessage</code> der fehlgeschlagenen Regel erscheint: am Control selbst, über ein <code>ErrorProvider</code> -Icon oder als Text in einem Label, jeweils erreicht über <code>validation.ErrorProvider</code> .
Hilfsklasse für die Regelkonfiguration	Eine kleine Methode, die die Regel-Arrays für gängige Felder aufbaut, sodass ein zweites Formular durch ihren Aufruf dieselben Prüfungen und dieselben Formulierungen erhält.

3. Der rote Faden der Lektion

Eine Komponente, zwei Wege der Konfiguration

Der `Validation` -Extender ist eine Komponente, wie ein Timer oder der `ErrorProvider` aus Modul 2: Er hat keine eigene sichtbare Oberfläche. Ziehen Sie ihn im Designer von Visual Studio aus der Toolbox auf das Aufnahmeformular, und er erscheint im Komponentenbereich unter dem Formular. Ab diesem Moment erhält jedes compatible Control im Formular im Eigenschaftfenster einen neuen Eintrag namens `ValidationRules`. Wählen Sie `txtEmail` aus, öffnen Sie diesen Auflistungs-Editor und fügen Sie die Regeln hinzu, die das Feld braucht. Der Designer schreibt sie in die Designer-Datei des Formulars, sodass jeder, der das Formular öffnet, sehen kann, welche Felder Pflichtfelder sind, ohne einen einzigen Event-Handler zu lesen.

Dasselbe geht auch im Code, und genau das verwendet das Lab. Sie erzeugen die Komponente, richten ihren `ContainerControl` auf das Formular aus und rufen `SetValidationRules` einmal pro Control auf, üblicherweise aus einer Methode `ConfigureValidationRules`, die im Konstruktor des Formulars aufgerufen wird. Beide Wege führen zum selben Ergebnis: Der Extender hängt sich an die Validierungs-Events des Controls und führt die Regeln aus, wenn der Benutzer das Feld verlässt – genau in dem Moment, den Sie in Modul 1 von Hand behandelt haben.

```

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." }
});

```

Die sieben integrierten Regeln

Jede Regel beantwortet eine Frage zum Text in einem Control. **RequiredValidationRule** fragt „Steht hier überhaupt etwas?“ und hat drei nützliche Eigenschaften: **Required** schaltet die Prüfung ein, **Trim** entfernt zuerst Leerzeichen, damit ein Feld, das nur Leerzeichen enthält, trotzdem als leer gilt, und **MinimumLength** verlangt eine Mindestanzahl an Zeichen. **EmailValidationRule** prüft, ob der Wert wie eine E-Mail-Adresse aussieht. **IntegerValidationRule** akzeptiert ganze Zahlen wie ein Alter, während **DecimalValidationRule** Zahlen mit Nachkommastellen akzeptiert. **CurrencyValidationRule** prüft einen Geldbetrag wie das Kreditlimit und kann den Wert formatieren, sobald er akzeptiert ist.

TelephoneValidationRule prüft eine Telefonnummer anhand einer Maske, und **RegexValidationRule** vergleicht den Wert mit einem regulären Ausdruck; so erzwingt ValidationClinic eine Kundennummer aus den Buchstaben CUS, einem Bindestrich und vier Ziffern.

Jede Regel trägt eine **InvalidMessage**. Diesen Text liest der Benutzer, schreiben Sie ihn also als Produkttext, der sagt, was als Nächstes zu tun ist: „Age must be a whole number.“ statt „Integer parse failed“.

Regelreihenfolge und der erste Fehlschlag

Die Regeln eines Controls laufen von der ersten bis zur letzten. Sobald eine fehlschlägt, werden die übrigen Regeln übersprungen, und das **Validating** -Event des Controls wird abgebrochen; das hat dieselbe Wirkung, als würden Sie wie in Modul 1 selbst **e.Cancel = true** setzen. Es wird nur eine Meldung angezeigt: die Meldung der Regel, die zuerst fehlgeschlagen ist.

Deshalb ist die Reihenfolge eine Designentscheidung. Setzen Sie **RequiredValidationRule** bei jedem Pflichtfeld an die erste Stelle. Bleibt das E-Mail-Feld leer, liest der Benutzer „Email is required.“, und das ist das eigentliche Problem. Stünde die Email-Regel vorn, würde dasselbe leere Feld „Enter a valid email address.“ melden – technisch richtig, aber deutlich weniger hilfreich. Im Lab weisen Sie das mit einem leeren **txtEmail** nach: Required schlägt fehl, Email läuft nie, und der Fokus verhält sich entsprechend dem **AutoValidate** -Modus, den Sie zuvor gewählt haben. Optionale Felder wie das Kreditlimit beginnen einfach mit ihrer Formatregel.

Festlegen, wo die Meldung erscheint

Ohne Anzeigeziel schreibt eine fehlgeschlagene Regel ihre Meldung als Ungültig-Tooltip zurück an das Control selbst. Weisen Sie einen `ErrorProvider` der Eigenschaft `validation.ErrorProvider` zu, ruft der Extender stattdessen dort `SetError` auf, und Sie erhalten Icon und Tooltip aus Modul 2, ohne in Ihrem eigenen Code `SetError` aufzurufen. Auch ein Label kann das Ziel sein: Es zeigt die Meldung als einfachen Text an und blendet sich aus, wenn es nichts zu sagen gibt, was zu einem kurzen Formular oder zu einem Feld passt, das sichtbaren Text statt eines Icons braucht. Jede Klasse, die `IErrorProvider` implementiert, kann diese Rolle übernehmen; Modul 7 baut einen, der ein Zusammenfassungs-Panel füllt.

Der Extender löst außerdem eigene `Validating` - und `Validated` -Events für jedes Control aus, das Regeln hat. Damit haben Sie eine zentrale Stelle, um `validationSummaryLabel` zu aktualisieren, sodass die sichtbare Zusammenfassung aus Modul 2 mit den Icons synchron bleibt, obwohl Sie die Prüfungen nicht mehr selbst schreiben.

Wiederverwendbare Regelkonfigurationen

E-Mail, Alter und Telefon kommen in mehr als einer Maske von ValidationClinic vor. Statt die Arrays zu kopieren, legen Sie sie in eine kleine Hilfsklasse, etwa eine statische Klasse `ClinicRules` mit Methoden wie `RequiredEmail()` und `RequiredAge()`, die jeweils ein neues `ValidationRule[]` zurückgeben. Das Aufnahmeformular und jedes weitere Formular rufen dieselben Methoden auf, sodass Reihenfolge und Formulierung jeder Meldung nur einmal festgelegt werden.

Beschränken Sie die Hilfsklasse auf die Form der Felder. Regeln, die zwei Felder vergleichen, etwas in einer Datenbank nachschlagen oder eine Geschäftsrichtlinie ausdrücken, gehören nicht in den Extender, weil sie nur laufen, wenn ein Control den Fokus verliert. Modul 4 verschiebt sie in Validierungsmethoden, einen Validator-Service und eigene `ValidationRule` -Klassen.



Gewohnheit: Validierungsebenen

Benennen Sie die Ebene, bevor Sie die Prüfung schreiben. Die Feldvalidierung schützt das Bearbeiten und läuft, wenn der Benutzer ein Control verlässt; die Modellvalidierung schützt die Daten und führt dieselben Regeln aus dem Formular, dem Grid und jedem Service aus; die Validierung beim Speichern schützt die Datenbank und führt jede Ebene noch einmal aus, bevor irgendetwas geschrieben wird. Zeigen Sie jeden Fehler nahe an seinem Feld und in einer sichtbaren Zusammenfassung, in verständlicher Sprache, die sagt, was als Nächstes zu tun ist, und halten Sie Fehler, die der Benutzer selbst korrigieren kann, getrennt von Systemfehlern.



Lernziele

Am Ende dieses Moduls können Sie:

- ✓ den Validation-Extender von Wisej.NET zur Entwurfszeit und zur Laufzeit einsetzen.
- ✓ die Regeln Required, Email, Integer, Decimal, Currency, Telephone und Regex anwenden.
- ✓ die Regelreihenfolge erklären und begründen, warum die Validierung beim ersten Fehlschlag stoppt.
- ✓ einen ErrorProvider oder ein Label zuweisen, um Validierungsmeldungen anzuzeigen.

- ✓ wiederverwendbare Regelkonfigurationen für gängige Formulare erstellen.

Lab- und Prüfungsleitfaden

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

Praxis-Lab

Ersetzen Sie die handgeschriebenen Feld-Handler in ValidationClinic durch den Validation-Extender. Fügen Sie dem Aufnahmeformular eine Validation-Komponente und einen ErrorProvider hinzu, setzen Sie ContainerControl und validation.ErrorProvider und rufen Sie dann SetValidationRules in einer Methode ConfigureValidationRules auf: Required und danach Email auf txtEmail, Required und danach Integer auf txtAge, Required und danach Telephone auf txtPhone, Currency auf txtCreditLimit und eine Regex-Regel für die Kundennummer. Setzen Sie Required überall an die erste Stelle, damit die hilfreichste Meldung gewinnt, hängen Sie sich an das Validating-Event des Extenders, um die Zusammenfassung zu aktualisieren, und verschieben Sie die Regel-Arrays in eine kleine wiederverwendbare Hilfsklasse, damit ein anderes Formular dieselbe Konfiguration anwenden kann.

Ausgangspunkt

Sie öffnen ValidationClinic in dem Zustand, in dem Modul 2 es hinterlassen hat: Das Aufnahmeformular hat txtName , txtEmail , txtPhone und txtAge , einen ErrorProvider , die Hilfsmethoden ValidateName , ValidateEmail und ValidatePhone sowie ein validationSummaryLabel . Fügen Sie für dieses Lab die Textboxen txtCreditLimit und txtCustomerCode hinzu, falls sie noch nicht im Formular sind.

Vorgeschlagenes Vorgehen

- ✓ Fügen Sie dem Aufnahmeformular im Designer eine Validation -Komponente hinzu, wählen Sie txtEmail aus und fügen Sie über den Auflistungs-Editor ValidationRules eine Required- und eine Email-Regel hinzu; starten Sie das Formular dann einmal, um den Extender in Aktion zu sehen, bevor Sie Code schreiben.
- ✓ Entfernen Sie diese Designer-Regeln wieder und schreiben Sie ConfigureValidationRules , aufgerufen aus dem Konstruktor: Setzen Sie errorProvider.ContainerControl , validation.ContainerControl und validation.ErrorProvider und rufen Sie dann SetValidationRules für E-Mail, Alter, Telefon, Kreditlimit und Kundennummer auf.
- ✓ Geben Sie jedem Pflichtfeld RequiredValidationRule als erste Regel, mit Trim = true , wo Leerzeichen nicht zählen sollen, und formulieren Sie jede InvalidMessage als Satz, der dem Benutzer sagt, was er eingeben soll.
- ✓ Behandeln Sie die Events Validating und Validated des Extenders an einer Stelle und bauen Sie validationSummaryLabel aus den aktuellen Fehlern neu auf; blenden Sie es aus, wenn keine mehr übrig sind.
- ✓ Verschieben Sie die Regel-Arrays in eine statische Hilfsklasse ClinicRules , deren Methoden ein neues ValidationRule[] zurückgeben, rufen Sie sie aus dem Aufnahmeformular auf und testen Sie dann eine leere E-Mail-Adresse, eine falsch aufgebaute E-Mail-Adresse, eine gültige E-Mail-Adresse, ein nicht numerisches Alter und eine falsche Kundennummer.

Geforderte Ergebnisse

- ✓ Validation-Komponente, die mit dem Formular und einem ErrorProvider als Anzeigeziel verbunden ist
- ✓ Methode ConfigureValidationRules, die SetValidationRules für E-Mail, Alter, Telefon, Kreditlimit und Kundennummer aufruft
- ✓ Required-Regel an erster Stelle auf jedem Pflichtfeld, mit verständlichem InvalidMessage-Text
- ✓ Validating-Handler des Extenders, der die Validierungszusammenfassung mit den Regeln synchron hält
- ✓ Wiederverwendbare Hilfsklasse für die Regelkonfiguration, die ein zweites Formular aufrufen kann

Wiederholungsfragen

- ✓ Sie vertauschen die Reihenfolge, sodass auf `txtEmail` die Email-Regel vor Required läuft. Beschreiben Sie genau, was ein Benutzer sieht, nachdem er das Feld leer verlassen hat, und warum diese Meldung weniger nützlich ist.
- ✓ Die Hilfsmethoden aus Modul 2 und der Extender enden beide mit einem Icon neben dem Feld. Was schreiben Sie nicht mehr von Hand, und was mussten Sie trotzdem selbst entscheiden?
- ✓ Eine Kollegin möchte „Das Enddatum muss nach dem Startdatum liegen“ in `ClinicRules` aufnehmen. Warum ist der Extender der falsche Ort für diese Prüfung, und wann würde sie nicht laufen?



Gewohnheit: Validierungsebenen

Benennen Sie die Ebene, bevor Sie die Prüfung schreiben. Die Feldvalidierung schützt das Bearbeiten und läuft, wenn der Benutzer ein Control verlässt; die Modellvalidierung schützt die Daten und führt dieselben Regeln aus dem Formular, dem Grid und jedem Service aus; die Validierung beim Speichern schützt die Datenbank und führt jede Ebene noch einmal aus, bevor irgendetwas geschrieben wird. Zeigen Sie jeden Fehler nahe an seinem Feld und in einer sichtbaren Zusammenfassung, in verständlicher Sprache, die sagt, was als Nächstes zu tun ist, und halten Sie Fehler, die der Benutzer selbst korrigieren kann, getrennt von Systemfehlern.