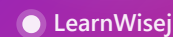


Règles de validation intégrées et extender Validation

Guide de la leçon · Guide du lab / de l'examen



Guide de la leçon

L'extender Validation de Wisej.NET : la propriété étendue ValidationRules au moment de la conception, SetValidationRules à l'exécution, les règles Required, Email, Integer, Decimal, Currency, Telephone et Regex, l'ordre des règles qui s'arrête au premier échec, et les cibles d'affichage ErrorProvider ou Label. Ce que cela signifie pour un développeur qui connaît les bases de C# et débute dans le développement de formulaires Wisej.NET — et comment aborder ce module.



Objectif production

L'extender Validation de Wisej.NET : la propriété étendue ValidationRules au moment de la conception, SetValidationRules à l'exécution, les règles Required, Email, Integer, Decimal, Currency, Telephone et Regex, l'ordre des règles qui s'arrête au premier échec, et les cibles d'affichage ErrorProvider ou Label.

1. Le problème en production

À la fin du Module 2, le formulaire d'inscription de ValidationClinic fonctionne, mais regardez la quantité de code nécessaire : un utilitaire pour le nom, un pour l'adresse e-mail, un pour le téléphone, et chacun répète les mêmes vérifications « est-ce vide, est-ce la bonne forme ». Ajoutez l'âge, la limite de crédit et un code client, et le formulaire s'alourdit d'une nouvelle poignée de méthodes presque identiques. Les vérifications ne sont pas fausses ; elles sont simplement retapées pour chaque champ et chaque formulaire. Ce module remplace cette répétition par l'extender **Validation**, où vous décrivez à quoi ressemble un champ valide et où Wisej.NET exécute les vérifications pour vous.

2. Concepts clés

Concept	Pourquoi c'est important
Extender <code>Validation</code>	Un composant non visuel que vous ajoutez à un formulaire ; il s'attache aux événements <code>Validating</code> et <code>Validated</code> de chaque contrôle qui a des règles, si bien que vous n'écrivez plus un gestionnaire par champ.
propriété étendue	Une propriété qu'un composant ajoute à d'autres contrôles ; dès que l'extender est sur le formulaire, chaque contrôle compatible affiche une entrée <code>ValidationRules</code> dans la fenêtre Propriétés.
<code>SetValidationRules</code>	L'équivalent en code du designer : <code>validation.SetValidationRules(txtEmail, new ValidationRule[] { ... })</code> attribue à un contrôle sa liste de règles à l'exécution.
règles intégrées	Des vérifications prêtes à l'emploi pour les saisies courantes : <code>RequiredValidationRule</code> , <code>EmailValidationRule</code> , <code>IntegerValidationRule</code> , <code>DecimalValidationRule</code> , <code>CurrencyValidationRule</code> , <code>TelephoneValidationRule</code> et <code>RegexValidationRule</code> .
ordre des règles	Les règles s'exécutent dans l'ordre où vous les listez ; la première qui échoue arrête les suivantes et annule l'événement <code>Validating</code> du contrôle.
cible d'affichage	L'endroit où apparaît l' <code>InvalidMessage</code> de la règle en échec : sur le contrôle lui-même, via une icône <code>ErrorProvider</code> ou sous forme de texte dans un Label, le tout atteint par <code>validation.ErrorProvider</code> .
utilitaire de configuration des règles	Une petite méthode qui construit les tableaux de règles des champs courants, pour qu'un second formulaire obtienne les mêmes vérifications et la même formulation en l'appelant.

3. Déroulé de la leçon

Un composant, deux manières de le configurer

L'extender `Validation` est un composant, comme un timer ou l'`ErrorProvider` du Module 2 : il n'a pas de surface visible propre. Déposez-le depuis la boîte à outils sur le formulaire d'inscription dans le designer de Visual Studio, et il apparaît dans la barre des composants sous le formulaire. Dès cet instant, chaque contrôle compatible du formulaire gagne dans la fenêtre Propriétés une nouvelle entrée appelée `ValidationRules`. Sélectionnez `txtEmail`, ouvrez cet éditeur de collection et ajoutez les règles dont le champ a besoin. Le designer les écrit dans le fichier designer du formulaire, si bien que quiconque ouvre le formulaire voit quels champs sont obligatoires sans lire un seul gestionnaire d'événements.

On peut faire la même chose en code, et c'est ce qu'utilise le lab. Vous créez le composant, faites pointer son `ContainerControl` vers le formulaire et appelez `SetValidationRules` une fois par contrôle, généralement depuis une méthode `ConfigureValidationRules` appelée dans le constructeur du formulaire. Les deux voies produisent le même résultat : l'extender s'accroche aux événements de validation du contrôle et exécute les règles quand l'utilisateur quitte le champ, exactement le moment que le Module 1 vous a appris à gérer à la main.

```

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." }
});

```

Les sept règles intégrées

Chaque règle répond à une question sur le texte d'un contrôle. **RequiredValidationRule** demande « y a-t-il quelque chose ici ? » et possède trois propriétés utiles : **Required** active la vérification, **Trim** supprime d'abord les espaces pour qu'un champ ne contenant que des blancs compte toujours comme vide, et **MinimumLength** exige un nombre minimal de caractères. **EmailValidationRule** vérifie que la valeur ressemble à une adresse e-mail. **IntegerValidationRule** accepte des nombres entiers comme un âge, tandis que **DecimalValidationRule** accepte des nombres avec une partie décimale.

CurrencyValidationRule vérifie un montant monétaire comme la limite de crédit et peut mettre en forme la valeur une fois acceptée. **TelephoneValidationRule** vérifie un numéro de téléphone par rapport à un masque, et **RegexValidationRule** compare la valeur à une expression régulière, ce qui permet à ValidationClinic d'imposer un code client composé des lettres CUS, d'un tiret et de quatre chiffres.

Chaque règle porte un **InvalidMessage**. C'est le texte que lit l'utilisateur ; rédigez-le donc comme un texte produit qui dit quoi faire ensuite : « Age must be a whole number. » plutôt que « Integer parse failed ».

L'ordre des règles et le premier échec

Les règles d'un contrôle s'exécutent de la première à la dernière. Dès que l'une échoue, les règles restantes sont ignorées et l'événement **Validating** du contrôle est annulé, ce qui a le même effet que de définir vous-même **e.Cancel = true** comme au Module 1. Un seul message est affiché : celui de la première règle en échec.

C'est pourquoi l'ordre est une décision de conception. Placez **RequiredValidationRule** en premier sur chaque champ obligatoire. Laissez le champ e-mail vide et l'utilisateur lit « Email is required. », qui est le vrai problème. Si la règle Email venait en premier, le même champ vide afficherait « Enter a valid email address. », ce qui est techniquement vrai et bien moins utile. Dans le lab, vous le prouverez avec un **txtEmail** vide : Required échoue, Email ne s'exécute jamais, et le focus se comporte selon le mode **AutoValidate** choisi précédemment. Les champs facultatifs, comme la limite de crédit, commencent simplement par leur règle de format.

Choisir où apparaît le message

Sans cible d’affichage, une règle en échec écrit son message sur le contrôle lui-même, dans son infobulle d’invalidité. Affectez un `ErrorProvider` à `validation.ErrorProvider` et l’extender appelle `SetError` sur celui-ci à la place, ce qui vous donne l’icône et l’infobulle du Module 2 sans aucun appel à `SetError` dans votre propre code. Un Label peut aussi servir de cible : il affiche le message en texte simple et se masque quand il n’y a rien à dire, ce qui convient à un formulaire court ou à un champ qui a besoin d’un texte visible plutôt que d’une icône. Toute classe qui implémente `IErrorProvider` peut jouer ce rôle ; le Module 7 en construit une qui alimente un panneau récapitulatif.

L’extender déclenche aussi ses propres événements `Validating` et `Validated` pour chaque contrôle qui a des règles. Cela vous donne un endroit central pour rafraîchir `validationSummaryLabel`, afin que le récapitulatif visible du Module 2 reste synchronisé avec les icônes alors même que vous n’écrivez plus les vérifications vous-même.

Des configurations de règles réutilisables

L’e-mail, l’âge et le téléphone apparaissent sur plusieurs écrans de ValidationClinic. Au lieu de copier les tableaux, placez-les dans un petit utilitaire, par exemple une classe statique `ClinicRules` avec des méthodes comme `RequiredEmail()` et `RequiredAge()` qui renvoient chacune un nouveau `ValidationRule[]`. Le formulaire d’inscription et tout second formulaire appellent les mêmes méthodes, donc l’ordre et la formulation de chaque message sont décidés une seule fois.

Limitez l’utilitaire à la forme des champs. Les règles qui comparent deux champs, recherchent quelque chose dans une base de données ou expriment une politique métier n’ont pas leur place dans l’extender, car elles ne s’exécuteraient que lorsqu’un contrôle perd le focus. Le Module 4 les déplace dans des méthodes de validation, un service de validation et des classes `ValidationRule` personnalisées.



Réflexe des couches de validation

Nommez la couche avant d’écrire la vérification. La validation des champs protège la saisie et s’exécute quand l’utilisateur quitte un contrôle ; la validation du modèle protège les données et applique les mêmes règles depuis le formulaire, la grille et n’importe quel service ; la validation à l’enregistrement protège la base de données et réexécute toutes les couches avant que quoi que ce soit ne soit écrit. Affichez chaque erreur près de son champ et dans un récapitulatif visible, dans un langage simple qui dit quoi faire ensuite, et séparez les erreurs que l’utilisateur peut corriger des pannes système.



Objectifs d’apprentissage

À la fin de ce module, vous serez capable de :

- ✓ Utiliser l’extender Validation de Wisej.NET au moment de la conception et à l’exécution.
- ✓ Appliquer les règles Required, Email, Integer, Decimal, Currency, Telephone et Regex.
- ✓ Expliquer l’ordre des règles et pourquoi la validation s’arrête au premier échec.
- ✓ Affecter un ErrorProvider ou un Label pour afficher les messages de validation.
- ✓ Créer des configurations de règles réutilisables pour les formulaires courants.

Guide du lab / de l'examen

Ce que vous allez construire dans le lab pratique, l'approche suggérée et les livrables attendus.



Lab pratique

Remplacez les gestionnaires de champ écrits à la main dans ValidationClinic par l'extender Validation. Ajoutez un composant Validation et un ErrorProvider au formulaire d'inscription, définissez ContainerControl et validation.ErrorProvider, puis appelez SetValidationRules dans une méthode ConfigureValidationRules : Required puis Email sur txtEmail, Required puis Integer sur txtAge, Required puis Telephone sur txtPhone, Currency sur txtCreditLimit et une règle Regex pour le code client. Placez Required en premier partout pour que le message le plus utile l'emporte, abonnez-vous à l'événement Validating de l'extender pour rafraîchir le récapitulatif, et déplacez les tableaux de règles dans un petit utilitaire réutilisable pour qu'un autre formulaire puisse appliquer la même configuration.

Point de départ

Vous ouvrez ValidationClinic tel que l'a laissé le Module 2 : le formulaire d'inscription comporte `txtName`, `txtEmail`, `txtPhone` et `txtAge`, un `ErrorProvider`, les utilitaires `ValidateName`, `ValidateEmail` et `ValidatePhone` et un `validationSummaryLabel`. Ajoutez une TextBox `txtCreditLimit` et une TextBox `txtCustomerCode` pour ce lab si elles ne sont pas encore sur le formulaire.

Approche suggérée

- ✓ Ajoutez un composant `Validation` au formulaire d'inscription dans le designer, sélectionnez `txtEmail` et ajoutez une règle Required et une règle Email via l'éditeur de collection `ValidationRules`, puis exécutez le formulaire une fois pour voir l'extender fonctionner avant d'écrire le moindre code.
- ✓ Supprimez ces règles du designer et écrivez `ConfigureValidationRules`, appelée depuis le constructeur : définissez `errorProvider.ContainerControl`, `validation.ContainerControl` et `validation.ErrorProvider`, puis appelez `SetValidationRules` pour l'e-mail, l'âge, le téléphone, la limite de crédit et le code client.
- ✓ Donnez à chaque champ obligatoire `RequiredValidationRule` comme première règle, avec `Trim = true` là où les espaces ne doivent pas compter, et rédigez chaque `InvalidMessage` comme une phrase qui indique à l'utilisateur quoi saisir.
- ✓ Gérez les événements `Validating` et `Validated` de l'extender en un seul endroit et reconstruisez `validationSummaryLabel` à partir des erreurs en cours, en le masquant quand il n'en reste aucune.
- ✓ Déplacez les tableaux de règles dans un utilitaire statique `ClinicRules` dont les méthodes renvoient un nouveau `ValidationRule[]`, appelez-le depuis le formulaire d'inscription, puis testez une adresse e-mail vide, une adresse mal formée, une adresse valide, un âge non numérique et un code client incorrect.

Livrables attendus

- ✓ Composant Validation relié au formulaire et à une cible d'affichage ErrorProvider

- ✓ Méthode ConfigureValidationRules appelant SetValidationRules pour l'e-mail, l'âge, le téléphone, la limite de crédit et le code client
- ✓ Règle Required placée en premier sur chaque champ obligatoire, avec un texte InvalidMessage en langage simple
- ✓ Gestionnaire Validating de l'extender qui garde le récapitulatif de validation synchronisé avec les règles
- ✓ Utilitaire de configuration des règles réutilisable qu'un second formulaire peut appeler

Questions de révision

- ✓ Vous inversez l'ordre pour que la règle Email s'exécute avant Required sur `txtEmail`. Décrivez précisément ce que voit un utilisateur après avoir quitté le champ vide, et pourquoi ce message est moins utile.
- ✓ Les utilitaires du Module 2 et l'extender aboutissent tous deux à une icône à côté du champ. Qu'avez-vous cessé d'écrire à la main, et que deviez-vous encore décider vous-même ?
- ✓ Un collègue veut ajouter « la date de fin doit être postérieure à la date de début » à `ClinicRules`. Pourquoi l'extender n'est-il pas le bon endroit pour cette vérification, et quand ne s'exécuterait-elle pas ?



Réflexe des couches de validation

Nommez la couche avant d'écrire la vérification. La validation des champs protège la saisie et s'exécute quand l'utilisateur quitte un contrôle ; la validation du modèle protège les données et applique les mêmes règles depuis le formulaire, la grille et n'importe quel service ; la validation à l'enregistrement protège la base de données et réexécute toutes les couches avant que quoi que ce soit ne soit écrit. Affichez chaque erreur près de son champ et dans un récapitulatif visible, dans un langage simple qui dit quoi faire ensuite, et séparez les erreurs que l'utilisateur peut corriger des pannes système.