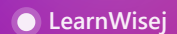


Éditeurs, boutons, validation et retour utilisateur

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



Guide de la leçon

Choisir l'éditeur adapté à la valeur, un modèle de commandes pour les boutons, une validation `ErrorProvider` qui aide plutôt qu'elle ne bloque, et un retour non bloquant avec `Toast` et `AlertBox`. Ce que cela signifie pour un développeur `Wisej.NET` — et comment aborder ce module.



Objectif en production

Choisir l'éditeur adapté à la valeur, un modèle de commandes pour les boutons, une validation `ErrorProvider` qui aide plutôt qu'elle ne bloque, et un retour non bloquant avec `Toast` et `AlertBox`.

1. Le problème en production

Les éditeurs sont le premier endroit où une bibliothèque de contrôles devient un système métier. Un formulaire qui semble correct mais accepte des saisies ambiguës ou invalides génère des tickets de support et des scripts de nettoyage longtemps après sa mise en production. Les éditeurs `Wisej.NET` suivent le modèle de composants `.NET` familier mais s'exécutent dans le navigateur, si bien qu'une même règle métier peut être appliquée à plusieurs niveaux : propriétés des éditeurs, extenders de validation, gestionnaires d'événements, liaison de données, validation du service et contraintes de la base de données. Le rôle de la couche d'interface est d'éviter les erreurs évidentes et d'expliquer clairement les erreurs ; elle ne doit jamais être la seule couche qui protège l'application.

2. Concepts clés

Concept	Pourquoi c'est important
éditeur adapté à la valeur	L'éditeur dont le type contraint déjà la saisie : <code>DateTimePicker</code> pour les dates, <code>NumericUpDown</code> pour les nombres bornés, <code>ComboBox</code> pour les valeurs de liste, afin que le serveur n'analyse jamais de texte ambigu.
modèle de commandes	Une répartition délibérée entre commandes principales (Save, Approve, Search, Send, Export), commandes secondaires (ToolBar, MenuBar, SplitButton, menu contextuel) et commandes destructrices qui exigent une confirmation ou un emplacement réfléchi.
ErrorProvider	L'extender qui attache un message précis et exploitable au contrôle erroné, au lieu d'un « Valeur invalide » générique ailleurs sur la page.
retour bloquant ou non bloquant	<code>MessageBox</code> interrompt pour une décision ou une confirmation critique ; <code>AlertBox</code> et <code>Toast</code> signalent un état sans arrêter le déroulement.
validateurs nommés	Des méthodes comme <code>ValidateEmail</code> ou <code>ValidateCreditLimit</code> qui portent chacune une règle, pour que la règle soit testable et explicable au lieu d'être enfouie dans un gestionnaire de clic.
état d'attente	Désactiver Save et afficher une progression pendant une opération, ce qui empêche les doubles envois et relève de la fiabilité, pas de la décoration.
UserControl d'édition	Une composition de libellés, d'éditeurs, d'un <code>ErrorProvider</code> et d'une méthode de validation qui renvoie un objet résultat, pour que le reste de l'application ne sache jamais comment le champ e-mail est vérifié.

3. Le fil de la leçon

Adapter l'éditeur à la valeur

La première décision sur tout écran de saisie consiste à choisir l'éditeur qui représente chaque valeur : prenez le contrôle dont le type exclut déjà les saisies impossibles. `TextBox` sert au texte ordinaire. N'activez `Multiline` que lorsque l'utilisateur doit saisir des paragraphes, pas parce qu'un champ se trouve être large, et utilisez `Watermark` comme exemple ou invite, jamais comme substitut à un libellé. Quand la forme de la saisie est prévisible, comme un code, un motif de type numéro de téléphone ou un numéro de référence, `MaskedTextBox` impose cette forme pendant la frappe. Quand l'analyse et la mise en forme d'une valeur typée sont au cœur du champ, `TypedTextBox` effectue la conversion pour vous.

Les dates et les nombres méritent leurs propres éditeurs. `DateTimePicker` supprime toute ambiguïté d'analyse et valide une sélection de date bien plus naturellement qu'un texte libre. `NumericUpDown`, `DomainUpDown` et `TimeUpDown` expriment des valeurs contraintes avec un minimum, un maximum et un pas. `ComboBox` est le bon choix quand une valeur provient d'une liste ; son `DropDownStyle` détermine si l'utilisateur peut aussi saisir une valeur absente de la liste. C'est un mauvais choix quand l'utilisateur doit rechercher ou parcourir un très gros jeu de données : il est conçu pour une sélection compacte dans une liste petite ou moyenne.

Dans Operations Console, le `CustomerEditor` applique exactement cette correspondance : le nom et l'e-mail sont des contrôles `TextBox`, le statut et le type de client des contrôles `ComboBox` à éléments fixes, la date de début un `DateTimePicker` et la limite de crédit un `NumericUpDown` avec une plage. Une habitude compte avec les listes : séparez le texte affiché de la clé stockée, pour que « Active » à l'écran ne devienne jamais la valeur que vous persistez.

Les boutons comme commandes, et les contrôles de retour

Les boutons ne sont pas de simples rectangles cliquables ; un écran métier a besoin d'un modèle de commandes. Les commandes principales (Save, Approve, Search, Send, Export) sont de simples contrôles `Button` placés là où l'œil les attend. Les commandes secondaires ont leur place dans une `ToolBar`, une `MenuBar`, un `SplitButton` ou un menu contextuel. Un `SplitButton` convient quand une commande a une action par défaut et des actions alternatives, comme Save à côté de Save and Close. Les commandes destructrices exigent une confirmation, un état désactivé ou un emplacement clair. `CheckBox` et `RadioButton` sont des contrôles d'état, pas des commandes : ils ont leur place là où l'utilisateur fait un choix, jamais là où une action se déclenche.

Les contrôles de retour ont chacun un rôle. `ToolTip` et `HelpTip` guident sur ce qu'un champ attend ; ce ne sont pas des canaux d'erreur. `ErrorProvider` signale le contrôle qui présente une erreur de validation, et pour les contrôles liés aux données, il doit être associé au bon conteneur, sinon ses icônes apparaissent au mauvais endroit. `MessageBox` est réservée à une décision bloquante ou à une confirmation critique. `AlertBox` et `Toast` servent aux messages d'état qui ne doivent pas interrompre le déroulement, et comme un `Toast` peut être réutilisé et mis à jour tant qu'il est visible, il sert aussi de retour de progression pour un enregistrement long, aux côtés d'une `ProgressBar` quand la progression réelle est connue.

Une validation qui aide sans bloquer

La validation doit être visible, précise et rattrapable, et intervenir assez tôt pour aider sans gêner la saisie. Le flux propre compte six étapes. Premièrement, définissez les propriétés des éditeurs (`MaxLength`, `Minimum`, `Maximum`, éléments fixes) pour que les valeurs impossibles ne puissent pas du tout être saisies. Deuxièmement, utilisez une analyse propre au type de saisie pour les valeurs typées. Troisièmement, lors de la validation du champ, dans le gestionnaire `Validating` du contrôle, définissez ou effacez le message `ErrorProvider` de ce seul contrôle. Quatrièmement, lors de Save, exécutez la validation au niveau du formulaire puis celle du service. Cinquièmement, en cas d'échec, gardez l'utilisateur sur l'écran, donnez le focus au premier contrôle invalide et affichez des erreurs précises. Sixièmement, quand l'enregistrement réussit, affichez une confirmation non bloquante et mettez à jour la zone d'état.

Le message lui-même doit dire quoi corriger : « Saisissez une limite de crédit comprise entre 0 et 250 000 » plutôt que « Valeur invalide ». Évitez de regrouper toute la validation dans les gestionnaires de clic ; l'écran devient difficile à tester et difficile à expliquer pour un collègue ou un assistant IA. Placez chaque règle près de son contrôle ou dans un validateur nommé comme `ValidateRequiredName`, `ValidateEmail` et `ValidateCreditLimit`, et laissez Save appeler ces méthodes. Gardez le chemin de Save explicite (valider, persister, rafraîchir l'état, notifier) et désactivez les commandes pendant son exécution pour qu'un double clic ne puisse pas envoyer deux fois.

```

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;
}

```

Le UserControl CustomerEditor

C'est le moment où un ensemble de contrôles devient un composant de niveau application. Le `CustomerEditor` du lab est un `UserControl` qui possède ses libellés, ses éditeurs, un `ErrorProvider`, les boutons Save, Reset et Validate et une méthode de validation qui renvoie un objet résultat clair : quels contrôles sont invalides et quels messages ont été affichés. La page du shell qui l'héberge se contente de demander « est-ce valide ? » et « donne-moi le client » ; elle ne sait jamais comment fonctionne la règle de l'e-mail. La méthode s'appelle `ValidateContent` pour ne pas être confondue avec les membres de validation qu'un contrôle conteneur expose déjà.

Les règles de conception des écrans de saisie découlent de cette frontière et constituent la liste de contrôle utilisée par le formateur pour l'évaluation :

- Chaque champ modifiable a un libellé visible ou un équivalent accessible.
- L'éditeur correspond au type de la valeur ; les dates et les nombres ne sont pas analysés à partir de contrôles `TextBox` ordinaires sans une bonne raison.
- Les messages de validation sont exploitables, `ErrorProvider` porte les erreurs de champ et `Toast` ou `AlertBox` les retours sur le déroulement.
- Les commandes sont désactivées pendant une opération, et le chemin de Save est explicite : valider, persister, rafraîchir l'état, notifier.



Pièges courants

Utiliser `MessageBox` pour chaque message d'état mineur. Enregistrer un état partiellement invalide parce que la validation n'a porté que sur les contrôles visibles. Cacher la validation dans des gestionnaires d'événements anonymes. Lire la valeur d'une `ComboBox` sans distinguer le texte affiché de la clé stockée. Oublier que les boutons désactivés et le retour d'attente relèvent de la fiabilité, pas de la décoration.



Réflexe bibliothèque de contrôles

Choisissez le contrôle natif le plus simple qui exprime la tâche de l'utilisateur avant d'écrire du code personnalisé. Les contrôles natifs offrent déjà le support au design, les thèmes, les événements et l'état serveur ; tournez-vous ensuite vers un extender ou un UserControl, et ne créez un widget JavaScript que si aucun contrôle natif ne convient.



Objectifs d'apprentissage

À la fin de ce module, vous saurez :

- ✓ Choisir le bon éditeur pour les saisies de texte, numériques, de date/heure, masquées et de sélection.
- ✓ Utiliser à bon escient Button, SplitButton, CheckBox, RadioButton, ToolTip, HelpTip, ErrorProvider, MessageBox, Toast et ProgressBar.
- ✓ Mettre en œuvre une validation assez immédiate pour aider l'utilisateur, sans être agressive au point de bloquer la saisie normale.
- ✓ Créer un UserControl d'édition réutilisable pour des données de client ou de contact.

Guide du lab / de l'examen

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



Lab pratique

Créez un UserControl `CustomerEditor` pour `Operations Console` avec des champs libellés pour le nom, l'e-mail, le statut, la date de début, la limite de crédit et le type de client, en utilisant l'éditeur adapté à chaque valeur plutôt qu'une `TextBox` pour tout. Ajoutez des messages `ErrorProvider` pour les champs obligatoires, le format de l'e-mail, la plage de la limite de crédit et les dates invalides ; des commandes `Save`, `Reset` et `Validate` désactivées et signalant l'attente pendant l'enregistrement ; et un `Toast` ou une `AlertBox` à la fin de la validation ou de l'enregistrement.

Point de départ

Ouvrez la solution `OperationsConsole` du Module 1 : la page du shell avec ses zones de navigation, de contenu, de commandes et d'état fonctionne déjà, et la page provisoire `Editors` est vide. Gardez le travail de ce module dans son propre dossier et namespace `Editors` pour que le projet de synthèse puisse réutiliser le contrôle tel quel.

Approche suggérée

- ✓ Esquissez les six champs et notez l'éditeur de chacun avant d'ouvrir le designer : `TextBox` pour le nom et l'e-mail, `ComboBox` pour le statut et le type de client, `DateTimePicker` pour la date de début, `NumericUpDown` pour la limite de crédit.
- ✓ Ajoutez un UserControl `CustomerEditor`, déposez les libellés et les éditeurs avec des noms clairs (`txtName`, `cboStatus`, `dtpStartDate`, `numCreditLimit`), définissez `Minimum`, `Maximum`, `MaxLength` et des éléments fixes pour qu'aucune valeur impossible ne puisse être saisie, et donnez à chaque éditeur un `ToolTip` qui décrit ce qu'il attend.
- ✓ Ajoutez un `ErrorProvider` et des validateurs nommés (`ValidateRequiredName`, `ValidateEmail`, `ValidateCreditLimit`, `ValidateStartDate`) qui définissent ou effacent un message précis ; appelez le validateur correspondant depuis l'événement `Validating` de chaque éditeur, et tous depuis une méthode `ValidateContent` qui renvoie un objet résultat.
- ✓ Ajoutez les boutons `Save`, `Reset` et `Validate` ; faites exécuter à `Save` d'abord `ValidateContent`, puis l'appel au service, puis un rafraîchissement de l'état, et désactivez les trois commandes en affichant un retour d'attente pendant l'enregistrement.
- ✓ Affichez un `Toast` ou une `AlertBox` à la fin de la validation ou de l'enregistrement, mettez à jour la zone d'état du shell, et hébergez le contrôle terminé sur la page `Editors`.

Livrables attendus

- ✓ UserControl `CustomerEditor` avec des champs libellés pour le nom, l'e-mail, le statut, la date de début, la limite de crédit et le type de client
- ✓ Éditeurs adaptés : `TextBox`, `DateTimePicker`, `NumericUpDown` et `ComboBox` au lieu de texte libre partout

- ✓ Messages ErrorProvider pour les champs obligatoires, le format de l'e-mail, la plage de la limite de crédit et les dates invalides
- ✓ Commandes Save, Reset et Validate désactivées et signalant l'attente pendant l'enregistrement
- ✓ Retour Toast ou AlertBox à la fin de la validation ou de l'enregistrement

Questions de révision

- ✓ Quelles erreurs de validation ne peuvent plus du tout se produire grâce à l'éditeur choisi, et lesquelles nécessitent encore un message ErrorProvider ?
- ✓ Si l'utilisateur clique deux fois sur Save en une seconde, que fait votre écran, et à quel endroit du code est-ce garanti ?
- ✓ Que sait la page du shell de la manière dont le champ e-mail est validé, et que faudrait-il changer si la règle passait dans un service ?



Réflexe bibliothèque de contrôles

Choisissez le contrôle natif le plus simple qui exprime la tâche de l'utilisateur avant d'écrire du code personnalisé. Les contrôles natifs offrent déjà le support au design, les thèmes, les événements et l'état serveur ; tournez-vous ensuite vers un extender ou un UserControl, et ne créez un widget JavaScript que si aucun contrôle natif ne convient.