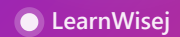


Le modèle d'image de Wisej.NET : Image, ImageSource et ImageList

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



Guide de la leçon

Les deux pipelines d'image : une propriété Image qui porte un System.Drawing.Image que le serveur envoie au navigateur sous forme de PNG, face à une chaîne ImageSource ou IconSource que le client résout comme une URL, un nom de thème, un chemin resource.wx ou un élément de pack d'icônes, auxquels s'ajoutent le contrat commun IImage et l'ImageList qui permet à plusieurs contrôles de partager un même jeu d'images adressées par clé. Ce que cela signifie pour un développeur qui connaît les bases de C# et a déjà travaillé avec les contrôles et le designer de Wisej.NET — et comment aborder ce module.



Objectif en production

Les deux pipelines d'image : une propriété Image qui porte un System.Drawing.Image que le serveur envoie au navigateur sous forme de PNG, face à une chaîne ImageSource ou IconSource que le client résout comme une URL, un nom de thème, un chemin resource.wx ou un élément de pack d'icônes, auxquels s'ajoutent le contrat commun IImage et l'ImageList qui permet à plusieurs contrôles de partager un même jeu d'images adressées par clé.

1. Le problème en production

Wisej.NET propose délibérément plus d'une manière de poser un graphique sur un contrôle, et toutes fonctionnent. L'ennui commence quand une équipe les utilise toutes par accident : une barre d'outils où un bouton lit un fichier sur le disque du serveur, le suivant pointe vers un nom de thème, et un troisième a discrètement perdu son icône parce que deux propriétés ont été définies sur le même contrôle. Six mois plus tard, personne ne sait expliquer d'où vient réellement une image donnée ni pourquoi elle a disparu après le déploiement.

2. Concepts clés

Concept	Pourquoi c'est important
propriété Image	Un objet <code>System.Drawing.Image</code> qui vit dans la mémoire du serveur ; Wisej.NET l'envoie au navigateur sous forme de données PNG, si bien que l'image voyage en octets que votre code a déjà décodés.
chaîne ImageSource	Une chaîne qui identifie une ressource à résoudre par le client : une URL relative, une URL absolue, une image de thème nommée, un chemin <code>resource.wx</code> ou un élément d'un assembly de pack d'icônes.
IconSource	La même idée de chaîne, sur les contrôles qui possèdent un emplacement d'icône dédié à côté de leur image principale ; vérifiez donc l'API du contrôle que vous utilisez au lieu de supposer que tous nomment la propriété de la même façon.
contrat IImage	La famille d'images commune qu'exposent les contrôles capables d'en porter une — <code>Image</code> , <code>ImageAlign</code> , <code>ImageIndex</code> , <code>ImageKey</code> , <code>ImageList</code> et <code>ImageSource</code> — et qui permet à une seule méthode utilitaire de décorer uniformément un Button, un Label ou n'importe quoi d'autre.
ImageList	Une collection réutilisable d'objets image vers laquelle plusieurs contrôles peuvent pointer, adressée par position ou par nom, et capable d'établir une taille d'image commune pour les contrôles qui l'utilisent.
ImageKey contre ImageIndex	Une clé telle que <code>"delete"</code> porte du sens et survit à un réordonnement ; l'index <code>3</code> devient silencieusement la mauvaise image le jour où quelqu'un insère une image au-dessus.
affectation mutuellement exclusive	Toutes ces propriétés alimentent un seul emplacement visuel : en affecter une peut donc réinitialiser les autres — et c'est pourquoi une mauvaise icône est généralement une seconde affectation, pas un fichier cassé.

3. Le fil de la leçon

Objet ou chaîne : la première question à trancher

Avant de taper quoi que ce soit, décidez si le serveur possède un *objet* image ou si le navigateur doit résoudre un *identifiant* d'image. Ce sont les deux pipelines, et presque toutes les confusions ultérieures de ce cours viennent de leur mélange. Lorsque vous affectez `Image`, vous confiez à Wisej.NET un `System.Drawing.Image` : de véritables données de pixels décodées, présentes dans la mémoire de votre session, que le framework envoie ensuite au navigateur en PNG. C'est la bonne voie quand votre C# a réellement produit ou reçu un objet image — un graphique que vous avez rendu, une vignette que vous avez redimensionnée, une photo arrivée d'un service sous forme de flux.

Lorsque vous affectez `ImageSource` ou `IconSource`, vous confiez à Wisej.NET une chaîne. Rien n'est décodé sur le serveur. La chaîne désigne quelque chose que le client peut récupérer et mettre en cache tout seul : `"Images/open.png"` relatif à l'application, une adresse absolue, un nom d'image de thème tel que `"icon-save"`, ou un chemin `resource.wx` dans un assembly. Comme la ressource conserve son

identité, un SVG reste un SVG jusqu'au navigateur, et c'est pourquoi les icônes d'interface restent nettes à toute taille et peuvent ensuite être recolorées. Dans IconDesk vous verrez les deux : `btnOpen` reçoit un objet image chargé par le serveur, `btnSave` reçoit un nom de thème sous forme de chaîne, et le libellé de diagnostic doit les distinguer.

Une famille de propriétés, nommée `IImage`

Wisej.NET ne donne pas à chaque contrôle son propre vocabulaire pour les images. Les contrôles capables d'en porter une partagent un contrat appelé `IImage` qui rassemble `Image`, `ImageAlign`, `ImageIndex`, `ImageKey`, `ImageList` et `ImageSource` en une seule famille. Reconnaître la famille a plus de valeur pour un débutant que de convertir vers elle : face à un nouveau contrôle, vous savez déjà quelles propriétés chercher et lesquelles, parmi ces quatre, se disputent le même emplacement.

La conversion reste utile pour le code d'infrastructure. Une méthode qui accepte un `Control`, teste `is IImage` puis définit `ImageSource` peut appliquer l'icône d'enregistrement du projet à un Button, un Label ou un élément de menu sans instruction switch. Le libellé de diagnostic d'IconDesk fait l'inverse dans le lab : il lit chaque bouton via `IImage`, vérifie laquelle des quatre propriétés est renseignée, et affiche le mécanisme derrière l'icône au lieu de le deviner à partir de l'image.

Choisir le mécanisme adapté à la ressource que vous avez devant vous

Quatre cas couvrent presque tout. Un fichier déployé à côté de l'application est une source d'image avec une URL relative. Une image hébergée ailleurs est une source d'image avec une URL absolue. Une icône que le thème actif définit déjà est une source d'image contenant le nom que le thème lui donne, ce qui signifie que le client la prend dans le cache de thème qu'il a de toute façon chargé. Une image que votre code a créée, transformée ou reçue en tant qu'objet .NET est une `Image`.

Le cinquième cas est celui du partage, et c'est la raison d'être de l' `ImageList`. Une `ImageList` contient une collection d'objets image ; les contrôles référencent une entrée par `ImageIndex` numérique ou par `ImageKey` textuel. Deux propriétés de cette indirection comptent dans une vraie application : de nombreux contrôles peuvent partager une seule collection, et l'image stockée sous une clé peut être remplacée sans toucher à un seul contrôle qui renvoie à cette clé. La liste peut aussi établir une taille d'image pour les contrôles qui l'utilisent, ce qui est pratique quand le thème déciderait autrement de la taille des icônes à votre place.

```
var icons = new ImageList();
icons.Images.Add("print", Image.FromFile("Images/print.png"));
icons.Images.Add("delete", Image.FromFile("Images/delete.png"));

btnPrint.ImageList = icons;
btnPrint.ImageKey = "print";
btnDelete.ImageList = icons;
btnDelete.ImageKey = "delete";

// btnSave takes its icon from a string instead - no ImageList involved.
btnSave.ImageSource = "icon-save";
```

Notez ce que les premières lignes ne font pas. `Image.FromFile` lit un fichier sur le serveur, jamais sur la machine de l'utilisateur : un chemin absolu qui fonctionnait sur votre portable est donc un défaut de

déploiement en puissance. Une chaîne de source d'image, à l'inverse, s'adresse comme le navigateur adresse tout le reste.

Pourquoi l'icône que vous définissez n'est pas toujours celle que vous voyez

Un contrôle a un seul emplacement visuel pour son image, et `Image`, `ImageSource`, `ImageIndex` et `ImageKey` se disputent tous son remplissage. Wisej.NET tranche en les rendant mutuellement exclusives : en définir une peut réinitialiser les autres, de sorte que la configuration n'est jamais ambiguë. Si `btnDelete` affiche tranquillement l'entrée de liste sous `ImageKey = "delete"` et qu'une ligne ultérieure définit `ImageSource`, la chaîne devient la représentation active et la clé cesse de compter.

Cette seule règle explique la plupart des rapports de bug « ce n'est pas la bonne icône ». Le designer a défini une propriété, le constructeur du formulaire une autre, et un thème ou une méthode utilitaire une troisième. Quand l'image est fausse, ne commencez pas par ouvrir le fichier — inspectez toutes les propriétés liées à l'image sur le contrôle et trouvez la dernière affectation exécutée.

- Une seule propriété d'image par contrôle, choisie délibérément.
- Des clés, pas des index, partout où l'identité a du sens.
- Un équivalent textuel pour toute icône qui porte du sens à elle seule.



Nommez d'abord le mécanisme

Avant d'affecter la moindre image, dites quel mécanisme vous utilisez : un objet `System.Drawing.Image` que le serveur rend en PNG, une chaîne de source d'image que le navigateur résout, une image de thème nommée, une ressource embarquée `resource.wx`, ou une police d'icônes dessinée en HTML. N'affectez qu'une seule propriété d'image par contrôle pour que les autres ne puissent pas la réinitialiser en silence, gardez une chaîne de source parlante plutôt qu'un chemin magique collé, préférez le SVG pour les icônes d'interface afin qu'elles restent nettes à toute taille et à toute échelle, et donnez à chaque icône porteuse de sens un équivalent textuel.



Objectifs d'apprentissage

À la fin de ce module, vous saurez :

- ✓ Expliquer la différence entre une valeur `System.Drawing.Image` et une chaîne d'image telle qu'`ImageSource` ou `IconSource`.
- ✓ Utiliser le modèle commun `Image` pour reconnaître les propriétés `Image`, `ImageSource`, `ImageList`, `ImageIndex` et `ImageKey`.
- ✓ Choisir le bon mécanisme d'image pour un bitmap local, une URL, une image de thème ou une icône réutilisable.
- ✓ Comprendre pourquoi affecter l'une de plusieurs propriétés d'image mutuellement exclusives peut réinitialiser les autres.

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 le projet IconDesk et dotez-le d'un panneau de commandes avec les boutons Open, Save, Print et Delete, chacun portant son icône par un mécanisme différent. Chargez `btnOpen` depuis un `System.Drawing.Image` côté serveur, affectez une image de thème nommée à `btnSave.ImageSource`, et faites pointer `btnPrint` et `btnDelete` vers une même `ImageList` partagée via `ImageKey`. Ajoutez un libellé de diagnostic qui inspecte chaque bouton via `IImage` et indique quel mécanisme est réellement en jeu, puis remplacez l'image stockée sous une clé de l'`ImageList` et vérifiez que tous les contrôles qui utilisent cette clé suivent le changement.

Point de départ

C'est le premier module : vous partez donc de rien. Créez une nouvelle application Wisej.NET 4 nommée `IconDesk` et ajoutez-y une page pour le panneau de commandes. Déposez à côté de l'application un petit dossier d'images — un visuel d'ouverture, d'enregistrement, d'impression et de suppression suffit — et vérifiez que le projet vide compile et s'ouvre dans le navigateur avant d'affecter la moindre icône.

Approche suggérée

- ✓ Disposez d'abord les quatre boutons avec du texte seulement, pour que le panneau soit lisible avant qu'aucune image n'existe.
- ✓ Donnez à `btnOpen` un `System.Drawing.Image` que le serveur charge, et à `btnSave` un nom d'image de thème sur `ImageSource` ; dites à voix haute quel pipeline chacun utilise.
- ✓ Construisez une seule `ImageList`, ajoutez-y les images d'impression et de suppression sous des clés parlantes, puis faites pointer les deux boutons vers la liste via `ImageKey` plutôt que via `ImageIndex`.
- ✓ Écrivez le libellé de diagnostic : parcourez les contrôles du panneau, testez chacun pour `IImage`, et indiquez laquelle des propriétés `Image`, `ImageSource`, `ImageList` plus `ImageKey` est réellement renseignée.
- ✓ Remplacez l'image stockée sous la clé de suppression, rafraîchissez la page, puis définissez délibérément une seconde propriété d'image sur un bouton afin de consigner ce qu'a fait la première affectation.

Livrables attendus

- ✓ Panneau de commandes IconDesk avec les boutons Open, Save, Print et Delete
- ✓ `btnOpen` portant un `System.Drawing.Image` chargé sur le serveur et `btnSave` une image de thème nommée sur `ImageSource`
- ✓ `btnPrint` et `btnDelete` partageant une même `ImageList` via `ImageKey` plutôt qu'`ImageIndex`
- ✓ Libellé de diagnostic qui lit chaque bouton via `IImage` et nomme le mécanisme derrière son icône

- ✓ Note de lab consignant ce qui s'est passé lors du remplacement d'une clé de l'ImageList et lorsque deux propriétés d'image ont été définies sur le même bouton

Questions de révision

- ✓ Votre libellé de diagnostic indique que `btnDelete` utilise une source d'image, alors que vous n'avez jamais défini que son `ImageKey`. Qu'est-ce qui a bien pu s'exécuter, et où le chercheriez-vous ?
- ✓ L'impression et la suppression ont toutes deux changé lorsque vous avez remplacé une entrée de l' `ImageList`. Serait-ce encore vrai si les boutons utilisaient `ImageIndex`, et que se passerait-il si une cinquième image était insérée en tête de liste ?
- ✓ Le bouton Open lit son fichier sur le serveur. Quelle partie de ce montage casse lorsque l'application est déployée sur une autre machine, et quelle partie ne casserait pas si la même image était une chaîne de source d'image ?



Nommez d'abord le mécanisme

Avant d'affecter la moindre image, dites quel mécanisme vous utilisez : un objet `System.Drawing.Image` que le serveur rend en PNG, une chaîne de source d'image que le navigateur résout, une image de thème nommée, une ressource embarquée `resource.wx`, ou une police d'icônes dessinée en HTML. N'affectez qu'une seule propriété d'image par contrôle pour que les autres ne puissent pas la réinitialiser en silence, gardez une chaîne de source parlante plutôt qu'un chemin magique collé, préférez le SVG pour les icônes d'interface afin qu'elles restent nettes à toute taille et à toute échelle, et donnez à chaque icône porteuse de sens un équivalent textuel.