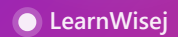


# Contrôles Wisej personnalisés, classes client et thèmes

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



## Guide de la leçon

Construire des contrôles d'apparence native avec des classes serveur, des classes de widget client, des clés d'apparence et des ressources incorporées. Ce que cela signifie pour un développeur qui intègre avec Wisej.NET — et comment aborder ce module.



### Objectif production

Construire des contrôles d'aspect natif avec des classes serveur, des classes de widget client, des clés d'apparence et des ressources incorporées.

## 1. Le problème d'intégration

Certaines intégrations doivent se comporter comme des contrôles Wisej.NET à part entière : réutilisables, compatibles avec les thèmes, utilisables dans le Designer et maintenables dans de nombreuses applications.

## 2. Concepts clés

| Concept        | Comment l'utiliser                                                                   |
|----------------|--------------------------------------------------------------------------------------|
| Classe Control | Utiliser un contrôle côté serveur lorsqu'une intégration d'aspect natif est requise. |
| AppearanceKey  | Relier le rendu visuel du contrôle au style défini par le thème.                     |

## 3. Leçon de production

Un contrôle Wisej.NET personnalisé est la forme d'intégration la plus aboutie. Il se justifie quand le widget fait partie de la plateforme produit, quand le thème compte, ou quand le contrôle doit avoir l'apparence et le comportement d'un contrôle Wisej.NET natif. Le pattern ressemble à la création d'un contrôle intégré : une classe serveur expose des propriétés et des événements .NET, et une classe client reçoit l'état rendu et possède l'objet côté navigateur. La classe client peut étendre `wisej.web.Control` ou `wisej.web.Widget` selon le comportement souhaité. Elle peut créer l'objet tiers, implémenter les méthodes apply des propriétés, gérer les changements de taille ou d'apparence, et traduire les callbacks tiers en événements Wisej.NET. La classe serveur peut redéfinir `OnWebRender` pour n'envoyer que les propriétés dont le client a besoin. Cette approche demande davantage de code, mais elle offre l'architecture la plus propre sur le long terme. C'est le bon choix pour les contrôles livrés dans un framework, une bibliothèque d'extensions

ou un produit d'entreprise, où le wrapper doit apparaître dans IntelliSense, fonctionner dans le Designer et rester stable au fil des versions de la bibliothèque tierce sous-jacente.

### Un contrôle serveur pour une intégration plus étroite

Dériver de `Wisej.Web.Control` au lieu de `Wisej.Web.Widget` change la nature de la classe. Un Widget est un hôte pour un script ; un Control est un membre à part entière du framework, avec la même mise en page, le même focus, le même thème et le même comportement dans le Designer qu'un bouton ou un panneau intégré. Le `SimpleGaugeControl` du lab suit cette voie pour que la jauge puisse être glissée, dimensionnée, ancrée et thématiquée comme n'importe quel autre contrôle.

La classe serveur expose des propriétés typées et des événements .NET, comme le faisait le wrapper, mais elle ne porte plus d'InitScript ni de sac d'Options. Elle déclare plutôt quelle classe client l'implémente et rend son état via `OnWebRender`. Les setters des propriétés signalent que le contrôle doit être mis à jour, et le framework décide quand envoyer la modification.

Choisissez cette profondeur quand le widget fait partie de la plateforme produit, quand le thème compte, ou quand le contrôle doit sembler natif aux personnes qui construisent des écrans avec lui. Le code supplémentaire est rentabilisé par chaque écran qui utilise le contrôle sans avoir à y penser.

### Une classe client sous Platform

La moitié navigateur d'un contrôle personnalisé est une classe JavaScript définie avec le modèle de classes client de Wisej.NET, fondé sur qooxdoo, et placée dans le dossier `/Platform` de l'application pour que le framework la charge avec le client. La classe étend `wisej.web.Control`, ou `wisej.web.Widget` lorsqu'on veut un comportement d'hôte de script, et déclare ses propriétés avec des méthodes apply.

Une méthode apply s'exécute chaque fois que le serveur modifie la propriété correspondante, et c'est là que doit se trouver l'appel tiers : l'apply de `value` appelle la méthode de la bibliothèque qui définit la valeur, et l'apply de `maximum` met à jour les options de la bibliothèque. Le constructeur de la classe crée l'instance tierce dans l'élément de contenu du contrôle, et le destructeur la détruit. Les changements de taille et d'apparence arrivent sous forme d'événements que la classe peut traiter en un seul endroit.

Gardez la classe client exempte de logique applicative. Elle traduit entre le framework et la bibliothèque tierce, et rien d'autre, afin de pouvoir être réutilisée par chaque contrôle serveur qui se rend vers elle.

### Rendre l'état avec OnWebRender

`OnWebRender` est l'occasion pour le serveur de décider exactement ce que voit le client. La redéfinition reçoit un objet de configuration dynamique, et le contrôle n'y écrit que les propriétés déclarées par la classe client : `config.value`, `config.minimum`, `config.maximum`, `config.caption`. Les noms sont en camelCase pour correspondre aux noms des propriétés client, et Wisej.NET n'envoie que les valeurs modifiées depuis le dernier rendu.

Il s'agit d'un état intentionnel et non accidentel. Une propriété qui existe côté serveur pour le Designer ou pour la logique métier n'est tout simplement pas écrite dans la configuration et n'atteint jamais le navigateur. Une valeur coûteuse à sérialiser, comme un grand tableau d'historique, peut être réduite à ce dont la jauge a besoin avant d'être affectée. L'auto-vérification du lab interroge sur le rôle de `OnWebRender`, et la réponse est qu'il constitue le contrat entre les deux moitiés du contrôle.

Les événements circulent dans l'autre sens via `OnWebEvent`, traité au Module 7. Ensemble, les deux redéfinitions décrivent tout le protocole réseau du contrôle.

## Clés d'apparence, thèmes et mode design

Un thème Wisej.NET stylise les contrôles par clé d'apparence. Donner au contrôle de jauge une `AppearanceKey` comme `gauge`, et ajouter une entrée portant cette clé au thème ou à un mixin de thème, relie les couleurs, les polices et les espacements du contrôle au thème de l'application. Changer de thème restylise alors la jauge en même temps que tous les autres contrôles : c'est toute la différence entre un contrôle qui semble natif et un contrôle qui semble plaqué.

Les ressources de thème ne se limitent pas aux couleurs. Les images et icônes dont le contrôle a besoin peuvent être déclarées comme ressources de thème, pour qu'un thème sombre fournisse des variantes sombres sans modification de code. La classe client lit son style depuis l'apparence au lieu de le coder en dur, et l'instance tierce est configurée à partir de ces valeurs dans les méthodes apply.

Le rendu en mode design est le test final. Le Designer instancie le contrôle et l'affiche en direct : la classe client doit donc supporter des données fictives, l'absence de serveur et des créations répétées pendant que le développeur le redimensionne. C'est pour cela que le lab demande une capture d'écran en mode design : si le contrôle s'affiche correctement dans le Designer, il sera adopté ; s'il lève une exception, il sera évité.



### Réflexe de production

Gardez clairement séparées la propriété de l'état côté Wisej.NET et le comportement du widget client. Documentez le contrat client/serveur avant de vous appuyer sur l'intégration en production.



### Erreur courante à éviter

Copier du JavaScript de démonstration dans les écrans sans documenter le cycle de vie, les ressources, les contrats d'événements, le nettoyage ni les frontières de sécurité.



### Objectifs d'apprentissage

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

- ✓ Créer un contrôle côté serveur dérivé de `Wisej.Web.Control` lorsqu'une intégration plus étroite est requise.
- ✓ Créer une classe côté client sous ``/Platform`` avec le modèle de classes Wisej.NET/qx.
- ✓ Rendre l'état via ``OnWebRender`` et le faire correspondre aux propriétés ou options du client.
- ✓ Comprendre `AppearanceKey`, les ressources de thème et les contraintes du rendu en mode design.

## Guide du lab / de l'examen

---

Ce que vous allez construire dans le lab pratique, la démarche recommandée, les livrables attendus et l'auto-vérification avant la remise.



### Lab pratique

Créez un contrôle personnalisé `SimpleGaugeControl`` avec une classe serveur et une classe client ``/Platform``. Ajoutez une clé d'apparence de base et faites-le s'afficher en mode design.

### Démarche recommandée

- ✓ Ouvrez le projet existant du cours d'intégration Wisej.NET et exécutez-le en local.
- ✓ Repérez le composant serveur, le widget client, le fichier de ressources ou le point de terminaison concernés par ce module.
- ✓ Faites une petite modification à la fois et testez-la dans le navigateur avant de passer à la suite.
- ✓ Gardez le JavaScript propre à la bibliothèque tierce dans l'adaptateur ou le wrapper au lieu de le disperser dans les pages.
- ✓ Mettez à jour l'interface pour que l'apprenant voie clairement le succès, l'échec, la progression ou les changements de communication.
- ✓ Prenez des captures d'écran ou des notes qui prouvent que l'intégration fonctionne et qui expliquent le choix de conception.

### Livrables attendus

- ✓ Classe serveur du contrôle
- ✓ Squelette de la classe client qx
- ✓ Entrée de thème/d'apparence
- ✓ Capture d'écran ou notes en mode design

### Auto-vérification avant la remise

- ✓ Quand un contrôle personnalisé vaut-il le travail supplémentaire ?
- ✓ Quel est le rôle de ``OnWebRender`` ?
- ✓ Pourquoi l'intégration au thème compte-t-elle pour l'adoption en entreprise ?