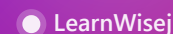


Costruire il tuo pacchetto di icone Wisej.NET

Guida della lezione · Guida al lab / esame



Guida della lezione

La ricetta in quattro passi dietro ogni pacchetto di icone: una class library, una cartella Resources, i file delle icone marcati Embedded Resource e un catalogo statico di stringhe resource.wx che dà al team IntelliSense invece di percorsi magici, più le pratiche di denominazione, versionamento, ricolorazione e override che mantengono il pacchetto manutenibile. Che cosa significa per uno sviluppatore che conosce le basi di C# e ha già lavorato un po' con i controlli e il designer di Wisej.NET — e come affrontare questo modulo.



Focus di produzione

La ricetta in quattro passi dietro ogni pacchetto di icone: una class library, una cartella Resources, i file delle icone marcati Embedded Resource e un catalogo statico di stringhe resource.wx che dà al team IntelliSense invece di percorsi magici, più le pratiche di denominazione, versionamento, ricolorazione e override che mantengono il pacchetto manutenibile.

1. Il problema in produzione

Al quinto modulo IconDesk mostra un logo incorporato, un'immagine del tema e le icone di due pacchetti ufficiali, ed è lì che comincia il disordine: le stringhe di risorsa compaiono inline nei gestori di eventi, due schermate scrivono lo stesso file in modo diverso e nessuno sa dire quali icone appartengono al prodotto. Un pacchetto di icone personalizzato è il rimedio a basso costo, e in più ti compra un confine: un assembly contiene la grafica, un catalogo C# le dà un nome e il resto dell'applicazione non scrive mai più un percorso.

2. Concetti chiave

Concetto	Perché è importante
assembly di risorse	Una semplice class library .NET il cui unico compito è portare i file delle icone come risorse incorporate; non ha bisogno di alcun riferimento all'applicazione né di codice Wisej.NET proprio.
la ricetta in quattro passi	Crea una class library, crea una cartella chiamata <code>Resources</code> , metti dentro i file delle icone, imposta la loro Build Action su <code>Embedded Resource</code> . Tutto quello che viene dopo è comodità.
catalogo delle icone	Una classe statica come <code>AppIcons</code> che espone ogni icona come costante stringa dal nome parlante, così che il compilatore e IntelliSense sostituiscano i percorsi copiati e incollati.
nome semantico contro nome del file	Il catalogo espone il vocabolario del tuo prodotto — save, delete, customer, warning — mentre il file mantiene un nome semplice, così rinominarlo cambia una riga e non quaranta schermate.
indirizzo di risorsa qualificato	Un'icona in un assembly separato si indirizza come <code>resource.wx/</code> più il nome dell'assembly più il percorso della risorsa: è per questo che ogni voce del catalogo condivide un'unica costante radice privata.
contratto di ricolorazione	Il suffisso <code>?color=</code> ricolora solo grafica monocromatica adatta; un SVG con valori di <code>fill</code> fissati nel file è il motivo abituale per cui un'icona si rifiuta di cambiare colore.
set a tema e override	Pacchetti paralleli che condividono un'unica API di catalogo su grafica diversa, più file distribuiti che hanno la precedenza su una risorsa incorporata, sostituiscono le icone senza toccare il codice dell'applicazione.

3. Il filo della lezione

Una class library il cui unico compito sono le icone

Wisej.NET documenta un pacchetto di icone personalizzato in quattro passi, e vale la pena leggerli alla lettera perché ci si aspetta sempre qualcosa di più cerimonioso. Crea una class library .NET. Creaci dentro una cartella chiamata `Resources`. Metti i file delle icone. Imposta la loro Build Action su `Embedded Resource`. Il meccanismo è tutto qui: il compilatore colloca i file dentro l'assembly e l'handler delle risorse incorporate del Modulo 5 li serve. Nel lab la libreria è `IconDesk.Icons`, ha un target framework compatibile con IconDesk e non contiene una sola riga di codice Wisej.NET.

Puoi esprimere il packaging nel file di progetto con una sola voce che include tutto ciò che sta sotto `Resources`, ma di solito è Visual Studio a scrivere quei metadati per te quando scegli la Build Action. Non fare entrambe le cose: un file dichiarato due volte è il motivo comune per cui un pacchetto appena creato compila e però produce nomi di risorsa duplicati o mancanti. Dopo la prima compilazione, richiedi una singola icona dall'applicazione in esecuzione: il percorso della risorsa che hai ipotizzato o è giusto o non lo è.

Perché un assembly e non una cartella copiata in ogni soluzione? Perché un assembly è un'unità: si può versionare, impacchettare come NuGet, condividere tra prodotti e sostituire in blocco, e il suo confine impedisce ai percorsi delle icone di filtrare nelle schermate.

Il catalogo è ciò che il tuo team usa davvero

Un assembly di risorse da solo è ancora un sacco di stringhe, e le stringhe non sopravvivono a una base di codice che cresce. La seconda metà di un pacchetto vero è un catalogo statico: una classe, una costante radice privata, una costante pubblica per icona. Chi lo usa scrive allora un nome che l'editor sa completare e che chi rivede il codice sa leggere.

```
namespace IconDesk.Icons;

public static class AppIcons
{
    private const string Root = "resource.wx/IconDesk.Icons/";

    public const string Save = Root + "save.svg";
    public const string Delete = Root + "delete.svg";
    public const string Success = Root + "status-success.svg";
}

// chi le usa non scrive mai più un percorso
btnSave.ImageSource = AppIcons.Save;
btnDelete.ImageSource = AppIcons.Delete + "?color=error";
```

Leggi con attenzione la costante radice, perché è la parte che si sbaglia più spesso: l'handler `resource.wx`, poi il **nome dell'assembly**, poi il percorso della risorsa dentro quell'assembly. Il percorso esatto dipende da come la tua libreria incorpora i suoi file, quindi verificalo contro l'applicazione in esecuzione invece di trattare una radice d'esempio come magica. Una volta confermato, vive in un'unica costante privata.

L'altra metà della progettazione del catalogo è la curatela. Un pacchetto non è una cartella con dentro ogni icona che qualcuno ha scaricato; è il vocabolario del tuo prodotto: azioni, entità, stati. Tieni semplici i nomi dei file: quando il design sostituisce `save.svg` con un glifo ridisegnato, la voce del catalogo non si muove.

Usare il pacchetto dal designer e da C#

Referenzia il progetto della class library da IconDesk e *compila la soluzione*. È quella compilazione il passo che si salta, ed è il motivo per cui a volte l'Image Selector sembra vuoto: il designer scopre le risorse dei pacchetti di icone dagli assembly referenziati dal progetto. Una volta che le ha, il tuo pacchetto compare accanto a quelli ufficiali, e scegliere un'icona scrive dentro `Designer.cs` la stessa stringa di sorgente che produce la tua costante del catalogo: apri una volta il file generato e confronta le due.

Usa entrambe le vie in modo consapevole. Il designer è la scelta giusta per le icone su un controllo che disponi visivamente; il catalogo è la scelta giusta per tutto ciò che viene deciso a runtime, come un glifo di stato scelto da un enum o una toolbar costruita in codice. Entrambe finiscono come una stringa di sorgente immagine che il browser risolve e mette in cache un'icona alla volta, e nessuna delle due è un

`System.Drawing.Image` : un SVG resta un SVG fino al browser, ed è per questo che resta nitido a 16 come a 48 pixel.

La ricolorazione merita una prova, non una supposizione. Il suffisso `?color=` ricolora la grafica disegnata per essere ricolorata: un tracciato a un solo colore, senza `fill` o `stroke` fissati che gli si oppongano. Nel lab verifichi tre icone contro un colore esadecimale, il nome di un colore del tema ed entrambi i temi.

Un'icona che si rifiuta di solito ha bisogno che l'attributo fill venga tolto dal suo SVG, mentre un marchio va lasciato in pace: i suoi colori sono il brand, non una scelta di stile.

Le regole che tengono vivo il pacchetto dopo il rilascio

Tutto quello che precede è il lavoro di un pomeriggio. Ciò che decide se il pacchetto sarà ancora utile fra due anni sono le regole che scrivi adesso. Prima viene la denominazione: un concetto significa un'icona, l'uso delle maiuscole è coerente, le view box sono normalizzate e ogni icona ha una licenza registrata. I concetti duplicati — due glifi di ricerca da due designer diversi — sono il modo in cui un pacchetto curato torna a essere una cartella.

Il versionamento discende dal trattare il catalogo come API pubblica. Aggiungere una costante è additivo. Sostituire la grafica dietro una costante esistente è una modifica visiva che va nelle note di rilascio, anche se nulla smette di compilare. Rimuovere o rinominare una costante è breaking, quindi deprecata invece di eliminarla. Gli override ti danno due leve documentate e una architetturale: una sorgente che omette il qualificatore di assembly lascia che Wisej.NET preferisca un file distribuito corrispondente, un percorso completamente qualificato può essere sovrascritto replicando la struttura di cartelle della distribuzione, e un pacchetto parallelo — un set ad alto contrasto o dedicato a un cliente — rilascia gli stessi nomi di catalogo su grafica diversa.

- Ogni icona ha una licenza registrata e una view box normalizzata.
- Il catalogo è il contratto; la grafica può cambiare, i nomi non possono sparire in silenzio.
- Una pagina di esempio mostra l'intero pacchetto in entrambi i temi prima che tu marchi una versione.



Dai prima un nome al meccanismo

Prima di assegnare qualsiasi immagine, dichiara quale meccanismo stai usando: un oggetto `System.Drawing.Image` che il server disegna come PNG, una stringa di sorgente immagine che il browser risolve, un'immagine del tema indicata per nome, una risorsa incorporata `resource.wx` o un font di icone disegnato come HTML. Assegna una sola proprietà immagine per controllo, così che le altre non possano azzerarla di nascosto, mantieni la stringa della sorgente significativa invece di un percorso magico incollato, preferisci l'SVG per le icone di interfaccia così che restino nitide a ogni dimensione e scala, e dai a ogni icona che porta un significato un'alternativa testuale.



Obiettivi di apprendimento

Alla fine di questo modulo sarai in grado di:

- ✓ Creare una class library che impacchetta le icone dell'applicazione come risorse incorporate.
- ✓ Esporre costanti e proprietà stabili di sorgente delle icone per IntelliSense e riuso del codice.

- ✓ Usare il pacchetto di icone personalizzato dal designer e da C#.
- ✓ Progettare pratiche di denominazione, versionamento, gestione dei temi e override delle icone per un prodotto manutenibile.

Guida al lab / esame

Che cosa costruirai nel lab pratico, l'approccio suggerito e i deliverable richiesti.



Lab pratico

Crea `IconDesk.Icons` come class library accanto all'applicazione, metti almeno dodici icone SVG in una cartella `Resources` e marcale `Embedded Resource`. Aggiungi un catalogo statico `AppIcons` che espone ogni icona come stringa `resource.wx` dal nome parlante, così che chi la usa non debba mai scrivere un percorso; referencia la libreria da `IconDesk`, poi assegna alcune icone tramite il designer e altre dal catalogo in C#. Verifica il suffisso di colore su tre icone e annota la pulizia dell'SVG di cui hanno avuto bisogno. Concludi con le regole di denominazione e versionamento che stabiliscono come una nuova icona entra nel pacchetto.

Punto di partenza

Apri la soluzione `IconDesk` come l'ha lasciata il Modulo 5: l'applicazione incorpora già le proprie immagini in una cartella `Resources` e le risolve tramite `resource.wx`, e due pacchetti di icone ufficiali sono referenziati dal Modulo 4. Nella soluzione non c'è ancora nulla che sia un pacchetto riutilizzabile, quindi ogni stringa di sorgente personalizzata è ancora scritta a mano nella pagina che ne ha bisogno.

Approccio suggerito

- ✓ Aggiungi alla soluzione una class library chiamata `IconDesk.Icons`, con un target framework compatibile con `IconDesk`, e dalle una cartella `Resources` che contiene almeno dodici icone SVG selezionate con cura.
- ✓ Imposta la Build Action di ogni file di icona su `Embedded Resource`, compila la libreria una volta e conferma il percorso della risorsa di una singola icona dall'applicazione in esecuzione, prima di scrivere qualsiasi voce del catalogo.
- ✓ Scrivi la classe statica `AppIcons` con una costante radice privata e una costante pubblica per ogni icona, chiamata in base a ciò che l'icona significa e non al file da cui arriva.
- ✓ Referenzia la libreria da `IconDesk`, ricompila la soluzione, poi assegna alcune icone tramite l'`Image Selector` del designer e le restanti da `AppIcons` in C#, su una pagina che mostra l'intero pacchetto.
- ✓ Aggiungi il suffisso di colore a tre icone, prova un colore esadecimale e il nome di un colore del tema sia in un tema chiaro sia in uno scuro, e correggi o dichiara esplicitamente come eccezione qualsiasi grafica che si rifiuti di ricolorarsi.

Deliverable richiesti

- ✓ Class library `IconDesk.Icons` con almeno dodici file SVG in una cartella `Resources`, tutti `Embedded Resource`
- ✓ Catalogo statico `AppIcons` che espone ogni icona come stringa `resource.wx` dal nome parlante
- ✓ Pagina `IconDesk` che assegna alcune icone tramite il designer e altre dal catalogo `AppIcons`

- ✓ Tre icone verificate con il suffisso di colore, più una nota sulla pulizia dell'SVG di cui hanno avuto bisogno
- ✓ Regole di denominazione e versionamento che descrivono come una nuova icona entra nel pacchetto e come un'applicazione ne sovrascrive una

Domande di autovalutazione

- ✓ Il tuo pacchetto compila e il designer non mostra nulla nell'Image Selector. Quale passo della ricetta controlleresti per primo, e quale prova ti direbbe che le risorse sono davvero dentro l'assembly?
- ✓ Due delle tue tre icone di prova si ricolorano e la terza resta grigia. Che cosa c'è dentro quell'SVG, e quando lasceresti di proposito un'icona non ricolorabile?
- ✓ Un cliente vuole tre icone sostituite solo nella sua distribuzione. Quale delle strade di override sceglieresti, e che cosa ti costa quella scelta alla prossima versione del pacchetto?



Dai prima un nome al meccanismo

Prima di assegnare qualsiasi immagine, dichiara quale meccanismo stai usando: un oggetto `System.Drawing.Image` che il server disegna come PNG, una stringa di sorgente immagine che il browser risolve, un'immagine del tema indicata per nome, una risorsa incorporata `resource.wx` o un font di icone disegnato come HTML. Assegna una sola proprietà immagine per controllo, così che le altre non possano azzerarla di nascosto, mantieni la stringa della sorgente significativa invece di un percorso magico incollato, preferisci l'SVG per le icone di interfaccia così che restino nitide a ogni dimensione e scala, e dai a ogni icona che porta un significato un'alternativa testuale.