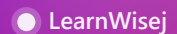


Eingebettete Ressourcen und resource.wx

Lektionsleitfaden · Lab- und Prüfungsleitfaden



Lektionsleitfaden

Bilder in der Assembly ausliefern: Build Action auf Embedded Resource, der resource.wx-Handler mit und ohne Assembly-Qualifizierer, die Erweiterungen css, js, html, jpg, jpeg, png, gif, svg und bmp, die er ausliefert, und die ausgelieferte Datei, die eine eingebettete Ressource ohne Neubau überschreiben kann. Was das für Entwicklerinnen und Entwickler bedeutet, die die Grundlagen von C# beherrschen und schon etwas mit den Steuerelementen und dem Designer von Wisej.NET gearbeitet haben – und wie Sie dieses Modul lesen.



Fokus: Produktivbetrieb

Bilder in der Assembly ausliefern: Build Action auf Embedded Resource, der resource.wx-Handler mit und ohne Assembly-Qualifizierer, die Erweiterungen css, js, html, jpg, jpeg, png, gif, svg und bmp, die er ausliefert, und die ausgelieferte Datei, die eine eingebettete Ressource ohne Neubau überschreiben kann.

1. Das Problem im Produktivbetrieb

Bis hierher war jedes Asset von IconDesk etwas, das die Anwendung auf der Festplatte gefunden oder beim Theme angefragt hat. Das trägt, bis jemand die Site ausliefert und einen Ordner vergisst: Der Browser fordert ein Icon an, der Server antwortet mit 404, und die Benutzerin sieht ein leeres Rechteck, wo das Status-Badge stehen sollte. Im Review fällt es niemandem auf, weil der Entwicklungsrechner die Datei noch hat.

Einbetten macht die Grafik zum Teil des Build-Ergebnisses: Das Bild reist mit der Assembly, ist über dieselbe Buildnummer versioniert und lässt sich in keinem Kopierschritt verlieren. Es ist außerdem der Mechanismus, auf dem jedes Wisej.NET-Icon-Pack aufbaut – und genau das macht Modul 6 möglich.

2. Zentrale Konzepte

Konzept	Warum es wichtig ist
Build Action = Embedded Resource	Die Dateieigenschaft in Visual Studio, die eine Datei in das Manifest der Assembly kompiliert, statt sie neben die ausführbare Datei zu kopieren – das Asset wird also innerhalb der DLL ausgeliefert.
resource.wx-Handler	Der HTTP-Handler von Wisej.NET, der eingebettete Ressourcen an den Browser ausliefert; zugewiesen wird ein Bildquellen-String, nie ein serverseitiges Bildobjekt.
Unqualifizierte Ressourcen-URL	<code>resource.wx/company-logo.svg</code> bittet Wisej.NET, die geladenen Assemblys nach einer Ressource dieses Namens zu durchsuchen – bequem und ein wenig mehrdeutig.
Mit Assembly qualifizierte Ressourcen-URL	<code>resource.wx/MyCompany.Branding/company-logo.svg</code> benennt die Assembly (ohne <code>.dll</code>), sodass die Auflösung nicht bei der gleichnamigen Datei eines anderen landen kann.
Erlaubte Ressourcentypen	Der Handler dokumentiert genau neun Erweiterungen: <code>.css</code> , <code>.js</code> , <code>.html</code> , <code>.jpg</code> , <code>.jpeg</code> , <code>.png</code> , <code>.gif</code> , <code>.svg</code> und <code>.bmp</code> ; alles andere kommt als 404 zurück.
Überschreiben durch ausgelieferte Datei	Eine mit der Anwendung ausgelieferte Datei kann unter demselben Pfad anstelle der eingebetteten Ressource ausgeliefert werden – so kann eine Installation Grafiken umbranden oder hotfixen, ohne die Assembly neu zu bauen.
Manifest-Ressourcenname	Der Name, den .NET tatsächlich speichert, abgeleitet aus Assembly-Name, Standardnamespace und dem Ordner der Datei; ihn zu raten ist der übliche Grund, warum eine Ressourcen-URL nichts liefert.

3. Der rote Faden der Lektion

Einbetten ist eine Build-Einstellung, kein API-Aufruf

Es gibt keine Methode aufzurufen und kein Attribut anzubringen. Sie fügen die Datei dem Projekt hinzu, öffnen das Eigenschaftenfenster und ändern Build Action vom Standardwert auf **Embedded Resource**. Der Compiler schreibt die Bytes in das Manifest der Assembly, die Datei erscheint nicht mehr im Ausgabeordner, und von da an ist das Asset Teil der Binärdatei. Im Lab tun Sie das fünfmal: `logo.svg`, `status-ok.svg`, `status-warning.svg`, `photo.png` und `badge.gif`, alle in einem Ordner `Resources` der Anwendung IconDesk.

Dieselbe Eigenschaft funktioniert in einer Klassenbibliothek, und das ist der interessante Fall. Eine Bibliothek, deren einzige Aufgabe es ist, Grafiken zu tragen, ist eine weiterverteilbare Sammlung von Icons: Referenzieren Sie sie, und jeder Bildschirm, der sie referenziert, kann diese Dateien adressieren. Wisej.NET-Icon-Packs sind genau das – mehrere hundert SVGs in einer Assembly –, und in Modul 6 bauen Sie Ihr eigenes. An der Technik ändert sich zwischen Anwendung und Bibliothek nichts; nur der Name, den Sie eventuell in die URL schreiben müssen.

Stellen Sie dem die Alternativen gegenüber. Eine Content-Datei mit *Copy to Output Directory* bleibt lose: leicht zu ersetzen, leicht zu verlieren. Eine externe URL verschiebt das Problem auf einen anderen Host.

Betten Sie ein, wenn das Asset zum Code gehört und mit ihm versioniert werden soll; die Entscheidung ist betrieblich, nicht stilistisch.

Adressieren: resource.wx mit und ohne Assembly-Namen

Sobald die Datei eingebettet ist, erreicht der Browser sie über den Handler `resource.wx`. Der Wert, den Sie `ImageSource` zuweisen, ist ein String, und der Client löst ihn als URL auf; der Server erzeugt dafür nie ein `System.Drawing.Image`. Diese Unterscheidung ist bei SVG am wichtigsten, denn SVG ist überhaupt kein `System.Drawing`-Format: Es ist Markup, das der Browser rendert – genau deshalb bleibt es bei 16, 24 oder 96 Pixeln scharf.

```
// IconDesk – ResourceLab.cs
// jede Datei unten hat Build Action = Embedded Resource

// unqualifiziert: Wisej.NET durchsucht die geladenen Assemblys
picStatusOk.ImageSource = "resource.wx/status-ok.svg";
picPhoto.ImageSource    = "resource.wx/photo.png";
picBadge.ImageSource     = "resource.wx/badge.gif";

// mit dem Assembly-Namen qualifiziert, ohne ".dll" – keine Mehrdeutigkeit
picLogo.ImageSource      = "resource.wx/IconDesk/logo.svg";

// dieselbe URL funktioniert außerhalb von Eigenschaften, z. B. in CSS:
// .brand-mark { background-image: url('resource.wx/IconDesk/logo.svg'); }
```

Lassen Sie das Assembly-Segment weg, durchsucht Wisej.NET die geladenen Assemblys nach einer passenden Ressource. Das ist in Ordnung, solange eine Assembly den Namen besitzt. Kommt eine zweite Icon-Bibliothek oder die Branding-Assembly eines Kunden hinzu, können zwei Dateien namens `logo.svg` beide antworten. Den Pfad zu qualifizieren beseitigt die Frage und dokumentiert für die nächste Leserin, wem die Datei gehört. Im Lab qualifizieren Sie genau eine Quelle, damit der Unterschied in Ihrem eigenen Code sichtbar wird.

Weil der Handler schlichtes HTTP spricht, kann dieselbe URL-Form in einem Stylesheet, einem HTML-Fragment oder einem Stück JavaScript auftauchen und nicht nur in einer Bildeigenschaft. Beachten Sie außerdem: Das Attribut `[assembly: WisejResources]` gehört zu einer umfassenderen Geschichte über das Finden und Bündeln eingebetteter Skripte, Styles und Themes; eine Ressourcen-URL funktioniert mit und ohne dieses Attribut – fügen Sie also keine Attribute mechanisch hinzu, nur um ein Bild anzuzeigen.

Was der Handler ausliefert und was nicht

Die dokumentierte Liste ist kurz und einprägenswert, denn der Fehlerfall ist leise. Der Handler liefert `.css`, `.js`, `.html`, `.jpg`, `.jpeg`, `.png`, `.gif`, `.svg` und `.bmp` aus. Sechs der neun sind Bilder; die übrigen drei gibt es, weil eine Bibliotheks-Assembly neben ihren Grafiken oft ein Stylesheet oder ein Skript mitbringt. Alles andere – eine Icon-Datei, eine Schriftart, ein PDF – kommt als 404 zurück, so korrekt es auch eingebettet wurde.

Die Erweiterung ist also eine Entscheidung, die fällt, bevor die Datei ins Projekt kommt: Ein Vektorlogo wird `.svg`, ein Foto `.png` oder `.jpg`, eine animierte Markierung `.gif`. Wenn Ihnen jemand ein Format übergibt, das der Handler nicht ausliefert, gehört das Konvertieren zur Arbeit dazu.

Wann eine ausgelieferte Datei gewinnt und wie Sie einen 404 diagnostizieren

Wisej.NET lässt eine mit der Anwendung ausgelieferte Datei unter demselben logischen Pfad Vorrang vor der eingebetteten Ressource haben. Das ist gewollt und nützlich: Eine Kundeninstallation kann ihr eigenes `Logo.svg` hinterlegen, und die Anwendung brandet sich um, ohne dass jemand die Ressourcen-Assembly neu baut oder ausliefert – derselbe Weg hotfixt auch ein falsches Icon. Im Lab führen Sie eine Überschreibung durch und halten den Zustand davor und danach fest.

Das Risiko ist, dass das Deployment-Verzeichnis nun Teil Ihrer Vertrauensgrenze ist: Wer dort Dateien schreiben darf, kann ändern, was die Anwendung anzeigt. Das zweite Risiko ist das Auseinanderdriften, weil die laufende Anwendung nicht mehr der Assembly entspricht, aus der sie gebaut wurde. Behandeln Sie die Überschreibbarkeit als Konfiguration, mit einem Verantwortlichen und einem Protokoll.

Wenn ein Bild nicht erscheint, raten Sie nicht an Manifest-Namen herum. Arbeiten Sie stattdessen eine feste Liste ab; jeder Schritt schließt eine Ursache aus:

- Prüfen Sie, dass Build Action auf Embedded Resource steht und nicht auf Content oder None.
- Prüfen Sie, dass die Erweiterung eine der neun ist, die der Handler ausliefert.
- Prüfen Sie, dass die Assembly im laufenden Prozess tatsächlich geladen ist.
- Prüfen Sie die Assembly- und Pfadsegmente in der URL Zeichen für Zeichen.
- Prüfen Sie, ob ein geänderter Standardnamespace, Ordner oder logischer Name den Manifest-Ressourcennamen verschoben hat.
- Bauen Sie neu, fordern Sie dann die exakte URL im Browser an und lesen Sie den Statuscode.



Zuerst den Mechanismus benennen

Bevor Sie ein Bild zuweisen, sagen Sie, welchen Mechanismus Sie verwenden: ein `System.Drawing.Image`-Objekt, das der Server als PNG rendert, einen Bildquellen-String, den der Browser auflöst, ein benanntes Theme-Bild, eine über `resource.wx` eingebettete Ressource oder einen Icon-Font, der als HTML gezeichnet wird. Weisen Sie pro Steuerelement genau eine Bildeigenschaft zu, damit die anderen sie nicht still zurücksetzen, halten Sie den Quell-String aussagekräftig statt als eingefügten magischen Pfad, bevorzugen Sie SVG für Oberflächen-Icons, damit sie in jeder Größe und Skalierung scharf bleiben, und geben Sie jedem bedeutungstragenden Icon eine Textalternative.



Lernziele

Am Ende dieses Moduls können Sie:

- ✓ Unterstützte Bilddateien in eine Anwendungs- oder Klassenbibliotheks-Assembly einbetten.
- ✓ `resource.wx` mit und ohne Assembly-Namen verwenden.
- ✓ Die Ressourcentypen aufzählen, die der Handler für eingebettete Ressourcen von Wisej.NET unterstützt.
- ✓ Erklären, wie ausgelieferte Dateien eingebettete Ressourcen überschreiben können und warum das nützlich sein kann.

Lab- und Prüfungsleitfaden

Was Sie im praktischen Lab bauen, das vorgeschlagene Vorgehen und die geforderten Ergebnisse.



Praktisches Lab

Legen Sie in der Anwendung IconDesk einen Ordner Resources mit logo.svg, status-ok.svg, status-warning.svg, photo.png und badge.gif an und setzen Sie für jede dieser Dateien Build Action auf Embedded Resource. Bauen Sie eine Seite ResourceLab, die alle fünf über resource.wx-Bildquellen anzeigt, schreiben Sie eine dieser Quellen mit dem Assembly-Namen der Anwendung vor dem Dateinamen, damit die Auflösung eindeutig ist, und lassen Sie die übrigen unqualifiziert. Legen Sie dann eine Ersatzdatei neben die ausgelieferte Anwendung, um eine eingebettete Ressource zu überschreiben, und halten Sie den Zustand davor und danach fest. Schreiben Sie zum Abschluss eine Ressourcen-URL absichtlich falsch und notieren Sie die Schritte, die den Fehler gefunden haben.

Ausgangspunkt

Öffnen Sie IconDesk in dem Zustand nach Modul 4: das Befehls-Panel, die Seite ImageLab, die im Designer gefüllte IconGallery und die Seite IconCompare, die zwei offizielle Icon-Packs nebeneinanderstellt. Jedes Asset kommt bislang aus dem Theme, aus einer losen Projektdatei, aus einer URL oder aus einer Pack-Assembly; nichts wird bisher in der eigenen Binärdatei von IconDesk ausgeliefert, und es gibt keinen Ordner **Resources** im Projekt.

Vorgeschlagenes Vorgehen

- ✓ Legen Sie zuerst den Ordner **Resources** und die fünf Dateien an, setzen Sie dann für jede Build Action auf Embedded Resource und bauen Sie neu, bevor Sie C# schreiben.
- ✓ Fügen Sie eine Seite **ResourceLab** mit fünf PictureBox- oder Label-Steuer-elementen hinzu und weisen Sie jeder **ImageSource** einen **resource.wx**-String zu, wobei Sie nur das Logo mit dem Assembly-Namen **IconDesk** qualifizieren.
- ✓ Führen Sie die Seite aus, öffnen Sie das Netzwerk-Panel des Browsers und prüfen Sie, dass jedes Asset 200 liefert und die SVGs beim Vergrößern scharf bleiben.
- ✓ Veröffentlichen oder kopieren Sie die Anwendung in einen Deployment-Ordner, legen Sie eine Ersatzdatei über ein eingebettetes Asset, laden Sie ohne Neubau neu und erstellen Sie Screenshots von beiden Zuständen.
- ✓ Machen Sie eine URL absichtlich kaputt, arbeiten Sie dann die Fehlersuchliste der Reihe nach ab und halten Sie fest, welcher Schritt die Ursache gezeigt hat.

Geforderte Ergebnisse

- ✓ Ordner Resources mit logo.svg, status-ok.svg, status-warning.svg, photo.png und badge.gif, alle als Embedded Resource markiert
- ✓ Seite ResourceLab, die jedes eingebettete Asset über eine resource.wx-Bildquelle anzeigt
- ✓ Eine Bildquelle, die mit dem Assembly-Namen der Anwendung qualifiziert ist, während die übrigen unqualifiziert bleiben

- ✓ Ausgelieferte Datei, die eine eingebettete Ressource überschreibt, dokumentiert davor und danach
- ✓ Fehlersuche-Notiz mit den Schritten, die eine falsch geschriebene Ressourcen-URL identifiziert haben

Wiederholungsfragen

- ✓ Zwei referenzierte Assemblys betten beide eine Datei namens `logo.svg` ein. Worauf löst eine unqualifizierte Quelle auf, und was würden Sie ändern, damit die Antwort nicht mehr von der Ladereihenfolge abhängt?
- ✓ Ihre Überschreibung funktioniert auf dem Testserver, aber nach dem nächsten Release nicht mehr. Welche Teile des Deployment-Prozesses könnten sie entfernt haben, und wo hätte die Überschreibung dokumentiert werden müssen?
- ✓ Ein Teammitglied bettet `icon.ico` ein und meldet, resource.wx sei kaputt. Was passiert tatsächlich, und was ist die kleinste Änderung, die das behebt?



Zuerst den Mechanismus benennen

Bevor Sie ein Bild zuweisen, sagen Sie, welchen Mechanismus Sie verwenden: ein `System.Drawing.Image`-Objekt, das der Server als PNG rendert, einen Bildquellen-String, den der Browser auflöst, ein benanntes Theme-Bild, eine über resource.wx eingebettete Ressource oder einen Icon-Font, der als HTML gezeichnet wird. Weisen Sie pro Steuerelement genau eine Bildeigenschaft zu, damit die anderen sie nicht still zurücksetzen, halten Sie den Quell-String aussagekräftig statt als eingefügten magischen Pfad, bevorzugen Sie SVG für Oberflächen-Icons, damit sie in jeder Größe und Skalierung scharf bleiben, und geben Sie jedem bedeutungstragenden Icon eine Textalternative.