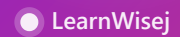


Lokalisierungsarchitektur für den Produktivbetrieb, Tests und Abschlussprojekt

Lektionsleitfaden · Lab- und Prüfungsleitfaden



Lektionsleitfaden

Lokalisierung, die den Produktivbetrieb übersteht: ein schlanker LocalizationService über dem ResourceManager, Domänencode, der Werte statt Sätze zurückgibt, eine beobachtbare Richtlinie für fehlende Schlüssel, eine Lokalisierungs-Testmatrix, Pseudo-Lokalisierung, die fest verdrahtete Zeichenfolgen aufdeckt, die Auslieferung von Satellitenassemblies und das GlobalDesk-Abschlussprojekt. Was das für Entwicklerinnen und Entwickler bedeutet, die die Grundlagen von C# und die Wisej.NET-Controls kennen und die Anwendung nun in mehr als einer Sprache ausliefern müssen – und wie Sie dieses Modul lesen.



Fokus: Produktivbetrieb

Lokalisierung, die den Produktivbetrieb übersteht: ein schlanker LocalizationService über dem ResourceManager, Domänencode, der Werte statt Sätze zurückgibt, eine beobachtbare Richtlinie für fehlende Schlüssel, eine Lokalisierungs-Testmatrix, Pseudo-Lokalisierung, die fest verdrahtete Zeichenfolgen aufdeckt, die Auslieferung von Satellitenassemblies und das GlobalDesk-Abschlussprojekt.

1. Das Problem im Produktivbetrieb

Nach sechs Modulen spricht GlobalDesk zwei Sprachen, doch die Mechanik ist verstreut: Designer-Ressourcen auf dem Kundeneditor, ein Helper über den gemeinsamen `.resx`-Dateien, eine Sprachauswahl, ein `CultureChanged`-Handler und eine RTL-Vorführung. Noch sagt nichts, was geschieht, wenn ein Schlüssel keine Übersetzung hat, und niemand hat belegt, dass der deutsche Build noch in ein schmales Fenster passt. Ein Lokalisierungssystem für den Produktivbetrieb ist vorhersehbar: Es weiß, woher jede Zeichenfolge kommt, welche Kultur in dieser Sitzung aktiv ist, wie eine fehlende Übersetzung aussieht und was ein Sprachwechsel mit geöffneten Bildschirmen macht. Dieses Modul macht aus diesen vier Antworten einen kleinen Service, eine schriftliche Testmatrix und das fertige Abschlussprojekt.

2. Zentrale Konzepte

Konzept	Warum es wichtig ist
<code>LocalizationService</code>	Eine kleine Klasse, die die Entscheidungen Ihrer Anwendung rund um die .NET-Lokalisierung bündelt, statt sie zu ersetzen: ein Ressourcen-Basisname, eine Richtlinie für fehlende Schlüssel, ein Satz Formatierungsmethoden.
Richtlinie für fehlende Schlüssel	Das gewählte, getestete Verhalten, wenn <code>GetString</code> nichts liefert: eine sichtbare Markierung <code>[Key]</code> während der Entwicklung, ein protokollierter Schlüssel plus neutraler Rückfalltext im Produktivbetrieb.
Produkt-, System- und Diagnosetext	Drei Familien mit drei Eigentümern: Ihre eigenen <code>.resx</code> -Zeichenfolgen, die Beschriftungen von Wisej.NET über <code>Resources-[LANG].resx</code> und Protokoll- oder Ausnahmetext, der in einer einzigen Entwicklersprache bleibt.
kulturfreie Domäne	Das Modell stellt einen Wert wie <code>TicketStatus.Closed</code> bereit und die Oberfläche bildet ihn auf eine lokalisierte Beschriftung ab, sodass Geschäftslogik und ihre Tests nie von einer Sprache abhängen.
Lokalisierungs-Testmatrix	Eine schriftliche Liste von Läufen — neutral, übersetzt, mit Regionsrückfall, fehlender Schlüssel, formatierte Werte, Wechsel zur Laufzeit, lange Zeichenfolgen, RTL, Systembeschriftungen —, sodass Abdeckung eine Checkliste ist und keine Erinnerung.
Pseudo-Lokalisierung	Ein Testressourcensatz, dessen Werte sichtbar verändert und verlängert sind, etwa <code>[!!! Save customer record !!!]</code> , gebaut, um Abschneidungen und fest verdrahtete Zeichenfolgen aufzudecken, und nicht, um etwas zu übersetzen.
Satellitenressourcen	.NET kompiliert kulturspezifische Ressourcen in Satellitenassemblies, deshalb müssen die vom Build erzeugten Kulturordner die Anwendung begleiten, wenn Sie sie veröffentlichen oder paketieren.

3. Der rote Faden der Lektion

Ein Service, dem jede Suche gehört

Bisher hat jeder Bildschirm direkt nach dem Ressourcen-Helfer gegriffen. Das funktioniert, aber es verstreut Entscheidungen: welcher Ressourcen-Basisname, was ein fehlender Schlüssel bewirkt, welche Kultur eine Währung formatiert. Ein schlanker `LocalizationService` sammelt diese Entscheidungen in einer Klasse und ändert sonst nichts — er liest weiterhin über `Wisej.Resources.ResourceManager`, verlässt sich weiterhin auf den gewöhnlichen .NET-Kulturrückfall und formatiert weiterhin mit `Wisej.Web.Application.CurrentCulture`. Was er Ihnen bringt, ist eine einzige Stelle, an der Sie Ihre Meinung ändern können, und eine einzige Stelle, die ein Reviewer lesen muss.

```

public sealed class LocalizationService
{
    private readonly Wisej.Resources.ResourceManager _resources =
        new("GlobalDesk.Resources.Strings", typeof(LocalizationService).Assembly);

    public string Text(string key)
        ⇒ _resources.GetString(key) ?? $"[{key}]";

    public string Currency(decimal value)
        ⇒ value.ToString("C", Wisej.Web.Application.CurrentCulture);

    public string Date(DateTime value)
        ⇒ value.ToString("d", Wisej.Web.Application.CurrentCulture);
}

```

Zwei Einzelheiten zählen mehr als der Code. Erstens gibt `Text` nie etwas Leeres zurück: Ein unbekannter Schlüssel kommt als `[CustomerEditor.Save]` zurück, was auf dem Bildschirm unmöglich zu übersehen und in einem Screenshot mühelos zu finden ist. Zweitens liest jede Formatierungsmethode `Application.CurrentCulture` im Moment des Aufrufs. Die Kultur ist in Wisej.NET Zustand pro Sitzung, deshalb würde eine in einem Feld — schlimmer noch: in einem statischen Feld — zwischengespeicherte `CultureInfo` stillschweigend einem Benutzer die Sprache eines anderen geben. Machen Sie den Service zur einzigen Stelle, an der die Oberfläche den `ResourceManager` berührt, und eine Suche nach diesem Typ in der Projektmappe sollte genau einen Treffer liefern.

Drei Textfamilien, drei verschiedene Eigentümer

Produkttext gehört Ihnen: Beschriftungen, Meldungen, Validierungsformulierungen, all die Sätze, die ein Übersetzer je zu sehen bekommt, abgelegt in Ihren `.resx`-Dateien unter stabilen semantischen Schlüsseln. Wisej.NET-Systemtext gehört dem Framework: die Schaltflächen einer `MessageBox`, die Beschriftungen der eingebauten Dialoge, je Sprache mit einer Datei `Resources-[LANG].resx` überschrieben. Diagnosetext ist für Entwickler: das, was Sie in ein Protokoll schreiben oder einer Ausnahme anhängen. Er bleibt bewusst in einer Sprache, denn ein ins Deutsche übersetzter Stacktrace ist ein Stacktrace, nach dem niemand suchen kann.

Dieselbe Trennung gilt innerhalb der Anwendung. Ein Ticket ist nicht „Closed“; es hat einen Status, und `TicketStatus.Closed` ist der Wert, den das Modell bereitstellt. Die Oberfläche bildet diesen Wert über den Lokalisierungsservice auf einen Ressourcenschlüssel ab und fragt nach der Beschriftung.

Domänencode, der Sätze zurückgibt, zieht eine Sprache in Ihre Geschäftsregeln und in jeden Test, der auf ihnen beharrt; Domänencode, der Werte zurückgibt, hält beides stabil, während die Oberfläche eine vierte Sprache bekommt.

Eine Testmatrix statt einer Stichprobe

Einmal auf die Sprachauswahl zu klicken und deutsche Beschriftungen zu sehen belegt sehr wenig. Eine brauchbare Lokalisierungs-Testmatrix ist aufgeschrieben und wird wiederholt: die neutrale Kultur; mindestens eine übersetzte Kultur; eine Sprach-Regions-Kultur, die auf ihre Sprachressource zurückfallen muss; eine absichtlich fehlende Übersetzung, damit Sie den Rückfall sehen und keine Leere; Datums-, Zahlen-, Prozent- und Währungswerte; ein Kulturwechsel zur Laufzeit mit bereits geöffneten Bildschirmen;

ein Durchgang mit langen Zeichenfolgen; eine RTL-Kultur mit gespiegelmtem Layout; Systembeschriftungen von Wisej.NET in einer `MessageBox`; und die Validierungs- und Fehlermeldungen, also die Zeichenfolgen, die Teams am häufigsten vergessen.

Pseudo-Lokalisierung braucht keine besonderen Werkzeuge. Bauen Sie eine Testressourcendatei, deren Werte sichtbar verändert und verlängert sind, etwa `[!!! Save customer record !!!]`, und führen Sie die Anwendung damit aus. Genauigkeit spielt keine Rolle; der Sinn ist, dass abgeschnittene Schaltflächen und beengte Labels offensichtlich werden und dass alles, was während dieses Laufs schlicht englisch bleibt, ein Kandidat für eine fest verdrahtete Zeichenfolge ist. Ein Hinweis zur Vorsicht: Manche Premium-Erweiterungen für Wisej.NET kapseln JavaScript-Bibliotheken von Drittanbietern, deren Lokalisierung eine eigene Kultureinstellung des Anbieters hat, und die Erweiterungskonfiguration unterstützt Platzhalter für Kultur und Gebietsschema bei Komponentenressourcen. Behandeln Sie das als Integrationsschicht über Ihrer Kultur und testen Sie die gekapselte Komponente nach einem Sprachwechsel, statt anzunehmen, dass ihre Beschriftungen Ihrer `.resx` folgen.

Was den veröffentlichten Build begleiten muss

.NET kompiliert kulturspezifische Ressourcen üblicherweise in Satellitenassemblies, die der Build in nach Kultur benannte Ordner neben der Anwendung legt. `ResourceManager` findet sie über die standardmäßige .NET-Suche; es gibt nichts von Hand zu laden. Die Lektion für die Bereitstellung ist deshalb kurz: Werfen Sie kulturspezifische Ausgabeordner nicht weg, wenn Sie die Anwendung veröffentlichen oder paketieren.

Das ist die Art Fehler, die ein Bereitstellungsskript stillschweigend macht. Ein Kopierschritt, der nur die Hauptassembly nimmt, oder ein Image, das aus einer kuratierten Dateiliste gebaut wird, ergibt eine Anwendung, die auf Ihrem Rechner perfekt aussieht und überall sonst auf Englisch zurückfällt. Ergänzen Sie die Checkliste des Abschlussprojekts um eine Zeile: nach dem Veröffentlichen den Ausgabeordner öffnen und bestätigen, dass die Kulturordner da sind.

GlobalDesk fertigstellen

Das Abschlussprojekt ist keine große Anwendung; es ist der Beleg, dass Sie eine von Anfang bis Ende lokalisieren können, statt ein Label zu übersetzen. GlobalDesk braucht eine neutrale `.resx` mit einem sinnvollen Satz semantischer Schlüssel, mindestens eine übersetzte Ressourcendatei, einen im Designer lokalisierten Bildschirm, den `LocalizationService`, eine Sprachauswahl, die `Application.CurrentCulture` setzt, eine kulturbewusste Anzeige von Datum, Zahl und Währung, eine Vorführung der Wisej.NET-Systemressourcen, eine RTL-Vorführung, Notizen zum Übersetzungsworkflow und die QA-Checkliste.

Proben Sie die Demo in einer festen Reihenfolge, denn die Reihenfolge ist das Argument: Beginnen Sie mit der automatisch erkannten Browserkultur, wechseln Sie zu Deutsch, zeigen Sie, wie übersetzte Beschriftungen und formatierte Werte gemeinsam umspringen, lösen Sie eine `MessageBox` aus, damit eine Systembeschriftung erscheint, zeigen Sie eine absichtlich fehlende Übersetzung beim Zurückfallen, wechseln Sie zur RTL-Kultur und betrachten Sie das gespiegelte Layout und zeigen Sie dann die Workflow-Notizen und die Checkliste. Wenn jeder Schritt funktioniert, ohne dass Sie etwas bearbeiten, ist die Lokalisierung fertig.



Gewohnheit für die Lokalisierungsschichten

Entscheiden Sie, wo jede Zeichenfolge lebt, bevor Sie sie tippen. Designer-Ressourcen gehören zu einem entworfenen Bildschirm und werden beim Erstellen seiner Controls angewendet; gemeinsame Anwendungsressourcen in .resx tragen den Text, den Code erzeugt oder den mehrere Bildschirme wiederverwenden; Resources-[LANG].resx überschreibt die Systembeschriftungen von Wisej.NET. Speichern Sie niemals ein formatiertes Datum, eine formatierte Zahl oder einen formatierten Währungsbetrag: Behalten Sie den Wert und formatieren Sie ihn zum Anzeigzeitpunkt für die aktive Kultur. Und denken Sie daran, dass die Kultur pro Sitzung gilt, sodass die Sprache eines Benutzers niemals den Bildschirm eines anderen verändern darf.



Lernziele

Am Ende dieses Moduls können Sie:

- ✓ einen kleinen Lokalisierungsservice um den ResourceManager herum erstellen.
- ✓ Produkttext, Systemtext und Diagnosetext voneinander trennen.
- ✓ Rückfall, Formatierung, RTL und den Kulturwechsel zur Laufzeit testen.
- ✓ die Bereitstellung von Satellitenressourcen im Überblick verstehen.
- ✓ das GlobalDesk-Lokalisierungs-Abschlussprojekt abschließen.

Lab- und Prüfungsleitfaden

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



Praktisches Lab

Bringen Sie das GlobalDesk-Abschlussprojekt zu Ende. Kapseln Sie jede Ressourcensuche in einem `LocalizationService` mit den Methoden `Text`, `Date`, `Number` und `Currency`, geben Sie ihm eine Richtlinie für fehlende Schlüssel, die in der Entwicklung einen in Klammern gesetzten Schlüssel anzeigt und ihn im Produktivbetrieb protokolliert, und machen Sie diesen Service zur einzigen Stelle, an der die Oberfläche den `ResourceManager` berührt. Halten Sie die Domäne heraus: Das Ticket-Modell stellt einen `TicketStatus`-Wert bereit, und die Oberfläche bildet ihn auf eine lokalisierte Beschriftung ab. Führen Sie dann die Lokalisierungs-Testmatrix über die fertige Anwendung aus – die neutrale Kultur, Deutsch, eine Sprach-Regions-Kultur, die zurückfällt, ein absichtlich fehlender Schlüssel, formatierte Datumsangaben und Währungsbeträge, ein Wechsel zur Laufzeit, ein Durchgang mit langen Zeichenfolgen und die RTL-Kultur – und nutzen Sie dafür einen pseudo-lokalisierten Ressourcensatz, der alles aufdeckt, was noch fest verdrahtet ist. Vergewissern Sie sich, dass die Kulturordner eine Veröffentlichung überstehen, und reichen Sie die QA-Checkliste als Ergebnis des Abschlussprojekts ein.

Ausgangspunkt

GlobalDesk so, wie Sie es nach Modul 6 hinterlassen haben: ein Dashboard und ein im Designer lokalisierter Kundeneditor, `Resources/Strings.resx` mit seinen deutschen und italienischen Begleitdateien, der Helper, der sie liest, eine Sprachauswahl, die `Application.CurrentCulture` setzt, ein `CultureChanged`-Handler, der Texte erneut anwendet und den Editor neu aufbaut, RTL-Unterstützung und Ihre Workflow-Notizen. Jede Schicht funktioniert, aber Ressourcensuchen finden noch an mehreren Stellen statt, ein paar Laufzeitmeldungen sind weiterhin englische Literale im Code, und niemand hat aufgeschrieben, wie die Anwendung getestet wird.

Vorgeschlagenes Vorgehen

- ✓ Schreiben Sie zuerst den `LocalizationService`: einen einzigen `Wisej.Resources.ResourceManager` über Ihrem `Strings`-Basisnamen und der Assembly, eine Methode `Text(key)`, die eine Klammermarkierung `[Key]` zurückgibt, wenn die Suche nichts liefert, und die Methoden `Date`, `Number` und `Currency`, die mit `Application.CurrentCulture` formatieren, gelesen zum Aufrufzeitpunkt. Geben Sie dem Pfad für fehlende Schlüssel einen einzigen Schalter: Markierung während der Entwicklung, protokollierter Schlüssel plus neutraler Text im Produktivbetrieb.
- ✓ Machen Sie ihn zur einzigen Tür. Ersetzen Sie jeden verbliebenen direkten Ressourcenaufruf und jedes für den Benutzer sichtbare englische Literal durch einen Serviceaufruf, suchen Sie dann in der Projektmappe nach `ResourceManager` und bestätigen Sie, dass der Service der einzige Treffer ist. Lassen Sie Protokoll- und Ausnahmetext unangetastet und ergänzen Sie einen Kommentar, der sagt, dass er bewusst Diagnosetext ist.
- ✓ Drängen Sie die Kultur aus der Domäne: Lassen Sie das Ticket-Modell einen `TicketStatus`-Wert zurückgeben, ergänzen Sie in der Oberfläche oder im Service eine kleine Zuordnung von

diesem Wert zu einem Ressourcenschlüssel wie `TicketStatus.Closed` und prüfen Sie, dass im Modell und in seinen Tests keine englische Beschriftung mehr steht.

- ✓ Bauen Sie den pseudo-lokalisierten Ressourcensatz — dieselben Schlüssel mit verlängerten und in Klammern gesetzten Werten, zum Beispiel `[!!! Save customer record !!!]` —, führen Sie die Anwendung damit aus und listen Sie jede Beschriftung auf, die schlicht englisch geblieben ist, und jedes Control, dessen Text abgeschnitten wurde. Beheben Sie beide Listen.
- ✓ Schreiben Sie die QA-Checkliste als Tabelle und gehen Sie sie durch: neutrale Kultur, Deutsch, eine Sprach-Regions-Kultur, die auf ihre Sprachressource zurückfällt, ein absichtlich fehlender Schlüssel, ein Datum, eine Zahl und ein Währungsbetrag, ein Wechsel zur Laufzeit bei geöffnetem Editor, der Durchgang mit langen Zeichenfolgen, die RTL-Kultur und eine Systembeschriftung in einer `MessageBox`. Veröffentlichen Sie dann die Anwendung und bestätigen Sie, dass die Kulturordner und ihre Ressourcenassemblies in der Ausgabe liegen.

Das Abschlussprojekt wird als eine funktionierende Anwendung geprüft und nicht als zehn getrennte Übungen: eine neutrale Ressourcendatei mit einem sinnvollen Satz semantischer Schlüssel, mindestens eine übersetzte Ressourcendatei, ein im Designer lokalisierter Bildschirm, der `LocalizationService`, eine Sprachauswahl, die `Application.CurrentCulture` ändert, eine kulturbewusste Anzeige von Datum, Zahl und Währung, eine Vorführung der Wisej.NET-Systemressourcen, eine RTL-Vorführung, Notizen zum Übersetzungsworkflow und die QA-Checkliste. Proben Sie die Demo in einer festen Reihenfolge: Beginnen Sie mit der automatisch erkannten Browserkultur, wechseln Sie zu Deutsch, zeigen Sie, wie übersetzte Beschriftungen und formatierte Werte gemeinsam umspringen, lösen Sie eine `MessageBox` aus, damit eine Systembeschriftung erscheint, zeigen Sie eine absichtlich fehlende Übersetzung beim Zurückfallen, wechseln Sie zur RTL-Kultur und zeigen Sie dann die Workflow-Notizen. Denken Sie daran, dass die Kultur Sitzungszustand ist: Wenn irgendetwas, das Sie geschrieben haben, eine `CultureInfo` außerhalb der Sitzung zwischenspeichert, besteht es Ihre eigene Demo und scheitert in dem Moment, in dem ein Kollege die Anwendung neben Ihnen öffnet.

Geforderte Ergebnisse

- ✓ `LocalizationService` mit den Methoden `Text`, `Date`, `Number` und `Currency` und einer dokumentierten Richtlinie für fehlende Schlüssel
- ✓ Domänenencode, der einen `TicketStatus`-Wert zurückgibt, den die Oberfläche auf eine lokalisierte Beschriftung abbildet
- ✓ Pseudo-lokalisierter Ressourcensatz, mit dem jede verbliebene fest verdrahtete Zeichenfolge gefunden wird
- ✓ Vollständige Lokalisierungs-Testmatrix für Rückfall, Formatierung, Laufzeitwechsel und RTL
- ✓ Veröffentlichte Ausgabe, auf ihre Kulturordner geprüft, dazu die GlobalDesk-Lokalisierungs-QA-Checkliste

Wiederholungsfragen

- ✓ Ein Reviewer öffnet GlobalDesk auf Französisch, einer Kultur, die Sie nie übersetzt haben, und sieht englische Beschriftungen mit korrekt formatierten französischen Datumsangaben. Erklären

Sie, welcher Mechanismus jede Hälfte dieses Bildschirms hervorgebracht hat, und sagen Sie, ob irgendetwas kaputt ist.

- ✓ Ihr pseudo-lokalisierter Lauf zeigt ein Label, das weiterhin schlicht englisch „Save“ liest. Nennen Sie die drei Stellen, aus denen dieser Text stammen könnte, und beschreiben Sie, wie Sie sie ohne Raten auseinanderhalten würden.
- ✓ Ein Kollege schlägt vor, die aufgelöste `CultureInfo` in einem statischen Feld auf `LocalizationService` zwischenspeichern, „weil es verschwenderisch ist, sie bei jedem Aufruf nachzuschlagen“. Beschreiben Sie, was ein zweiter angemeldeter Benutzer erleben würde, und was Sie stattdessen vorschlagen würden.



Gewohnheit für die Lokalisierungsschichten

Entscheiden Sie, wo jede Zeichenfolge lebt, bevor Sie sie tippen. Designer-Ressourcen gehören zu einem entworfenen Bildschirm und werden beim Erstellen seiner Controls angewendet; gemeinsame Anwendungsressourcen in `.resx` tragen den Text, den Code erzeugt oder den mehrere Bildschirme wiederverwenden; `Resources-[LANG].resx` überschreibt die Systembeschriftungen von Wisej.NET. Speichern Sie niemals ein formatiertes Datum, eine formatierte Zahl oder einen formatierten Währungsbetrag: Behalten Sie den Wert und formatieren Sie ihn zum Anzeigzeitpunkt für die aktive Kultur. Und denken Sie daran, dass die Kultur pro Sitzung gilt, sodass die Sprache eines Benutzers niemals den Bildschirm eines anderen verändern darf.