

Lokalisierung von Microsoft™ .NET Anwendungen

Lokalisierung / Internationalisierung von Software

Unter dem Lokalisieren von Software Anwendungen versteht man das Übersetzen von Software mit speziellen auch kulturellen Anpassungen an die Zielsprache.

Das betrifft nicht allein die Texte von Steuerelementen in Windows Dialogen, sondern auch das Anpassen oder das Ändern von Icons, Bildern, die Position und Größe von Steuerelementen und Dialogen wie auch die Anpassung von Fehlertexten oder externen Dateien (z.B. XML Daten, Protokolldateien,...).

Lokalisierung von Ressourcdateien

In den bisherigen Entwicklungsumgebungen (IDEs) von Microsoft war das Lokalisieren von Anwendungen kompliziert und zeitaufwendig.

Viele Ressourcdateien¹ (*.rc) mussten erstellt und verwaltet werden. Dabei mussten verschiedene Ressourcentypen wie Grafiken, Texte, Icons mit unterschiedlichen Tools erstellt werden. Der Aufwand konnte explodieren, wenn zu Beginn eines Projektes die Möglichkeit einer Anpassung an eine andere Spracheinstellung nicht berücksichtigt wurde oder die Forderung nach einer Lokalisierung während des Projekts auftauchte.

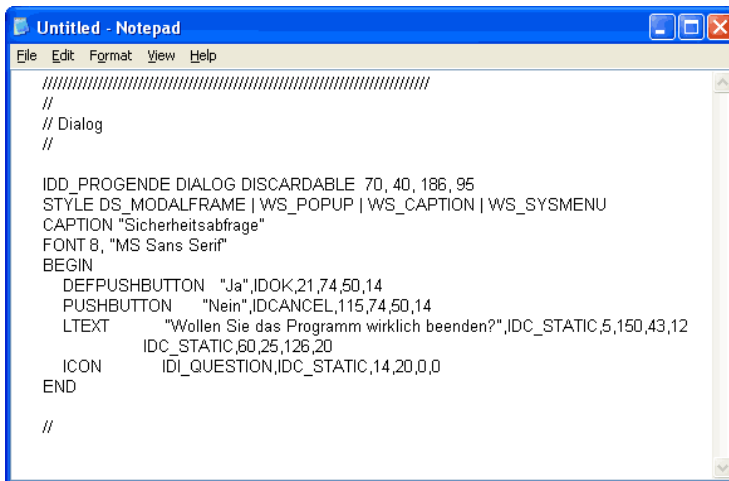


Abb. einer RC-Datei

Dieser Entwicklungs- und Übersetzungsprozess für „Standard“ MS Windows Anwendungen ist der Lokalisierungsindustrie ein wohl bekannter und verstandener Begriff.

Während einige Organisationen es vorziehen Ressource-Dateien (*.rc) zu lokalisieren und ihre Übersetzungen im lesbaren Quellcode Format aufrechtzuerhalten, entscheiden sich heute innovative Organisationen auf Basis von Windows Binärdateien (EXEs oder DLLs) zu übersetzen.

¹ Mit Ressourcen bezeichnet man alle Text- und grafischen Elemente von Software, die von der Spracheinstellung der Laufzeitumgebung betroffen sind.

Lokalisierung von Binärdateien

- + Beschleunigung des Lokalisierungsprozess
- + Reduktion der Kosten
- + Entlastung der Entwickler

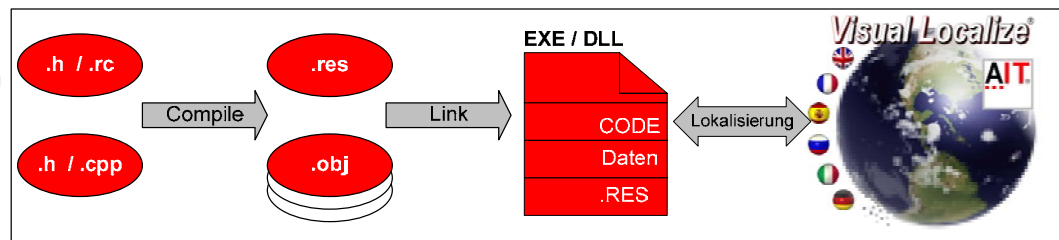
Diese fortschrittliche Entscheidung beschleunigt den Übersetzungs- und Lokalisierungsprozess, reduziert die dabei anfallenden Kosten und verbessert die Geschwindigkeit bei der Markteinführung von verschiedensprachiger Software.

Der bedeutendste technische Vorteil, der aus der Übersetzung von binären Anwendungsdateien (EXEs, DLLs) abgeleitet werden kann, ist die Verringerung der Komplexität aufgrund der gewaltigen Reduktion an Dateien, die bearbeitet werden müssen.

Während eine Binärdatei aus mehreren hundert kleineren Ressource Dateien bestehen kann, muss der Sprach-Ingenieur bei der Lokalisierung auf Binärdatei-Ebene nur mit dieser EINEN, fertig kompilierten Binärdatei arbeiten.

Diese enorme Verminderung in der Anzahl von Dateien reduziert die Komplexität bei der Verwaltung von mehreren (gleichzeitigen) Übersetzungen. Stellen Sie sich den Unterschied zwischen 15 Binärdateien z.B. EXEs und DLLs oder deren entsprechenden 1000 Quell-Ressourcen vor.

Workflow: Lokalisierung von „Standard Microsoft™ Anwendungen“

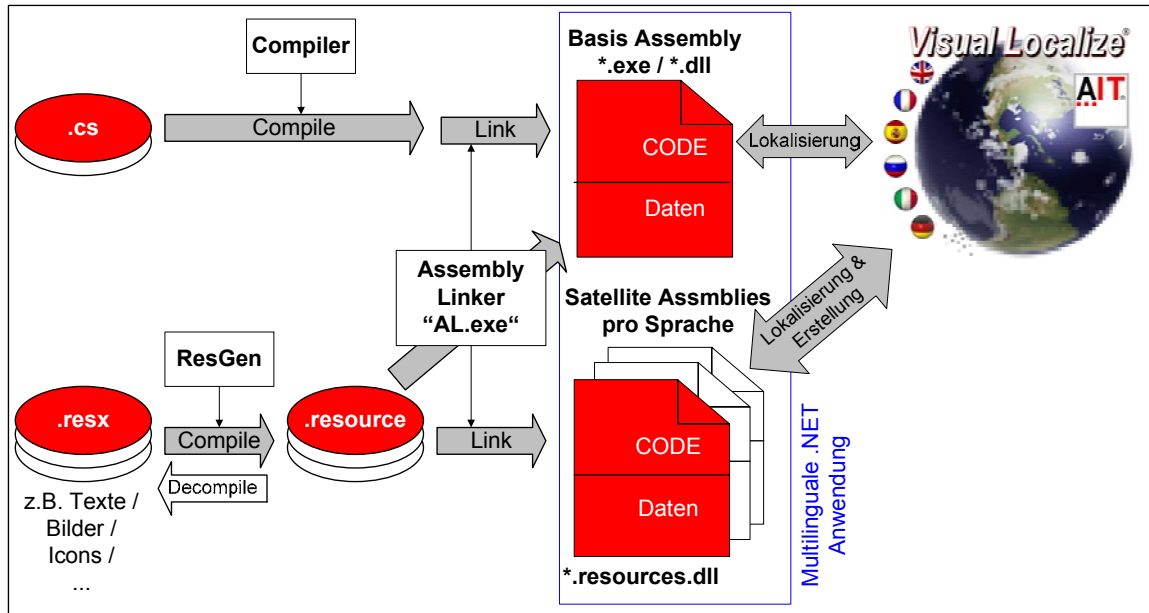


Das Software Lokalisierungstool Visual Localize (www.visloc.com) hilft Ihnen mit der ständig fortschreitenden Entwicklung Schritt zu halten und bietet eine komplette und integrierte Lösungsplattform zur Übersetzung von Binärdateien, XML Daten und Microsoft Datenbanken.

Workflow: Lokalisierung von Microsoft™ .NET Anwendungen

Im Gegensatz zur Lokalisierung von „Standard Windows Anwendungen“ ist der Lokalisierungsprozess für Microsoft .NET Anwendungen für viele Übersetzungsagenturen und Softwareentwickler noch unbekannt und stellt eine erhebliche Herausforderung dar.

Während das Microsoft™ .NET Framework eine neue Technologie bietet, um die Globalisierung aus Entwicklungssicht einfacher zu gestalten, definiert diese neue Technologie gleichzeitig den Lokalisierungsprozess neu.



Microsoft™ .NET Anwendungen können mittels Visual Basic .NET (VB .NET) oder C# erstellt werden.

Anwendungs-Ressourcen werden in einem neuen Dateityp der so genannten .RESX-Datei definiert. Diese Dateien sind XML Container, welche die Bestandteile der Anwendungsoberfläche wie Menüs, Dialoge und Strings, beschreiben. Es gibt *eine* .RESX-Datei für *jeden* Ressourcetyp (Dialog, Menü, Texte, Icons,...).

Erstellung von Ressourcen

Um „embedded“ also eingebundene Ressourcen-Dateien zu erstellen geht die Entwicklungsumgebung (IDE) Microsoft™ Visual Studio .NET“ in zwei Schritten vor:

1. Eine .resx-Datei, welche die XML basierende Beschreibung einer Anwendungs-Ressource enthält, wird mittels des Microsoft „ResGen“ Werkzeugs in eine .resource-Datei kompiliert. Dieses Tool wird als Teil des Microsoft .NET Framework SDK ausgeliefert. Die erzeugte .resource-Datei enthält ausschließlich binäre Ressourcen.

2. Diese .resource-Datei wird in einem zweiten Schritt in ein Assembly eingebaut (engl. embedded). Ein Assembly ist die technische Beschreibung von Microsoft, um eine Kombination aus Anwendungscode und Anwendungs-Ressource zu beschreiben, die verbunden werden um eine einzelne .NET Anwendung zu beschreiben. Hierzu wird entweder das Werkzeug "Assembly Linker" oder ein entsprechender Sprach-Compiler (wie CSC für den C# Compiler) verwendet.

Kultur Informationen In .NET Anwendungen

Die „Kultur“ (engl. *cultur*“) der Anwendung stellt lokal-spezifische Informationen einschließlich des verwendeten Schreibsystems, des Sortierverfahrens, des Datumsformats, der verwendeten Währung, Zeitformate, Postleitzahlenformate u. v. a. m. dar.

Das .NET Framework bietet hierzu eine *CultureInfo* Klasse, um einen einfachen programmatischen Zugriff auf die einzelnen „Sprachsysteme“ und Text-Bearbeitungsgrundlagen zu haben.

Innerhalb dieser *CultureInfo* Klasse bestimmt das *UICulture data member* wie die Anwendungs-Ressourcen geladen werden. Dieses *data member* spezifiziert somit die Sprache der Bedienoberfläche (engl. User Interface) für die .NET Anwendung. Durch den Wechsel der *data member* kann der Entwickler die Bedienoberfläche von seiner .NET Anwendung beliebig umschalten.

Nomalerweise wird der Wert der *UICulture* auf die Standard-Sprache (engl. *Default Language*) des Betriebssystems gesetzt, wenn der Entwickler den Wert der *UICulture* nicht händisch deklariert.

Sobald die *UICulture* definiert wurde, lädt die .NET Anwendung die gewünschte Ziel-Sprache und zeigt die Bedienoberfläche in dieser Sprache an.

Nach einmaligem Einstellen der *UICulture* findet die .NET Anwendung die gewählten Sprach-Dateien für die Bedienoberfläche selbst und zeigt diese an.

Die sprachabhängigen Dateien für die Bedienoberfläche sind bei einer .NET Software Anwendung in so genannten Satellite Resource DLLs (*.resources.dll) enthalten und werden von der Anwendung automatisch nach dem RFC 1766² Standard geladen.

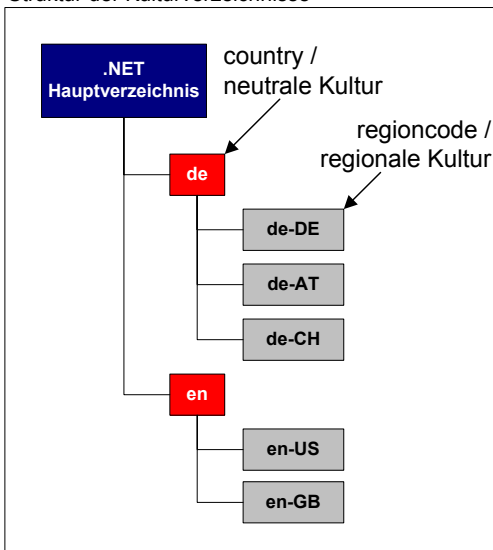
Die Satellite Resource DLLs sind dabei in einer vordefinierten Verzeichnisstruktur des Hauptverzeichnisses, von der die .NET Anwendung aus gestartet wird, abgelegt. (Siehe Abb. links).

Durch das Festlegen dieser Verzeichnisstruktur hat Microsoft ein Mechanismus entwickelt, um mehrsprachige Applikationen zu erhalten, die sich z.B. auf einem Server befinden.

Die Bedienoberfläche der Anwendung wird zur Laufzeit eingestellt, wenn die .NET Anwendung unter Berücksichtigung, der zuvor gewählten *UICulture* eines Clients, die Sprachinformationen anfordert.

Mit der Verwendung von diesem Mechanismus ist es möglich, dass eine .NET Anwendung auf dem Server abgelegt ist, während mehrere Clients verschiedene und voneinander unabhängige Sprachen und Bedienoberflächen darstellen.

Struktur der Kulturverzeichnisse

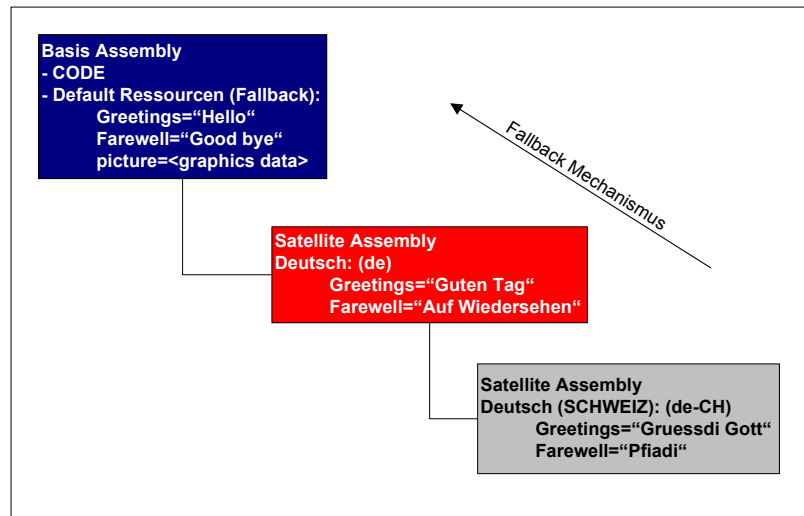


² <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpref/html/frifsystemglobalizationcultureinfoclasstopic.asp>

Weiter ist zu berücksichtigen, dass eine solche mehrsprachige Anwendung ohne codieren erreicht werden kann, da sie Teil des neuen .NET Frameworks ist.

Wenn eine .NET Anwendung ein sprachabhängiges Ressourcenset anfordert, welches nicht verfügbar ist, tritt ein Fallback Mechanismus in Kraft. (Siehe Abb. unten).

Die Abfrage wird hierbei von "unten" nach "oben" geleitet. Erst wenn in höchster Hierarchieebene im Basis Assembly keine Ressource gefunden wird, spricht das Ausnahmehandling an:



Konventionen der Laufzeitumgebung

- Ein Basis Assembly kann für jede Spracheinstellung ein Satellite Assembly besitzen
- Das Satellite Assembly muss immer denselben Basisnamen wie sein Basis Assembly haben und hat den Postfix .resources.dll
z.B. Basis Assembly: „DOTNETSample.exe“
Satellite Assembly: „DOTNETSample.resources.dll“
- Die Satellite Assembly muss in einem direkten Unterverzeichnis relativ zum Basis Assembly abgelegt sein, wobei der Name des Unterverzeichnisses identisch mit dem Kürzel der regionalen bzw. neutralen Kultur sein muss. (Siehe Abb. "Struktur der Kulturverzeichnisse")
- Ein Basis Assembly kann das Attribut *NeutralResourceLanguage* auf ein Kürzel einer neutralen Kultur setzen. Die .NET-Runtime geht dann davon aus, dass die Ressourcen für die neutrale Kultur im Basis Assembly eingebettet sind. Erkennt die .NET Runtime zur Laufzeit, dass die aktuelle Spracheinstellung dem Wert des Attributes *NeutralResourceLanguage* entspricht, wird direkt auf die eingebetteten Ressourcen der Basis Assembly zugegriffen, was einen Performancegewinn bei der gebräuchlichsten Kultur bewirkt.

Erstellung einer lokalisierbaren .NET Anwendung

Betrachten wir nun die Microsoft Visual Studio IDE und die Erstellung einer .NET Anwendung genauer.

1. Erstellung einer „Dialog form“ Ressource

Sobald eine „Dialog form“ (oder *WinForm*) mittels dem Designer erstellt wird, wird automatisch parallel eine RESX-Dateiressource im Microsoft Visual Studio .NET erstellt.

```

/// <summary>
/// Required method for Designer support - do not modify
/// the contents of this method with the code editor.
/// </summary>
private void InitializeComponent()
{
    this.button1 = new System.Windows.Forms.Button();
    this.SuspendLayout();
    //
    // button1
    //
    this.button1.Location = new System.Drawing.Point(16, 232);
    this.button1.Name = "button1";
    this.button1.TabIndex = 0;
    this.button1.Text = "button1";
    //
    // Form1
    //
    this.AutoScaleBaseSize = new System.Drawing.Size(5, 13);
    this.ClientSize = new System.Drawing.Size(292, 266);
    this.Controls.AddRange(new System.Windows.Forms.Control[] {
        this.button1});
    this.Name = "Form1";
    this.Text = "Form1";
    this.ResumeLayout(false);
}
    
```

Jedoch ist sind die Text Strings von dieser *WinForm* nicht in einer RESX-Datei abgelegt. Sie wurden in der Methode **InitializeComponent()** innerhalb der C#-Quellcode Datei (siehe blau markierten Bereich oben) gespeichert. Beim Kompilieren wird dieser Code in eine binäre .NET Anwendungsdatei umgewandelt.

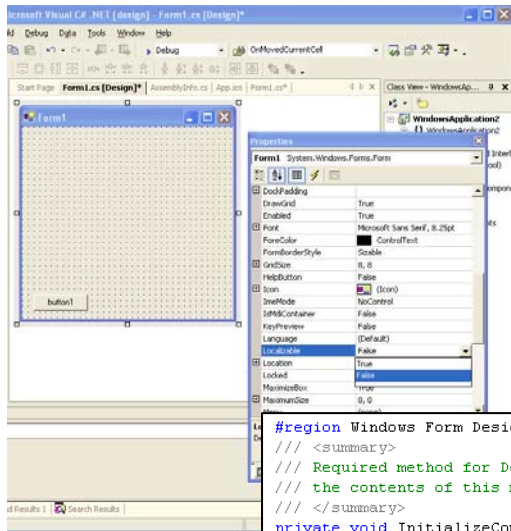
Dieses Vorgehen hilft dem Lokalisierungsprozess offensichtlich nicht, da die sprachabhängigen Texte "hart-codiert" innerhalb der Anwendung abgelegt sind.

Um die oben erstellte *WinForm* effizient übersetzen zu können benötigen wir ein Vorgehen, das alle sprachrelevanten Phrasen und Bezeichnungen in eine vom Quellcode unabhängige Datei extrahiert.

Diese externe Datei kann nachfolgend übersetzt werden, ohne dass der C#-Quellcode bearbeitet bzw. neu kompiliert werden muss.

Die Entwicklungsumgebung Microsoft VisualStudio .NET beinhaltet eine Möglichkeit automatisch diese externen Ressourcen Dateien zu erstellen. Der Entwickler muss hierzu alle *WinForms* als lokalisierbar deklarieren (Eigenschaft: **Localizable = TRUE**).

2. Definieren eine Ressource zur Lokalisierung



Diese Anweisung instruiert die Microsoft .NET IDE den Text und andere UI (User Interface) Elemente der Anwendung zu extrahieren und in einer externen RESX-Datei zu speichern.

VisualStudio .NET stellt denselben Mechanismus zur Speicherung externer Ressourcen für beide Programmiersprachen C# und VB .NET zur Verfügung. Deshalb sollte die Lokalisierung einer beliebigen Applikation analog erfolgen können.

Wenn Sie beide Quellcode-Auszüge (Abb. SC1 & SC2) miteinander vergleichen, sehen Sie die Unterschiede im C#-Code nach Änderung der Property „Localizable“ der WinForm.

```
#region Windows Form Designer generated code
/// <summary>
/// Required method for Designer support - do not modify
/// the contents of this method with the code editor.
/// </summary>
private void InitializeComponent()
{
    System.Resources.ResourceManager resources = new System.Resources.ResourceManager(typeof(Form1));
    this.button1 = new System.Windows.Forms.Button();
    this.SuspendLayout();
    //
    // button1
    //
    this.button1.Location = ((System.Drawing.Point) (resources.GetObject("button1.Location")));
    this.button1.Name = "button1";
    this.button1.RightToLeft = ((System.Windows.Forms.RightToLeft) (resources.GetObject("button1.RightToLeft")));
    this.button1.Size = ((System.Drawing.Size) (resources.GetObject("button1.Size")));
    this.button1.TabIndex = ((int) (resources.GetObject("button1.TabIndex")));
    this.button1.Text = resources.GetString("button1.Text");
    this.button1.TextAlign = ((System.Drawing.ContentAlignment) (resources.GetObject("button1.TextAlign")));
    this.button1.Visible = ((bool) (resources.GetObject("button1.Visible")));
    //
    // Form1
    //
    this.AccessibleDescription = ((string) (resources.GetObject("$this.AccessibleDescription")));
    this.AccessibleName = ((string) (resources.GetObject("$this.AccessibleName")));
    this.Anchor = ((System.Windows.Forms.AnchorStyles) (resources.GetObject("$this.Anchor")));
    this.AutoScaleBaseSize = ((System.Drawing.Size) (resources.GetObject("$this.AutoScaleBaseSize")));
    this.AutoScroll = ((bool) (resources.GetObject("$this.AutoScroll")));
```

Abb. SC2 (in RESX Datei ausgelagert)

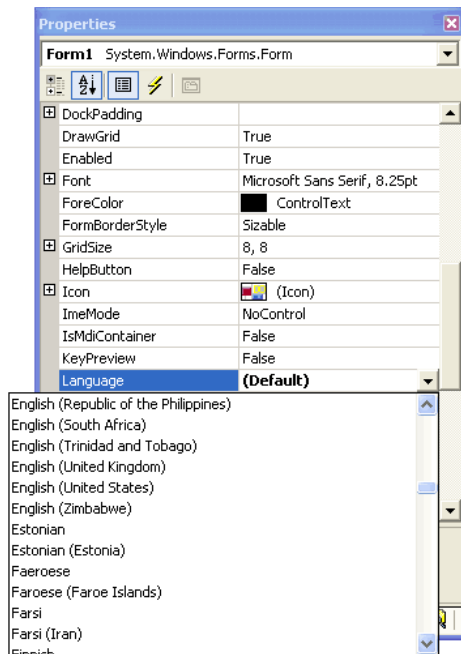
In der ersten Instanz (Abb. SC1) sind die Text-Strings hart in den C#-Quellcode codiert. Dieses Verfahren gestaltet den Lokalisierungsprozess kompliziert und macht ihn unnötig teuer.

In der zweiten Instanz wurde der C#-Quellcode verändert und benutzt jetzt C#-Methoden (aus der Klasse *ResourceManager*), um Strings aus einer externen Ressourcen Datei zu extrahieren.

3. Erstellung von mehreren Ressourcen Dateien

Pro lokalisierbare WinForm wird EINE neue .RESX-Datei erstellt: Wenn Sie beispielsweise festlegen, dass eine WinForm in die englische und deutsche Sprache übersetzt werden soll, so legt die Entwicklungsumgebung MS Visual Studio .NET zwei zusätzliche .RESX-Dateien für die gewünschten Zielsprachen an.

Um dieses Verhalten nachvollziehen zu können, klicken Sie bitte auf eine WinForm und ändern Ihre Lokalisierungs-Eigenschaft (engl. **Localizable**) in „JA“ (engl. **True**). Danach wählen Sie bitte Ihre Zielsprache für die WinForm.



Auswahl der Zielsprache

.RESX Sprachdateien

Pro Sprache, die vom Entwickler gewählt wird, wird eine neue RESX-Datei von der Microsoft IDE erstellt.

Die Namen der einzelnen RESX-Dateien basieren auf einem festen Schema, welches aus dem Namen der ursprünglichen RESX-Datei abgeleitet wird und ein Sprachen-Postfix am Ende enthalten:

Wenn die ursprüngliche RESX-Basisdatei beispielsweise „Form1.resx“ hieß, so trägt die deutsche Variante von dieser Form den Namen „Form1.de.resx“ und die regionale deutsch-schweizerische Variante den Namen „Form1.de-CH.resx“.

Während die Entwicklungsumgebung Microsoft Visual Studio .NET alle Sprach-Varianten der RESX-Datei für den Entwickler erstellt, muss darauf hingewiesen werden, dass dieser Prozess für JEDE *WinForm* in einer Anwendung wiederholt werden muss.

Bei sehr großen Applikationen sollte berücksichtigt werden, dass dieser Aufwand sehr zeitaufwändig ist und entsprechend in die Entwicklung mit eingeplant werden.

Stellen Sie sich den Aufwand für einen Entwickler vor, der mehrer hundert RESX-Dateien in 10 Sprach-Varianten verwalten soll. Schnell müssen hier mehrere tausend RESX-Dateivarianten fehlerfrei verwaltet werden.

Diese .RESX-Sprachdateien sind platz sparend und übertragungs-technisch optimiert, das heißt, dass diese Dateien nur die Änderungen bzw. das Delta zu Ihrer RESX-Basisdatei beinhalten. Diese Methode erweckt den Anschein, dass dies eine elegante Lösung ist um Speicherplatz einzusparen, doch die Realität sieht anders aus:

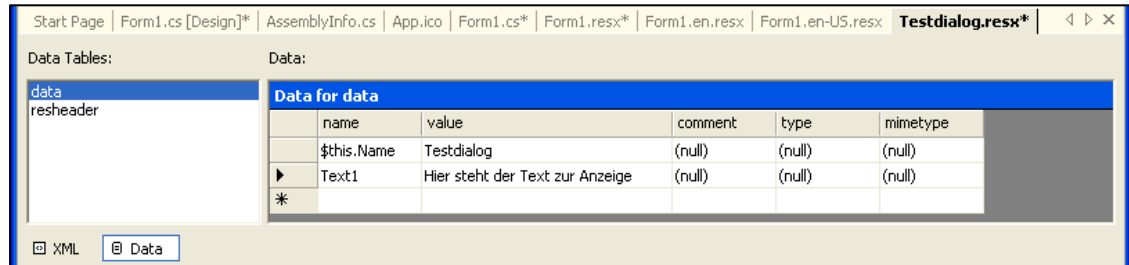
Für einen Lokalisierungsdienstleister oder einen Übersetzer, stellt diese Speichertechnik eine erhebliche Herausforderung dar, da anfänglich jede RESX-Sprachdatei ohne Inhalt erstellt wird.

Das bedeutet, dass diese RESX-Sprachdateien nicht lokalisiert werden können, ohne zuvor Referenzen zur Basis- oder unveränderten Ressourcen-Datei in der Microsoft Visual Studio Entwicklungsumgebung erstellt zu haben.

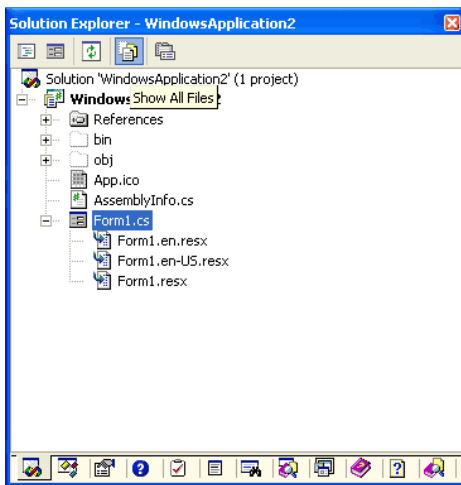
Darüber hinaus können nur Ressourcen in Assembly-Ressourcen-Dateien lokalisiert werden, die mittels der Klasse *ResourceManager* ermittelt werden! Deshalb sollten...

- alle grafischen Ressourcen (z.B. Icons, Bitmaps) in eine Ressourcendatei, die einem Windows-Formular (*WinForm*) zugeordnet ist, mit aufgenommen werden. Dazu kann auch ein extra dafür hinzugefügtes Windows-Formular benutzt werden. In diese *WinForm* werden dann die Grafiken in *ImageList*-Objekten eingefügt.

- Zusätzliche Textressourcen werden in einer eigenen hinzugefügten Assembly-Ressource-Datei aufgenommen. (Siehe Abb. unten).



Um die RESX-Basisdatei und alle sprachabhängigen Varianten in der Entwicklungsumgebung (IDE) anzuzeigen, klicken Sie bitte auf das **Show All Files** Icon in der Toolbar des Solution Explorers. (Siehe Abb. links).



Anzeige aller *.RESX Sprachvarianten

Sobald die .NET Anwendung kompiliert wird, erstellt die Entwicklungsumgebung eine Anzahl von Sprach-DLLs, die Satellite Assemblies³ genannt werden.

Diese Satellite Assemblies befähigen den Entwickler zur Laufzeit die Sprache der Benutzeroberfläche umzuschalten, indem die *UICulture data member* innerhalb der *CultureInfo* Klasse eingestellt wird.

³ Ein Satellite Assembly ist ein Assembly, das nur eingebettete Ressourcen enthält. Ein Satellite Assembly ist immer einem Basis-Assembly zugeordnet. Es enthält keine weiteren Typdefinitionen oder IL-Code.

4. Wahl der Ziel-Sprache der Bedienoberfläche (UI)

Ein Entwickler kann die Sprache seiner Anwendung wählen, indem er die Zielsprache des laufenden Threads im Konstruktor der Anwendung spezifiziert. Code-Beispiel:

```
Thread.CurrentThread.CurrentUICulture=new CultureInfo("de-DE");
```

Diese Anweisung setzt die Sprache der Applikation auf den Wert „Deutsch“ und lädt die Strings der Bedienoberfläche aus der Datei, die sich innerhalb des „de“ Verzeichnisses der .NET Anwendung befindet.

Um die Sprache ins Englische zu wechseln ändert der Entwickler die oben aufgeführte Anweisung wie folgt ab:

```
Thread.CurrentThread.CurrentUICulture=new CultureInfo("en-US");
```

Sofort erscheint die .NET Anwendung in englischer Sprache.

Wenn keine Sprache innerhalb der .NET Anwendung ausgewählt oder definiert wurde, wird automatisch die Standard-Bedienoberfläche angezeigt.

Diese Standard-Bedienoberfläche wurde in der RESX-Basisdatei (hier „Form1.resx“) erstellt. Diese Ressourcen werden automatisch an die Haupt-Applikationsdatei während des Compiliervorgangs gebunden.

Allgemeine Vorbereitung zur Lokalisierung einer .NET Anwendung (Zusammenfassung)

1. Die Eigenschaft **Localizable** aller Windows-Formulare (*WinForms*) müssen auf **true** gesetzt werden. Damit werden automatisch alle lokalisierbaren Eigenschaften des Formulars (z.B. Icons, Formulartitel, Texte der Steuerelemente, ...) in der Formular-Ressourcendatei (*.resx) abgelegt und über ein vom Designer hinzugefügtes Objekt dem *RessourcenManager* von dort zur Laufzeit ausgelesen.
2. Auf Ebene der Assemblies sollte eine geeignete Einstellung für die beiden Attribute *NeutralResourcesLanguage* bzw. *SatelliteContractVersion* definiert werden. Diese Attribute sind für den Fallback Mechanismus in der Sprachverwaltung der .NET Anwendung verantwortlich. Der Wert für das Attribut *NeutralResourcesLanguage* sollte vorzugsweise auf die (neutrale) Kultur mit der größten Verbreitung gesetzt werden.
Den Wert des zweiten Attributes setzt man auf den Wert der Basis-Assembly für die gewünschte Zielkultur bei der Auslieferung. Dieser Wert ist das Standard-Sprachsystem, welches beim Start der Applikation verwendet wird.
3. Für alle Phrasen, die nicht in den *WinForm*-Ressourcen-Dateien enthalten sind, fügt man dem .NET Projekt ein oder mehrere Assembly-Ressourcendateien hinzu und nimmt die Phrasen dort auf. Im Code verwendet man immer das *ResourceManager* Objekt, um auf diese Texte zuzugreifen.

Zusammenfassung des Lokalisierungsprozesses in der Entwicklungs-umgebung

Ohne Zweifel stellt Microsoft .NET eine effektive Art und Weise bereit, um mehrsprachige Anwendungen in der Design-Phase zu erstellen. Von Microsoft wird eine Umgebung bereitgestellt, die einfache Werkzeuge für die Lokalisierung von .NET Anwendungen dem Entwickler zur Verfügung stellt.

Jedoch wird dieser Lokalisierungsprozess innerhalb der IDE schnell unhandlich, wenn viele Formulare (*WinForms*) mit vielen Steuerelementen in verschiedenen Landessprachen benötigt werden.

Es ist geradezu unmöglich jede größere Anwendung auf diese Weise zu lokalisieren. Die Belastung der Entwickler beim Verwalten von RESX-Dateien und mehrfachen Assemblies ist dabei gewaltig und wird die Entwickler von Ihren eigentlichen Aufgaben, die Anwendung zu erstellen, ernsthaft abhalten.

Als Einschränkungen bei dieser Vorgehensweise innerhalb der IDE können folgende Punkte genannt werden:

- Komplexer .NET Lokalisierungsprozess

- Seitdem jede *WinForm* eine individuelle RESX-Datei und jede Zielsprache eine Variante von dieser Datei benötigt, wird das Verwalten von großen Anwendungen mit einer Vielzahl an *WinForms* und Sprachen nahezu unmöglich. In diesem Zusammenhang muss ebenfalls der Entwicklungs-Overhead bei der Pflege von mehreren Sprachvarianten berücksichtigt werden. Der Aufwand, welcher anfällt um diese Dateien synchron abzustimmen, ist nicht überschaubar - selbst in einem mittelgroßen Projekt.

- Inkrementelle Ressourcdateien können nicht direkt übersetzt werden

- Weil die RESX-Sprachdateien und die mehrsprachigen Satellite Assemblies, die von einer .NET Applikation benutzt werden, keine vollständigen Kopien der kompletten Ressource sind, können diese nicht direkt übersetzt werden. Dieser Mangel an vollständigem Inhalt macht diese Dateien geradezu redundant im Hinblick auf einen brauchbaren Lokalisierungsprozess.

- Keine Versionskontrolle bei der Lokalisierung

- Die Entwicklungsumgebung Microsoft VisualStudio .NET stellt keine Versionskontrolle zur Verfügung, um Änderungen des Haupt-Projekts für die Übersetzungen und deren Sprach-Layouts in den RESX-Sprachvarianten zu überwachen. Diese Kontrolle muss händisch und sehr konsequent durchgeführt werden, da diese zu einem hohen Fehlrisiko und wiederholten Qualitätssicherungsproblemen neigt.



visuallocalize

- Manuelle
Bearbeitung bei
Änderung der
Ausgangsdateien

Zusätzlich kann gesagt werden, dass alle Änderungen in der Basis der Anwendungs-Ressource nicht automatisch in den bereits übersetzten Varianten-Dateien übernommen werden.

Diese Aufgabe muss manuell vom Entwickler durchgeführt werden und ist in den meisten Fällen nahezu unmöglich.

Dieses Fehlen einer Versionskontrolle gestaltet das parallele Lokalisieren einer Anwendung während der Entwicklungsphase unmöglich. Selbst bei einer sehr kleinen Applikation.

Zusammenfassend kann gesagt werden, dass für die Übersetzung von .NET Anwendungen externe Übersetzungs-Werkzeuge, so genannte Software Lokalisierungstools, deutlich besser geeignet sind.

Im Nachfolgenden Abschnitt werden die Vorteile eines solchen Software-Lokalisierungstool genauer erläutert.

Lokalisierung von .NET Anwendungen mit Visual Localize .NET

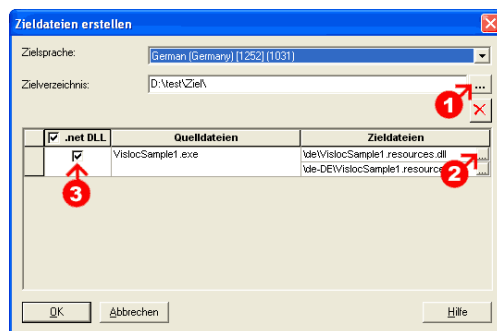
Lokalisierungs-Komplettlösung für .NET

+ direkte Bearbeitung der .NET Anwendung

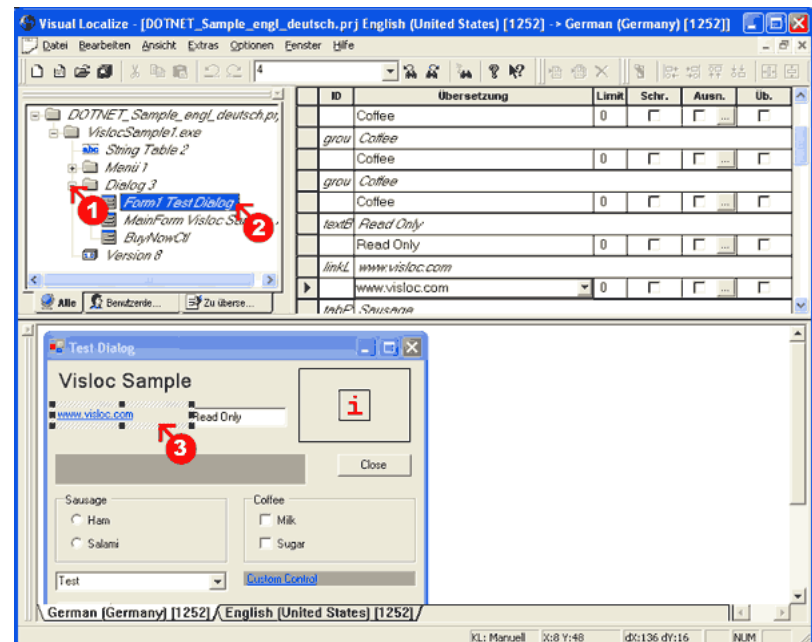
Mit der Entwicklung der Visual Localize® (.NET) Edition wurde ein Durchbruch in der Lokalisierung von .NET Anwendungen erreicht. Visual Localize's einfach zu bedienenden WYSIWYG-Benutzeroberfläche hilft dem Anwender bei der Erstellung von lokalisierten .NET Anwendungen und stellt durch die Praxis orientierte Vorgehensweise eine deutliche Vereinfachung bei der Übersetzung und Anpassung von Microsoft .NET Anwendungen dar. Dabei konnten alle Nachteile, die im Lokalisierungsprozess mittels der Entwicklungsumgebung (IDE) entstehen, gelöst werden:

Da jede Ressource innerhalb einer .NET Anwendung eine einzelne RESX-Datei benötigt, kann die Anzahl der zu verwaltenden Dateien bei der Lokalisierung beschwerlich groß für den Lokalisierungs-dienstleister und Software-Entwickler werden. Sobald Updates der .NET Anwendung veröffentlicht werden eskaliert der Verwaltungsaufwand für mehrere Versionen und Sprachen.

Um diesem Problem Herr zu werden kann Visual Localize .NET kompilierte .NET Anwendungen DIREKT bearbeiten!



Erstellung von .NET Satellite Assemblies



Lokalisierung einer .NET Anwendung in WYSIWYG Editor von Visual Localize

Das heißt, Visual Localize verwaltet in einer einzelnen Projektdatei alle fertig kompilierten .NET Assemblies (EXEs, DLLs). Nach der Lokalisierung in eine beliebige Landessprache können auf Knopfdruck die fertig angepassten Zieldateien (EXEs, DLLs) bzw. Satellite Assemblies (.resource.DLLs) erstellt werden – einfach, schnell und sicher.



+ Vereinfachung der Lokalisierungs-Workflows

+ Reduktion der Projektkosten und -dauer

+ Bessere Qualität durch sichtbaren kontextueller Zusammenhang

+ Versionskontrolle bei Software Updates

Entwicklungsumgebung contra Software- Lokalisierungstool

Das Übersetzen, Testen und Anpassen von .NET Binärdateien entspricht dabei exakt dem Bearbeiten von .RESX-Dateien, außer dass die Anzahl der zu bearbeitenden Dateien erheblich geringer ist.

Diese Tatsache reduziert die Komplexität in der Verwaltung des Lokalisierungsprozesses und reduziert dadurch die Projektdauer sowie damit verbundene Lokalisierungskosten.

Weiter kann Visual Localize aus einem .NET Assembly (EXE/DLL), welches alle kulturelevanten Informationen der Anwendung erhält, selbstständig komplette Satellite Assemblies erstellen. Somit ist sichergestellt, dass der Übersetzer den Kontext für seine Übersetzung korrekt und vollständig angezeigt bekommt. Ein solches Vorgehen entspricht dem tatsächlichen Übersetzungsprozess, da der Softwarehersteller zum Zeitpunkt der Entwicklung meist noch nicht weiß, für welche kulturellen Zielmärkte das Produkt später lokalisiert werden soll.

Nach einem Update der zu lokalisierenden .NET Software erkennt Visual Localize automatisch alle Änderungen und Neuerungen. Dadurch muss nur die Differenz zur Vorgängerversion übersetzt und an die Zielsprache angepasst werden.

Durch diese integrierte Versionskontrolle können mehrere Programmversion und –sprachen komfortabel verwaltet werden und enorme Übersetzungskosten eingespart werden.

Einer schnellen Markteinführung bzw. Update Ihrer Software steht also nichts mehr im Wege.

Entwicklungsumgebung	Visual Localize .NET
- Unterstützt nur das Bearbeiten von MS Resource Dateien (RC, RESX).	+ Visual Localize unterstützt die Lokalisierung von Binärdateien (EXE, DLL, OCX), MS Datenbank Dateien & XML Daten.
- Die Übersetzer müssen MS Visual Studio erwerben, um den Resource Editor zu erhalten. Eine andere Möglichkeit ist die fehlerbehaftete manuelle Bearbeitung der Ressourcen Skripte.	+ Visual Localize arbeitet auf Basis der ausführbaren Programmdateien. Dadurch wird das Fehlerisiko deutlich minimiert. Weiter werden keine zusätzlichen Lizenzen benötigt.
- Es gibt keine Möglichkeit zu kontrollieren, was übersetzt werden soll und was nicht. Im schlimmsten Fall ist die gesamte Software fehlerhaft.	+ Visual Localize zeigt dem Übersetzer nur die sprachabhängigen Phrasen in seiner Anwendung an. Der Quellcode kann nicht verändert werden.
- Die Übersetzer sind auf ein bestimmtes Format und Vorgehensweise festgelegt. Es besteht keine Möglichkeit andere computerunterstützte Übersetzungstool (CAT) wie z.B. TRADOS™ oder Wörterbücher zu verwenden.	+ Visual Localize speichert alle Übersetzungen in Wörterbüchern, womit eine flexible Wiederverwendung gewährleistet werden kann. Durch verschiedene Im- und Exportfunktionen können die Übersetzungen in anderen Tools weiterbearbeitet werden.
- Es gibt keine Möglichkeit frühere Übersetzungen wieder zu verwenden.	+ Visual Localize speichert alle Übersetzungen in 1 Projektdatenbankdatei, so dass alle Übersetzungen einfach wieder verwendet werden können.
- Keine Versionskontrolle; im Falle einer Änderung / Neuerung bei einem Software-Update müssen alle betroffenen Strings manuell gefunden und bearbeitet werden.	+ Visual Localize erkennt automatisch alle Änderungen / Neuerungen nach einem Software-Update, sodass nur diese Phrasen neu übersetzt und angepasst werden müssen.



Visual Localize® .NET

Visual Localize® ist eine Anwendung die den kompletten Software-Lokalisierungsprozess von Microsoft™ Windows Anwendungen (inkl. .NET Anwendungen) MS Datenbanken und XML Dateien unterstützt. Dabei werden anfallende Lokalisierungs- und Übersetzungskosten sowie die Komplexität des Prozesses drastisch reduziert.

Weitere Informationen zu Visual Localize® .NET und Dienstleistung in diesem Bereich finden Sie auf der Produkthomepage:

<http://www.visloc.com/>

AIT AG

AIT AG gehört zu den führenden, deutschen Pionieren auf dem Gebiet „Anwendungsentwicklung für MS Windows“ im industriellen Bereich.

Seit vielen Jahren entwickelt die AIT AG Konzepte und Werkzeuge für die Software-Lokalisierung und ist einer der führenden Lösungsanbieter für verteilte Systeme auf Basis von MS .NET Technologie.

Weitere Informationen über AIT AG und Ihre Dienstleistungen finden Sie auf der Firmenwebseite unter:

<http://www.aitag.com/>

Weitere Informationen

Weitere Informationen zu den Produkten und Dienstleistungen von AIT finden Sie unter:

<http://www.aitag.com> (AIT AG Firmen Webseite)

<http://www.visloc.com> (Visual Localize® .NET – Produkthomepage)

Weitere Informationen zu den Produkten und Dienstleistungen von Microsoft™ finden Sie unter:

<http://www.microsoft.com>

Weitere Informationen zu den Produkten und Dienstleistungen von Trados™ finden Sie unter:

<http://www.trados.com>

© 2003 AIT AG. Alle Rechte vorbehalten.