

Abschlussprüfung Sommer 2018

Fachinformatiker für Anwendungsentwicklung

Dokumentation zu der betrieblichen Projektarbeit

WPF-Translate

Desktopanwendung zu dem Übersetzen von WPF-Ressourcen

Abgabetermin 25.04.2018

Prüfungsbewerber:

Jan Wiesemann
Wilhemstraße 2
34513 Waldeck



Ausbildungsbetrieb:

Landau Software GmbH
Masurenallee 9a
34537 Bad Wildungen

Inhalt

Anhangsverzeichnis-----	III
Tabellenverzeichnis-----	IV
Abkürzungsverzeichnis-----	V
1. Einleitung-----	1
1.1. Projektbeschreibung-----	1
1.2. Projektziel-----	1
1.3. Projektumfeld-----	2
1.4. Projektbegründung-----	2
1.5. Projektschnittstellen-----	2
1.6. Projektabgrenzung-----	2
2. Startphase-----	3
2.1. Ist-Analyse-----	3
2.2. Darstellung und Abstimmung des Auftrags / der Ziele-----	3
3. Projektplanung-----	4
3.1. Projektphasen-----	4
3.2. Ressourcenplanung-----	4
3.3. Vorkalkulierte Projektkosten-----	5
3.4. Entwurfsphase-----	5
3.4.1. Zielplattform-----	5
3.4.2. Pflichtenheft-----	6
3.4.3. Bibliothekssichtung-----	6
4. Implementierungsphase-----	7
4.1. Erstellen des Projektes-----	7
4.2. Backend Entwicklung-----	8
4.2.1. Lesen und Schreiben der Ressourcen-Dateien-----	8
4.2.2. Gruppieren der Daten nach Sprache und Keys-----	8
4.2.3. Anbindung an die Oberfläche-----	10
4.2.4. Anbindung an Google Translate-----	10
4.3. Frontend Entwicklung-----	10
4.4. Schnittstelle für Externe Anwendungen-----	11
4.5. Erstellung eines Handbuchs-----	11

5. Abschlussphase-----	12
5.1. Qualitätssicherung und Tests -----	12
5.2. Soll-/Ist-Vergleich -----	12
5.3. Nachkalkulation -----	13
5.4. Projektdokumentation -----	13
5.5. Präsentation und Abnahme der Ergebnisse -----	14
6. Fazit -----	15
Literaturverzeichnis -----	16
Internetquellen -----	16
Internetquellen - Bibliotheken -----	16

Anhangsverzeichnis

EPK Ändern eines Wertes -----	i
Auszug aus dem Lastenheft -----	ii
Projektplanung mit Zeitplanung in Stunden-----	iii
Verwendete Ressourcen-----	iv
Vergleich Programmiersprachen -----	v
Auszug aus dem Pflichtenheft -----	vi
Verwendete Bibliotheken-----	viii
Aufbau XAML-Ressourcenwörterbuch -----	ix
NotifyBase Klasse -----	x
RelayICommand Klasse-----	xi
ResourceHelper in LSDOTNETINFOMAN -----	xiv
Handbuch WPF-Translate -----	xx
Entwicklungsdokumentation -----	xxvii
Code Map der IO Funktionen -----	xxviii
UML Klassenmodel der Model und ViewModel Klassen-----	xxix
Screenshots der Anwendung -----	xxx

Tabellenverzeichnis

Tabelle: Grobe Zeitplanung -----	4
Tabelle: Stundensätze -----	5
Tabelle: Vorkalkulation -----	5
Tabelle: Wichtige XAML-Kontenpunkte -----	8
Tabelle: Soll-/Ist-Vergleich -----	12
Tabelle: Nachkalkulation -----	13

Abkürzungsverzeichnis

API	Application Programming Interface
CSV	Comma - separated values
EPK	Ereignisgesteuerte Prozesskette
JSON	JavaScript Object Notation
LS	Landau-Software GmbH
MVVM	Model View ViewModel
TFS	Team Foundation Server
UI	User Interface
UML	Unified Modeling Language
WPF	Windows Presentation Foundation
XAML	Extensible Application Markup Language
XML	Extensible Markup Language

1. Einleitung

Die folgende Projektdokumentation schildert den Ablauf des IHK-Abschlussprojektes, welches ich im Rahmen meines Abschlusses für die Fachrichtung Fachinformatiker Anwendungsentwicklung durchgeführt habe. Mein Ausbildungsbetrieb ist die Landau-Software GmbH, welche sich in Reinhardshausen befindet. Zu der Zeit beschäftigt LS fünf Mitarbeiter im Unternehmen. Zu den Produkten der Firma zählen LS-INFOMAN, ein Warenwirtschaftssystem für die Lederbranche und LS-INFOCASH, die zugehörige Kassensoftware.

1.1. Projektbeschreibung

INFOMAN und INFOCASH sind seit Januar 2017, aufgrund der gesetzlichen Lage, für Kassensysteme unserer Kunden verpflichtend geworden. Durch die Anpassungen an diese Gesetzeslage ist das Oberflächendesign der Software immer mehr in den Hintergrund gerückt. Im Rahmen einer Neuentwicklung der Oberfläche fallen XAML-Ressourcen an. Diese Ressourcen dienen zu der Beschreibung von Oberflächen, Fenstern und Steuerelementen, können aber auch einfachere Daten, wie z.B. Zeichenketten, definieren. Durch das Zusammenführen von mehreren XAML-String-Ressourcen, ist es möglich eine mehrsprachige Oberfläche zu schaffen. Durch den Einsatz unserer Software in verschiedensten Europäischen Ländern ist diese Mehrsprachigkeit dringendste erforderlich.

In meinem Projekt soll eine Anwendung geschaffen werden, die das Verwalten und Übersetzen dieser String-Ressourcen erleichtert.

1.2. Projektziel

Ziel des Projektes ist es, das Arbeiten mit XAML-String-Ressourcen zu erleichtern, um das anpassen dieser Dateien in verschiedenste Landersprachen zu ermöglichen. Um das Nachladen einer solchen Ressourcen-Datei zu erleichtern, müssen sich die Zeichenketten, für die verschiedenen Sprachen, in verschiedenen Dateien befinden. Da die einzelnen Zeichenketten in diesen nicht geordnet dargestellt werden oder unter Umständen erst gar nicht vorhanden sind, ist das Arbeiten mit genannten Dateien, in mehreren Sprachen, sehr zeitaufwändig. Durch das geschaffene Programm soll der Zeitaufwand minimiert und das Bearbeiten dieser Dateien vereinfacht werden.

1.3. Projektumfeld

Auftraggeber des Projektes sind alle Entwickler der **LS**, die sich mit der Entwicklung in **WPF** beschäftigen.

1.4. Projektbegründung

Die Hauptschwachstelle an der Verwendung dieser Methodik, zu der Implementierung einer mehrsprachigen Oberfläche, ist der hohe Zeitaufwand bei dem Anpassen, Ändern oder Löschen von Zeichenketten. Ein Beispiel hierfür ist das Anpassen einer Fehlermeldung nach einer Programmänderung. Der Entwickler muss hierfür alle Sprachdateien öffnen, in jeder nicht sortierten Datei nach dem passenden Eintrag suchen, diesen anpassen, anschließend speichern und danach wieder schließen.

Ein weiteres Problem ist die Übersichtlichkeit dieser Dateien, welche mit wachsender Größe immer weiter verloren geht. Auf Grund dieser Problematik und durch die steigende Fehleranfälligkeit, haben wir uns zu der Entwicklung dieser Desktopanwendung entschieden.

1.5. Projektschnittstellen

Damit die erzeugten Dateien später einfach und schnell in eine Anwendung integriert werden können, soll zusätzlich eine weitere Klasse in der LSDOTNETCORE-Bibliothek, eine Bibliothek, welche häufig benutzte Funktionen für **WPF** und andere C# Sprachfeatures beinhaltet, geschaffen werden.

1.6. Projektabgrenzung

Da der Projektumfang beschränkt ist soll die Implementierung der von dieser Anwendung geschaffenen Features in ein anderes Projekt nicht Bestandteil dieses Abschlussprojektes sein.

2. Startphase

2.1. Ist-Analyse

Wie bereits im Abschnitt [Projektbeschreibung \(1.1.\)](#) erwähnt, sind die Entwickler bei LS für das Übersetzen und Hinzufügen von Zeichenketten, in eine andere Sprache, verantwortlich. Die momentane Lösung für dieses Problem bringt einen hohen Zeitaufwand mit sich, der das Arbeiten an einer WPF-Oberfläche sehr in die Länge zieht.

Der momentane Verarbeitungsprozess wurde zu Beginn des Projektes in Form eines EPK dargestellt, welches im Anhang [EPK Ändern eines Wertes \(S. i\)](#) zu finden ist, um das hohe Maß an Zeitaufwand noch einmal zu verdeutlichen.

2.2. Darstellung und Abstimmung des Auftrags / der Ziele

Am Ende der Analysephase wurde, zusammen mit den Auftraggebern, ein Lastenheft erstellt. Dieses stellt die Anforderungen des Auftraggebers an die Anwendung dar. Das Lastenheft befindet sich im Anhang [Auszug aus dem Lastenheft \(S. ii\)](#).

3. Projektplanung

3.1. Projektphasen

Für die Umsetzung des beschriebenen Projektes, standen mir 70 Stunden zu der Verfügung. Diese sind vor Beantragung in verschiedene Phasen, welche während der Software Entwicklung durchlaufen werden, unterteilt worden. Einen groben Zeitplan, sowie die Hauptphasen, lassen sich der Tabelle [Grobe Zeitplanung](#) entnehmen. Diese können in weitere Unterpunkte zerlegt werden. Eine detaillierte Übersicht der genannten Phasen befindet sich im Anhang [Projektplanung mit Zeitplanung](#) (S. iii).

Tabelle: Grobe Zeitplanung

Projektphase	Geplante Zeit
Startphase	3h
Projektplanung	13h
Implementierungsphase	34h
Abschlussphase	20h
Gesamt	70h

3.2. Ressourcenplanung

In der Übersicht, die sich im Anhang [Verwendete Ressourcen](#) (S. iv) befindet, sind jene Ressourcen aufgelistet, welche für dieses Projekt eingesetzt wurden. Diese beziehen sich sowohl auf Hard- als auch auf Softwareressourcen. Bei der Auswahl der Software wurde darauf geachtet, dass diese kostenfrei (z.B. Open source) zu der Verfügung steht oder **LS** bereits über die passenden Lizenzen verfügt. Die Projektkosten sollen dadurch möglichst gering gehalten werden.

3.3. Vorkalkulierte Projektkosten

Die Kosten, welche während der Entwicklung des Projektes anfallen, lassen sich der Tabelle [Vorkalkulation](#) entnehmen. Die angenommenen Stundensätze können der Tabelle [Stundensätze](#) entnommen werden.

Tabelle: Stundensätze

Kosten	Kosten/Stunde
Auszubildender	10 €
Mitarbeiter	70 €
Ressourcenkosten (z.B. Strom)	15 €

Tabelle: Vorkalkulation

Vorgang	Mitarbeiter	Zeit	Personal	Ressourcen	Gesamt
Entwicklungskosten	1x Auszubildender	70h	700,00 €	1.050,00 €	1.750,00 €
Code Review	1x Mitarbeiter	2h 30m	175,00 €	37,50 €	212,50 €
Abnahme	1x Mitarbeiter	0h 30m	35,00 €	7,50 €	42,50 €
Projektkosten gesamt					2.005,00 €

3.4. Entwurfsphase

3.4.1. Zielplattform

Mein Abschlussprojekt soll, wie bereits im Abschnitt [Projektziele \(1.2.\)](#) erläutert, als eigenständige Desktopanwendung umgesetzt werden.

Um die Programmiersprache festzulegen, wurde eine Nutzwertanalyse zu der Auswahl der Programmiersprache durchgeführt. Eine Nutzwertanalyse dient z.B. dem übersichtlichen Vergleichen von zwei Programmiersprachen. Dabei werden einzelne Punkte festgelegt, welche in die Bewertung einfließen sollen. Anschließend wird zu jedem dieser Punkte eine Gewichtung festgelegt. Sowohl dafür als auch für die jeweilige Bewertung werden Werte zwischen eins und fünf festgelegt.

Da im Unternehmen die Programmiersprachen C# mit [WPF](#) und C/C++ etabliert sind und diese für das Projekt geeignet sind, wurde die Auswahl auf diese Sprachen beschränkt. Der detaillierte Vergleich, ist im Anhang [Vergleich Programmiersprachen \(S. v\)](#) zu finden.

3.4.2. Pflichtenheft

Das Pflichtenheft legt die in der Entwicklung verfolgten Ziele des Produktes fest. Es wird verwendet, um den Entscheidungsraum für die Realisierung abzustecken. Diese Abgrenzung erfolgt durch das Festlegen von Wunsch-, Muss- und Absteckungskriterien. Ein Auszug aus dem für dieses Projekt erstellten Pflichtenheft befindet sich im Anhang [Auszug aus dem Pflichtenheft \(S. vi\)](#).

3.4.3. Bibliothekssichtung

Damit die Entwicklung der Anwendung erleichtert wird, wurden einige externe Bibliotheken für das Projekt verwendet. Um welche Bibliotheken es sich handeln und welche Funktionen diese implementieren, ist im Anhang [Verwendete Bibliotheken \(S. viii\)](#) abzulesen. Es wurde darauf geachtet, dass alle Bibliotheken kostenfrei zu der Verfügung stehen, der Quellcode offen einsehbar ist und die Bibliotheken über den Paketmanager NuGet¹ geladen werden können. Die Sichtung der Bibliotheken verlief über die Recherche in Foren, die Suche auf Google, NuGet.org und unserem lokalen NuGet-Server.

¹ vgl. <http://nuget.org/>

4. Implementierungsphase

4.1. Erstellen des Projektes

Bevor der Auszubildende mit der eigentlichen Entwicklung des Programms beginnen konnte, musste ich zunächst ein Projekt anlegen. Hierfür wurde ein neues Teamprojekt auf unserem Team Foundation Server² erstellt. Dieses dient, in diesem Fall, ausschließlich zu der Sicherung des Projektes. Das Verwenden einer Quellcodeverwaltung, wie TFS, ermöglicht das einfache Zusammenarbeiten an großen Projekten. Zusätzlich ist es möglich, fehlerhafte Änderungen wieder rückgängig zu machen. Bei TFS gibt es mehrere Möglichkeiten der Versionskontrolle. Dieser stellt hierfür zwei Möglichkeiten zu der Verfügung. Bei dem Projekt wurde Git³ verwendet, da dieses das Verwalten mit externen Anwendungen, wie bspw. SourceTree⁴, ermöglicht.

Nach der Erstellung des Team Projektes wurde dieses selbst auf einen lokalen Speicherort geklont und anschließend ein neues Visual Studio Projekt erstellt.

Nach der Erstellung des Projektes wurden die benötigten, externen Bibliotheken über NuGet in das Projekt geladen. Die Bibliotheken wurden bereits im Abschnitt [Bibliothekssichtung \(3.4.3.\)](#) aufgelistet und beschrieben.

² Server zu dem Ausführen von Git <https://www.visualstudio.com/tfs/>

³ Software für Versionskontrolle von Quellcode <https://git-scm.com>

⁴ Desktopanwendung für Git <https://www.sourcetreeapp.com>

4.2. Backend Entwicklung

4.2.1. Lesen und Schreiben der Ressourcen-Dateien

Um eine Datei lesen und schreiben zu können, muss man sich zunächst mit deren Aufbau beschäftigen. Im Anhang [Aufbau XAML-Ressourcenwörterbuch \(S. ix\)](#) ist der Aufbau einer solchen Datei beschrieben. Wie an diesem Beispiel zu sehen ist, handelt es sich bei den Ressourcen um [XML-Dateien](#). Aus diesem Grund ist es einfach, mit Hilfe der XmlDocument Klasse, diese Dateien mittels C# zu lesen. Be dem Einlesen dieser Dateien gibt es vier wesentliche Unterknoten. Diese Unterknoten sind alle im genannten Beispiel zu finden. Eine Beschreibung der selben ist in der Tabelle [Wichtige XAML-Kontenpunkte](#) abzulesen.

Tabelle: Wichtige XAML-Kontenpunkte

XML	Beschreibung
<ResourceDictionary.MergedDictionaries/>	In diesem Knoten werden alle untergeordneten Wörterbücher aufgelistet.
<sys:String/>	String-Eintrag (sys: steht für den Namespace des Objektes)
x:Static Member="sys:String.Empty"/>	Ein Verweis auf eine statische Ressource. In diesem Fall auf einen leeren String.
Andere	Andere Einträge wie Farben uvm.

Während der Entwicklung der Schreibfunktion sind einige Probleme aufgetreten. Zunächst wurde versucht, die Daten erneut über ein XmlDocument Objekt zu schreiben, allerdings sind dabei mehrere Probleme mit Namespaces und Zeilenumbrüchen aufgetreten. Anschließend wurde das Schreiben über die Verwendung eines XmlWriters realisiert. Da dieser jedoch nicht in der Lage ist die vorliegenden, rohen Daten zu formatieren, muss das Dokument nach der Erstellung erneut eingelesen, formatiert und gespeichert werden.

4.2.2. Gruppieren der Daten nach Sprache und Keys

Um das Darstellen der Daten in einer Liste zu erleichtern, werden die nachfolgenden Klassen direkt für das Binden der Oberfläche entwickelt. Dies wirft die Frage auf, wofür man Bindings benötigt und was diese mit den Klassen zu tun haben. In [WPF](#) gibt es mehrere Möglichkeiten Daten auf der Oberfläche anzuzeigen. Eine davon ist das direkte Aufrufen der Steuerelemente und das Setzen von Eigenschaften. Diese Methode ist sehr schreibintensiv und fehleranfällig, da der Programmierer durch sie schnell eine falsche

Eigenschaft setzen, das falsche Element ansprechen oder diese vielleicht gänzlich vergessen kann.

Aus genannten Gründen wurde stattdessen das **MVVM**⁵ (Model View ViewModel) Muster implementiert. Dieses sorgt für eine strikte Trennung zwischen **UI-Design** und Daten. Eine Implementierung des Musters eignet sich eher für den Einsatz in größeren Anwendungen. Dieses wurde jedoch gezielt gewählt, um das Anzeigen der Sprachdaten in einer Liste zu erleichtern. Zusätzlich ist es durch diese Trennung möglich, dass Designer und Entwickler parallel an dem selben Dialog arbeiten können, wodurch ein höherer Grad an Effizienz ermöglicht wird. Da das ViewModel auch ohne Oberfläche existieren kann, ist das Testen der Klassen somit einfacher. Das **MVVM** Muster basiert auf drei Komponenten:

- View** Benutzeroberfläche (**XAML** + Codebehind)
- ViewModel** Eine Klasse, welche das Model kapselt und Eigenschaften zu dem Binden an die Oberfläche bereit stellt.
- Model** Das Datenmodel

Jede dieser ViewModel- und Model- Klassen erbt jeweils von der Klasse NotifyBase, welche Teil der Bibliothek LSDOTNETCORE ist. Diese Klasse implementiert das Interface INotifyPropertyChanged, welches wiederum ein Event implementiert. Dieses Event dient dazu, die Oberfläche über Änderungen an einer Eigenschaft zu benachrichtigen. Der Ausschnitt dieser Klasse kann in dem Anhang **NotifyBase Klasse (S. x)** gefunden werden. Da das Programm auf einer GridView basiert, deren Spalten nicht an eine Eigenschaft gebunden werden können, wurde zusätzlich ein Event hinzugefügt, um die Oberfläche über eine Änderung in den Spalten zu informieren. Wie im Anhang **UML Klassenmodel der Model und ViewModel Klassen (S. xxiv)** zu sehen, wurde zusätzlich eine weitere Klasse DialogCoordinatorNotifyBase implementiert, um das Darstellen von MahApps Dialogen⁶ zu erlauben.

Um das Sichern dieser Daten in dem neuen Format zu erlauben, wurde eine Methode geschaffen, welche das Umwandeln der **UI-Daten** zu Ressourcen Wörterbüchern

⁵ vgl. T. Claudius, *WPF – Das umfassende Handbuch*, Bonn, 2016, S.520

⁶ vgl. <http://mahapps.com/controls/dialogs.html>

übernimmt. Diese Methode wurde direkt in dem Command zu dem Speichern implementiert.

4.2.3. Anbindung an die Oberfläche

Um das Anbinden von Commands⁷, dem Äquivalent zu Klick-Events in [MVVM](#), zu erleichtern, wurde die Klasse RelayCommand verwendet, welche aus LSDOTNETCORE stammt. Diese implementiert das Interface ICommand, sowie die Methode CanExecute, zu dem Überprüfen, ob der genutzte Command ausgeführt werden kann. Zudem implementiert diese die Methode Execute, welche Code zu dem Ausführen beinhaltet. Die genannte Klasse erlaubt das einfache Erstellen von Commands durch Lambda-Ausdrücke⁸, auch bekannt als anonyme Methoden. Ein Abschnitt aus dieser Klasse kann im Anhang [RelayCommand Klasse \(S. xi\)](#) gefunden werden.

4.2.4. Anbindung an Google Translate

Google bietet eine .NET-Bibliothek an, welche bspw. eine Anbindung an Google Translate ermöglicht. Diese Schnittstelle ist jedoch sehr kompliziert zu implementieren, und aus diesem Grund wurde für das Projekt ein anderer Weg gewählt. Dafür wurden die Anfragen von translate.google.com analysiert und anschließend, mit Hilfe der WebClient-Klasse, simuliert. Die von der Schnittstelle gelieferten Daten sind [JSON](#) formatiert und wurden mit [JSON.NET](#)⁹ verarbeitet. Da es sich hier nicht um die offizielle Anbindung an Google Translate handelt, soll diese Schnittstelle in Zukunft geändert werden. Aus Kostengründen wurde die Anbindung an die offizielle Google-API nicht gewählt und dient hier ausschließlich der Demonstration.

4.3. Frontend Entwicklung

Wie im vorherigen Abschnitt [Gruppieren der Daten nach Sprache und Key \(4.2.2.\)](#) erwähnt wurde, wird das Hinzufügen der Spalten zu der Oberfläche über ein Event gelöst. Dieses erstellt die einzelnen Spalten für das Darstellen der Werte. Da zu dem Anzeigen von Fehlermeldungen ein asynchroner Task¹⁰ verwendet werden muss und dieser die

⁷ vgl. T. Claudius, [WPF — Das umfassende Handbuch](#), Bonn, 2016, S.488

⁸ vgl. A. Kühnel, [C# 6 mit Visual Studio 2015 — Das umfassende Handbuch](#), Bonn, 2016, S. 409

⁹ vgl. <https://www.newtonsoft.com/json>

¹⁰ Eine Art weniger leistungshungriger Threads um paralleles Arbeiten zu ermöglichen.

Oberfläche des Programms nicht direkt ändern kann, muss der Dispatcher¹¹ dazwischen geschaltet werden. Der Dispatcher ist eine Klasse, welche die Kommunikation über mehrere Threads zu einem UI-Thread verwaltet. Er wird über die WPF-Bibliothek implementiert.

4.4. Schnittstelle für Externe Anwendungen

Wie bereits im Abschnitt [Projektschnittstellen \(1.5.\)](#) beschrieben, soll in der LSDOTNETCORE Bibliothek eine Klasse geschaffen werden, um das Verwenden der Sprachressourcen zu erleichtern. Um das Implementieren dieser Klasse zu ermöglichen, wurde das LSDOTNETCORE Projekt, mittels Git, auf dem Entwicklungs - PC geklont und kann anschließend mit Visual Studio geöffnet werden. Ein Ausschnitt dieser Klasse befindet sich in dem Anhang [ResourceHelper in LSDOTNETCORE \(S. xiv\)](#). Um das Laden der Daten zu erlauben, wurden mehrere Methoden geschaffen. Eine dieser Methoden arbeitet über eine festgelegte Dateiendung, welche die genutzte Sprache identifiziert. Die zweite aufwändigere, aber flexiblere Möglichkeit, nimmt eine Methode, bzw. einen Lambda-Ausdruck¹², als Filter an. In dieser Klasse befinden sich noch weitere Funktionen, die unabhängig von der Sprache eingesetzt werden können und bspw. das einfache Formatieren von Zeichenketten mit Platzhaltern erlauben.

4.5. Erstellung eines Handbuchs

Das Handbuch wurde mit Word erstellt und anschließend als PDF in das Programm eingebunden. Es kann dort über die Menüfunktion „Hilfe“ aufgerufen werden. Dieses ist in dem Anhang [Handbuch WPF-Translate \(S. xx\)](#) zu finden.

¹¹ vgl. T. Claudius, [WPF – Das umfassende Handbuch](#), Bonn, 2016, S.82

¹² vgl. A. Kühnel, [C# 6 mit Visual Studio 2015 – Das umfassende Handbuch](#), Bonn, 2016, S. 409

5. Abschlussphase

5.1. Qualitätssicherung und Tests

Damit die Qualität und die Übersichtlichkeit des Codes gesichert ist, wurden nach Beendigung des Projektes nochmals alle Klassen überprüft und gegebenenfalls angepasst. Anschließend wurden alle, im Pflichtenheft festgelegten, Testfälle anhand des Black-Box-Test¹³ abgearbeitet. Bei der Kontrolle dieser Testfälle konnten keine weiteren Fehler festgestellt werden.

5.2. Soll-/Ist-Vergleich

In der Tabelle Soll-/Ist-Vergleich wurde ein Vergleich zwischen den geplanten und eingesetzten Zeiten durchgeführt. Durch eine strukturierte Planung sind keine größeren Abweichungen in den jeweiligen Phasen aufgetreten. Es entstanden leichte Differenzen welche jedoch durch den Ausgleich mit anderen Phasen kompensiert werden konnten.

Tabelle: Soll-/Ist-Vergleich

Projektphase	Soll	Ist	Differenz
Startphase	3h	3h 30m	0h 30m
Projektplanung	13h	12h 30m	-0h 30m
Implementierungsphase	34h	35h	1h 0m
Abschlussphase	20h	19h	-1h 0m
Gesamt	70 Stunden	70 Stunden	0 Stunden

¹³ Bei diesem Test wird das Programm mit Testdaten versorgt und überprüft, ob das Ergebnis dem Wunschergebnis entspricht. Es sind hierbei keine Hintergrundinformationen über den Code erforderlich.

5.3. Nachkalkulation

Die Nachkalkulation wurde anhand der angenommenen Kosten aus dem Abschnitt [Vorkalkulierte Projektkosten \(3.3.\)](#) vorgenommen. Da es während des Code Reviews zu einer minimalen Verkürzung der benötigten Zeit gekommen ist, konnten die Projektkosten gesenkt werden.

Tabelle: Nachkalkulation

Vorgang	Geplante Zeit	Geplante Kosten	Abweichung Zeit	Abweichung Kosten	Gesamt
Entwicklungskosten	70h 0m	1.750,00 €	0h 0m	0,00 €	1.750,00 €
Code Review	2h 30m	212,50 €	-0h 30m	-42,50 €	170,00 €
Abnahme	0h 30m	42,50 €	0h 0m	0,00 €	42,50 €
Neue Projektkosten					1.962,50 €
Differenz					-42,50 €

5.4. Projektdokumentation

Die Dokumentation des Projektes lässt sich in drei wesentliche Bestandteile unterteilen.

Benutzerdokumentation: Die Erstellung des Benutzerhandbuches wurde auf Grund der implementierten Hilfe bereits im Abschnitt [Erstellung eines Handbuchs \(4.5.\)](#) erläutert.

Projektdokumentation: Die Dokumentation umfasst einen Verlauf und eine Beschreibung aller Phasen des Projektes. Sie wurde im Rahmen der Ausbildung erstellt.

Entwicklungsdokumentation: Um diese Dokumentation zu generieren, wurden während der Entwicklung [XML-Dokumentationskommentare](#) verfasst. Diese Kommentare wurden anschließend, mit Hilfe der Bibliothek DocFx¹⁴, exportiert. Ein Abschnitt dieser Dokumentation, ist im Anhang [Entwicklungsdokumentation \(S. xxii\)](#) zu finden. Sie dient den Entwicklern anschließend als Nachschlagewerk. Zusätzlich wurden ein Klassendiagramm und eine Code Map¹⁵ über Visual Studio erstellt. Auch diese Diagramme sind in den Anhängen [Code Map der IO Funktionen \(S. xxiii\)](#) und [UML Klassenmodel der Model und ViewModel Klassen \(S. xxvi\)](#) zu finden.

¹⁴ vgl. <https://dotnet.github.io/docfx/>

¹⁵ Diagram zu der Darstellung von Codezusammenhängen
<https://msdn.microsoft.com/de-de/library/jj739835.aspx>

5.5. Präsentation und Abnahme der Ergebnisse

Nach der Fertigstellung der Anwendung wurde diese mit dem Fachpersonal von LS besprochen. Durch eine gute Setzung der Ziele und Funktionen im Lasten- und Pflichtenheft sind keine Mängel aufgetreten. Durch eine einfach gestaltete Oberfläche ist die Eingewöhnungsphase kurz und es kann schnell Vertrautheit mit der Anwendung erlangt werden. Zusätzlich wurde der Code von dem Fachpersonal von LS abgenommen, wodurch eine einheitliche und übersichtliche Programmierung sichergestellt wurde.

6. Fazit

Im Zuge des durchgeführten Projektes konnte ich einige wertvolle Erfahrung für die Planung und Durchführung eines selbigen sammeln. Dabei wurde mir besonders deutlich, wie wichtig es ist, eine gute Planung vor die Durchführung eines Projektes zu stellen, um einen reibungslosen Ablauf zu gewährleisten. Durch den Bezug auf externe Bibliotheken, wie beispielsweise MahApps.Metro¹⁶, konnte ich mich auf die wesentlichen Bestandteile der Entwicklung konzentrieren und habe einen tieferen Einblick in die Nutzung genannter Bibliotheken bekommen.

Obwohl alle im Lastenheft beschriebenen Anforderungen realisiert werden konnten, plane ich in Zukunft noch Änderungen an der Anwendung vorzunehmen. Ich möchte bspw. ermöglichen, dass das Programm zusätzlich CSV-Dateien einlesen und speichern kann. Des Weiteren muss die Anbindung an Google Translate in Zukunft überarbeitet werden, da wie in Abschnitt [Anbindung an Google Translate \(4.2.4.\)](#) beschrieben, eine Alternativlösung gefunden werden muss.

Um eine Übersicht über die Anwendung zu erhalten, wurden einige Screenshots in den Anhang [Screenshots der Anwendung \(S. xxv\)](#) hinzugefügt.

¹⁶ vgl. <http://mahapps.com>

Literaturverzeichnis

Claudius, Thoams: WPF — Das umfassende Handbuch

Windows Presentation Foundation — Das umfassende Handbuch; Thomas Claudius Huber, 4. Auflage, Rheinwerk Verlag 2016, ISBN 978-3-8362-3756-7

Kühnel, Andreas: C# 6 mit Visual Studio 2015 — Das umfassende Handbuch

C# mit Visual Studio 2015 — Das umfassende Handbuch; Andreas Kühnel, 7. Auflage, Rheinwerk Verlag 2016, ISBN 978-3-8362-3714-7

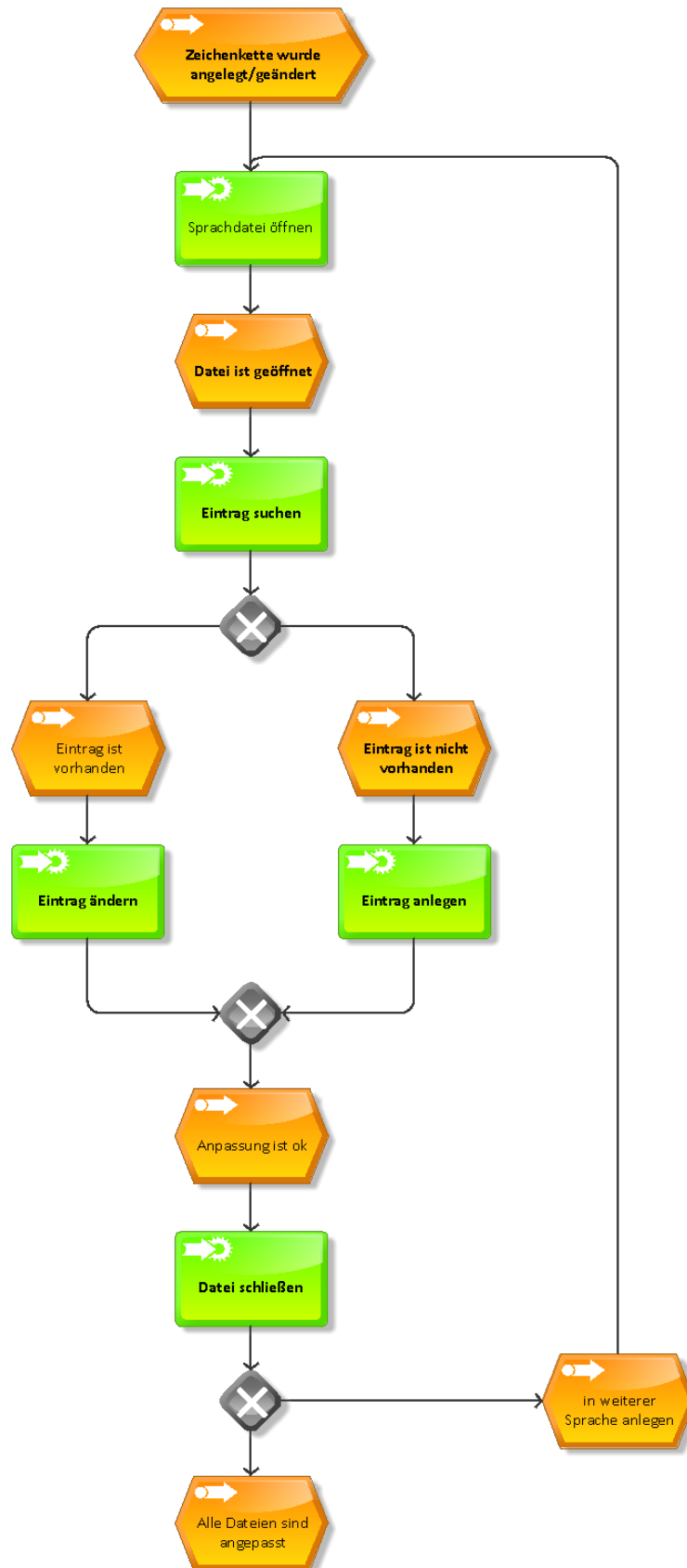
Internetquellen

- <http://nuget.org/> (zuletzt abgerufen 15.03.2018)
- Server zu dem Ausführen von Git <https://www.visualstudio.com/tfs/> (zuletzt abgerufen am 15.03.2018)
- Software für Versionskontrolle von Quellcode <https://git-scm.com> (zuletzt abgerufen am 15.03.2018)
- Desktopanwendung für Git <https://www.sourcetreeapp.com> (zuletzt abgerufen am 15.03.2018)
- <https://dotnet.github.io/docfx/> (zuletzt abgerufen am 15.03.2018)
- <http://mahapps.com> (zuletzt abgerufen am 15.03.2018)
- <http://mahapps.com/controls/dialogs.html> (zuletzt abgerufen am 15.03.2018)
- <https://msdn.microsoft.com/de-de/library/jj739835.aspx> (zuletzt abgerufen am 15.03.2018)

Internetquellen - Bibliotheken

- <https://github.com/MahApps/MahApps.Metro.Resources>
- <https://www.newtonsoft.com/json>
- <https://github.com/ControlzEx/ControlzEx>
- <https://dotnet.github.io/docfx/>
- <http://mahapps.com>

EPK Ändern eines Wertes



Auszug aus dem Lastenheft

WPF-Translate

Lastenheft

1. Zielsetzung

Das Programm soll zur einfachen Verwaltung und Übersetzung von WPF-String-Ressourcen verwendet werden.

2. Produkteinsatz

Das Produkt soll von den Programmierern des Unternehmens eingesetzt werden.

3. Produktfunktionen

/LF10/ Laden und Speichern von WPF-Ressourcendateien

/LF20/ Anlegen und Löschen von Sprachen

/LF30/ Anlegen und Löschen von Einträgen

/LF40/ Sortieren von Keys und Values (Zeichenkette)

/LF50/ Automatisches Übersetzen von einzelnen Einträgen / Sprachen

4. Produktdaten

Das Programm muss alle Daten aus einer WPF-Ressourcen-Datei verarbeiten können und diese ohne Änderungen wieder speichern können. Die Daten lassen sich in folgender Tabelle ablesen.

/LD10/ Keys

/LD20/ Values

/LD30/ Vorhandene unter Ressourcen (Abgeleitete/Vererbte Wörterbücher)

/LD40/ Sonstige Daten

5. Produktleistungen

Das Programm soll beliebig viele Sprachen und Einträge verwalten können.

6. Qualitätsanforderungen

Qualität	Sehr gut	Gut	Normal	Nicht relevant
Funktionalität	x			
Zuverlässigkeit	x			
Benutzbarkeit		x		
Effizienz			x	
Änderbarkeit		x		

Projektplanung mit Zeitplanung in Stunden

1. Startphase	3 Stunden
a. Darstellung und Abstimmung des Auftrags/Ziele	1 Stunde
b. IST-Analyse durchführen	2 Stunden
2. Projektplanung	13 Stunden
a. Soll-Konzept	6 Stunden
b. Pflichtenheft	4 Stunden
c. Vorkalkulation	2 Stunden
d. Ressourcenplanung	1 Stunde
3. Implementierungsphase	34 Stunden
a. Backend Klassen und Methoden implementieren	24 Stunden
b. Frontendentwicklung	6 Stunden
c. Schnittstelle für externe Anwendungen schaffen	2 Stunden
d. Benutzerhandbuch erstellen	2 Stunden
4. Abschlussphase	20 Stunden
a. Qualitätssicherung und Tests	5 Stunden
b. Soll-/Ist-Vergleich	1 Stunde
c. Nachkalkulation	1 Stunde
d. Projektdokumentation	10 Stunden
e. Präsentation und Abnahme der Ergebnisse	3 Stunden

Gesamtzeit des Projektes: 70 Stunden

Verwendete Ressourcen

Hardware

- Büroarbeitsplatz mit Windows Rechner

Software

- Windows 10 Professional — Betriebssystem
- Visual Studio Professional 2017 — Entwicklungsumgebung C#
- Git — Versionskontrolle
- NuGet — Paketmanager
- ARIS Express — Programm zu dem Erstellen von Diagrammen (hier [EPK](#))
- SourceTree — Git Oberfläche

Personal

- Entwickler — Umsetzung des Projektes
- Anwendungsentwickler — Abnahme und Review des Projektes

Vergleich Programmiersprachen

Eigenschaft	Gewichtung	C/C++ (bew.)	C# mit WPF (bew.)	C/C++ (gew.)	C# mit WPF (gew.)
Testbarkeit	3	4	5	12	15
Plattform: Windows	5	4	5	20	25
Kenntnisstand (Entwickler LS)	3	4	5	12	15
Kosten	4	4	4	16	16
Entwicklungsumgebung	3	5	5	15	15
Benutzeroberfläche	5	3	5	15	25
Perfromance	3	5	4	15	12
Umfang verfügbarer Bibliotheken	3	5	5	15	15
XML	5	2	5	10	25
Gesamt	34			130	163
Nutzwert				3,82	4,79

Auszug aus dem Pflichtenheft

WPF-Translate

2. Produkteinsatz

2.1. Anwendungsbereich

Das Programm soll im Bereich der Entwicklung eingesetzt werden.

2.2. Zielgruppe

Da das Programm hauptsächlich in der Entwicklung eingesetzt wird, wird das Programm einen Großteil seiner Arbeit bei Programmierern verrichten. Es soll aber auch von anderen Bürokräften genutzt werden können.

2.3. Betriebsbedingungen

Das Programm soll sich nicht wesentlich von anderen .NET-Desktopanwendungen unterscheiden. Es muss ausschließlich die .NET-Framework installiert sein. Auf die zusätzliche Installation kann ab Windows 8 verzichtet werden, da diese bereits installiert ist.

3. Produktumgebung

3.1. Software

- .NET-Framework v4.0 oder neuer

3.2. Hardware

- .NET fähige Hardware (Siehe .NET Systemanforderungen <https://docs.microsoft.com/de-de/dotnet/framework/get-started/system-requirements>)

3.3. Orgware

- Optional: Excel oder vergleichbares Tabellenprogramm zum Bearbeiten von .csv Dateien.

4. Produktfunktionen

- /F10/ Laden und Speichern von WPF-Ressourcendateien
- /F20/ Anlegen und Löschen von Sprachen
- /F30/ Anlegen und Löschen von Einträgen
- /F40/ Suchen in den Dateien nach Keys und Values
- /F50/ Automatisches Übersetzen von einzelnen Einträgen / Sprachen
- /F60/ Intigriertes Handbuch

WPF-Translate

5. Produktdaten

Da das Programm keine Daten speichern soll ist in der folgenden Tabelle ein Ausschnitt aus den WPF-Ressourcen mit den benötigten Werten zu sehen.

/D10/ Keys
/D20/ Values
/D30/ Vorhandene unter Ressourcen
/D40/ Sonstige Daten

6. Produktleistungen

/L100/ Das Programm muss über eine tolerante Lesefunktion verfügen, um verschiedenste Szenarien abzudecken.

7. Benutzeroberfläche

Da die Benutzeroberfläche in einem modernen und einfach zu bedienenden Stil gehalten werden soll, wird die Bibliothek MapApps.Metro für diese verwendet. Zusätzlich sollen alle Funktion einfach und schnell zu erreichen sein. In den folgenden Punkten wird eine grobe Übersicht über die Dialogstruktur gegeben.

7.1. Menüstruktur

Datei		Sprache		Hilfe (/F60/)
↳	Öffnen (/F10/)	↳	Sprache anlegen (/F20/)	
↳	Speichern (/F10/)	↳	Sprache löschen (/F20/)	
↳	Neu (/F20/)	↳	Sprache übersetzen (/F50/)	
↳	Suche (/F40/)			
↳	Inhalt löschen			
↳	Beenden			

Verwendete Bibliotheken

Bibliothek	Funktion
LSDOTNETCORE	Implementiert Grundfunktionen für das einfache Verwenden von WPF wie bspw. das Sortieren von WPF-ListView's. Eigenes Projekt von LS
MahApps.Metro	Modern-UI Bibliothek für WPF http://mahapps.com
MahApps.Metro.Resources	Mordern-UI-Icons Bibliothek https://github.com/MahApps/MahApps.Metro.Resources
JSON.NET	Bibliothek zu dem Arbeiten mit JSON https://www.newtonsoft.com/json
ControlzEx	Behebt Fehler für einige WPF Controls und ist eine Voraussetzung für MahApps.Metro https://github.com/ControlzEx/ControlzEx
DocFX	Tool zu der Generierung einer Entwicklerdokumentation aus XML -Kommentaren https://dotnet.github.io/docfx/

Aufbau XAML-Ressourcenwörterbuch

```
<ResourceDictionary
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  1. xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  xmlns:sys="clr-namespace:System;assembly=mscorlib">
  <ResourceDictionary.MergedDictionaries>
  2.   <ResourceDictionary Source="pack://application:,,,/LSDOTNETINFOMAN;component/Resources/Strings_de-de.xaml" />
   <ResourceDictionary Source="pack://application:,,,/PermissionManager;component/Strings.xaml" />
  </ResourceDictionary.MergedDictionaries>
  3. <SolidColorBrush x:Key="LabelBackgroundColorBrush" Color="#FFC0FFFF" />
  <Style BasedOn="{StaticResource {x:Type Label}}" TargetType="{x:Type Label}">
  4.   <Setter Property="VerticalContentAlignment" Value="Center" />
   <!--<Setter Property="FontSize" Value="{DynamicResource FontSizeKey}" />-->
  </Style>
  5. <x:String x:Key="abbrechen">Abbrechen</x:String>
  6. <x:String x:Key="reparaturSpeichernFehler" xml:space="preserve">Die Änderungen können nicht in die Reparatur übertragen werden!
  Fehler: {0}</x:String>
  7. <x:Static x:Key="vText5Doppelpunkt" Member="sys:String.Empty" />
</ResourceDictionary>
```

1. Namespaces
2. Zusätzliche Ressourcenwörterbücher
3. Anderer Typ - hier zu dem Festlegen einer Farbe
4. Anderer Typ - hier wird der Style für alle Label überschriebene und angepasst
5. String Ressource - normaler String
6. String Ressource - mit Zeilenumbruch
7. String Ressource - leere Zeichenkette

NotifyBase Klasse

```
/// <summary>
/// WPF NotifyBase Klasse eine Klasse, welche das INotifyPropertyChanged
/// Interface implementiert
/// </summary>
[Serializable]
public class NotifyBase : INotifyPropertyChanged
{
    /// <summary>
    /// Wird aufgerufen, wenn sich eine Eigenschaft ändert
    /// </summary>
    [field: NonSerialized] //Verbietet das Serialisieren des Events.
    public event PropertyChangedEventHandler PropertyChanged;

    /// <summary>
    /// Benachrichtigt die Oberfläche über eine geänderte Eigenschaft
    /// </summary>
    /// <param name="name">Name der Eigenschaft (wenn string.empty oder null
    /// werden alle Eigenschaften geupdatet). Wenn kein Parameter angegeben,
    /// wird der Compiler versuchen einen zu generieren,</param>
    public virtual void RaisePropertyChanged([CallerMemberName]string name = "")
    {
        PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(name));
    }
}
```


RelayICommand Klasse

```
/// <summary>
/// Eine Klasse zu dem einfachen Verwenden von ICommand
/// </summary>
public class RelayCommand<T> : ICommand
{
    /// <summary>
    /// CanExecute Code
    /// </summary>
    protected Predicate<T> canExecute;

    /// <summary>
    /// Execute Code
    /// </summary>
    protected Action<T> execute;

    /// <summary>
    /// Eine Klasse zu dem einfachen Verwenden von ICommand
    /// </summary>
    /// <param name="canExecute">CanExecute Methode</param>
    /// <param name="execute">Execute Methode</param>
    public RelayCommand(Predicate<T> canExecute, Action<T> execute)
    {
        this.canExecute = canExecute;
        this.execute = execute;
    }

    /// <summary>
    /// Eine Klasse zu dem einfachen Verwenden von ICommand.
    /// (CanExecute gibt immer true zurück)
    /// </summary>
    /// <param name="execute">Execute Methode</param>
    public RelayCommand(Action<T> execute) : this(T => true, execute)
    { }
}
```

```
/// <summary>
/// Can Execute Changed
/// </summary>
public event EventHandler CanExecuteChanged
{
    add
    {
        CommandManager.RequerySuggested += value;
    }
    remove
    {
        CommandManager.RequerySuggested -= value;
    }
}

/// <summary>
/// Prüft, ob der Command ausgeführt werden darf
/// </summary>
/// <param name="parameter">Parameter</param>
/// <returns>Can Execute</returns>
public bool CanExecute(object parameter)
{
    return canExecute(ParamToT0rDefault0fT(parameter));
}

/// <summary>
/// Führt den Command aus.
/// </summary>
/// <param name="parameter">Parameter</param>
public void Execute(object parameter)
{
    execute(ParamToT0rDefault0fT(parameter));
}
```

```
/// <summary>
/// Kastet den Parameter zu einem T.
/// Wenn Parameter null ist wird default(T) zurück gegeben
/// </summary>
/// <param name="parameter"></param>
/// <returns></returns>
protected T ParamToTOrDefaultOfT(object parameter)
{
    return parameter == null ? default(T) : (T)parameter;
}

/// <summary>
/// Eine Klasse zu dem einfachen Verwenden von ICommand
/// </summary>
public class RelayCommand : RelayCommand<object>
{
    /// <summary>
    /// Eine Klasse zu dem einfachen Verwenden von ICommand
    /// </summary>
    /// <param name="canExecute">CanExecute Methode</param>
    /// <param name="execute">Execute Methode</param>
    public RelayCommand(Predicate<object> canExecute, Action<object> execute) :
        base(canExecute, execute)
    { }

    /// <summary>
    /// Eine Klasse zu dem einfachen Verwenden von ICommand.
    /// (CanExecute gibt immer true zurück)
    /// </summary>
    /// <param name="execute">Execute Methode</param>
    public RelayCommand(Action<object> execute) : base(execute)
    { }
}
```

ResourceHelper in LSDOTNETINFOMAN

```
/// <summary>
/// Bietet Funktionen zu dem einfachen Verwenden von Wpf Ressourcen und dem
///     einfachen Nachladen von Sprach-Dateien.
/// </summary>
public static class ResourceHelper
{
    /// <summary>
    /// Fügt ein Ressourcenwörterbuch zu den Anwendungsressourcen hinzu
    /// </summary>
    /// <param name="resource">Ressourcenwörterbuch</param>
    public static void AddResource(ResourceDictionary resource)
    {
        Application.Current.Resources.MergedDictionaries.Add(resource);
    }

    /// <summary>
    /// Sucht nach einem Wörterbuch
    /// </summary>
    /// <param name="filter"></param>
    /// <returns></returns>
    public static ResourceDictionary
        FindDictionary(Func<string, string, bool> filter)
    {
        Stream baml = GetResourceStreams(Assembly.GetEntryAssembly(),
                                           filter).FirstOrDefault();

        if (baml == null)
            return null;

        return LoadResourceDictionaryFormBaml(baml);
    }
}
```

```
/// <summary>
/// Sucht nach einem Wörterbuch für eine Sprache
/// ( Sprachen müssen enden mit z.B. langkey.xaml)
/// </summary>
/// <param name="langID">Sprach ID z.B. de-de</param>
/// <returns>ResourceDictionary</returns>
public static ResourceDictionary FindLanguageDictionary(string langID)
{
    langID = langID.ToLower();

    return FindDictionary((rname, fname) =>
        fname.EndsWith(langID + ".baml"));
}

/// <summary>
/// Sucht in der aktuellen Anwendung nach einer Ressource
/// </summary>
/// <param name="key">Key</param>
/// <returns>Ressource</returns>
public static object FindRessource(object key)
{
    return Application.Current?.TryFindResource(key);
}

/// <summary>
/// Sucht in der aktuellen Anwendung nach einer String-Ressource
/// </summary>
/// <param name="key">Key</param>
/// <returns></returns>
public static string FindString(object key)
{
    return FindRessource(key).ToString();
}
```

```
/// <summary>
/// Sucht nach einem Ressourcenschlüssel
/// </summary>
/// <param name="dic">Wörterbuch</param>
/// <param name="value">Wert</param>
/// <returns>Schlüssel</returns>
public static object FindResourceKeyByValue(ResourceDictionary dic,
                                             object value)
{
    foreach (object item in dic.Keys)
    {
        object v = dic[item];

        if (v == value)
            return item;
    }

    if (dic.MergedDictionaries != null)
    {
        foreach (ResourceDictionary item in dic.MergedDictionaries)
        {
            object ret = FindResourceKeyByValue(item, value);

            if (ret != null)
                return ret;
        }
    }

    return null;
}
```

```
/// <summary>
/// Ruft alle Ressourcenstreams der Anwendung ab
/// </summary>
/// <returns>Liste an Streams</returns>
public static List<Stream> GetResourceStreams()
{
    return GetResourceStreams(Assembly.GetEntryAssembly());
}

/// <summary>
/// Ruft alle Ressourcenstreams der Anwendung ab
/// </summary>
/// <param name="assembly">Assembly</param>
/// <returns>Liste an Streams</returns>
public static List<Stream> GetResourceStreams(Assembly assembly)
{
    return GetResourceStreams(assembly, (r, s) => true);
}

/// <summary>
/// Wandelt einen Baml Stream in ein Ressourcenwörterbuch
/// </summary>
/// <param name="stream">Stream</param>
/// <returns>Wörterbuch</returns>
public static ResourceDictionary
    LoadResourceDictionaryFormBaml(Stream stream)
{
    return LoadBaml<ResourceDictionary>(stream);
}
```

```
/// <summary>
/// Ruft all Ressourcenstreams der Anwendung ab
/// </summary>
/// <param name="assembly">Assembly</param>
/// <param name="filter">Filter (Resourcen file, Key String)</param>
/// <returns>Liste an Streams</returns>
public static List<Stream> GetResourceStreams(Assembly assembly,
                                             Func<string, string, bool> filter)
{
    List<Stream> bamlStreams = new List<Stream>();

    string[] resourceDictionaries = assembly.GetManifestResourceNames();
    foreach (string resourceName in resourceDictionaries)
    {
        ManifestResourceInfo info =
            assembly.GetManifestResourceInfo(resourceName);

        if (info.ResourceLocation !=
            ResourceLocation.ContainedInAnotherAssembly)
        {
            Stream resourceStream =
                assembly.GetManifestResourceStream(resourceName);

            using (ResourceReader reader =
                new ResourceReader(resourceStream))
            {
                foreach (DictionaryEntry entry in reader)
                {
                    if (filter(resourceName,
                                entry.Key.ToString().ToLower()))
                    {
                        bamlStreams.Add(entry.Value as Stream);
                    }
                }
            }
        }
    }
    return bamlStreams;
}
```



```
/// <summary>
/// Wandelt einen BAML-Stream (geprüfte XAML-Datei) in ein Objekt um
/// </summary>
/// <typeparam name="T">Type</typeparam>
/// <param name="stream">Stream</param>
/// <returns>Neues T</returns>
public static T LoadBaml<T>(Stream stream)
{
    Baml2006Reader reader = new Baml2006Reader(stream);
   .XamlObjectWriter writer = new.XamlObjectWriter(reader.SchemaContext);

    while (reader.Read())
    {
        writer.WriteNode(reader);
    }

    return (T)writer.Result;
}

/// <summary>
/// Läuft alle Sprachwörterbücher
/// </summary>
/// <param name="langID">SprachID</param>
public static void LoadLanguageDictionary(string langID)
{
    ResourceDictionary dic = FindLanguageDictionary(langID);

    if (dic != null)
        AddResource(dic);
}
}
```

Handbuch WPF-Translate

Inhalt

Oberfläche.....	3
Menüfunktionen.....	4
Datei.....	4
Öffnen	4
Speichern.....	4
Neu	4
Suche	4
Inhalt löschen.....	4
Beenden	4
Sprache	5
Sprache anlegen.....	5
Sprache löschen	5
Sprache übersetzen.....	5
Funktionen auf der Oberfläche.....	6
Eintrag anlegen.....	6
Eintrag löschen	6
Eintrag übersetzen.....	6
Exemplarisches Vorgehen zum anlegen neuer Sprachdateien	7
Schnittstelle	8

Oberfläche

Datei

Öffnen Strg+O

Speichern Strg+S

Neu Strg+N

Suche Strg+F

Inhalt Löschen

Beenden

Sprache

Sprache anlegen Strg+N

Sprache löschen

Sprache übersetzen

	KEY	DE-DE	NL-NL
X T	abbrechen	Abbrechen	
X T	abbrechenESC	ESC - Abbrechen	ESC - Annuleren
X T	abDatumDoppelpunkt	ab Datum:	vanaf datum:
X T	abgelehnt	Abgelehnt	verworpen
X T	abgelehnt2	2 - Abgelehnt	2 - Weigerde
X T	abgelehnt5	5 - Abgelehnt	5 - Weigerde
X T	abweichenderFormularnameDoppelpunkt	abweichender Formularname:	
X T	abweichenderOrdnernameBilderDoppelpunkt	abweichender Ordnername für Bilder:	
X T	abweichenderOrdnernameINETShopDoppelpunkt	abweichender Ordnername für INETSHOP:	
X T	adressdaten	Adressdaten	adresgegevens
X T	adressdatenholen	Adressdaten holen	
X T	adresse	Adresse	adres
X T	adresseDoppelpunkt	Adresse:	
X T	adresseKopieren	Adresse kopieren	Kopieer het adres
X T	adressen	Adressen	
X T	adresseNichtGeladen	Die Adresse wurde nicht geladen!	
X T	adressfeldFuellen	Adressfeld füllen	Vul adresveld in

↖

Löscht einen Eintrag

↖

Öffnet ein Eintrag
im Übersetzer

↗

Fügt einen neuen
Eintrag hinzu

Menüfunktionen

Datei

Öffnen

Öffnet eine oder mehrere Dateien. Es können nur .xaml-Dateien geöffnet werden. Bei den Dateien muss es sich um WPF-RessourceDictionaries handeln. Alle nicht String Daten, wie z.B. Styles oder MergedDictionaries, werden zwischengespeichert und während des [Speicherns](#) wieder in die Dateien geschrieben.

Die Funktion kann auch über die Tastenkombination **STRG+O** aufgerufen werden.

Speichern

Speichert alle offenen Dateien. Alle Zusatzdaten, wie z.B. Styles oder MergedDictionaries, werden wieder mit in diese Datei geschrieben.

Die Funktion kann auch über die Tastenkombination **STRG+S** aufgerufen werden.

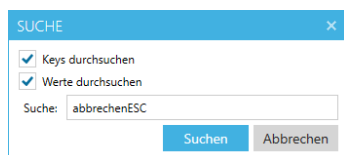
Neu

Legt eine neue Sprache an.

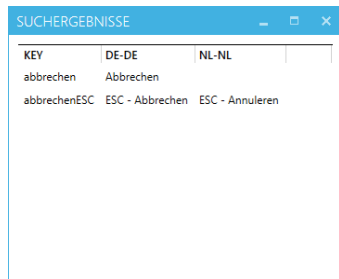
Die Funktion kann auch über die Tastenkombination **STRG+N** aufgerufen werden.

Suche

Diese Funktion erlaubt das schnelle Suchen nach Einträgen.



Wurden in der Suche mehrere Ergebnisse gefunden werden diese in einem Extra Dialog angezeigt.



KEY	DE-DE	NL-NL
abbrechen	Abbrechen	
abbrechenESC	ESC - Abbrechen	ESC - Annuleren

Die Funktion kann auch über die Tastenkombination **STRG+F** aufgerufen werden.

Inhalt löschen

Diese Funktion entfernt alle Dateien aus dem Programm und erlaubt das Laden und Arbeiten mit neuen Dateien.

Beenden

Beendet die Anwendung.

Sprache

Sprache anlegen

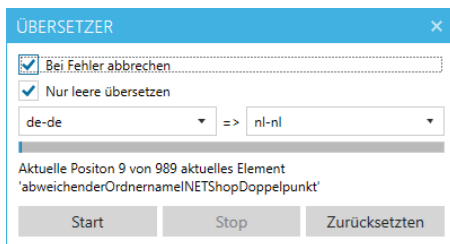
Siehe [Menüfunktionen->Datei->Neu](#)

Sprache löschen

Löscht eine Sprache aus der Oberfläche. Die Dateien werden nicht von der Festplatte gelöscht.

Sprache übersetzen

Mit Hilfe dieser Funktion können ganze Sprachen oder einzelne Einträge in eine neue Sprache übersetzt werden.



Funktionen auf der Oberfläche

Eintrag anlegen

In der [Oberfläche](#) kann ein neuer Eintrag hinzugefügt werden. Damit dieser Button geklickt werden kann, muss mindestens eine Sprache geladen werden.

Eintrag löschen

In der Oberfläche kann ein Eintrag, aus allen Sprachen, durch ein einfachen Klick, auf den entsprechenden Button, entfernt werden.

Eintrag übersetzen

Diese Funktion ähnelt dem [Übersetzten von Sprachen](#). Es wird allerdings nur ein Eintrag angezeigt und übersetzt.

Exemplarisches Vorgehen zum anlegen neuer Sprachdateien

1. Legen sie neue Sprachdateien über [Menüfunktionen->Datei-Neu](#) an. Während des Anlegens werden Sie aufgefordert einen Sprachekey festzulegen. Dieser Key wird zum Identifizieren der Sprache verwendet kann jedoch auch frei gewählt werden. Sprachkeys sollten nach dem MS-Windows Language Code Identifier Muster festgelegt werden. Wiederholen Sie diesen Vorgang für alle benötigten Sprachen. Das Anlegen weiterer Sprachen ist jederzeit möglich.
2. Fügen sie über die Funktion [Funktionen auf der Oberfläche->Eintrag anlegen](#) neue Einträge an, geben sie einen Key und die Entsprechenden Werte an. Zum Übersetzten der Sprachen kann zusätzlich die Funktion [Menüfunktionen->Sprache->Sprache übersetzten](#) oder [Funktionen auf der Oberfläche->Eintrag übersetzten](#) verwendet werden.
3. Speichern Sie die Dateien oder Änderungen über [Menüfunktionen->Datei->Speichern](#).

Schnittstelle

Über die Bibliothek LSDOTNETCORE ist es möglich das einfache Laden, der erstellten Wörterbücher, zu ermöglichen. Die Klasse *ResourceHelper* befindet sich in dem Namespace *de.LandauSoftware.Core.WPF*. Die Funktion der Methoden kann der XML-Dokumentation entnommen werden.

Entwicklungsdokumentation

Class Language

Stellt eine Sprache dar

Inheritance

↳ System.Object
↳ Language

Namespace: [de.LandauSoftware.WPFTranslate](#)

Assembly: WPF-Translate.dll

Syntax

```
public class Language : NotifyBase
```

Constructors

Language()

Initialisiert eine neue Instanz der Sprache

Declaration

```
public Language()
```

Language(String)

Initialisiert eine neue Instanz der Sprache

Declaration

```
public Language(string key)
```

Parameters

Type	Name	Description
System.String	key	Sprachkey z.B. en-us

Properties

LangKey

Ruft den Sprachkey ab oder setzt diesen

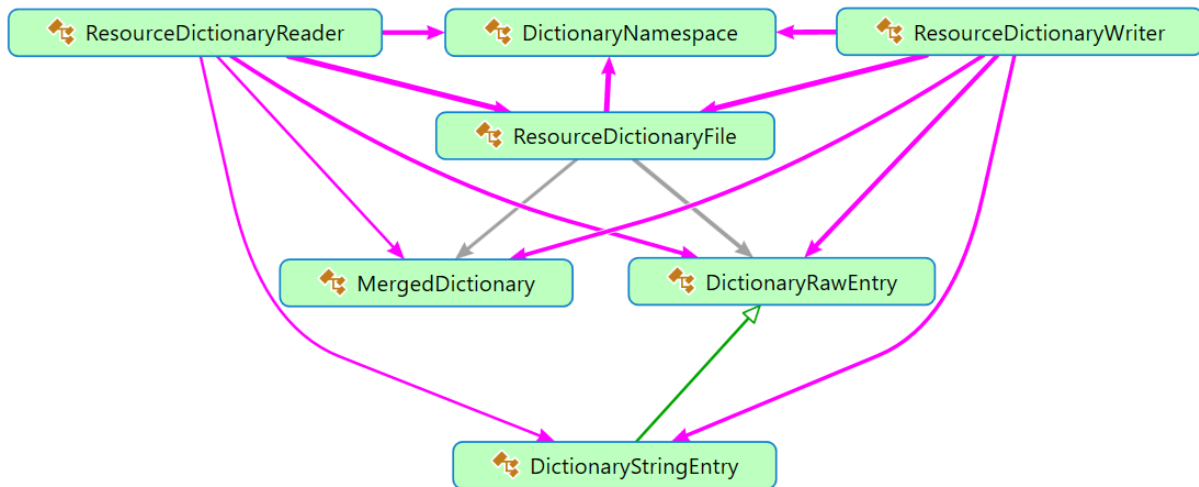
Declaration

```
public string LangKey { get; set; }
```

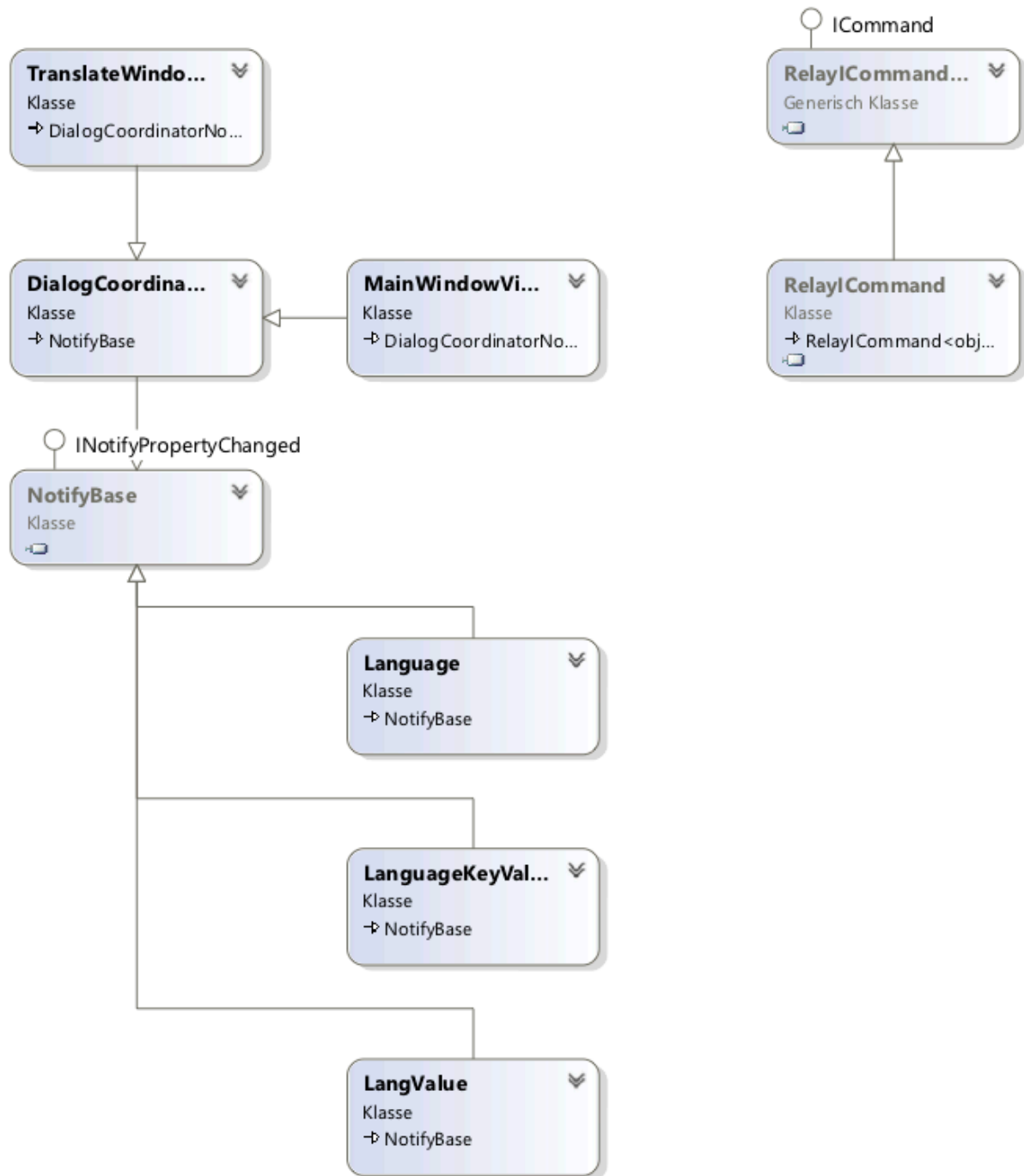
Property Value

Type	Description
System.String	

Code Map der IO Funktionen



UML Klassenmodell der Model und ViewModel Klassen



Screenshots der Anwendung

WPF-TRANSLATE				
Datei Sprache Hilfe				
	KEY	DE-DE	NL-NL	
X T	abbrechenESC	ESC - Abbrechen	ESC - annuleren	
X T	abDatumDoppelpunkt	ab Datum:	vanaf datum:	
X T	abgelehnt	Abgelehnt	verworpen	
X T	abgelehnt2	2 - Abgelehnt	2 - Verworpen	
X T	abgelehnt5	5 - Abgelehnt	5 - Verworpen	
X T	abweichenderFormularnameDoppelpunkt	abweichender Formularname:	andere vormnaam:	
X T	abweichenderOrdnernameBilderDoppelpunkt	abweichender Ordnername für Bilder:	andere mapnaam voor afbeeldingen:	
X T	abweichenderOrdnernameINETShopDoppelpunkt	abweichender Ordnername für INETSHOP:	verschillende mapnaam voor INETSHOP:	
X T	adressdaten	Adressdaten	adresgegevens	
X T	adressdatenholen	Adressdaten holen	Krijg adresgegevens	
X T	adresse	Adresse	adres	
X T	adresseDoppelpunkt	Adresse:	adres:	
X T	adresseKopieren	Adresse kopieren	Adres kopiëren	
X T	adressen	Adressen	adressen	
X T	adresseNichtGeladen	Die Adresse wurde nicht geladen!	Het adres is niet geladen!	
X T	adressfeldFuellen	Adressfeld Füllen	Vul adresveld in	
X T	adressfeldFuerinfomanFormulareDoppelpunkt	Adressfeldanzeige für INFOMAN Formulare:	Adresveldweergave voor INFOMAN-formulieren:	
X T	aendern	Ändern	verandering	
X T	aenderungenNichtInReparaturÜbertragen	Die Änderungen können nicht in die Reparatur {0}		
X T	aenderungenVerwerfenFrage	Sie haben umgespeicherte Änderungen! Sollen diese verworfen werden?	U heeft niet-opgeslagen wijzigingen! Moeten ze worden weggegooid?	

ÜBERSETZER

☒ Bei Fehler abbrechen
☐ Nur leere übersetzen

de-de => nl-nl

Aktuelle Positon 23 von 989 aktuelles Element
 'aktuellerKundeDoppelpunkt'

Start Stop Zurücksetzen

SUCHE

☒ Keys durchsuchen
☒ Werte durchsuchen

Suche:

Suchen Abbrechen