

Hausarbeit

Anwendungsprogrammierung

WS 21/22

Vor- und Nachname:	Dimitri Dorn
Matrikelnummer:	690199
Prüfungszeitraum:	WS-2021/22-II
Einzelprojektnummer:	017
Gruppennummer:	017

Hinweise zur Verwendung dieser Vorlage: Sie können diese Vorlage verwenden oder mit einer Textverarbeitung Ihrer Wahl nachbauen. Ändern Sie auf keinen Fall die Formatierung! Schriftart (Liberations Serif oder TimesNewRoman), Schriftgröße, Abstände, Schriftfarbe, Seitenabstand, Einrückung etc. sind beizubehalten; oder im Falle eines "Nachbaus" auf identische Weise herzustellen. Auch dieser Hinweis darf nicht entfernt werden!

Inhaltsverzeichnis

1. Übersicht des Programms aus Nutzersicht.....	1
1.1. Walkthrough.....	1
1.1.1. Featureauswahl im Hauptfenster.....	1
1.1.2. Übersicht des Tex Book Collection Windows.....	1
1.1.3. Auswahl eines Bib Files.....	2
1.1.4. Import der Bücher in die System Collection.....	3
1.2. Ergänzende Hinweise.....	4
1.2.1. Aufgabe des Data Grids.....	4
1.2.2. Default Werte für Bucheigenschaften.....	4
1.2.3. Spaltenerweiterung nach dem Einlesen.....	4
2. Übersicht über die Solution.....	5
2.1. UML-Diagramm.....	5
2.2. Schichten des MVVM-Patterns.....	5
2.2.1. View.....	5
2.2.2. View Model.....	5
2.2.3. Services.....	6
2.3. Struktureller Aufbau der einzelnen Klassen.....	6
2.3.1. Ui.Desktop.....	6
TexBookCollectionWindow.....	6
Data Grid Columns.....	7
ErrorWindow.....	8
2.3.2. Logic.Ui.....	8
Konstruktoraufruf.....	9
OpenFileMethod.....	9
ImportFileMethod.....	10
CheckImportedBooks.....	11
2.3.3. Services.TexImport.....	12
SplittPath.....	12
AddItemsToCollection.....	12
SplittByItems.....	13
SplittTheBook.....	14
PropertyFinder.....	15
Property.....	15
PropertyContent.....	16
PropertyAssigner.....	17
Konstruktor: PropertyAssigner.....	17
Case-Anweisungen.....	17
3. Besondere Features und Implementierungsdetails.....	18
3.1. WPF-Fenster.....	18
3.1.1. Unpassender Name für das Import Fenster.....	18
3.1.2. Kleine Fenstergröße des TexBookCollectionWindows.....	18
3.1.3. String Format für das Release Date.....	18
3.1.4. Kein WindowViewModel für das ErrorWindow.....	18
3.2. View Model.....	18
3.2.1. Hinweis zu der OpenFileMehod.....	18
3.2.2. Aufruf von Debug.WriteLine in OpenFileMethod.....	19
3.3. Services.....	19
3.3.1. Import-Datei.....	19
3.3.2. Unpassender Name für die Split-Klasse.....	19
3.3.3. String-Parameter der SplittByItems-Methode.....	19

3.3.4. Entfernen des Usernamen in der SplittTheBook Methode.....	19
3.3.5. Unspezifischer Property-Name in der Klasse PropertyFinder.....	19
3.3.6. Mögliche Bucheigenschaften in PropertyFinder.....	20

1. Übersicht des Programms aus Nutzersicht

Aufgabe: Erläutern Sie durch Screenshots im Walkthrough möglichst kurz, prägnant und übersichtlich alle Funktionen und Features, die Ihr Programm(-teil) bietet. Für das Hauptfenster können Sie auf die gemeinsame Kurzanleitung verweisen. In die Ergänzenden Hinweise erläutern Sie alles, was nicht in den Walkthrough passt.

1.1. Walkthrough

Beim Walkthrough versetzen Sie sich in einen hypothetischen User und zeigen exemplarisch den konkreten Handlungsablauf, welcher Ihre Funktionen möglichst abdeckt. Im Folgenden wird der Walkthrough für den Programmteil *Import eines Tex-Files (.bib)* beschrieben.

1.1.1. Featureauswahl im Hauptfenster

Der Walkthrough startet im Hauptfenster. Eine Kurzanleitung zu den Funktionen und Features des Hauptfensters wird in *AWP_017_Kurzanleitung.pdf* erläutert. In dem Hauptfenster befinden sich drei Buttons, die auf die jeweiligen Hauptfunktionen der Bookmanager-App verweisen. Damit die Teilfunktion des Tex-Imports aufgerufen werden kann, muss der Button *Oben Bib Import* betätigt werden.



1.1.2. Übersicht des Tex Book Collection Windows

Nachdem im Hauptfenster der Bib Import ausgewählt wurde, erscheint ein weiteres Window, welches für die beschriebene Teilfunktion zuständig ist. Der Titel dieses Fensters lautet *TexBookCollectionWindow*.

Den größten Platz nimmt im diesem Fenster das Data Grid¹ ein, welches die für den Upload ausgewählte Bücher und deren Eigenschaften anzeigt. Sobald das Tex Book Collection Window geöffnet wird, besteht die Tabelle nur aus zwei Zeilen und sechzehn Spalten. Dabei beschriften die Spalten in der ersten Zeile die möglichen Bucheigenschaften, die in diesem Programm festgelegt werden können. Da anfangs noch keine Bücher für den Import markiert worden sind, bleiben alle Spalten in der zweiten Zeile leer.

In der unteren linken Ecke des Fensters sind untereinander noch zwei Buttons positioniert. Der obere Button ist hierbei mit dem Titel *Choose File* und der untere mit dem Titel *Import Books* benannt.



Abbildung 1.2: Darstellung des Tex Book Collection Windows

1.1.3. Auswahl eines Bib Files

Damit der Nutzer die gewünschten Bücher aus der jeweiligen Bib-Datei für den Import auswählen kann, muss zunächst der Button *Choose File* bedient werden. Daraufhin wird ein File Dialog geöffnet, wodurch sich der Benutzer im Verzeichnis seines Rechners durch navigieren kann, um anschließend die gewünschte Tex-Datei für den Import auszusuchen.

Nachdem die Datei ausgesucht und die Bucheigenschaften vom Programm ausgelesen wurden, füllt sich das Data Grid mit den entsprechenden Werten². Je nachdem wie viele Bücher in der ausgewählten Tex Datei vorhanden sind, wird die Tabelle um diese Bücheranzahl mit Zeilen erweitert. Dabei wird jede Spalte mit der entsprechenden Bucheigenschaft, die aus einem Buch in der Bib-Datei ausgelesen wurde, befüllt³.

1 Steuerelement, welches automatisch Zeilen und Spalten für die übergebenen Dateneigenschaften erzeugt. Genauere Beschreibung wird in dem Abschnitt *Aufgabe des DataGrids* erläutert.
 2 Abschnitt: *Default Werte für Bucheigenschaften*
 3 Abschnitt Spaltenerweiterung nach dem Einlesen

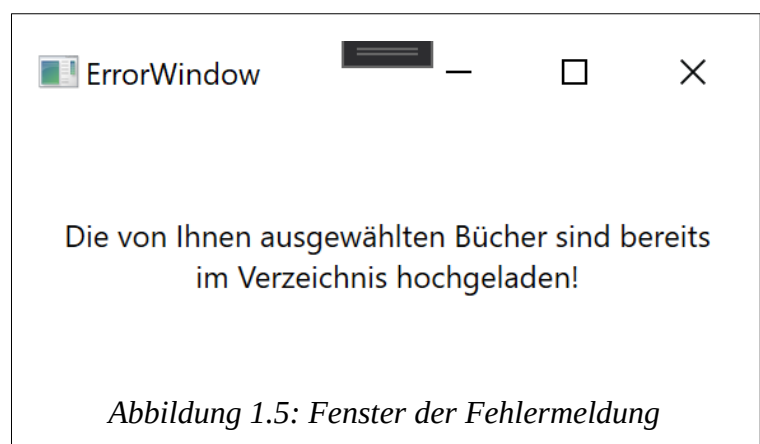
Author	Price	ReleaseDate	Publisher	Description	Pages
Wolff, Manfred AND Hauck, Peter AND Küchlin, Wolfgang	0	2004	Springer-Verlag		500
Norman L. Biggs, E. Keith Lloyd, Robin J. Wilson	0	1999			0
Kastens, U. and B{\\"u	0	2005	Hanser		0
Stephan Kleuker	0	2011	Vieweg+Teubner		0
Anton Nijholt	0	1988	Elsevier Science Ltd		0
Gregor Büchel	0	2012	Vieweg+Teubner		0
Dr. Katrin Erk, Prof. Dr. Lutz Priese	0	2008	Springer		0
John E. Hopcroft, Rajeev Motwani, Jeffrey D. Ullman	0	2003	Vieweg+Teubner		0
Christina Büsing	0	2010	Springer Spektrum		0
Abbildung 1.3: Temporäre Buch Kollektion mit Werten					

1.1.4. Import der Bücher in die System Collection

Nachdem der Nutzer festgelegt hat, welche Bücher in die Buch Kollektion des gesamten Systems gespeichert werden sollen, müssen diese durch einen bestimmten Befehl importiert werden. Denn eigentlich findet der Buchimport erst dann statt. Hervorgerufen wird der Import, indem der untere Button *Import Books* angeklickt wird. Daraufhin leert sich die temporäre Liste, sodass das Data Grid wie beim ersten Aufruf nur aus zwei Zeilen besteht, bei der die Spalten der letzten Zeile leer sind.

Author	Price	ReleaseDate	Publisher	Description	Pages
Abbildung 1.4: Leere Temporäre Kollektion nach dem Import					

Ein weiteres mögliches Ereignis, welches nach dem Button-Klick auftreten kann, ist das Erscheinen einer Fehlermeldung. Die Fehlermeldung erscheint in Form eines neuen Fensters und tritt im dem Fall auf, wenn die temporäre Buch Kollektion mindestens ein Buch besitzt, welches schon im Model enthalten ist.



1.2. Ergänzende Hinweise

Hier können Sie mögliche Nachbehandlungen, Ergebnisse im Dateisystem oder sonstige Ergänzungen dokumentieren, welche im Walkthrough keinen Platz gefunden haben.

1.2.1. Aufgabe des DataGrids

Das DataGrid des Tex Book Collection Windows ist im Prinzip eine automatisch erzeugte Tabelle, die mit den Büchern befüllt wird, die vom Nutzer für den Import ausgewählt werden. Die Tabelle übernimmt somit die Rolle einer temporären Liste und erzeugt eine Übersicht von den Büchern, die aus den ausgewählten .bib-Dateien ausgelesen wurden. Dabei ist es wichtig zu sehen, dass die Bücher dieser temporären Kollektion noch nicht in die System Kollektion importiert wurden. Es können außerdem noch weitere Bücher aus anderen Tex-Dateien ins DataGrid hinzugefügt werden, indem erneut auf den *Choose File* Button gedrückt wird und im Verzeichnis eine neue .bib-Datei ausgelesen wird.

1.2.2. Default Werte für Bucheigenschaften

Wenn es in der ausgewählte Tex-Datei Bücher gibt, die nicht zu jeder Buch-Property einen Wert stehen haben, dann werden den Properties im Model definierte Default Werte zugewiesen. Bei String Properties ist ein Leerstring als Default Wert festgelegt und bei Integer Properties Zahl 0. Falls in der Tex-Datei eine Enum Property undefiniert bleibt, so wird dieser Bucheigenschaft der erste Enumwert zugewiesen.

1.2.3. Spaltenerweiterung nach dem Einlesen

Zu dem Zeitpunkt, wo das Import-Window geöffnet wird, sind die Breiten der DataGrid-Spalten an die Länge der jeweiligen Spaltenüberschrift angepasst. Nachdem im File Dialog eine .bib-Datei ausgesucht wurde, nimmt die Breite automatisch zu, sodass sich jede Spalte dem längsten Eigenschaftswert anpasst. Dadurch wird auch die Tabelle der temporären Liste deutlich in die Breite gezogen, sodass nicht mehr alle Spalten innerhalb der Import-Window-Reichweite bleiben. Mithilfe einer horizontaler Scrollleiste können jedoch trotzdem alle Bestandteile des DataGrids betrachtet werden.

2. Übersicht über die Solution

Aufgabe: Beschreiben Sie den strukturellen Aufbau Ihrer Implementierung. Gehen sie auf die Schichten des MVVM-Patterns, auf die einzelnen Namespaces sowie die einzelnen Klassen ein. Begründen Sie alle Designentscheidungen. Für den strukturellen Aufbau zeichnen Sie außerdem ein UML-Diagramm mit den Projekten als Packages, sowie den Namespaces als Packages. Achten Sie darauf, dass das UML-Diagramm gut lesbar ist - bei Schwierigkeiten geben Sie das UML-Diagramm **zusätzlich** als hochauflösende Bilddatei oder pdf-Datei ab.

2.1. UML-Diagramm

Das UML-Diagramm ist unter *017-UML-Diagramm-AWP-Hausarbeit.png* als eine separate Bilddatei gespeichert.

2.2. Schichten des MVVM-Patterns

Für die Implementierung *Import eines Tex-Files* wurde nur in den Schichten *View* und *View Model* Namespaces und Klassen angelegt. Trotzdem spielte für diese Implementierung auch das *Model* eine große Rolle, da dort besonders in dem Namespace *Business.Model.BusinessObjects* auf die Klassen *Book* und *BookCollection* zugegriffen wurde, um die Eigenschaften der zu importierenden Büchern festzulegen, sowie die Buch Kollektion des Systems (also die *BookCollection*) mit neuen Büchern zu füllen. Wie diese Prozesse im einzelnen stattfinden, wird in 2.3 detaillierter erläutert.

2.2.1. View

Die *View* stellt das dar, was der Benutzer davon erwartet. In der Klassenbibliothek *Ui.Desktop* befinden sich XAML-Dateien, die die Struktur und Interface der Fenster (also die UI) definieren. Für die Implementierung des Tex-Imports wurden zwei User Interfaces erstellt.

Das erste Fenster heißt *TexBookCollectionWindow* und ist als eine XAML-Klasse unter den Namen *TexBookCollectionWindow.xaml* gespeichert. Dieses Window beinhaltet alle UI-Steuerelemente, die für den Tex-Import von Bedeutung sind. Ein weiteres Fenster, das zu der Implementierung gehört, heißt *ErrorWindow* und dient als eine einfache Fehlermeldung, ohne jegliche Funktionalitäten. Auch dieses Fenster befindet sich im selben Namespace und ist in der XAML-Klasse *ErrorWindow.xaml* definiert.

Des weiteren befindet sich in der *View* ein Ordner *MessageBusLogic*, in dem sich die Klasse *MessageListener* befindet. Diese Klasse ist ausschlaggebend für die Delegate Übertragung an den *MessageBus* und die Initialisierung der Windows der Tex-Import-Implementierung.

2.2.2. View Model

Das *View Model* stellt das Model für die *View* dar. Es erzeugt somit eine Datenbindung zu der *View* und übergibt alle Änderungen. Außerdem stellt das *View Model* Funktionalitäten per *Commands* zur Verfügung. Diese werden ebenfalls an die *View* gebunden, wodurch die *Code Behind* Datei reduziert wird.

Besonders wichtig für die Implementierung des Tex-Imports ist es, dass das WPF-Fenster *TexBookCollectionWindow* im ViewModel sein eigenes *ViewModelWindow* besitzt. Die ViewModels für die Windows befinden sich alle im Namespace *Logic.Ui.ViewModels*. Dabei ist die Klasse *TexBookCollectionWindowViewModel* das zuständige ViewModel des WPF-Fensters *TexBookCollectionWindow*. Hier befindet sich der wesentliche Code, der für die Logic UI wichtig ist. Dazu zählen die Relay Commands und Methoden, die Änderungen an die View oder an das Model weiterleitet. Darauf wird aber in 2.4 weiter eingegangen.

Außerdem befinden sich im ViewModel unter dem Namespace *Logic.Ui.MessageBusMessages* die Window Messages, die die Instanzen der Tex-Import-WPF-Fenster erzeugen. Im *ViewModelLocator*, der ebenfalls im ViewModel liegt, wird das ViewModel *TexBookCollectionWindowViewModel* initialisiert. Dabei wird dem *WindowViewModel* das *BookCollectionViewModel* übergeben, welches ebenfalls im Locator initialisiert wurde.

2.2.3. Services

Die Services-Schicht ist kein typisches Bestandteil des MVVM-Patterns, jedoch ist sie bedeutsam für die Implementierung des Tex-Imports. Denn genau dort liegt die Klassenbibliothek *Services.TexImport* mit den drei Klassen, die wichtige Teilaufgaben des Imports übernehmen. Diese sind *SplittPath*, *PropertyFinder* und *PropertyAssigner*. Welche Teilaufgaben in den einzelnen Klassen verarbeitet werden, wird in 2.5 genauer ausgeführt.

Einige dieser Klassen benötigen einen Zugang auf das Model, damit neue *Book*-Objekte erstellt werden können und daraufhin deren Properties festgelegt werden können. Außerdem wird von der Klasse *SplittPath* auf die *BookCollection* zugegriffen, damit eine temporäre Buch Collection generiert werden kann. Aus diesen Gründen ist der Klassenbibliothek *Services.TexImport* eine Referenz an das Model bzw. an die Klassenbibliothek *Business.Model* zugewiesen worden. Das ist auch die einzige Referenz, die von Tex-Import-Services ausgeht.

2.3. Struktureller Aufbau der einzelnen Klassen

Nachdem der grobe Aufbau der Implementierung in 2.2 beschrieben wurde, werden nun die einzelnen Klassen im Detail betrachtet und dessen Designentscheidungen begründet. Natürlich werden hierbei nur die Klassen in Betracht gezogen, die ausschließlich für den Tex-Import gebraucht werden. Dabei werden die Klassen der drei benutzten Klassenbibliotheken (*Ui.Desktop*, *Logic.Ui* und *Services.TexImport*) nach der Reihenfolge analysiert.

2.3.1. Ui.Desktop

In dieser Klassenbibliothek sind zwei XAML-Dateien gespeichert, die jeweils den Aufbau eines WPF-Fensters bestimmen. Die Code Behind Datei ist bei beiden Fenstern leer geblieben, da die Funktionalitäten in das ViewModel ausgelagert wurden.

TexBookCollectionWindow

Die Oberfläche dieses Fensters besteht insgesamt aus drei Steuerelementen. Nämlich aus zwei Buttons und einem Data Grid. Alle diese Steuerelemente sind innerhalb eines *Grid*-Containers positioniert. Auf dem ersten Button steht *Choose File*. Wie der Name schon sagt, sorgt dieses

Element dafür, dass der Nutzer durch das Klicken auf den File Dialog seines Rechners zugreifen kann, um anschließend die gewünschte Bib-Datei für den Import auszuwählen. Der zweite Button mit dem Namen *Import Books* sorgt dafür, dass die ausgesuchten Bücher ins Programmverzeichnis importiert werden. Das Data Grid ist hierbei eine Tabelle, die die ausgewählten Bücher mit deren Eigenschaften darstellt. Dafür benötigt das Data Grid ein *BookCollectionViewModel*, das die Bücher mit den Eigenschaften beinhaltet. Deswegen enthält das Data Grid eine Item Source, welche auf das *BookCollectionViewModel TempBooks* zeigt. *TempBooks* ist eine Collection, die im ViewModel definiert ist. Wie die UI an das View Model gebündet wird, wird im Folgenden erklärt.

Wie bereits schon erklärt, gibt es für das *TexBookCollectionWindow* keine Code Behind Datei, in der die Logik der Steuerelementen definiert ist. Den Programmcode entnimmt die UI aus dem *TexBookCollectionWinowViewModel* aus der Klassenbibliothek *Logic.Ui* mithilfe des *ViewModel Locators*, der als eine globale Ressource innerhalb der gesamten Klassenbibliothek zur Verfügung steht. Um auf diese Ressource zugreifen zu können, muss das WPF-Fenster unter *DataContext* an den *ViewModelLocator* gebündet werden. Durch das Argument *Path* wird das richtige ViewModel angegeben (nämlich *TheTexBookCollectionWinowViewModel*), mit dem das WPF-Fenster verknüpft werden soll.

```
<Window.DataContext>
  <Binding Source="{StaticResource ViewModelLocator}" Path="TheTexBookCollectionWindowViewModel"></Binding>
</Window.DataContext>
```

Abbildung 2.1: DataContext für das Bindng an den ViewMdelLocator

Nachdem das WPF-Fenster mit dem zuständigen WindowViewModel verknüpft wurde, können den Steuerelemente die dazugehörigen *RelayCommands* aus dem ViewModel zugewiesen werden, sodass sie eine Funktionalität bekommen.

```
<Button Command="{Binding OpenFile}" Margin="10,244,683,75.5">Choose File</Button>
<Button Command="{Binding ImportFile}" Margin="10,302,683,18.5">Import Books</Button>
```

Abbildung 2.2: Binding Source der Buttons

Data Grid Columns

In dem DataGrid-Container wurde durch das Argument *AutoGenerateColumns* eingestellt, dass die Spaltengenerierung nicht automatisch von der gebündeten *TempBooks*-Collection durchgeführt wird. Grund dafür ist, dass die Collection einige Informationen beinhaltet, die für den Nutzer irrelevant sind. Dazu zählen die Informationen, die angeben, in welchem Namespace sich die Models der Bücher befinden oder welche Enum-Werte die Enum-Klassen beinhalten. Damit für diese Informationen keine eigene Spalten im Data Grid erstellt werden, wurde das Argument *AutoGenerateColumns* auf *False* gesetzt.

Die Spalten für die Buchproperties, die angezeigt werden sollen, werden also manuell im *DataGrid.Columns*-Container festgelegt. Jeder Spalte wird ein Binding-Path zugewiesen, die richtige Buch-Propertie bestimmt. Außerdem wird im *Header*-Argument der Spaltentitel festgelegt.

```

<DataGrid
  Name="uploadedFiles"
  ItemsSource="{Binding Path=TempBooks}"
  Margin="130,0,0,0.5"
  AutoGenerateColumns="False"
  GridLinesVisibility="All">
  <DataGrid.Columns>
    <DataGridTextColumn Binding="{Binding Path=Title}" Header="Title"></DataGridTextColumn>
    <DataGridTextColumn Binding="{Binding Path=Author}" Header="Author"></DataGridTextColumn>
    <DataGridTextColumn Binding="{Binding Path=Price}" Header="Price"></DataGridTextColumn>
    <DataGridTextColumn Binding="{Binding Path=ReleaseDate, StringFormat={}{0:yyyy}}" Header="ReleaseDate"></DataGridTextColumn>
    <DataGridTextColumn Binding="{Binding Path=Publisher}" Header="Publisher"></DataGridTextColumn>
    <DataGridTextColumn Binding="{Binding Path=Description}" Header="Description"></DataGridTextColumn>
    <DataGridTextColumn Binding="{Binding Path=Pages}" Header="Pages"></DataGridTextColumn>
    <DataGridTextColumn Binding="{Binding Path=Weight}" Header="Weight"></DataGridTextColumn>
    <DataGridTextColumn Binding="{Binding Path=Isbn}" Header="ISBN"></DataGridTextColumn>
    <DataGridTextColumn Binding="{Binding Path=Rating}" Header="Rating"></DataGridTextColumn>
    <DataGridTextColumn Binding="{Binding Path=Edition}" Header="Edition"></DataGridTextColumn>
    <DataGridTextColumn Binding="{Binding Path=Bestseller}" Header="Bestseller"></DataGridTextColumn>
    <DataGridTextColumn Binding="{Binding Path=Extract}" Header="Extract"></DataGridTextColumn>
    <DataGridTextColumn Binding="{Binding Path=Genre}" Header="Genre"></DataGridTextColumn>
    <DataGridTextColumn Binding="{Binding Path=Format}" Header="Format"></DataGridTextColumn>
    <DataGridTextColumn Binding="{Binding Path=Language}" Header="Language"></DataGridTextColumn>
  </DataGrid.Columns>
</DataGrid>

```

Abbildung 2.3: DataGrid Container mit Column-Einstellungen

ErrorWindow

In dem Error Window ist nur ein Steuerelement, nämlich ein Text Block, enthalten. In diesem Block sieht die Fehlermeldung, dass die ausgewählten Bücher bereits im Verzeichnis vorhanden sind. Dabei ist die Nachricht zentral positioniert und ist über das ganze WPF-Fenster verteilt, da es auch das einzige Element im Grid Container ist.

```

<Grid>
  <TextBlock Margin="10" TextWrapping="Wrap" TextAlignment="Center" VerticalAlignment="Center" HorizontalAlignment="Center">
    Die von Ihnen ausgewählten Bücher sind bereits im Verzeichnis hochgeladen!</TextBlock>
</Grid>

```

Abbildung 2.4: Grid Container des ErrorWindows

2.3.2. Logic.Ui

Die Klassenbibliothek *Logic.Ui* enthält nur eine für den Tex-Import wichtige Klasse. Nämlich das *TexBookCollectionWindowViewModel*. Wie der Name schon sagt, ist diese Klasse ein Window View Model für das Fenster *TexBookCollectionWindow*. So wie alle Winfow View Models, befinden sich auch dieses in dem Ordner *ViewModels*.

Das *TexBookCollectionWindowViewModel* beinhalten Implementierungen für die UI-Steuererelemente, die als Delegates an Relay Commands (*OpenFile* und *ImportFile*) übergeben werden. Ähnlich wie die Relay Commands werden zwei *BookCollectionViweModels* als Klassenattribute deklariert. Das erste *BookCollectionWindowViewModel* heißt *TempBooks* und dient als eine temporäre Buch Kollektion, die mit den vom Nutzer ausgewählten Büchern befüllt ist. Dabei ist diese Kollektion an das Data Grid der UI gebündet, sodass sie im WPF-Fentser dargestellt wird. Da nur Properties mit einem Data Grid verknüpft werden können, ist die Kollektion

TempBooks eine Property. Die zweite Book Kollektion heißt *bookCollectionViewModel*. Diesem Attribut wird die Instanz auf das *BookCollectionViewModel* zugewiesen, das vom *ViewModelLocator* übergeben wurde. Somit wird auf diese Kollektion zugegriffen, sobald Änderungen in dem Model *BookCollection* übernommen werden sollen und dafür zunächst die Wrapper Klasse aktualisiert wird.

```
5 references
public BookCollectionViewModel TempBooks { get; set; }

private BookCollectionViewModel bookCollectionViewModel;
1 reference
public ICommand OpenFile { get; }
1 reference
public ICommand ImportFile { get; set; }
```

Abbildung 2.5: Attribute des *TexBookCollectionWindowViewModels*

Konstruktoraufruf

Sobald das Window View Model des Tex-Imports und somit auch der Konstruktor aufgerufen wird, werden die globalen Command- und Collection-Variablen initialisiert. Für die Relay Commands werden die zustehenden Funktionen als Delegate übergeben. Dem ICommand *OpenFile* wird die Funktion *OpenFileMethod* und dem ICommand *ImportFile* die Funktion *ImportFileMethod* als Delegate überreicht.

Dem Konstruktor muss außerdem ein Parameter vom Typ *BookCollectionViewModel* übergeben werden. Das geschieht im *ViewModelLocator*, wenn dort *TexBookCollectionWindowViewModel* initialisiert wird und als Parameter die einmalige Instanz des *BookCollectionViewModels* übergeben wird. Genau mit dieser Instanz wird im Konstruktor die globale Variable *bookCollectionViewModel* initialisiert. Die temporäre Book Collection *TempBooks* wird im Konstruktor als ein *BookCollectionViewModel* initialisiert, sodass auf sie in den kommenden Methoden zugegriffen werden kann.

```
1 reference
public TexBookCollectionWindowViewModel(BookCollectionViewModel viewModelCollection)
{
    OpenFile = new RelayCommand(OpenFileMethod);
    ImportFile = new RelayCommand(ImportFileMethod);
    bookCollectionViewModel = viewModelCollection;
    TempBooks = new BookCollectionViewModel();
}
```

Abbildung 2.6: Konstruktor von *TexBookCollectionWindowViewModel*

OpenFileMethod

OpenFileMethod ist eine parameterlose Funktion, die dem Relay Command *OpenFile* als Delegate übergeben wird und somit nach dem Klicken des *Choose File* Buttons ausgeführt wird. Die Funktionalität, die in dieser Methode bestimmt wird, widmet sich der Auswahl einer Bib-Datei im

File Dialog. Das geschieht, indem zunächst eine Referenz auf die Klasse des *OpenFileDialogs* gesetzt wird und anschließend mit der Methode *ShowDialog* das Erscheinen des Dialog Fensters hervorgerufen wird.

Die nächste Aufgabe dieser Funktion ist, den **gesamten** Inhalt der ausgewählten Bib-Datei mit Hilfe der *ReadAllText*-Funktion aus der *File*-Klasse als String auszulesen. Als nächstes wird mit dem String des gesamten Inhalts eine neues *SplittPath*-Objekt initialisiert. Es wird also eine Referenz zu der Klasse *SplittPath* erzeugt, die sich in der Klassenbibliothek *Services.TextImport* befindet und als Parameter wird der gesamte Stringinhalt *readText* übergeben, sodass im erzeugten Objekt nur die ausgewählten Bücher betrachtet werden.

Nachdem das Objekt erstellt wurde, wird die Methode *AddItemsToCollection* in *SplittPath* aktiviert. *TempBooks* wird in diesem Aufruf als Parameter übergeben. Mit diesem Funktionsaufruf wird in *SplittPath* der Prozess aktiviert, bei dem der Stringinhalt in Bücher und Properties unterteilt wird und anschließend die *TempBooks* Collection mit den jeweiligen Büchern erweitert. Der genaue Ablauf wird in 2.3.3 erläutert. Wichtig ist hierbei, dass das Wrapper-Objekt *TempBooks* entkapselt und in Form einer *BookCollection* an die Funktion überreicht wird. Deshalb wird der temporären Collection bei dem Methodenaufruf eine *.Model*-Endung dran gehangen.

```
1 reference
private void OpenFileMethod()
{
    OpenFileDialog openFileDialog = new OpenFileDialog();
    openFileDialog.ShowDialog();
    var path = openFileDialog.FileName;
    string readText = File.ReadAllText(path);
    SplittPath mySplittPath = new SplittPath(readText);
    mySplittPath.AddItemsToCollection(TempBooks.Model);
    Debug.WriteLine(readText);
}
```

Abbildung 2.7: *OpenFileMethod*

ImportFileMethod

Die Aufgabe dieser parameterlosen Funktion besteht darin, dass die in *TempBooks* vorhandenen Bücher in die *BookCollection* hinzugefügt werden, wenn keins davon bereits im Model vorhanden ist. In dem Fall, wenn in der temporären Collection mindestens ein Buch in vorhanden ist, das einen gleichen Titel besitzt, wie ein Buch aus dem *BookCollectionViewModel*, soll eine Fehlermeldung hervorgehoben werden. Die *ImportFileMethod* wird dem Relay Command *ImportFile* als Delegate übergeben und ist somit für die Logic des Buttons *Import Books* aus der Ui zuständig.

Der Aufbau dieser Funktion ist ganz simpel. Als erstes wird mit Hilfe einer if-Abfrage überprüft, ob kein Buch aus *TempBooks* bereits in der *BookCollection* vorhanden ist. Das geschieht, indem die Methode *CheckImportedBooks* in der Abfrage aufgerufen wird. Wenn das zutrifft, wird in einer foreach-Schleife jedes Buch aus *TempBooks* dem globalen Collection *bookCollectionViewModel* hinzugefügt. Da *bookCollectionViewModel* ein Wrapper Objekt ist, werden die Änderungen in der *BookCollection* des Models übernommen. Im else-Zweig wird dem Message Bus die Instanz einer neuen *OpenErrorMessage* zugesendet, sodass ein weiteres WPF-Fenster aufgerufen wird,

das den Nutzer über die bereits importierten Bücher informiert. Zum Schluss werden alle Bücher aus der temporären Collection mithilfe der Collection Funktion *Clear* entfernt, wodurch das auch Data Grid in der Ui keine weiteren Bücher mehr anzeigt. Somit erfährt der Nutzer, dass der Import erfolgreich war.

```
1 reference
public void ImportFileMethod()
{
    if (!CheckImportedBooks())
    {
        foreach (var book in TempBooks)
        {
            this.bookCollectionViewModel.Add(book);
        }
    }
    else
    {
        ServiceBus.Instance.Send(new OpenNewErrorMessage());
    }
    TempBooks.Clear();
}
```

Abbildung 2.8: ImportFileMethod

CheckImportedBooks

Diese Methode dient als eine Hilfsfunktion für die *ImportFileMethod*. Wie bereits im Abschnitt *ImportFileMethod* beschrieben, wird dort die *CheckImportedBooks* Methode bei der if-Abfrage aufgerufen, um die Collections *TempBooks* und *bookCollectionViewModel* nach identischen Buchtiteln zu überprüfen. Als Rückgabewert gibt die Funktion einen Boolean *foundSameBooks* zurück, der im Falle identischer Buchtitel den Wert *True* besitzt.

Der Vergleich der zwei Collections findet durch eine doppelte Schleife statt. In der äußeren Schleife wird als Index das betrachtende Buch dem *bookCollectionViewModel* bestimmt. Der Buchtitel des aktuellen Buches wird in der inneren Schleife mit den Buchtiteln aus *TempBooks* verglichen und auf Identität geprüft. Ist eine Identität vorhanden, so wird der Wert des Booleans *foundSameBooks* auf *True* gesetzt. Wenn *TempBooks* ausschließlich neue Bücher enthält, wird die doppelte Schleife komplett durchlaufen, ohne dass *foundSameBooks* verändert wird und weiterhin den *False*-Status beibehält.

```
1 reference
private bool CheckImportedBooks()
{
    bool foundSameBook = false;
    for (int i = 0; i < bookCollectionViewModel.Count; i++)
    {
        foreach (var book in TempBooks)
        {
            if (bookCollectionViewModel.ElementAt(i).Title == book.Title)
            {
                foundSameBook = true;
                return foundSameBook;
            }
        }
    }
    return foundSameBook;
}
```

Abbildung 2.9: CheckImportedBooks Methode

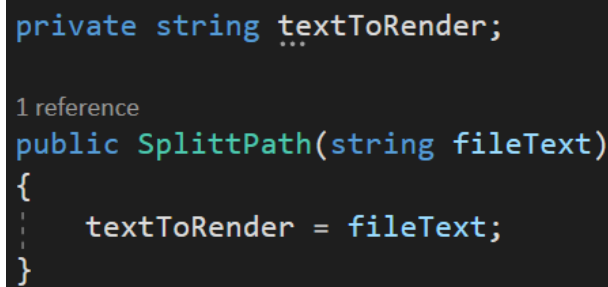
2.3.3. Services.TextImport

Die Klassenbibliothek *Services.TextImport* liegt in der Services-Schicht und beschäftigt sich mit der Aufteilung des Gesamtstrings, der die Buchinformationen der Tex-Datei beinhaltet. Dabei werden zunächst in der Klasse *SplittPath* die einzelnen Bücher aus der Datei ermittelt, woraufhin die dort vorhandenen Datei-Zeilen der Klasse *PropertyFinder* übergeben wird, um diese auf Properties zu untersuchen. Als nächstes werden ggf. die erkannten Properties in der Klasse *PropertyAssigner* einem *Book*-Objekt zugewiesen/verändert. Zum Schluss werden die in *SplittPath* initialisierten Bücher der temporären Collection aus *TexBookCollectionWindowViewModel* hinzugefügt. Wichtig zu sehen ist dabei, dass alle Klassen der Service-Schicht nur mit dem Model verknüpft sind. Deshalb muss das ViewModel den Wrapper der übergebenden Collection zunächst in ein Model entkapseln, da die Services Klassen keinen Zugriff auf ein *BookCollectionViewModel* haben.

SplittPath

SplittPath gilt als die Hauptklasse der Klassenbibliothek *Services.TextImport*. Zum einen wird genau dieser Klasse vom View Model aus die in ein String geparte Tex-Datei übergeben, sodass diese nach Büchern und Eigenschaften aufgeteilt werden kann. Außerdem werden in *SplittPath* die restlichen Klassen aktiviert, die für die Property-Suche und -Zuweisung zuständig sind (also *PropertyFinder* und *PropertyAssigner*).

Bei der Initialisierung eines *SplittPath*-Objekts wird ein String-Parameter übergeben, der die in ein String umgewandelte Tex-Datei bereitstellt. Im Konstruktor wird dann der Parameter der globalen String-Variable *textToRender* zugeteilt. Somit ist *textToRender* ein Klassenattribut, das in jeder *SplittPath*-Funktion genutzt werden kann.



```
private string textToRender;

1 reference
public SplittPath(string fileText)
{
    textToRender = fileText;
}
```

Abbildung 2.10: String-Attribut und Konstruktor von *SplittPath*

AddItemsToCollection

In dieser Methode wird die übergebene *BookCollection* mit den Büchern befüllt, die in *textToRender* enthalten sind. *AddItemToCollection* ist die einzige Methode in der Klasse, dessen Zugriffsmodifizierer auf *public* gesetzt ist. Das hat den Grund, weil die Funktion im ViewModel *TexBookCollectionWindowViewModel* aufgerufen wird, sodass von dort aus die temporäre Buch Collection an die Funktion überreicht wird.

Als erstes wird in der Methode mithilfe der Funktion *SplittByItems* ein String-Array initialisiert. In diesem Array befinden sich einzelne String-Segmente, die in *SplittByItems* aufgeteilt wurden. Innerhalb einer foreach-Schleife wird dann jedes Segment überprüft, ob es sich um ein Buch-

Segment handelt oder nicht. Dabei wird in einer If-Abfrage die String-Methode *Contains* eingesetzt, die jedes Segment überprüft, ob das einen Teilstring *BOOK{* enthält. Wenn diese Bedingung zutrifft, gibt die *Contains*-Methode *True* als Ausgabewert zurück und somit weiß das Programm, dass es sich im aktuellen Stringsegment um ein Buch handelt.

Im Falle eines Buch-Segmentes wird zunächst ein neues *Book*-Objekt initialisiert. Das Buchobjekt wird mit dem dazugehörigen Teilstring an die Funktion *SplittTheBook* als Parameter weitergegeben. Nach diesem Funktionsaufruf wurden die Bucheigenschaften ermittelt und zugewiesen. Wie es im Detail abläuft, wird in den Abschnitten *PropertyFinder* und *PropertyAssigner* beschrieben. Zum Schluss wird der *BookCollection* das fertige Buch hinzugefügt.

```
1 reference
public void AddItemsToCollection(BookCollection collection)
{
    string[] items = SplittByItems(textToRender);
    foreach (var item in items)
    {
        if (item.Contains("BOOK{"))
        {
            Book newBook = new Book();
            SplittTheBook(item, newBook);
            collection.Add(newBook);
        }
    }
}
```

Abbildung 2.11: AddItemToCollction Methode

SplittByItems

SplittByItems ist eine Hilfsmethode für die Funktion *AddItemsToCollection*, die als Parameter *path* den Inhalt der Tex-Datei als String übergeben bekommt. In einer Tex-Datei können sich Item-Segmente befinden, die nicht nur Buchinformationen beinhalten, sondern auch der Kategorie *misc* (*verschiedenes*) dazugehören können. Im Programm wird davon ausgegangen, dass in der Tex-Datei alle Item-Segmente mit dem *@*-Zeichen markiert sind.

Für die Aufteilung eines Strings in Substrings wird eine String-Methode *Split* verwendet. Natürlich wird die *Split*-Methode auf den Gesamtstring *path* angewandt. Für den ersten Parameter wird der String-Array *splittItem* gesetzt. Der Arrayinhalt besteht nur aus einem Inhalt, nämlich nur aus einem *@*-Zeichen. Somit gibt *splittItem* in der *Split*-Funktion an, dass nach jedem *@* ein neuer Substring erzeugt werden soll. Zusätzlich wird im zweiten Parameter die *StringSplitOptions* deaktiviert, sodass alle Teilzeichenfolgen bei der Aufteilung mit eingeschlossen werden. Als Ergebnis wird ein String-Array mit den aufgeteilten Items zurückgegeben. Dieser String-Array wird unter dem Namen *devidedItems* gespeichert und von der *SplittByItems*-Methode als Rückgabewert zurückgegeben.


```

1 reference
private string[] SplittByItems(string path)
{
    string[] splittItem = new string[1] { "@" };
    var devidedItems = path.Split(splittItem, StringSplitOptions.None);
    return devidedItems;
}

```

Abbildung 2.12: *SplittByItems Methode*

SplittTheBook

SplittTheBook ist eine Funktion, die ebenfalls in der Funktion *AddItemsToCollection* aufgerufen wird. Denn nachdem dort ein Teilstring als ein Buchinhalt identifiziert wurde, wird dafür ein neues Book-Objekt initialisiert. Sowohl der Teilstring *stringItem* als auch das Book-Objekt *currentBook* werden als Parameter weitergegeben, wenn *SplittTheBook* aufgerufen wird.

Als erstes wird jede Zeile des Buchinhalt-Strings in einen String Array *devidedLines* gespeichert. Die Zeilenaufteilung findet ähnlich wie bei *SplittByItems* durch die String-Methode *Split* statt. Als Separator-Parameter, der vorgibt bei welchem Zeichen eine Trennung stattfinden soll, wird der String Array *splittLine* verwendet. Dieser beinhaltet die mögliche Zeichenfolgen, die in C# eine Escapesequenz in Form eines Zeilenumbruchs auslöst.

Im nächsten Schritt wird in einer If-Abfrage jeweils eine Zeile aus dem Array überprüft, ob es sich nicht um das Schließelement eines Buch-Items handelt, das in der verwendeten Tex-Datei mit dem Zeichen „}“ gekennzeichnet ist. Außerdem prüft die If-Bedingung, dass die aktuelle Zeile kein Leerstring oder ein Spacefeld ist. Somit wird sichergestellt, dass die jeweilige Zeile eine Buchproperty beinhaltet.

Falls die Zeile des aktuellen Schleifen-Indexes eine Property wiedergibt, wird für diese Zeile ein Objekt der Klasse *PropertyFinder* initialisiert. Dort wird einerseits ermittelt, um welche Buch-Property es sich genau handelt und wie der Property-Content lautet. Diese beiden Informationen werden aus den Properties der Klasse *PoprtyFinder* und jeweils unter den Variablen *currentProperty* und *content* gespeichert. Anschließend werden die ermittelten Informationen und das Book-Objekt *currentBook* der Klasse *PropertyAssigner* weitergegeben, damit die Eigenschaften des Book- Objekts angepasst werden.

Somit teilt die *SplittTheBook*-Methode die übergebenen Bücher in Zeilenstrings auf und dient danach als Zeilenübermittler an die Klassen *PropertyFinder* und *ProperlyAssigner*, sodass das Buchobjekt mit den Properties aus den Zeilenstrings befüllt wird.

```

1 reference
private void SplittTheBook(string stringItem, Book currentBook)
{
    string[] splittLine = new string[3] { "\n\t", "\n", "" };
    var devidedLines = stringItem.Split(splittLine, StringSplitOptions.None);
    devidedLines[0] = "";
    foreach (var line in devidedLines)
    {
        if (line != "" && line != "}" && line != " ")
        {
            PropertyFinder finder = new PropertyFinder(line);
            string currentProperty = finder.Property;
            string content = finder.PropertyContent;
            PropertyAssigner assigner = new PropertyAssigner(currentProperty, content, currentBook);
        }
    }
}

```

Abbildung 2.13: *SplittTheBook Methode*

PropertyFinder

PropertyFinder ist die 2. Klasse von Services.TexImport. Zuständig ist sie für die Ermittlung einer Property-Art und des dazugehörigen Property-Inhalts, die sich in der übergebenen Zeile aus *SplittTheBook* befinden können. Um ein Objekt dieser Klasse erstellen zu können, muss ein String beim Konstruktoraufwurf überreicht werden, der die zu überprüfende Buch-Item-Ziele enthält. Im Konstruktor wird diese dann dem Klassenattribut *currentLine* zugewiesen, auf die Zeile innerhalb der gesamten Klasse zugegriffen werden kann. Die Ermittlung der Buch-Property und dessen Inhalt findet jeweils in einer eigenen Property statt. Innerhalb des jeweiligen get-Eigenschaftensaccessor werden Konvertierungen oder Untersuchungen des Zeilenstrings durchgeführt. Somit wird der Klasse *SplittTheBook* direkt die richtig angepassten *Property*- und *PropertyContent*-Werte übergeben, ohne dass für die Ermittlung eine zusätzliche Methode verwendet wird.

Property

Die *Property*-Eigenschaft überprüft die übergebene Zeile auf Bucheigenschaften. Alle möglichen Bucheigenschaften sind im String-Array-Attribut *categories* definiert. Innerhalb einer For-Schleife wird bei einer If-Abfrage auf *currentLine* die *Contains*-Methode angewandt. Als Parameter wird die Buch-Property aus *categories* festgelegt, die im aktuellen Schleifenduhrlauf dran ist. Somit wird mit der *Contains*-Methode überprüft, ob die übergebene Zeile eine der möglichen Bucheigenschaften entspricht. Sobald das zutrifft, wird die Porperty mit der *ToUpper*-Methode auf die Uppercase-Schreibweise normiert und unter dem *property*-Attribut gespeichert. Wenn currnetLine keine Buch-Property besitzt, wird die Schleife so lange durchlaufen, bis alle Elemente von *categories* eingesetzt wurden. Anschließend wird dem *property*-Attribut ein null-Wert zugewiesen.

```

private string property;..
private string[] categories = {"TITLE", "AUTHOR", "PRICE", "YEAR", "PUBLISHER",
    "DESCRIPTION", "PAGES", "WEIGHT", "ISBN", "isbn", "RATING", "EDITION", "BESTSELLER",
    "EXTRACT", "GENRE", "FORMAT", "LANGUAGE"};

```

Abbildung 2.14: Attribute "property" und "categories"

PropertyAssigner

PropertyAssigner ist die dritte Klasse der Bibliothek *Services.TexImport*. Ihre Aufgabe besteht daraus, die in *PropertyFinder* ermittelte Buch-Property und dessen Content einem überreichten Book-Objekt zuzuordnen. In der Methode *SplittTheBook* der Klasse *SplittPath* wird ein Objekt zu PropertyAssigner initialisiert. Dabei müssen als Parameter die Buchproperty *property*, der Property Inhalt *propertyContent* und das jeweilige Book-Objekt *book* übergeben werden.

Konstruktor: PropertyAssigner

Im Konstruktor dieser Klasse ist eine Switch-Case Anweisung implementiert. Dabei ist *property* der zu übergebende Ausdruck, der untersucht werden muss und dessen Wert dann mit den case-Sprungzielen verglichen wird. Jeder Case stellt dabei eine mögliche Buch-Property als String dar. Die einzelnen Fälle werden dann mit der *property* verglichen und bei Übereinstimmung wird der jeweilige case ausgeführt.

Case-Anweisungen

Jeder Case ist jeweils für die Zuordnung einer Book-Property zuständig. Wenn ein Case ausgeführt wird, dann wird die richtige Book-Objekt-Property mit dem dazugehörigen Inhalt *propertyContent* hinterlegt. Die Rückgabewerte der Bucheigenschaften, wie *Title* oder *Author*, werden in Form eines Strings erwartet. Das betrifft jedoch nicht alle Bucheigenschaften. Zum Beispiel muss die Property von *ReleaseDate* als Typ *DateTime* vorkommen oder von *Pages* als Integer. Deshalb müssen bestimmte case-Sprungzeilen den String *property* in den richtigen Typ mithilfe von Methoden der *Convert*-Klasse bringen. Für die Konvertierung des Strings in ein Enum-Wert wird die *TryParse*-Methode von der Klasse *Enum* verwendet.

```
public PropertyAssigner(string property, string propertyContent, Book book)
{
    switch (property)
    {
        case "TITLE":
            book.Title = propertyContent;
            break;

        case "AUTHOR":
            book.Author = propertyContent;
            break;

        case "PRICE":
            book.Price = propertyContent != "" ?
                Convert.ToInt32(propertyContent) : book.Price;
            break;

        case "YEAR":
            book.ReleaseDate = DateTime.Parse("01/" + propertyContent);
            break;
    }
}
```

Abbildung 2.17: Konstruktor von PropertyAssigner mit ersten vier Case-Anweisungen

3. Besondere Features und Implementierungsdetails

Aufgabe: Erläutern Sie hier alle Features bzw. Implementierungsdetails, die Sie bisher noch nicht erläutert haben.

3.1. WPF-Fenster

3.1.1. Unpassender Name für das Import Fenster

Der Name, der für das Tex-Import-Fenster verwendet wird, lautet *TexBookCollectionWindow*. Zwar wird dort im Data Grid eine temporäre Buch Collection angezeigt, die der Nutzer ausgesucht hat, jedoch ist das nur eine Teilaufgabe, die in diesem Fenster dargestellt wird. Die wesentliche Aufgabe ist der Import von Büchern, die aus einer Tex-Datei entnommen werden. Dementsprechend wäre es angebrachter gewesen, wenn diese Funktion im Namen lesbar wäre. Beispielsweise würde *TexImportWindow* den alten Namen gut ersetzen.

3.1.2. Kleine Fenstergröße des *TexBookCollectionWindows*

Die Data Grid Tabelle nimmt an der Breite zu, sobald eine Tex-Datei eingelesen wurde, da sich die Spalten mit Namen/Werten füllen. Es gibt zwar in dem Fenster eine horizontale Scrollbar, womit die gesamte Breite der Tabelle getrachtet werden kann, allerdings wäre eine größere Fensterbreite angenehmer und angebrachter gewesen, um den gesamten Content zu sehen.

3.1.3. String Format für das Release Date

In dem *DataGrid.Column* Container für die Property *ReleaseDate* ist ein zusätzliches Argument für das String Format angelegt. Dort wird nämlich eingestellt, dass die formatierte String-Darstellung des Datums nur das Jahr ohne Monat und Tag anzeigt.

3.1.4. Kein WindowViewModel für das ErrorWindow

Das ErrorWindow dient lediglich als eine Fehlermeldung. In der XAML-Klasse ist nur ein Text Block hinterlegt, der den Nutzer drauf hinweist, dass bestimmte Bücher bereits im Verzeichnis existieren. Somit befinden sich im ErrorWindow keine Logic UI Elemente, sodass diesem WPF-Fenster kein eigenes WindowViewModel zugewiesen werden muss. Natürlich wird im ViewModelLocator dann auch kein WindowViewModel für das ErrorWindow initialisiert.

3.2. View Model

3.2.1. Hinweis zu der OpenFileDialog

Wichtig zu sehen bei dieser Methode ist, dass hier lediglich der gesamte Inhalt der Tex-Datei als ein String ausgelesen und anschließend dem *SplittPath*-Objekt überreicht wird. Der Prozess, in dem die einzelnen Bücher und Properties ermittelt werden, findet in *Services.TexImport* statt. *OpenFileMethod* dient dabei nur als Parameter-Übermittler an das *SplittPath*-Objekt.

3.2.2. Aufruf von Debug.WriteLine in OpenFileMethod

Diese Zeile ist für die Funktionalität überflüssig. Durch diesen Befehl wird ein bestimmter String, in diesem Fall *readText*, beim Debuggen in eine Terminal-Nachricht umgewandelt und im Terminal Fenster aufgelistet. Dieser Vorgang diene als eine Hilfeleistung während des Programmierens und kann eigentlich entfernt werden.

3.3. Services

3.3.1. Import-Datei

Während des Umsetzungsprozess des Tex-Imports wurde für zwischen läufige Test eine bestimmte Tex-Datei als Vorlage genommen. Diese Datei heißt *Bib-Datei* und dessen Pfad liegt in dem Verzeichnis *CleintApp011*. Anhand dieser Datei richtete sich größtenteils die Import-Implementierung. Besonders die Formatierung der Bücher und Bucheigenschaften, die in der Vorlagedatei gegliedert sind, sind bei der Codierung sehr ausschlaggebend für die Segmentierung der einzelnen Buch-Items. Somit ist die Implementierung des Tex-Imports an die Vorlagedatei gerichtet und kann im Falle eines Imports von anderen Tex-Dateien einige Fehler aufweisen.

3.3.2. Unpassender Name für die Split-Klasse

Für die Aufteilung der Tex-Datei in Bücher und Zeilen wurde die Klasse *SplittPath* erstellt. Dabei entspricht der verwendete Name nicht ganz der Klassenaufgabe, da in der Klasse nicht der Dateipfad, sondern der in ein String umgewandelter Dateiinhalt aufgesplittet wird. Zudem muss das Wort *Splitt* ohne Doppelt-t geschrieben sein. *SplitText* wäre somit ein passender Name für die Klasse gewesen.

3.3.3. String-Parameter der SplittByItems-Methode

Als Parameter wird dieser Funktion ein String übergeben, der den gesamten Inhalt einer Tex-Datei enthält. Dabei ist die Parameter-Übergabe überflüssig, da der Gesamtstring bereits im Konstruktoraufruf in ein Attribut unter *textToRender* gespeichert wurde. Somit kann die Methode *SplittByItems* direkt auf die globale Variable zugreifen, ohne ein Parameter anzufordern.

3.3.4. Entfernen des Usernamen in der SplittTheBook Methode

Das erste Element des erstellten String-Zeilen-Arrays *devidedLines* wird in ein Leerstring umgewandelt. Der Grund dafür bezieht sich auf die Tex-Datei, auf die die Implementierung des Tex-Imports angepasst wurde. Dort gibt nämlich die erste Zeile den Usernamen eines Buches an. Da es im Model keine Property für einen Usernamen gibt, wird diese Zeile zunächst in ein Leerstring umgewandelt und nach der darauffolgenden If-Abfrage nicht mehr weiterverwendet.

3.3.5. Unspezifischer Property-Name in der Klasse PropertyFinder

Der Propertyname *Property* wurde unpassend gewählt. Dabei bezieht sich zwar der Name auf die Buch-Property, die in einer Buch-Item-Zeile ermittelt werden soll, jedoch ist *Property* ein schwammiger Name für die Benennung einer Klassenproperty. Es wurde leider bei der finalen Implementierung vergessen, diesen Namen umzuändern.

3.3.6. Mögliche Bucheigenschaften in PropertyFinder

Als Klassenattribut ist in PropertyFinder ein String-Array definiert, der die in einer Tex-Datei möglich auftretenden Bucheigenschaften beinhaltet. Auffällig ist dabei, dass alle Eigenschaften ausschließlich in Großbuchstaben geschrieben sind. Ausnahmsweise gibt es für die *Isbn*-Property sowohl eine Großbuchstaben- als auch eine Kleinbuchstaben-Schreibweise. Diese Festlegung bezieht sich auf die Import Datei *Bib Datei*, in der die Bucheigenschaften immer großgeschrieben sind. In dem Buch *Modellierung: Grundlagen und formale Methoden* ist die Isbn-Property hingegen kleingeschrieben. Deswegen besitzt der String-Array *categories* beide Schreibweisen dieser Property.