

Winterprüfung 2023

Ausbildungsberuf:

Fachinformatiker (VO 2020)

Fachrichtung:

Anwendungsentwicklung

Dokumentation der Projektarbeit

KeyValuePairLib

Klassenbibliothek zur Extraktion von Daten aus PDF-Dateien in .NET 7.0

Prüfling:

Patrick Hildebrandt

Hauptstr. 15 C, 58332 Schwelm

Prüflingsnummer:

19268

Ausbildungsbetrieb:

BFW Dortmund

Hacheneyer Str. 180, 44265 Dortmund

Kooperationsbetrieb:

D&TS GmbH

Wilhelmstraße 41-43, 58332 Schwelm



Inhaltsverzeichnis

Inhaltsverzeichnis	2
1 Ausgangssituation	4
1.1 Projektumfeld	4
1.2 Projektziel	4
1.2.1 IST-Analyse	5
1.2.2 SOLL-Konzept	6
1.2.3 Abweichungen zum Projektantrag	6
1.3 Prozess-Schnittstellen	7
1.3.1 Ansprechpartner	7
1.3.2 Einstieg und Ausstieg	7
2 Ressourcen- und Ablaufplanung	7
2.1 Personalplanung	7
2.2 Sachmittelplanung	8
2.2.1 Hardware	8
2.2.2 Software	8
2.3 Ablaufplanung	9
2.3.1 Projektphasen	9
2.4 Terminplanung	10
2.4.1 Kick-off-Meeting	10
2.4.2 Black-Box-Test	10
2.4.3 Projektabnahme	10
2.5 Kostenplanung	11
2.6 Amortisation	11
2.6.1 Faktoren der Amortisation	11
2.6.2 Szenario 1: Auswertung einer Serie durch Mitarbeiter	12
2.6.3 Szenario 2: Auswertung von drei Serien durch Mitarbeiter	13
2.6.4 Szenario 3: Auswertung einer Serie durch Dienstleister	13
2.6.5 Szenario 4: Auswertung von drei Serien durch Dienstleister	13
2.6.6 Kostenreduzierungspotential ab Break-even-Point	13
3 Durchführung und Auftragsbearbeitung	14
3.1 Vorgehensweise	14
3.2 Planungsphase	15
3.3 Entwurfsphase	16
3.3.1 Entwurf der Klasse KeyValuePairReader	16
3.3.2 Entwurf der Assoziationsklasse PdfFileData	18
3.3.3 Entwurf der Assoziationsklassen zu KeyValuePairPairs und Gruppen	19
3.3.4 Entwurf der Assoziationsklasse Config	19
3.3.5 Entwurf der dynamischen Methode ReadFile()	21
3.3.6 Entwurf der dynamischen Methode ExtractKeyValuePairPairs()	22
3.3.7 Entwurf der Methoden zu KeyValuePairPairs und Gruppen	23
3.3.8 Entwurf der Methode SetConfig()	23
3.3.9 Erstellen von Klassendiagrammen	24
3.3.10 Erstellen eines Klassendiagramms (verkürzt) zu Assoziationsbeziehungen	29
3.3.11 Erstellen eines Sequenzdiagramms zum Algorithmus der Datenextraktion	29
3.3.12 Erstellen eines Aktivitätsdiagramms zum Ablauf der Klassenfunktionalitäten	31
3.4 Implementierungsphase	32

3.4.1	Implementierung der Klasse KeyValuePairReader und der dynamischen Methode ExtractSearchKeyValue()	32
3.4.2	Implementierung der Assoziationsklasse PdfFileData	33
3.4.3	Implementierung der Assoziationsklasse und Methoden zu KeyValuePairs und Gruppen	34
3.4.4	Implementierung der Assoziationsklasse Config und Methode ReadConfig()	34
3.4.5	Implementierung der dynamischen Methode ReadFile().....	35
3.5	Testphase.....	35
3.5.1	Black-Box-Test.....	35
3.5.2	White-Box-Text	36
3.5.3	Begleitendes Debugging.....	36
4	Projektergebnis.....	38
4.1	SOLL-IST-Vergleich	38
4.2	Erklärung der Abweichungen.....	40
4.3	Fazit	41
5	Abschlussphase	41
5.1	Erstellung der Anwenderdokumentation.....	41
5.2	Abnahme des Projekts	42
5.3	Selbsterklärung.....	42
5.4	Datenschutz.....	42
6	Anhang.....	43
6.1	Abbildungsverzeichnis.....	43
6.2	Quellenverzeichnis	44
6.3	Selbsterklärung.....	44
6.4	Abnahmeprotokoll.....	44
6.5	Anwenderdokumentation.....	44
6.6	Große Abbildungen	44

1 Ausgangssituation

1.1 Projektumfeld

Der Kooperationsbetrieb für das Betriebspraktikum als Teil meiner Umschulung zum Fachinformatiker für Anwendungsentwicklung ist die D&TS GmbH in Schwelm. Hier bin ich in der Rolle eines Auszubildenden in der IT-Abteilung eingesetzt. Es wird ein agiles Vorgehensmodell mit Aspekten aus Scrum und Power-Programming und möglichst schlanker Bürokratie praktiziert. Der Auftraggeber für meine Projektarbeit ist ebenfalls die D&TS GmbH.

Die „D&TS GmbH - Dataservice & IT-Solutions“ ist tätig im Bereich des Stammdatenmanagements und der Klassifizierung von Materialien nach dem ECLASS-Standard. Unsere Schwerpunkte liegen somit im Bereich der Digitalisierung und Industrie 4.0. Unser Produktpotfolio umfasst hierbei Consulting, Dataservice & IT-Solutions zu den genannten Kernthemen. In diesen Bereichen hat das Unternehmen über 20 Jahre Erfahrung und zählt zu den führenden Anbietern.

Unser Hauptsitz ist in Schwelm. Darüber hinaus haben wir noch einen Standort in Lemberg (Lwiw, Ukraine) und unterhalten das EclassOffice für Spanien und Portugal. Unser Team setzt sich international zusammen aus Ingenieuren, Informatikern und Betriebswirten. Unsere Kunden sind große und mittelständische Unternehmen aus den Branchen Maschinen- und Anlagenbauer, Komponentenhersteller und produzierende Kunden. Dieser Kundenstamm verteilt sich auf aktuell vier Kontinente.

1.2 Projektziel

Es soll eine Klassenbibliothek (Dynamic Link Library) entwickelt werden, welche unabhängig von layout- und formatbasierten Abweichungen Wertschlüsselpaare (Key-Value-Pairs) aus PDF-Dateien extrahieren kann. Die Entscheidung für eine Klassenbibliothek begründet sich damit, dass diese mit geringem Aufwand vielseitig in unterschiedliche Umgebungen implementiert werden kann. Aus dem gleichen Grund sollen die extrahierten Daten als Objekt mit generischen Eigenschaften zurückgegeben werden, welche lediglich anhand der übergebenen Suchbegriff-Parameter angesprochen werden können. Hierbei ist anzumerken, dass die im PDF-Standard nutzbaren Formularfeld-Attribute nicht als vorhanden betrachtet werden können, wodurch ansonsten im Voraus strukturierte Daten vorliegen würden.

1.2.1 IST-Analyse

Die D&TS GmbH besitzt ein hauseigenes Ticketsystem, welches Grunddaten von Materialien des Kunden innerhalb eines Tickets speichert. Dieses Ticket wird dann durch Prozesse automatisch verarbeitet oder ggf. an die zuständigen Sachbearbeiter weitergeleitet, sodass eine Klassifizierung der angefragten Materialien schnellstmöglich erfolgen kann. Da Kunden üblicherweise schon bestehende Prozesse verwenden, um Materialien in Ihrem ERP-System anzulegen, wird für fast jeden Kunden ein individuell angepasstes Skript benötigt, welches die Daten im Format des Kunden entgegen nimmt und für die Systeme der D&TS GmbH aufbereitet bzw. weiterleitet.

Bei einem Kunden werden die Daten beispielsweise als PDF-Dokumente via E-Mail an das Ticketsystem gesendet, wozu ebenfalls ein individuell auf die Dokumente des Kunden angepasstes Skript entwickelt wurde. Veränderungen an Dokumenten seitens des Kunden erfordern somit auch einen Entwicklungsaufwand auf Seiten der D&TS GmbH. Hieraus resultiert für die D&TS GmbH ein hoher Entwicklungsaufwand für jeden neuen Ticketsystem-Kunden und jede Änderung an Dokumenten auf Kundenseite.

Für den oben genannten Kunden wurde von mir in der Vergangenheit bereits eine Klassenbibliothek entwickelt, welche spezifisch an den Kunden angepasst PDF-Dateien ausliest und an die E-Mail-Connector-Schnittstelle des Ticketsystems übergibt. Nachdem der Kunde im darauffolgendem Verlauf Änderungen an der Struktur seiner Dokumente vornahm, stellte sich heraus, dass der bisherige Algorithmus zur Extraktion der Wertschlüsselpaare nicht mehr in der Lage war, die Daten auszulesen.

Zudem war die Ursache hierfür so tief im Grundprinzip des Algorithmus verortet, dass auch keine Anpassung auf diese Änderungen umgesetzt werden konnte. Dieser Umstand lag darin begründet, dass die Änderungen des Kunden den Dokumentenfluss im Fließtext der PDF-Dateien ab der Stelle der ersten Änderung umstellten und neue Inhalte oder bisherige Werte ohne auswertbare, trennende Informationen ans Ende vom Dokumentenfluss anhängten. Es war also ein komplett neuer Lösungsansatz für eine vermeintlich kleine Änderung notwendig geworden.

Ich entwickelte hierfür im Elverfahren einen spezifisch an den Kunden angepassten Algorithmus, welcher positionsbasiert die Wertschlüsselpaare aus den Datenblättern auslesen konnte. Dieser war somit auch nicht mehr anfällig gegenüber Änderungen oder Variationen in der Dokumentenstruktur.

Das hierbei entstandene Skript umfasste rund 600 Zeilen Quellcode. Aufgrund dieses unerwarteten Mehraufwands für diese Anpassung wurde erkannt, dass für zukünftige Anwendungen und Anpassungen eine selbstkonfigurierende Klassenbibliothek benötigt wird. Zudem wurde das Potential in dem neuen, positionsbasierten Lösungsansatz als eine universelle Auslesemethode erkannt.

Die Kernaufgabe meiner Projektarbeit wird somit sein, diesen Algorithmus in eine stark typisierte und objektorientierte Klassenbibliothek zu refaktorieren, welche ohne ein tieferes Verständnis der Funktionsweise von einem anderen Anwender genutzt werden kann und zudem unabhängig von der Ausgangslage der Kunden-PDFs funktioniert. Ferner ist es hierzu angedacht, dass zukünftig keine quellcodebasierten Anpassungen mehr notwendig sind und die Kundenmanager mit Parametern über Eingabemasken selbstständig die Felder zur Verarbeitung bestimmen können. Dies ist jedoch nicht Gegenstand der Projektarbeit und ist als mögliche Weiterentwicklung zu verstehen.

1.2.2 SOLL-Konzept

Es soll eine Klassenbibliothek in C# basierend auf .NET 7.0 entwickelt werden, um möglichst hohe Kompatibilität für unterschiedliche Einsatzumgebungen zu erreichen. Diese Klassenbibliothek soll eine instanziierbare Klasse bereitstellen, welche bei der Deklaration oder Initialisierung den Pfad einer PDF-Datei entgegennimmt. Weiterhin sollen Parameter übergeben werden, welche die Keys der gesuchten Key-Value-Pairs darstellen. Um die vertikale oder horizontale Ausrichtung der Key-Values-Pairs zu berücksichtigen, muss der Anwender einen booleschen Wert zum jeweiligen Parameter angeben. Gruppen von sich wiederholenden Suchschlüsseln sollen in einer Unterklassie gesammelt erfasst werden, welche als zusätzliche Parameter definiert werden müssen. Weitere Attribute der Klasse sollen der Feinjustierung des Algorithmus auf Abweichungen wie zu ignorierende Felder, Mindestabstände oder Separierungszeichen dienen. Die zurückgegebene Instanz soll alle Key-Value-Pairs der PDF als generische Eigenschaften bereitstellen. Weiterhin soll eine Konfigurationsklasse implementiert werden, welche entsprechende Parameter aus einer CFG-Datei im JSON-Format abrufen kann bzw. den jeweiligen Use-Case festsetzt.

1.2.3 Abweichungen zum Projektantrag

Im Projektantrag wurde fälschlicherweise der Begriff Subklasse in der Zeitplanung verwendet. Tatsächlich war hiermit der Begriff der Assoziationsklasse gemeint. Weiterhin wurde der Begriff Klassenstrukturdiagramm von mir verwendet, welcher irreführend sein könnte. Hiermit war ein verkürz-

tes Klassendiagramm mit Assoziationsbeziehungen gemeint. Diese Abweichungen wurden von mir erst während der Dokumentationserstellung entdeckt und zudem als geringfügig erachtet.

1.3 Prozess-Schnittstellen

1.3.1 Ansprechpartner

Als Ansprechpartner für die Projektarbeit dienen mir zum Ersten der Abteilungsleiter der IT-Abteilung und zum Zweiten ein weiterer Fachinformatiker aus der IT-Abteilung, welcher auch zuständig für das Ticket-System und mit meinen bisherigen Arbeiten vertraut ist. Weiterhin kann ich auch fachbezogene Informationen vom Abteilungsleiter des Projektmanagements (Betriebswirt) und unserem Geschäftsführer (Ingenieur) beziehen.

1.3.2 Einstieg und Ausstieg

Der Einstieg und Ausstieg für meine Projektarbeit lassen sich in diesem Fall sehr klar abgrenzen, da es sich um eine Klassenbibliothek handelt, welche wiederverwendbar sein soll. Der Anwender soll die Klassenbibliothek instanziieren können, ihr eine Konfiguration übergeben, die Anweisung zum Auslesen erteilen und anschließend auf die ausgelesenen Wertschlüsselpaare als Eigenschaften zugreifen können. Über genau diese Funktionalität und deren Dokumentation erstreckt sich mein Verantwortungsbereich für das Projekt.

2 Ressourcen- und Ablaufplanung

2.1 Personalplanung

Für die Projektplanung und Durchführung sind folgende, anonymisierte Mitarbeiter vorgesehen:

Mitarbeiter (anonym)	Aufgabe	Umfang
Projektbeauftragter	Projektdurchführung	68 h
IT-Abteilungsleiter	Kick-off-Meeting	2 h
IT-Mitarbeiter	Kick-off-Meeting	2 h
IT-Mitarbeiter	Black-Box-Test	2 h
IT-Abteilungsleiter	Projektabnahme	1 h
Projekt-Mitarbeiter 1	Black-Box-Test	2 h
Projekt-Mitarbeiter 2	Black-Box-Test	2 h
Projekt-Mitarbeiter 3	Black-Box-Test	2 h
Gesamt		81 h

2.2 Sachmittelplanung

Für die Durchführung des Projektes werden folgend noch aufgelistete Sachmittel benötigt. Bei diesen Sachmitteln handelt es sich um meine vorhandene Arbeitsplatzausstattung, sodass keine zusätzlichen Anschaffungen bzw. Kosten erforderlich sind. Aus vorangegangenen Projekten sind zudem ausreichend PDF-Datenblätter mit variierender Struktur vorhanden.

2.2.1 Hardware

- ein Laptop "ThinkPad E15 G2 20T8000TGE" von Lenovo
- ein Laptop-Ständer
- eine Dockingstation von Lenovo
- zwei 31 Zoll Monitore
- eine „Membranical“-Tastatur "ROC-12-300-BK" von Roccat
- eine Mouse "Skiller SGM1" von Sharkoon

2.2.2 Software

- das OS Windows 11 Pro
- die IDE Visual Studio Code
- die IDE Visual Studio 2022
- die „Desktop“-App und Addons von GitHub
- die App Notepad++
- die App Teams
- die App Office
- die App „Software Ideas Modeler“
- diverse PDF-Datenblätter mit typografischen Variationen

2.3 Ablaufplanung

2.3.1 Projektphasen

Die Projektphasen werden entsprechend den nachfolgend aufgelisteten Teilschritten samt Zeitprognosen durchgeführt, wobei hier bereits die Abweichungen bis zum Ende der Planungsphase eingeflossen sind, sodass 2 Stunden Puffer entstanden sind:

Anforderungsphase	4 h
Kick-off-Meeting	1 h
Prüfung SOLL-Konzept	1 h
Aufgabenanalyse	2 h

Planungsphase	4 h
Zeitplanung	1 h
Personalplanung	1 h
Kosten- und Amortisationsplanung	2 h

Entwurfsphase	17 h
Entwurf der Klasse KeyValuePairReader	2 h
Entwurf der Assoziationsklasse PdfFileData	1 h
Entwurf der Assoziationsklassen zu KeyValuePairs und Gruppen	1 h
Entwurf der Assoziationsklasse Config	1 h
Entwurf der dynamischen Methode ReadFile()	2 h
Entwurf der dynamischen Methode ExtractKeyValuePairs()	2 h
Entwurf der Methoden zu KeyValuePairs und Gruppen	1 h
Entwurf der Methode SetConfig()	1 h
Erstellen von Klassendiagrammen	1 h
Erstellen eines Klassendiagramms (verkürzt) zu Assoziationsbeziehungen	1 h
Erstellen eines Sequenzdiagramms zum Algorithmus der Datenextraktion	2 h
Erstellen eines Aktivitätsdiagramms zum Ablauf der Klassenfunktionalitäten	2 h

Implementierungsphase	26 h
Implementierung der Klasse KeyValuePairReader	4 h
Implementierung der Assoziationsklasse PdfFileData	2 h
Implementierung der Assoziationsklassen zu KeyValuePairs und Gruppen	3 h
Implementierung der Assoziationsklasse Config	2 h
Implementierung der dynamischen Methode ReadFile()	4 h
Implementierung der dynamischen Methode ExtractKeyValuePairs()	4 h
Implementierung der Methoden zu KeyValuePairs und Gruppen	3 h
Implementierung der Methode SetConfig()	4 h

Testphase	8 h
Black-Box-Test	2 h
White-Box-Test	2 h
Begleitendes Debugging	4 h

Projektergebnis	4 h
Projektabnahme	1 h
SOLL-IST-Vergleich	2 h
Erklärung der Abweichungen	1 h

Abschlussphase	15 h
Dokumentation Klassenbibliothek für administrative Zwecke	2 h
Release-Erstellung GitHub Repository	1 h
Projektdokumentation	12 h

Summe Projektphasen (Projektantrag: 80 h)	78 h
--	-------------

2.4 Terminplanung

2.4.1 Kick-off-Meeting

Für das Kick-off-Meeting wurde ein Termin über zwei Stunden am 18.09.2023 um 09:00 Uhr in Outlook eingestellt. Teilnehmer werden der Abteilungsleiter der IT-Abteilung, der oben genannte Mitarbeiter der IT-Abteilung und meine Person sein.

2.4.2 Black-Box-Test

Weiterhin wurde ein Black-Box-Test für den 26.09.2023 geplant. Dieser wird vom Mitarbeiter der IT-Abteilung und mir ausgerichtet. Zudem sind drei weitere Projekt-Mitarbeiter vorgesehen, welche anschließend instruiert werden und zeitnah entsprechend ihrem jeweiligen Workload den Test durchführen. Für jeden Teilnehmer sind jeweils zwei Stunden vorgesehen.

2.4.3 Projektabnahme

Zuletzt wurde für die Projektabnahme ein Termin über eine Stunde am 27.09.2023 um 15:00 Uhr in Outlook eingestellt. Die Abnahme wird vom Abteilungsleiter der IT-Abteilung und mir durchgeführt.

2.5 Kostenplanung

Der Kostenplanung zugrunde liegt der sogenannte „durchschnittliche interne Verrechnungssatz“, welcher mir vom Betrieb für diesen Zweck genannt wurde. Dieser beträgt 50,00 € pro Stunde und wird nachfolgend mit d. i. V. abgekürzt. In diesem werden alle betrieblichen Kosten zwecks Vereinfachung von Kalkulationen harmonisiert. Entsprechend der Personalplanung unter Punkt 2.1 wird dieser mit 81 Stunden multipliziert:

Projektkosten
Personaleinsatz in Stunden * durchschnittlicher interner Verrechnungssatz
81 h * 50,00 €/h = 4.050,00 €

2.6 Amortisation

Da es sich beim Auslesen von PDF-Datenblättern zwar um eine wiederkehrende Problematik handelt, diese an sich jedoch keinen täglichen Routineprozess darstellt, kann eine Amortisation nur im Rahmen von Projekten bzw. Kundenaufträgen erfolgen. Es lässt sich folglich keine lineare Kostenreduzierung für ein Geschäftsjahr oder einen festgelegten Zeitraum ermitteln. Um eine Vergleichbarkeit zu schaffen, müssen weitere Faktoren in die Betrachtung aufgenommen werden, welche ich zuvor unter Punkt 2.6.1 erläutern werde.

2.6.1 Faktoren der Amortisation

Abgesehen vom Auslesen durch einen Algorithmus, welcher zuvor entwickelt werden muss, können PDF-Datenblätter entweder händisch von einem Mitarbeiter per Copy-Paste-Methode in Formularfeldern erfasst oder in großen Serien von einem Dienstleister in Asien ausgewertet werden. Die Auswertung durch den Dienstleister stellt die kostengünstigere Variante dar und wird auf große Serien angewendet. Dieser Prozess hat jedoch eine hohe Fehleranfälligkeit, da mit automatisierten Tools Tabellen erzeugt bzw. Spalten nebeneinandergelegt werden, bei denen in großen Datensätzen keine ausreichende Detailprüfung möglich ist und Varianz in den Datenblättern zu Datenschwund führen kann. Bei einer händischen Erfassung durch Mitarbeiter ist die Detailprüfung zwar deutlich besser, aber die Kosten für diesen Arbeitseinsatz fallen natürlich auch entsprechend höher aus.

Weil Datenblätter in ihrer Datendichte, Aufbau und Länge enorm unterschiedlich sein können, ist es hierfür auch nicht zweckmäßig, einen projektunabhängigen Durchschnittswert für die Bearbeitungszeit anzunehmen. In den nachfolgenden Kalkulationen gehe ich deshalb einmal von einer ungünsti-

gen Bearbeitungszeit von 20 Seiten pro Stunde und weiterhin von einer sehr effizienten Bearbeitungszeit von 50 Seiten pro Stunde aus. Die Kosten für die Auswertung durch den Dienstleister wurden mir vom Betrieb mit einem Durchschnitt von 1,00 € pro PDF-Datei beziffert, wobei hierzu noch die variierende Anzahl an Seiten je Datei berücksichtigt werden muss. Um dies abzubilden, werde ich beispielhaft mit Dezimalwerten unter einem Euro kalkulieren.

Die Klassifizierung von Materialien erfolgt in der Regel in Projekten mit Serien von ca. 500 bis 5000 Materialien bzw. PDF-Dateien, welche üblicherweise eine Waren- oder Teilegruppe darstellen. Wie die nachfolgenden Amortisationsszenarien verdeutlichen, ist eine Amortisation bereits im Rahmen eines Projektes bzw. einer Serie durchaus plausibel. Da die Klassenbibliothek den Personaleinsatz projektunabhängig auf durchschnittlich eine Stunde für die Erstellung einer Konfiguration reduziert, muss diese Stunde folglich als zusätzliche Folgekosten berücksichtigt werden. Für die Amortisation muss somit entweder der Schwellenwert für den Break-even-Point mit einer Konfiguration bzw. Serie erreicht werden oder die Kosten für die Erstellung einer Konfiguration müssen mit der Anzahl an Projekten multipliziert werden, bis sich die Amortisation akkumuliert hat.

Sichten Sie hierzu bitte die vier nachfolgend bereitgestellten Szenarien einer Amortisation in Tabellenform. Die konkreten Werte entnehmen Sie bitte den jeweiligen Tabellen. Da die Faktoren der Kalkulation in diesem Abschnitt ausgiebig erläutert wurden, verzichte ich folgend auf doppelte Erläuterungen.

2.6.2 Szenario 1: Auswertung einer Serie durch Mitarbeiter

Hierbei handelt es sich um eine Betrachtung der Amortisation anhand der notwendigen Arbeitszeit. Zwecks Vergleichbarkeit mit der Auswertung durch Dienstleister ergänze ich hierzu eine Beispielrechnung mit der entsprechenden Anzahl an Datenblättern.

Amortisationsdauer
(Projektkosten + Personaleinsatz für Konfigurationserstellung * d. i. V.) / d. i. V. (4.050,00 € + 1 h * 50,00 €/h) / 50,00 €/h = 82 h

Beispiel: Anzahl Datenblätter
Datenblätter pro Stunde * Amortisationsdauer 20 S./h * 82 h = 1.640 S.

2.6.3 Szenario 2: Auswertung von drei Serien durch Mitarbeiter

Wie in Szenario eins.

Amortisationsdauer

$$\text{(Projektkosten + Personaleinsatz für Konfigurationserstellung * d. i. V.) / d. i. V.}$$

$$(4.050,00 \text{ €} + 3 \text{ h} * 50,00 \text{ €/h}) / 50,00 \text{ €/h} = 84 \text{ h}$$

Beispiel: Anzahl Datenblätter

$$\text{Datenblätter pro Stunde * tatsächliche Amortisationsdauer}$$

$$50 \text{ S./h} * 84 \text{ h} = 4.200 \text{ S.}$$

2.6.4 Szenario 3: Auswertung einer Serie durch Dienstleister

In dieser Betrachtung wird ermittelt, ab welcher Anzahl an Datenblättern eine Amortisation erfolgt.

Amortisation ab Anzahl Datenblätter

$$\text{(Projektkosten + Personaleinsatz für Konfigurationserstellung * d. i. V.) / Kosten pro Datenblatt}$$

$$(4.050,00 \text{ €} + 1 \text{ h} * 50,00 \text{ €/h}) / 0,50 \text{ €/S.} = 8.200 \text{ S.}$$

2.6.5 Szenario 4: Auswertung von drei Serien durch Dienstleister

Wie in Szenario drei.

Amortisation ab Anzahl Datenblätter

$$\text{(Projektkosten + Personaleinsatz für Konfigurationserstellung * d. i. V.) / Kosten pro Datenblatt}$$

$$(4.050,00 \text{ €} + 3 \text{ h} * 50,00 \text{ €/h}) / 0,70 \text{ €/S.} = 6.000 \text{ S.}$$

2.6.6 Kostenreduzierungspotential ab Break-even-Point

Da sich die Kostenreduzierung durch den Einsatz der Klassenbibliothek nicht linear zu den vorherigen Kosten verhält und zudem abhängig von Variablen ist, lässt sich das Kostenreduzierungspotential am besten mit den folgenden Gleichungen darstellen:

$$\text{Kosten}(x) = \text{d. i. V.}$$

$$\Delta \text{ Kosten (Personaleinsatz in Stunden)} = \text{d. i. V.} * (\text{Personaleinsatz in Stunden} - 1)$$

$$\Delta \text{ Kosten (Anzahl Datenblätter)} = \text{Kosten pro Datenblatt} * \text{Anzahl Datenblätter} - \text{d. i. V.}$$

Wie sich anhand der Gleichungen ableiten lässt, ist das Kostenreduzierungspotential der Klassenbibliothek enorm hoch und umso effektiver, je größer der zu vergleichende Auftrag geartet ist. Dies erklärt sich anhand der Tatsache, dass jedweder personelle Aufwand um einen variablen Faktor auf

einen durchschnittlichen Personaleinsatz von ca. einer Stunde für die Erstellung einer Konfiguration reduziert werden kann.

3 Durchführung und Auftragsbearbeitung

3.1 Vorgehensweise

Die Zeitprognose für das Kick-Off-Meeting erwies sich in der Praxis als zu großzügig. Aufgrund der Planung im Projektantrag waren alle Beteiligten vollumfänglich im Bilde, sodass lediglich die übergeordnete Fragestellung geklärt wurde, ob Abweichungen vorliegen oder notwendig sind, ehe ich beginnen konnte. Der Zeitaufwand reduzierte sich somit um eine Stunde.

Nach dem Kick-Off-Meeting fand zunächst eine erneute Prüfung des SOLL-Konzeptes statt. Hierzu sichtete ich den Quelltext vom oben beschriebenen Vorläufer-Projekt, um eine Eingrenzung vorzunehmen, welche Teile des existierenden Algorithmus in Skriptform für eine Refaktorisierung geeignet waren. Ich entschied mich dazu, den Algorithmus nur parallel als Leitfaden zu verwenden und lediglich das Wording von einigen Variablen und Methoden zu übernehmen, welche ich als bezeichnend empfand. Hierzu einige Screenshots des ca. 600 Zeilen umfassenden, ursprünglichen Algorithmus in Skriptform, welche Ihnen lediglich einen Eindruck des Umfangs vermitteln sollen:

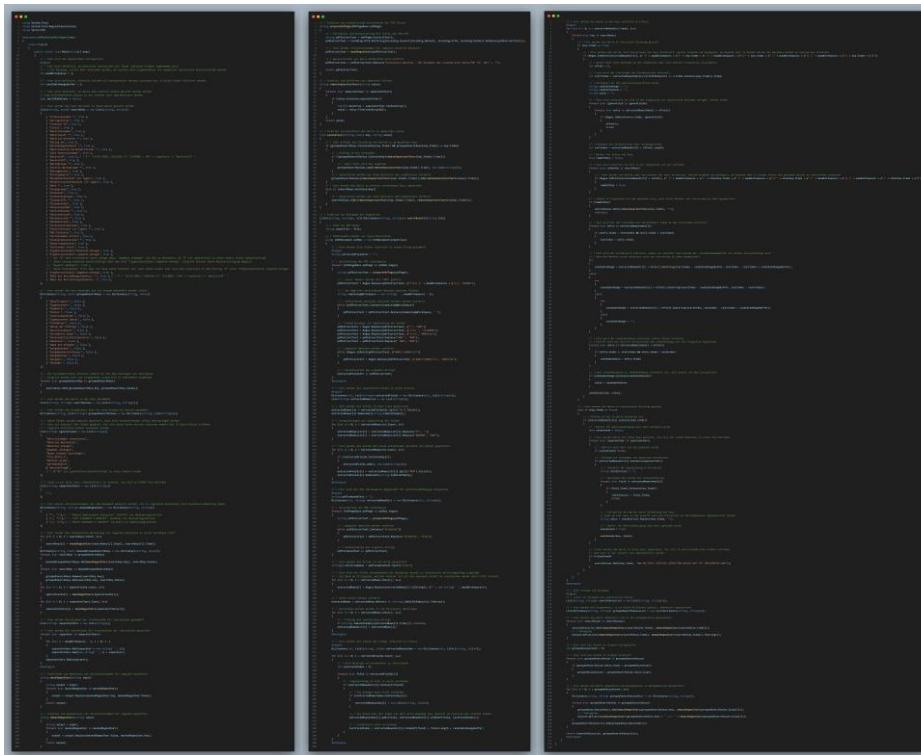
The image consists of three vertically stacked screenshots of a terminal window. Each screenshot displays a large amount of text, which appears to be a script or algorithm written in a programming language like Python. The text is dense and contains many lines of code, illustrating the complexity of the original algorithm. The terminal interface includes a dark background with white text, standard file navigation commands at the bottom, and a scroll bar on the left side of the window.

Abb. 1: Ursprünglicher Algorithmus

Da ich bereits mit der Funktionsweise der Klassenbibliothek „Spire.PDF“ vertraut war, welche in dem Vorläufer-Projekt verwendet wurde, und ich mir der Funktionalität des zugrundeliegenden Lösungsansatzes sicher sein konnte, waren an dieser Stelle keine weiteren Recherchen oder Entscheidungen notwendig. Der Zeitaufwand verkürzte sich hierfür ebenfalls um eine Stunde.

Hieran anschließend fand die Aufgabenanalyse statt. Ich erfasste die notwendigen Arbeitsschritte von mir und beteiligten Personen und verglich diese Ergebnisse mit der Projektplanung aus dem Projektantrag. Da ich keine notwendigen Änderungen zum Projektantrag identifizieren konnte, wurde dessen Zeitplanung unverändert übernommen. Zusätzliche Mikraufgaben, welche ich identifizierte, ließen sich eindeutig zu den Teilschritten aus der Projektplanung zuordnen und wurden somit nur in Form von To-Do-Listen festgehalten. Als Beispiele hierzu wären die Assoziationsklassen SearchKey oder RegexWrappedCollection zu nennen, welche als Notwendigkeit aus der Klasse Config hervorgingen, um eine starke Typisierung umzusetzen.

Um der Komplexität des Projektes zu entsprechen und möglichst gute Ergebnisse erzielen zu können, war es generell unerlässlich auf Hilfsmittel wie Excel-Tabellen oder OneNote-Notizblättern zurückzugreifen, um die Fülle an Informationen und Mikraufgaben strukturiert zu erfassen und bewältigen zu können. Zudem wurde nach Abschluss der Planungsphase die Ablaufplanung als strikter Leitfaden genutzt um den SMART-Prinzipien (Spezifisch, Messbar, Attraktiv, Realistisch und Termintisiert) zu entsprechen, soweit dies im Rahmen von Softwareentwicklung möglich ist. Meiner bisherigen persönlichen Erfahrung nach ist es bei derart komplexen und zusammenhängenden Konstrukten in der objekt-orientierten Programmierung kaum möglich, einzelne Klassen komplett abzuschließen, ehe man die nächste beginnt bzw. ohne Änderungen an zuvor erstellten Klassen fortzufahren.

3.2 Planungsphase

In der Planungsphase wurde der Ablauf des Projektes wie unter Punkt zwei dargestellt ausgearbeitet und die Kosten des Projektes bzw. die Amortisationsdauer kalkuliert. Hierzu wurden die notwendigen Personen kontaktiert, Termine in Outlook eingestellt und notwendige Rahmenkennzahlen bzw. Informationen eingeholt, welche für die Planung und Dokumentation erforderlich waren, wie beispielsweise der Begriff des „durchschnittlichen internen Verrechnungssatzes“ aus der Kostenplanung. Diese Informationen und Kalkulationen wurden aus Zeitgründen zunächst in formfreien Excel-Tabellen oder OneNote-Notizblättern erfasst bzw. kalkuliert, da es bereits absehbar war, dass diese in der endgültigen Dokumentation in angepasster Form mit, dem Layout entsprechender, sauberer

Strukturierung wiederverwendet werden sollten. Dies erleichterte auch die Erfassung von Abweichungen, welche somit ohne Formvorgaben unkompliziert und schnell festgehalten werden konnten, ohne Informationen zu verlieren. Aufgrund der Vorleistung im Projektantrag fiel der Zeitaufwand für die Zeitplanung um eine Stunde kürzer aus. Dieser Zeitgewinn wurde in eine detaillierte Ausarbeitung von Amortisationsszenarien umgeleitet.

3.3 Entwurfsphase

In der Entwurfsphase erstellte ich zunächst Rohfassungen der benötigten Klassen und Methoden in Form von UML-Klassendiagrammen (Unified Modelling Language) und einem Klassenstrukturdigramm. Weiterhin erstellte ich ein Sequenzdiagramm um den Algorithmus der Datenextraktion vereinfacht darzustellen und ein Aktivitätsdiagramms zum Ablauf der Klassenfunktionalitäten in Bezug derer Verwendung. Diese wurden im weiteren Verlauf sukzessiv um die konkreten Implementierungsschritte und Beziehungen erweitert und im Anschluss für die Projektdokumentation detailliert ausgearbeitet und strukturiert. Die Ergebnisse dieser Prozesse stelle ich zwecks Übersichtlichkeit und Konsistenz in der Reihenfolge des Projektantrages zusammen mit einer kurzen Erläuterung in den folgenden Unterpunkten vor. Hierbei stelle ich diejenigen Klassendiagramme, deren Lesbarkeit zu stark unter der Einbindung in eine Din A4-Seite leidet, zusätzlich im Anhang bereit.

3.3.1 Entwurf der Klasse KeyValuePairReader

Dies stellt die Oberklasse des Projektes dar, welche vom Anwender instanziert und angesprochen werden kann. In ihr laufen alle anderen Assoziationsklassen des Projektes in Form von Kompositionsbereihungen zusammen. Der Anwender übergibt ihr den Pfad der Konfigurationsdatei und PDF-Datei zum Auslesen und erhält nach Verarbeitung durch den internen Algorithmus eine Instanzierung mit generischen Eigenschaften, welche die Wert-Schlüssel-Paare der PDF-Datei abbilden. Weiterhin wurden während der Modellierung bereits zusätzliche Dateitypen über den Typ PDF hinaus berücksichtigt in Form einer Enumeration. Dies sollte auch die Formulierung „Entwurf der dynamischen Methode ReadFile()“ im Projektantrag zum Ausdruck bringen. Da PDF-Dateien bei der Verarbeitung in eine tabellarische Form überführt werden, sah ich hierin ein Potential für zukünftige Erweiterungen auf die Dateitypen CSV, XLSX und XML durch eine verhältnismäßig einfache Adaption des Extraktionsalgorithmus. Dieser Umstand begründete auch das Wording der Assoziationsklasse PdfFileData, welche nur einen möglichen Typ für die dynamische Auflösung des Attributs „_fileData: Dynamic“ darstellt. Das UML-Klassendiagramm werde ich zwecks Lesbarkeit zusätzlich im Anhang

bereitstellen. Dies mache ich folgend immer mit „(Diagramm in Anhang enthalten)“ am Ende eines Absatzes kenntlich.

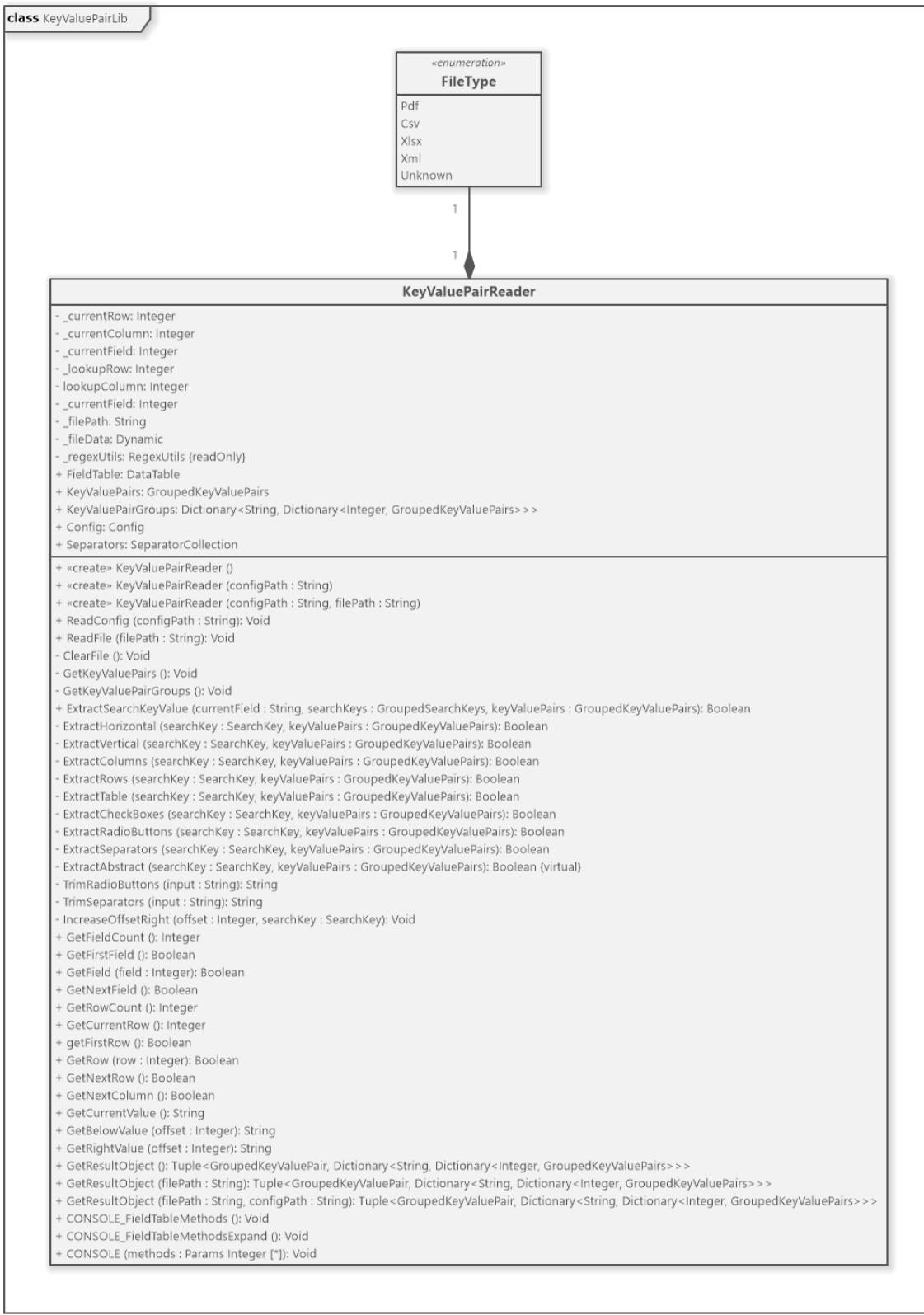


Abb. 2: UML-Klassendiagramm - KeyValuePairReader

3.3.2 Entwurf der Assoziationsklasse PdfFileData

Mit dieser Klasse wird der Prozess des Auslesens von PDF-Dateien mithilfe der Klassenbibliothek Spire.PDF umgesetzt. In dieser Klassenbibliothek liegt auch der Ursprung für meinen Lösungsansatz zu einer Format- und Layout-unabhängigen Auswertung begründet. Übliche Klassenbibliotheken, wie z.B. die populäre iTextSharp-Bibliothek, verfolgen typischerweise zwei Ansätze. Entweder lesen sie Dokumente anhand ihres Fließtextes aus der PDF-Struktur aus, wobei aufgrund der Formatierungsmöglichkeiten in PDF-Dateien mitunter stark frakturierte Ergebnisse möglich sind. Oder sie basieren auf einem OCR-Prinzip (Optical Text Recognition) mit Koordinatensystem, bei dem es zu Fehlern in der Erkennung von korrekten Zeichen kommen kann. Beispiele hierfür wären die Verwechslung vom „g“ mit der „9“ oder „o“ mit der „0“. Zudem ist die Methode zur Erfassung von bestimmten Abschnitten eines Dokumentes mithilfe des Koordinatensystems stark anfällig gegenüber Veränderungen in der Dokumentenstruktur. Die Spire.PDF-Bibliothek liest PDF-Dateien hingegen Fließtext-basiert aus und bildet die Positionen von Textteilen mithilfe von Leerzeichen als Abstandhaltern proportional ab. Diese Eigenschaft habe ich mir in meinem Algorithmus zunutze gemacht, um die Struktur von PDF-Dokumenten als eine zweidimensionale Tabelle zu interpretieren, wie ich es unter Punkt 3.4 noch erläutern werde. (Diagramm in Anhang enthalten)

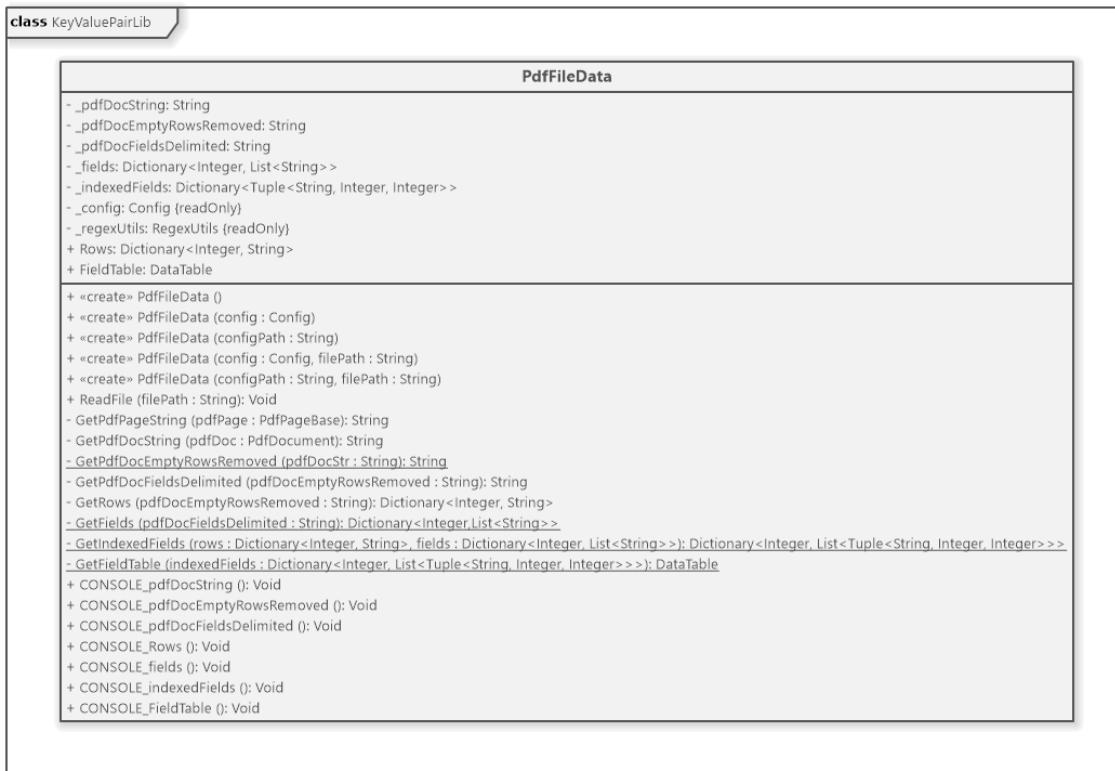


Abb. 3: UML-Klassendiagramm - PdfFileData

3.3.3 Entwurf der Assoziationsklassen zu KeyValuePairPairs und Gruppen

In dieser Klasse werden die Ergebnisse der Extraktion von Wert-Schlüssel-Paaren gruppiert und durch Erzeugung von generischen Eigenschaften gespeichert. Während der Ausarbeitung des Konzeptes kam ich zu dem Schluss, dass für diese Problematik nicht mehrere Assoziationsklassen erforderlich waren, wie die Formulierung im Projektantrag und der Ablaufplanung andeuten sollte. Durch Verwendung einer Basisklasse als Typ in einem Dictionary konnte der benötigte Sachverhalt ebenfalls zutreffend abgebildet werden.

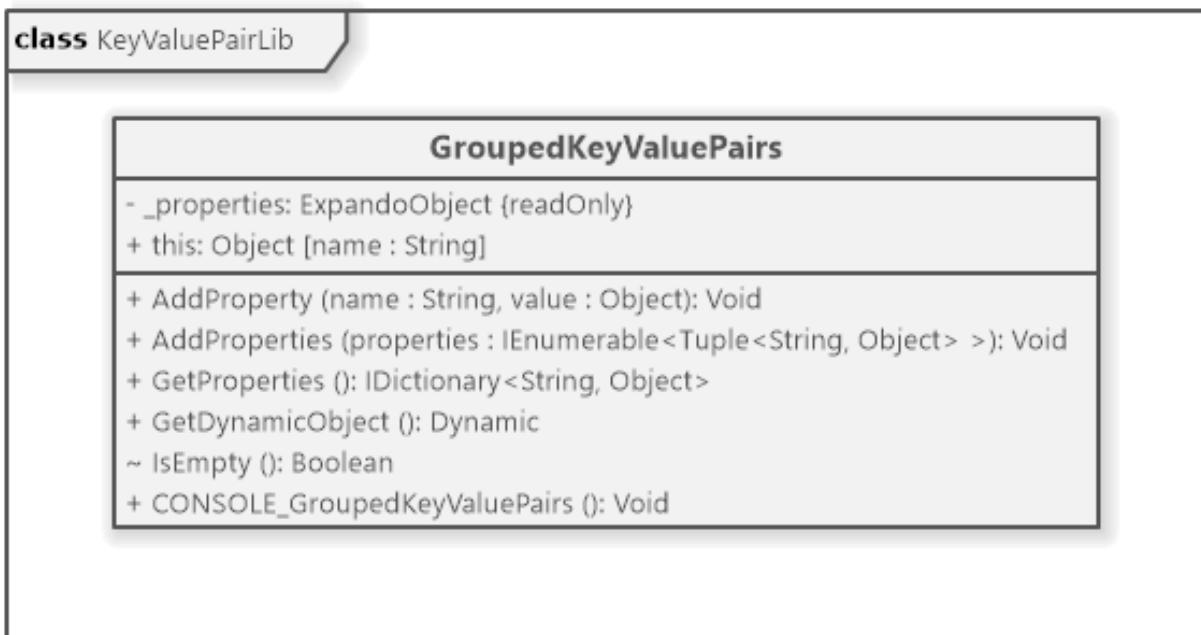


Abb. 4: UML-Klassendiagramm - `GroupedKeyValuePairPairs`

3.3.4 Entwurf der Assoziationsklasse Config

Mit dieser Klasse werden die Parameter zum Auslesen von und Extrahieren aus PDF-Dateien abgebildet. In ihr lässt sich auch noch am deutlichsten der Bezug zum ursprünglichen Algorithmus erkennen. Es wurde bereits während der Modellierung darauf geachtet, lediglich öffentliche Eigenschaften anstatt Felder zu verwenden, um eine Steuerung durch übergeordnete Projekte und einfache Refaktorisierung zu ermöglichen.

Für diejenigen Leser, welche nicht mit C# vertraut sind, halte ich an dieser Stelle einen kleinen Exkurs in Bezug auf meine letzte Ausführung für sinnvoll:

In C# werden Attribute in zwei Kategorien differenziert. Zum einen in sogenannte Felder, welche den klassischen Attributen mit typischerweise, jedoch nicht zwingender, privater Sichtbarkeit entsprechen. Zum anderen in Eigenschaften, welche der gleichen Konvention nach i.d.R. öffentliche Sichtbarkeit aufweisen und zudem integrierte Getter und Setter besitzen.

```
class KeyValuePairLib
```

Config

- + MaxWhiteSpaces: Integer
 - + RangeBuffer: Integer
 - + MinSeparatorSpaces: Integer
 - + MaxSeparatorSpaces: Integer
 - + CutTop: Integer
 - + CutBottom: Integer
 - + CutLeft: Integer
 - + CutRight: Integer
 - + AddEmptyProperties: Boolean
 - + SearchKeys: GroupedSearchKeys
 - + SearchKeyGroups: Dictionary<String, GroupedSearchKeys>
 - + IgnoreFields: RegexWrappedCollection
 - + IgnoreRows: RegexWrappedCollection
 - + BreakFields: RegexWrappedCollection
 - + BreakRows: RegexWrappedCollection
 - + CheckBoxes: RegexWrappedCollection
 - + RadioButtons: RegexWrappedCollection
 - + Separators: HashSet<String>
- + «create» Config ()
 - + «create» Config (filePath : String)
 - + ReadConfigFile (filePath : String): Void
 - + CONSOLE_Config (): Void

Abb. 5: UML-Klassendiagramm - Config

3.3.5 Entwurf der dynamischen Methode ReadFile()

Den dynamischen Aspekt dieser Methode habe ich bereits unter Punkt 3.3.1 vorweggegriffen. Mit dieser Methode wird eine Instanz der Klasse PdfFileData erzeugt, welche die Daten aus den PDF-Dateien aufbereitet und für die Extraktionsmethoden GetKeyValuePairs() und GetKeyValuePairGroups() bereit stellt. Diese Methoden werden ebenfalls von ReadFile() nach der besagten Instanziierung aufgerufen. Die nächsten vier abgebildeten Diagramme wurden dem noch folgenden Aktivitätsdiagramm zum Ablauf der Klassenfunktionalitäten entliehen.

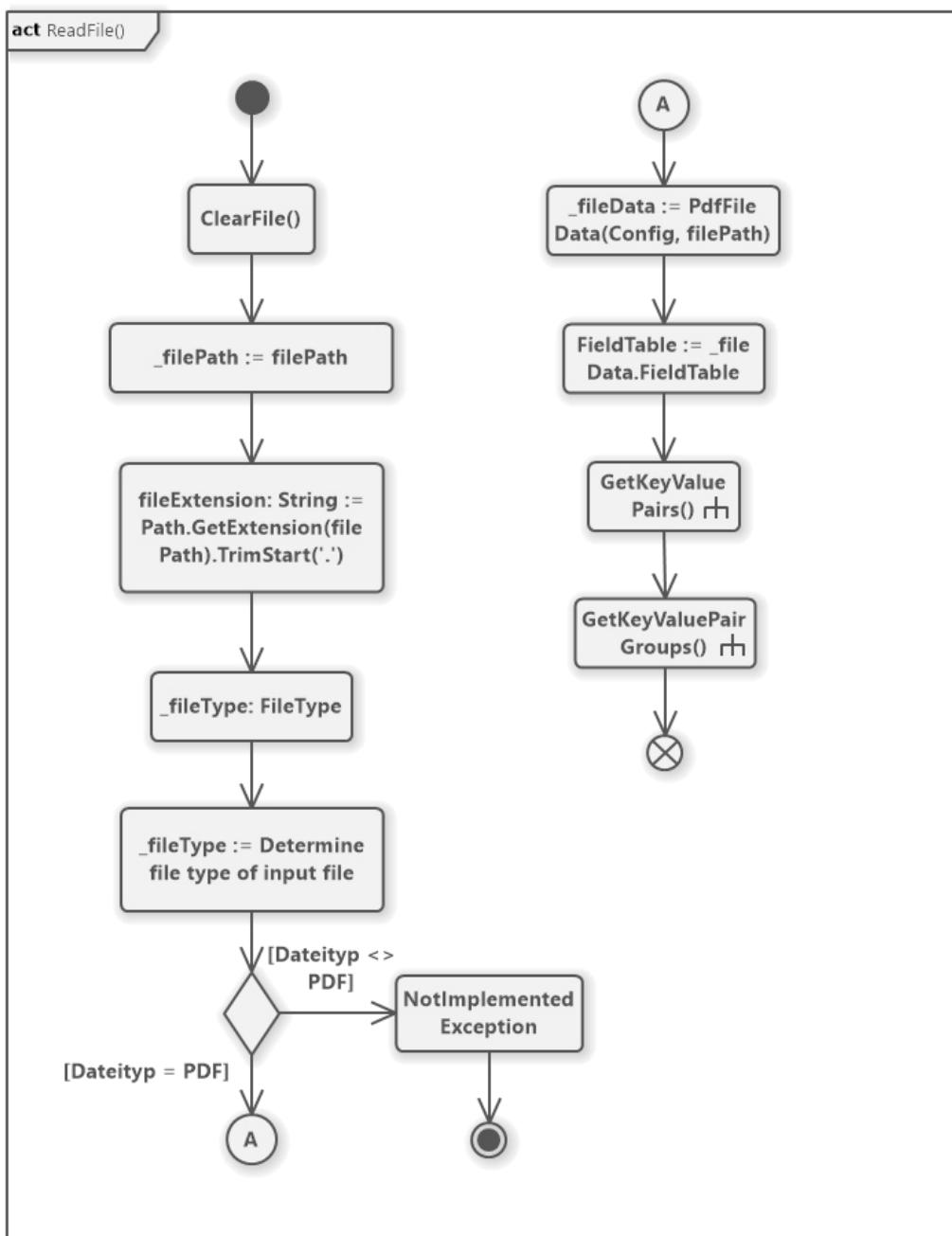


Abb. 6: UML-Aktivitätsdiagramm - ReadFile()

3.3.6 Entwurf der dynamischen Methode ExtractKeyValuePairs()

Entgegen der Planung aus dem Projektantrag wurden für diese Problematik zwei Methoden modelliert, um der Entscheidung zu entsprechen, KeyValuePair bsw. Gruppen mit einer Basisklasse abzubilden. Zudem wurde das Wording an die restlichen Member angepasst hin zu GetKeyValuePairs() und GetKeyValuePairGroups(). Beide Methoden sind insofern dynamisch, dass ihr Verhalten entsprechend den übergebenen Parametern in Form von Instanziierungen der Klasse SearchKey unterschiedliche Algorithmen zur Extraktion anwenden. Das folgende Aktivitätsdiagramm stellt eine vereinheitlichte und vereinfachte Darstellung beider Methoden in Bezug auf ihr zugrundeliegendes Prinzip dar.

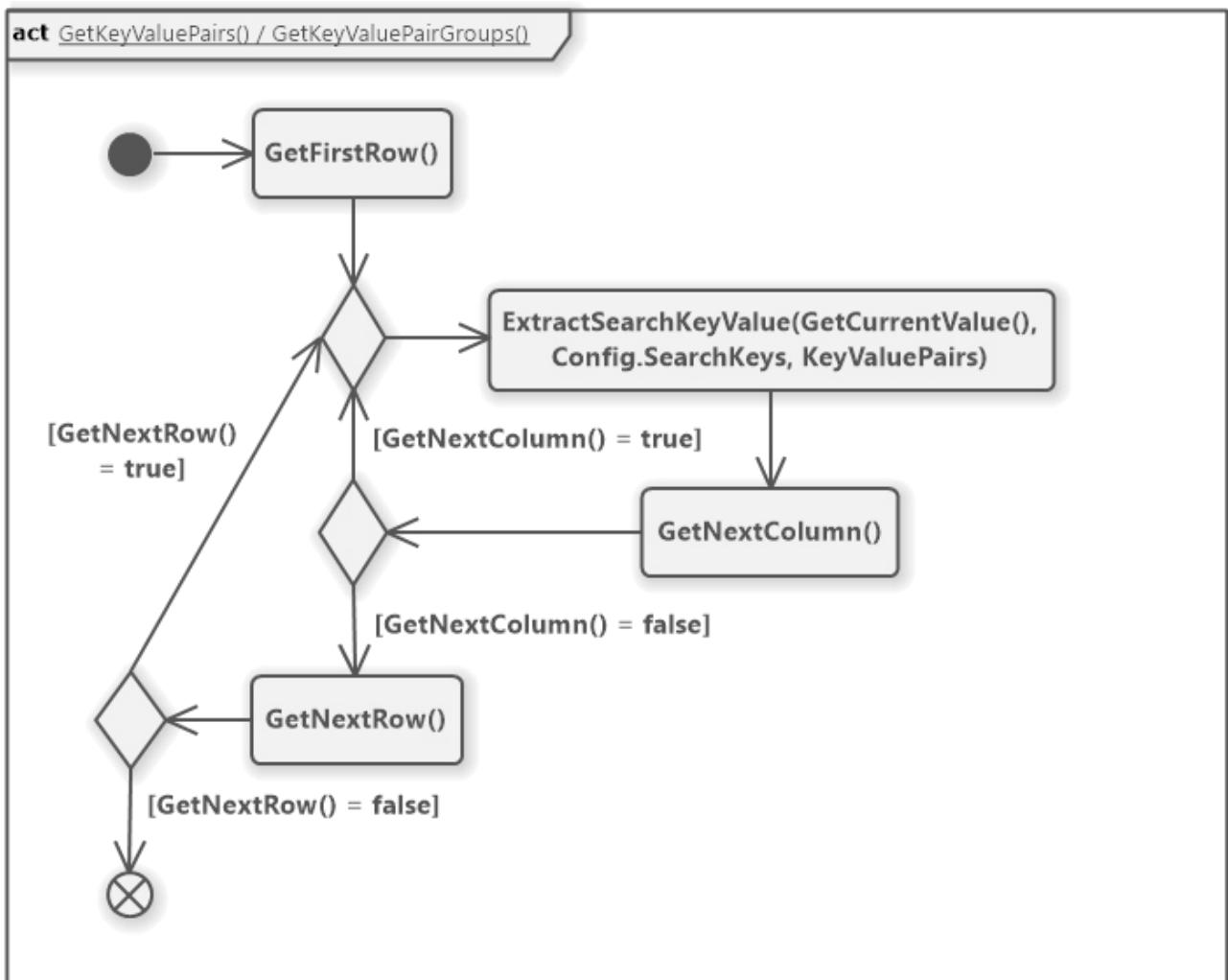


Abb. 7: UML-Aktivitätsdiagramm - GetKeyValuePairs() / GetKeyValuePairGroups()

3.3.7 Entwurf der Methoden zu KeyValuePairPairs und Gruppen

Durch die vorangegangenen Entscheidungen zur Klasse GroupedKeyValuePairPairs fielen die konkret benötigten Methoden der Klasse auch deutlich geringer aus, als ich es zur Zeit des Projektantrages angedacht hatte. Das abgebildete Aktivitätsdiagramm bildet nur die Rückgabemöglichkeiten der Klasse ab, da es aus einem anderen Kontext stammt, wie zuvor erläutert.

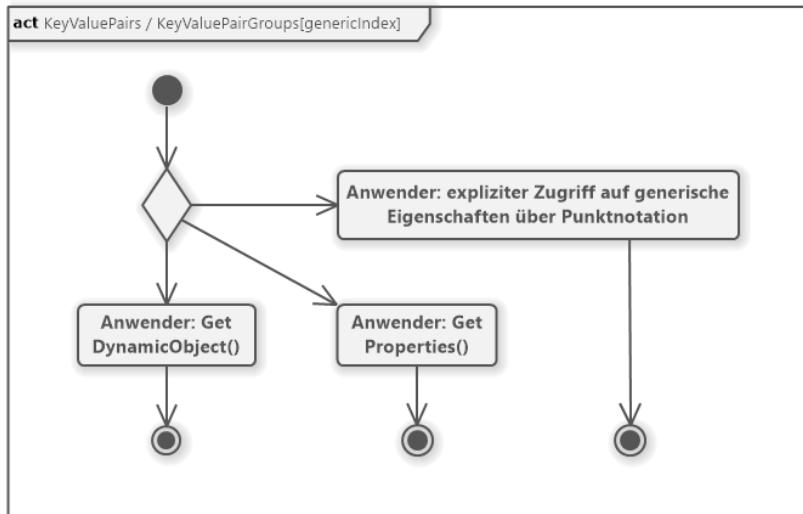


Abb. 8: UML-Aktivitätsdiagramm – Rückgabemöglichkeiten der Klasse GroupedKeyValuePairPairs

3.3.8 Entwurf der Methode SetConfig()

Für diese Methode wurde das Wording zu ReadConfig() geändert, da der Gedanke verworfen wurde, hierfür Methodenüberlagerungen bereit zu stellen, welche zusätzlich zu einer Konfigurationsdatei auch einzelne Parameter entgegennehmen könnten.

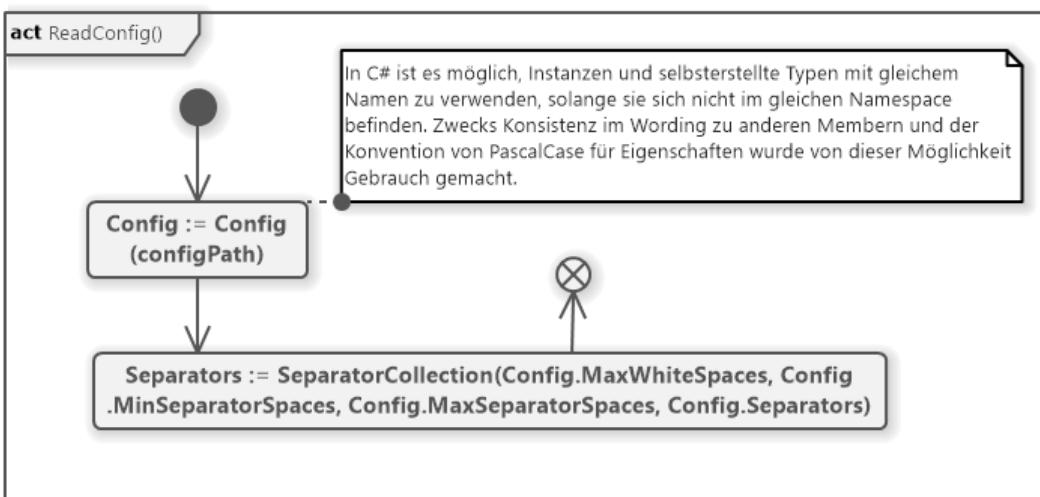


Abb. 9: UML-Aktivitätsdiagramm – ReadConfig()

3.3.9 Erstellen von Klassendiagrammen

In der Klasse GroupedSearchKeys werden die Schlüssel der zu suchenden Schlüssel-Wert-Paare gesammelt. Sie steht in Kombinationsbeziehung zur Klasse Config. In ihr wurden eine Vielzahl an Methoden- und Konstruktorenüberladungen modelliert, um möglichst hohe Interoperabilität an diesem zentralen Punkt der Konfiguration zu erreichen. (Diagramm in Anhang enthalten)

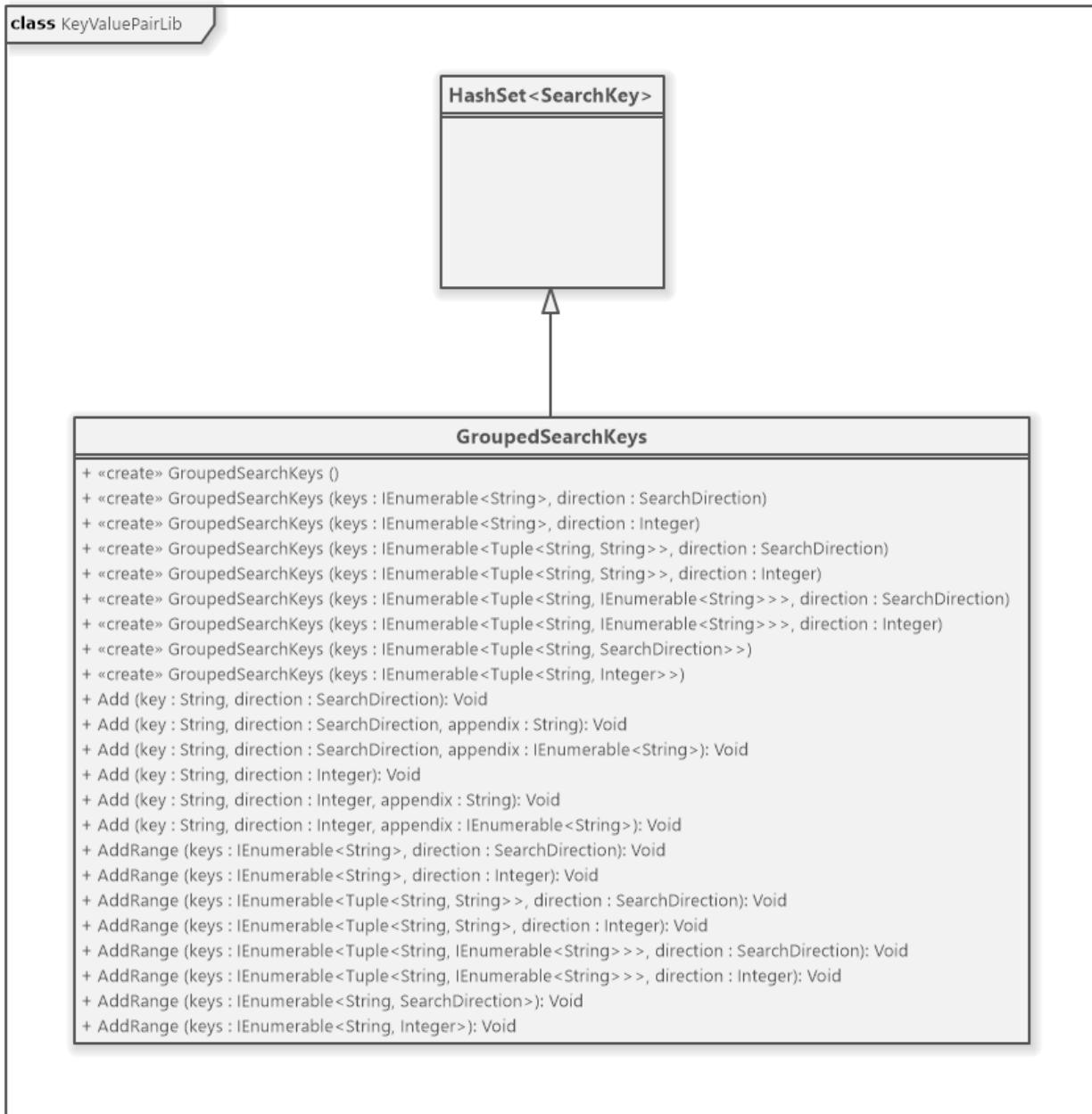


Abb. 10: UML-Klassendiagramm – GroupedSearchKeys

Die Klasse SearchKey steht wiederum in Kombinationsbeziehung zur Klasse GroupedSearchKeys. In ihr wird der konkrete einzelne Schlüssel der zu suchendenen Schlüssel-Wert-Paare neben weiteren Attributen festgehalten. Ursprünglich wurde im Projektantrag formuliert, die vertikalen und horizontalen Suchrichtungen mithilfe der booleschen Werte true und false zu unterscheiden. Weiterhin waren hierfür zusätzliche Konfigurationsattribute angedacht, welche jedoch eine globale Wirkungsweise im Algorithmus gehabt hätten. Aufgrund von Beispieldateien eines Neukunden, der ebenfalls für die Extraktion seiner PDF-Dateien geplant war, zeichnete sich jedoch frühzeitig eine Unzulänglichkeit in diesem Konzept ab, um bestimmte typografische Konstellationen in seinen Dokumenten abbilden zu können bzw. innerhalb eines jeweiligen Datenblattes unterscheidbar zu machen.

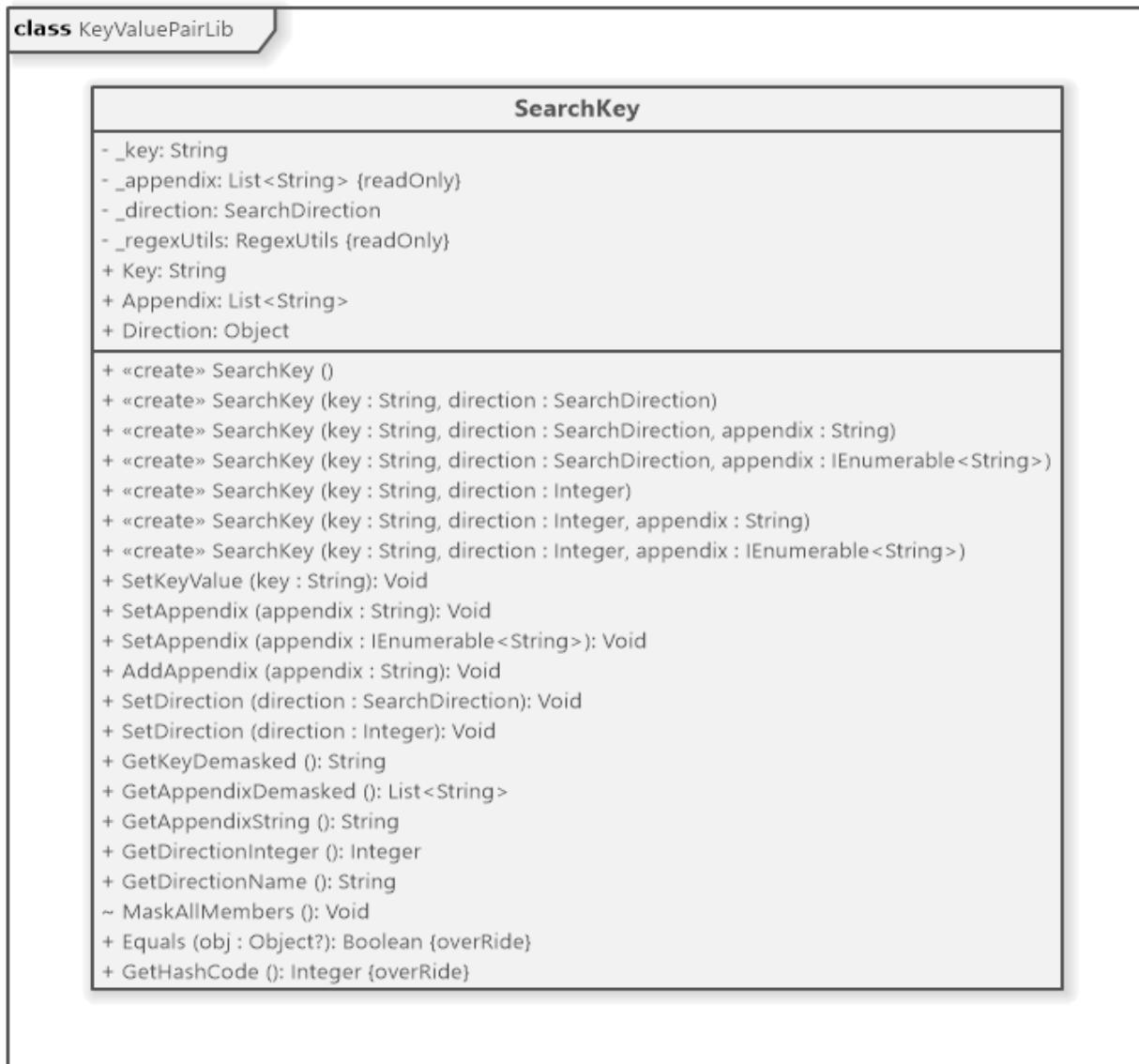


Abb. 11: UML-Klassendiagramm – SearchKey

Die Klasse SearchDirection steht weiterhin in Kombinationsbeziehung zur Klasse SearchKey und stellt die Umsetzung dieser Problematik dar. Sie ermöglicht die Unterscheidung von deutlich mehr Fällen in Form einer Enumeration und bietet dem Anwender zudem eine deutlich menschlichere Sprache in der Anwendung.

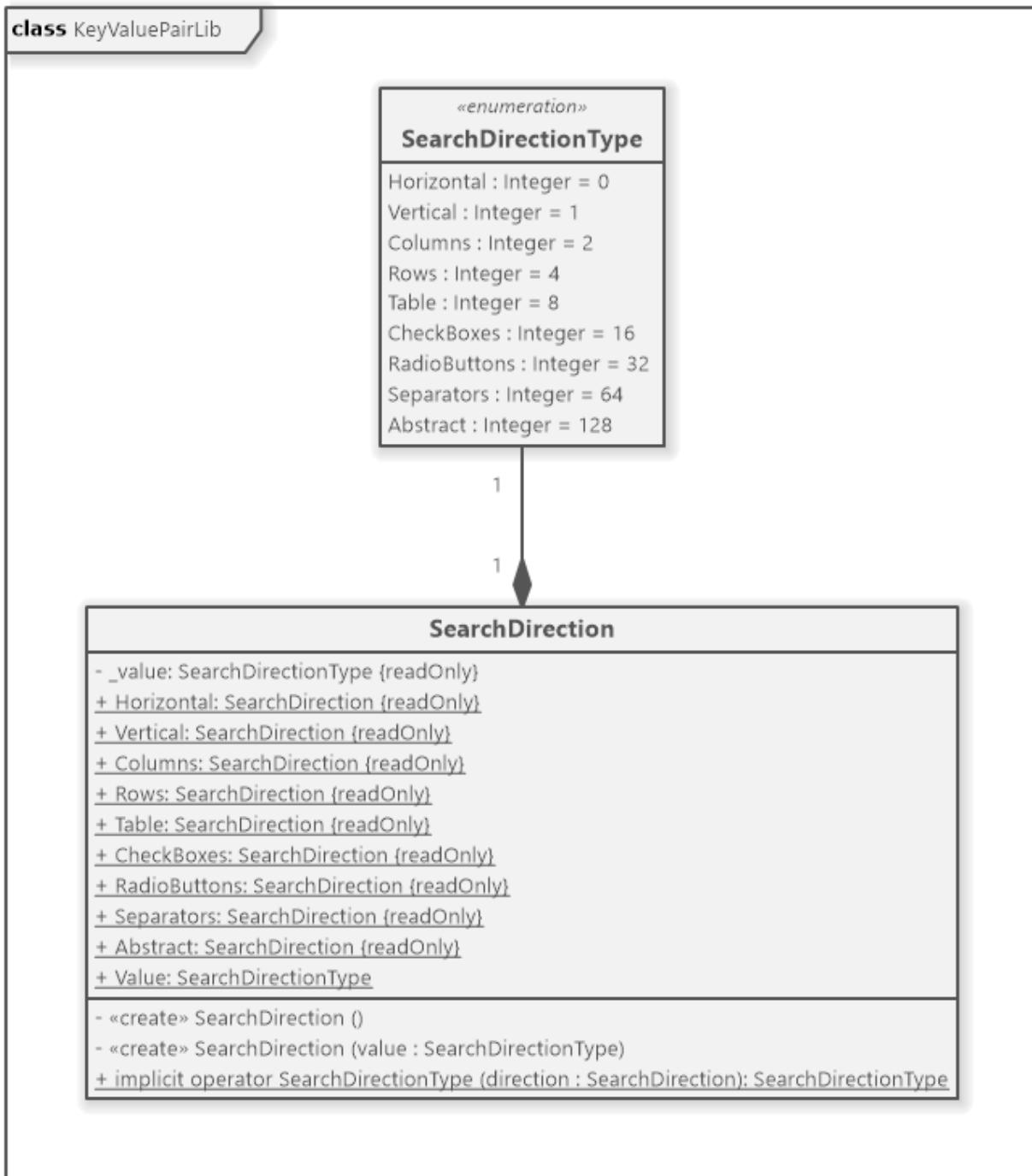


Abb. 12: UML-Klassendiagramm – SearchDirection

Aufgrund der häufigen Verwendung von regulären Ausdrücken, welche im Präfix und Suffix identisch sind, wurde die Basisklasse RegexWrappedCollection für mehrere Attribute der Konfiguration modelliert. In ihr werden Sammlungen von Strings gespeichert, denen automatisch das jeweils übergebene Präfix oder Suffix vorne- bzw. hintenangestellt werden. Es stellt eine Vererbung vom Verweistyp HashSet aus C# dar.

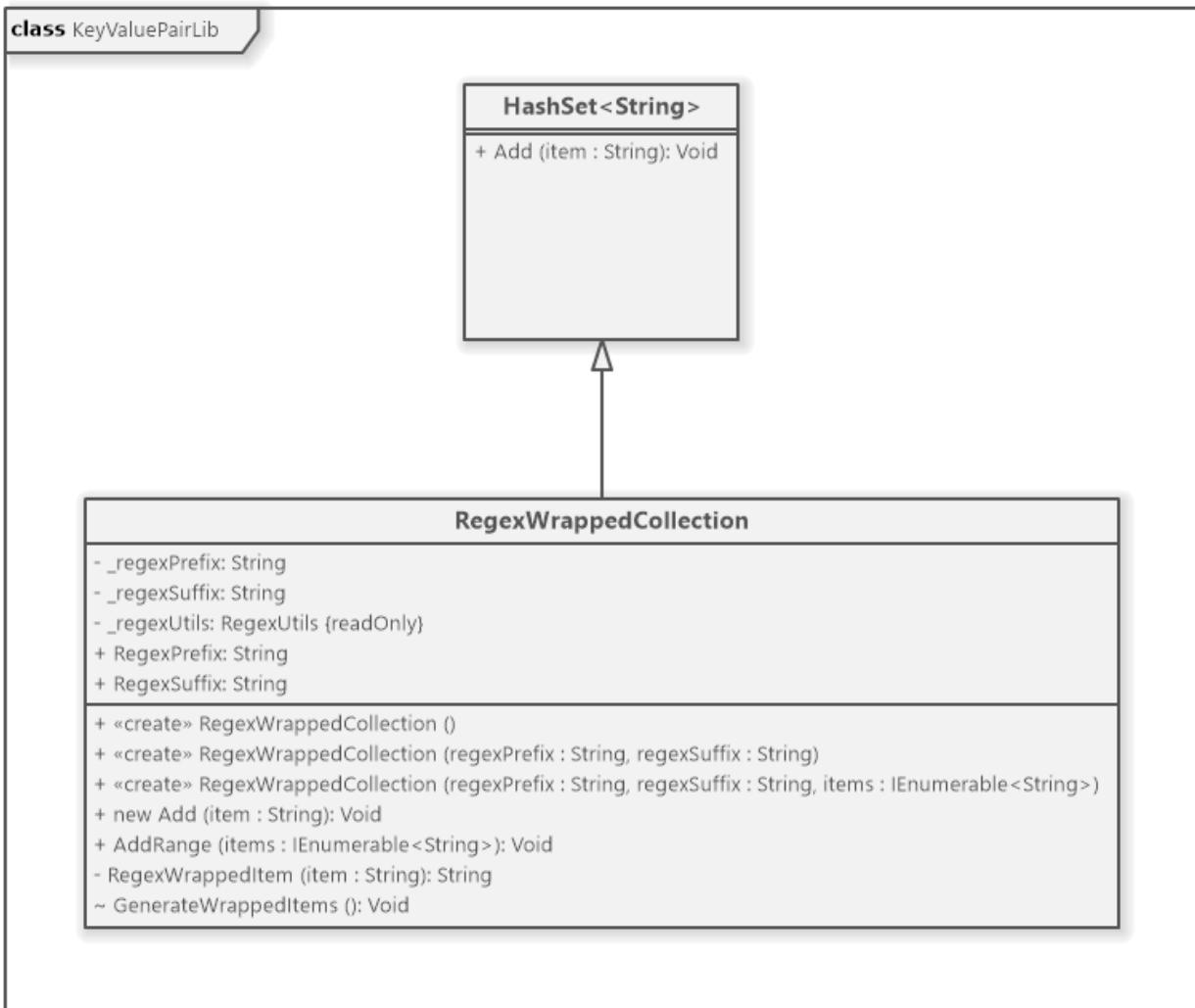


Abb. 13: UML-Klassendiagramm – `RegexWrappedCollection`

Die Klasse SeparatorCollection wurde für eine spezielle Form von typografischen Strukturen in Dokumenten benötigt. Stellen Sie sich hierzu einen Schlüsselbegriff auf der linken Seite eines Datenblattes vor, welcher durch eine Punktlinie bis zur rechten Seite mit seinem Wert verbunden ist. Da es hierbei keine räumliche Trennung gibt wird dieses Konstrukt als ein Feld interpretiert und eingelesen. Um ein solches Feld nun dennoch in seinen Schlüssel und Wert aufzuspalten zu können, wer-

den in dieser Klasse Sequenzen generiert, nach denen im String gesucht bzw. getrennt wird. (Diagramm in Anhang enthalten)

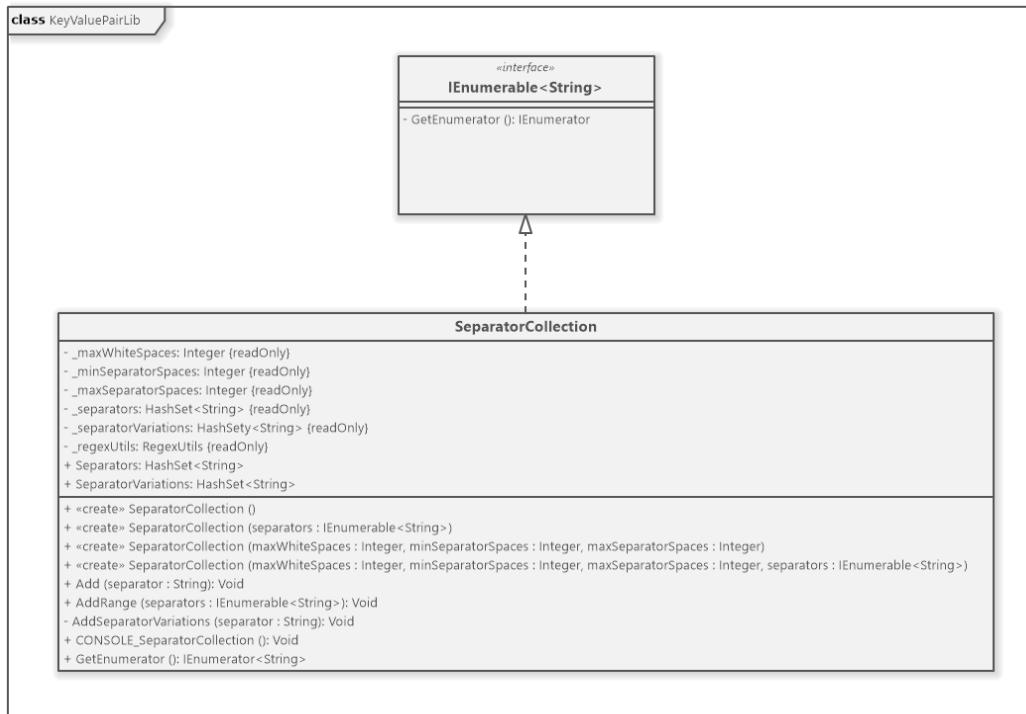


Abb. 14: UML-Klassendiagramm – SeparatorCollection

Zuletzt wurde noch eine Helferklasse modelliert, welche ebenfalls aufgrund der häufigen Verwendung von regulären Ausdrücken notwendig war. In ihr werden alle in regulären Ausdrücken interpretierbaren Zeichen festgehalten und einem Maskierungszeichen zugeordnet. Weiterhin wurde eine Maskierungs- und Demaskierungsmethode für diese Zeichen modelliert.

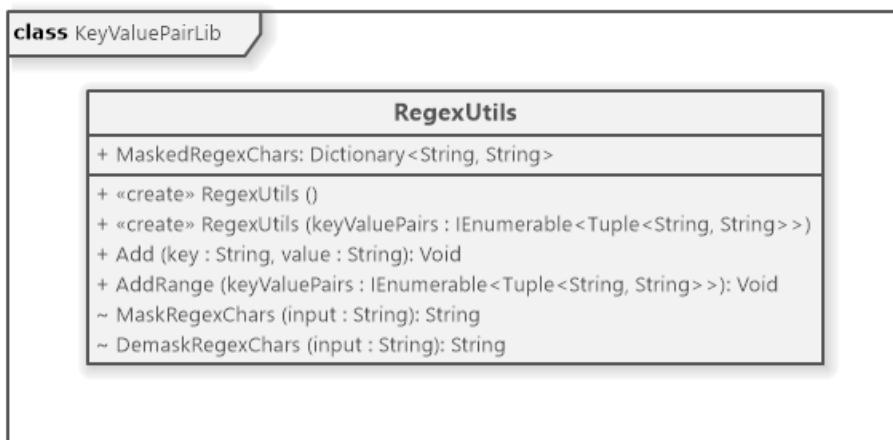


Abb. 15: UML-Klassendiagramm – RegexUtils

3.3.10 Erstellen eines Klassendiagramms (verkürzt) zu Assoziationsbeziehungen

Hiermit wurden die Beziehungen der selbsterstellten Klassen zueinander und ihre geerbten bzw. implementieren Verweistypen festgehalten.

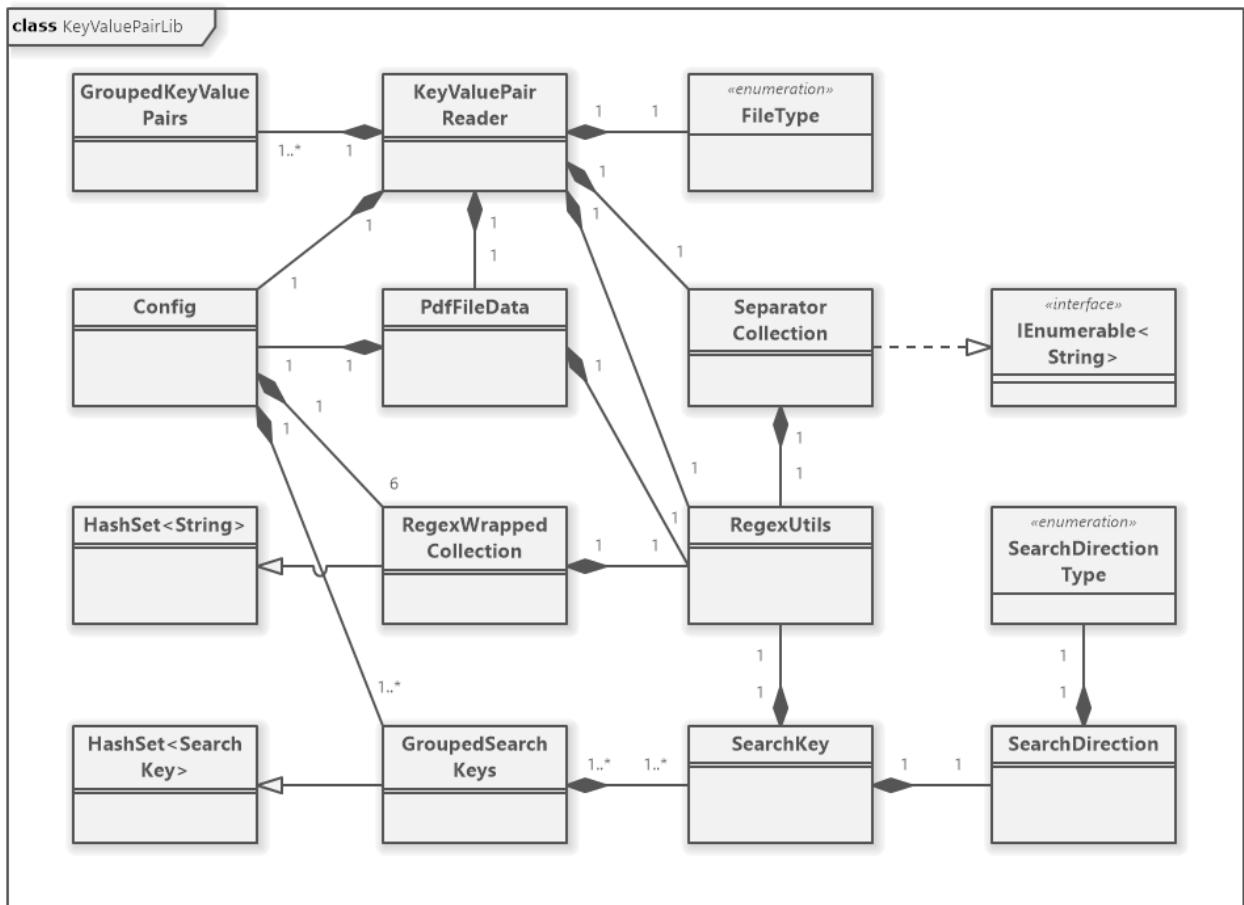


Abb. 16: UML-Klassendiagramm – Klassendiagramms (verkürzt) zu Assoziationsbeziehungen

3.3.11 Erstellen eines Sequenzdiagramms zum Algorithmus der Datenextraktion

In diesem Sequenzdiagramm wurde der Ablauf vom Einlesen einer PDF-Datei über die Aufbereitung der Daten und der Extraktion der gesuchten Schlüssel-Wert-Paare hinweg bis zur Speicherung der Ergebnisse in einem Container modelliert. (Diagramm in Anhang enthalten)

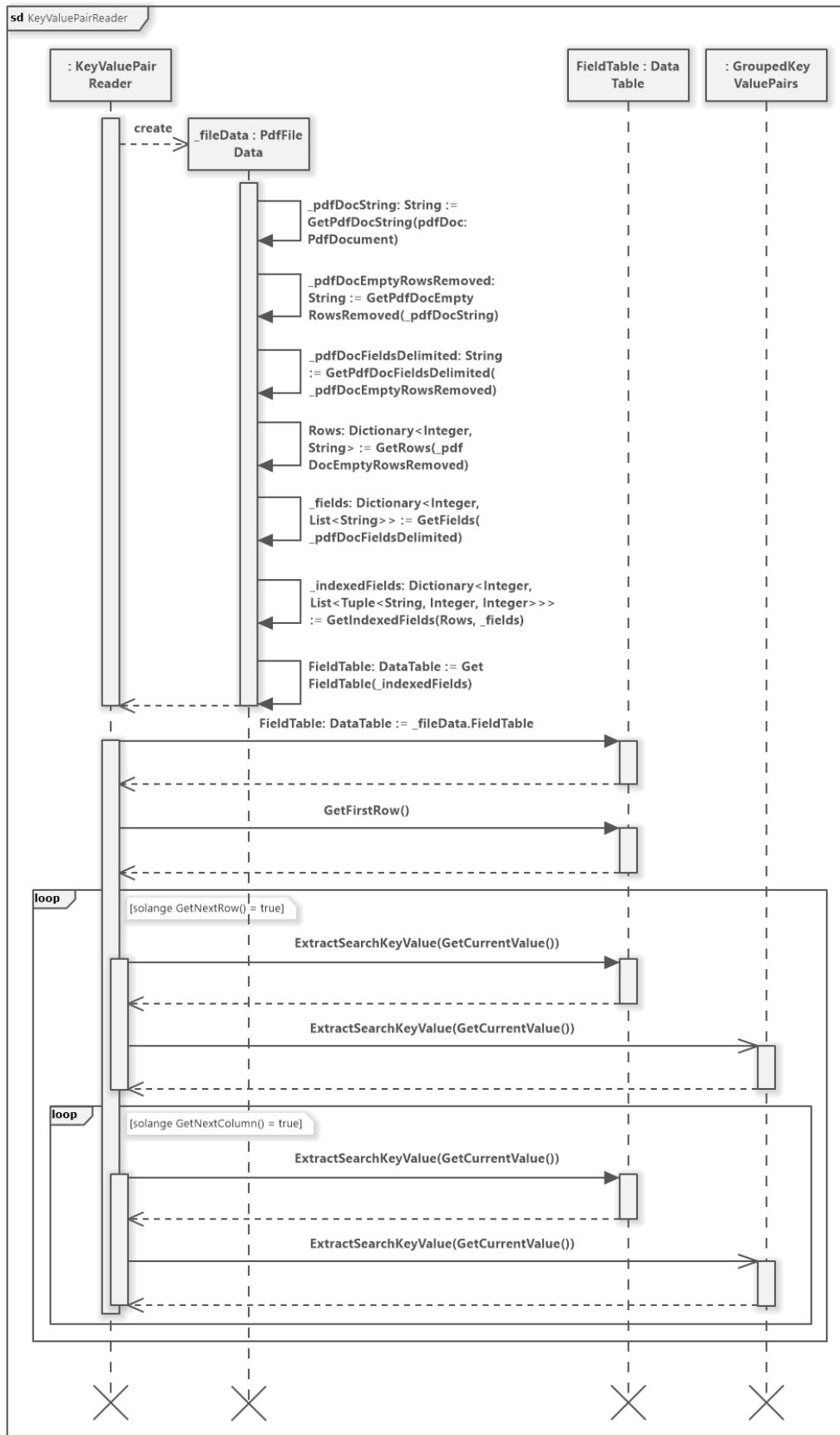


Abb. 17: UML-Sequenzdiagramm zum Algorithmus der Datenextraktion

3.3.12 Erstellen eines Aktivitätsdiagramms zum Ablauf der Klassenfunktionalitäten

In diesem Aktivitätsdiagramm wurden die möglichen Pfade in Bezug auf die Handhabung der Klassenbibliothek modelliert. Hierbei habe ich bewusst auf Bedingungen an den Verzweigungen im Hauptdiagramm verzichtet. Dies hatte den Hintergrund, dass ich nicht nur semantische Abläufe, sondern auch mögliche Entscheidungen des Anwenders abbilden wollte. Beschriftungen an den Verzweigungen hätten zum einen eine Doppelung der Begriffe in den Aktionen dargestellt. Zum anderen littten die Lesbarkeit und Übersichtlichkeit des Diagramms darunter, sodass ich mich zu dieser Interpretation von UML entschieden habe. (Diagramm in Anhang enthalten)

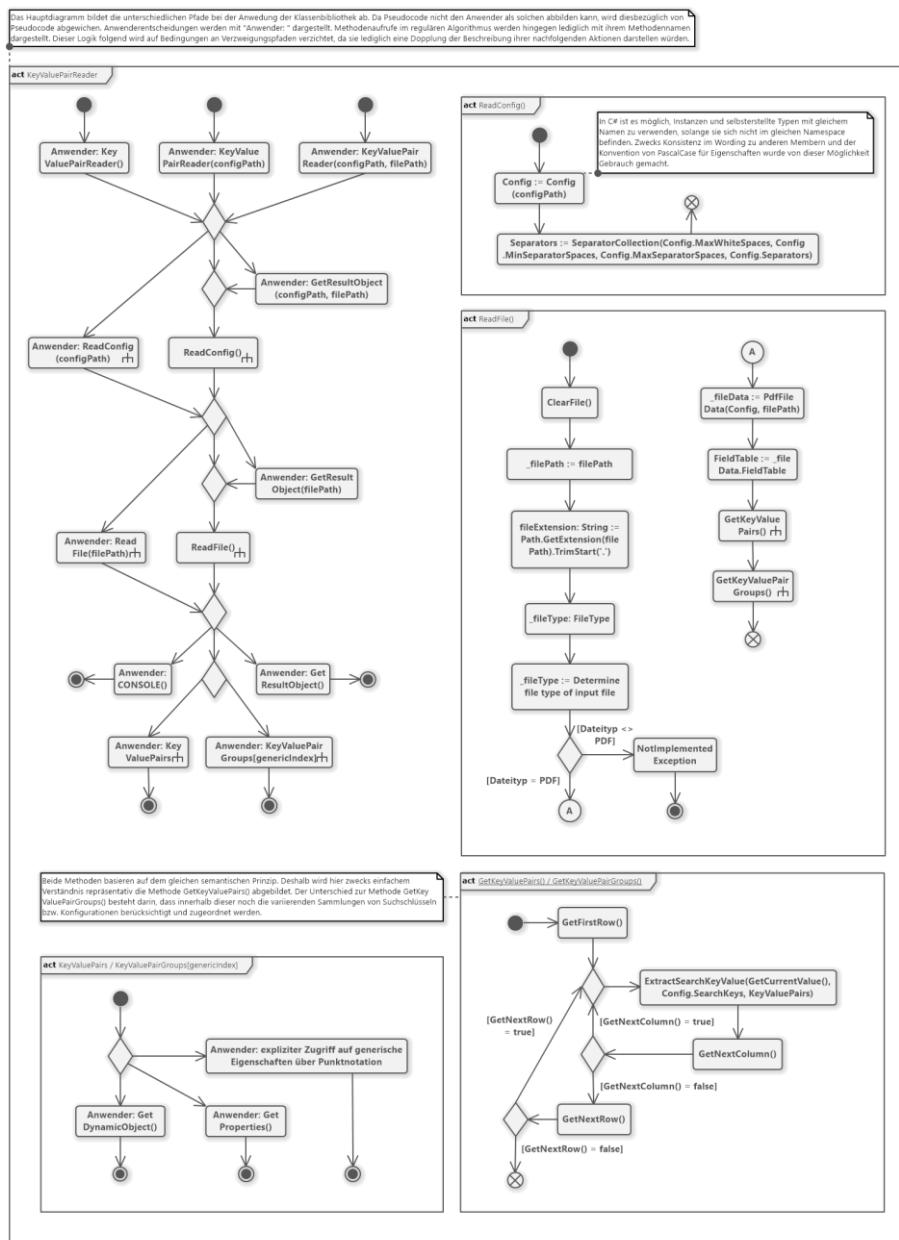


Abb. 18: UML-Aktivitätsdiagramm zum Ablauf der Klassenfunktionalitäten

3.4 Implementierungsphase

Der Zeit entsprechend wurde auch ChatGPT während der Entwicklung verwendet, wie ich es auch im Quellenverzeichnis anführe. Um Missverständnisse zu vermeiden, erkläre ich kurz die Methodik, mit welcher ich ChatGPT verwende. Ich habe die Erfahrung gemacht, dass mit ChatGPT deutlich schneller semantische Beispiele recherchiert werden können, als es z.B. auf StackOverflow möglich ist. Mir ist bewusst, dass die Aktualität und Korrektheit hierbei zu hinterfragen ist, wobei sich dies im Kontext von Quelltexten als quasi irrelevant herausgestellt hat, da die korrekte Funktionsweise unmittelbar überprüft werden kann. Ich verwende ChatGPT folglich ausschließlich zu Recherchezwecken, die ich unmittelbar fachlich überprüfen kann. Quelltexte werden weiterhin von mir selbst mit eigenem Stil und Wording verfasst, um auszuschließen, dass parallele Logiken stattfinden, die mir nicht bekannt sind. Für diese Dokumentation wurde ChatGPT ausdrücklich nicht verwendet.

Im Folgenden werde ich je einen Codeausschnitt pro Unterpunkt der Implementierungsphase präsentieren, welchen ich persönlich am prägnantesten empfand. Die roten Linien, welche Ihnen evtl. auffallen, stammen von dem Addon „Editor Guidelines“, welches die 120 Zeichen Konvention pro Zeile highlightet. An dieser Stelle möchte ich auch auf die XML-Kommentare hinweisen, die ich zum Zweck einer qualitativ hochwertigen Klassenbibliothek mitgepflegt habe. Mit XML-Kommentaren kann man Dokumentationselemente für andere Anwender bzw. Entwickler direkt innerhalb der IDE zur Verfügung stellen und somit eine deutlich angenehmere User Experience schaffen.

3.4.1 Implementierung der Klasse KeyValuePairReader und der dynamischen Methode ExtractSearchKeyValue()

Anhand des Klassendiagramms ist ihnen sicher aufgefallen, dass diese Klasse ein deutlich größeres Volumen als die restlichen Klassen entwickelt hat. Hierzu ist noch anzumerken, dass ich in der Vergangenheit eine eigene Klassenbibliothek entwickelt hatte, welche innerhalb des Verweistypen DataTable aus C# eine zweidimensionale Tabelle abbildet und Methoden zur Navigation durch deren Quasi-Zellen bereitstellt. Aus dieser konnte ich einige Methoden für mein aktuelles Projekt refaktorisieren. Im ursprünglichen Algorithmus wurde diese Problematik noch durch eine relativ wilde Verschachtelung von Dictionaries innerhalb Dictionaries umgesetzt, was ich für mein Projekt nachhaltiger lösen wollte.

In der folgend abgebildeten Methode ExtractSearchKeyValue() wird für jede Zelle, welcher der Algorithmus durchläuft, eine Prüfung vollzogen, ob ein Suchschlüssel in dieser Zelle enthalten ist. Wenn

dies der Fall ist, wird das SearchDirection-Objekt des SearchKey-Objekts ausgelesen und der weitere Algorithmus zur Extraktion dynamisch ausgelöst:

```
/// <summary>
/// This method compares the provided search keys in searchKeys with the current position of the cursor in currentField and, if
/// Diese Methode vergleicht die bereitgestellten Suchschlüssel in searchKeys mit der aktuellen Position des Cursors in current
/// </summary>
/// <param name="currentField">The current field value as string. Der aktuelle Feld-Wert als Zeichenkette.</param>
/// <param name="searchKeys">The collection of search keys to be compared. Die Sammlung von Suchschlüsseln, die verglichen werden.
/// <param name="keyValuePairs">The collection of key-value pairs to which the results are added. Die Sammlung von Schlüssel-Wert-Paaren, zu denen die Ergebnisse hinzugefügt werden.
/// <returns>True if search key and field matched, otherwise false. True, wenn Suchschlüssel und Feld übereinstimmen, sonst false.
3 Verweise
public bool ExtractSearchKeyValue(string currentField, GroupedSearchKeys searchKeys, GroupedKeyValuePairs
    keyValuePairs)
{
    foreach (SearchKey searchKey in searchKeys)
    {
        if (currentField == searchKey.Key && searchKey.Direction == SearchDirection.Horizontal)
        {
            ExtractHorizontal(searchKey, keyValuePairs);
            return true;
        }
        if (currentField == searchKey.Key && searchKey.Direction == SearchDirection.Vertical)...
        if (currentField == searchKey.Key && searchKey.Direction == SearchDirection.Columns)...
        if (currentField == searchKey.Key && searchKey.Direction == SearchDirection.Rows)...
        if (currentField == searchKey.Key && searchKey.Direction == SearchDirection.Table)...
        if (currentField.Contains(searchKey.Key) && searchKey.Direction == SearchDirection.CheckBoxes)...
        if (currentField.Contains(searchKey.Key) && searchKey.Direction == SearchDirection.RadioButtons)...
        if (currentField.Contains(searchKey.Key) && searchKey.Direction == SearchDirection.Separators)...
        if (currentField.Contains(searchKey.Key) && searchKey.Direction == SearchDirection.Abstract)...
    }
    return false;
}
```

Abb. 19: Quelltext der Methode ExtractSearchKeyValue() aus KeyValuePairReader

3.4.2 Implementierung der Assoziationsklasse PdfFileData

In diesem Codeausschnitt wird das Ergebnis der vorherigen Verarbeitungsschritte nach dem Einlesen einer PDF-Dateien final in eine DataTable gespeichert. Die Werte des übergebenen Dictionaries bilden in Reihenfolge die Zeile, die Indexposition des Sub-Strings, den Inhalt des Sub-Strings und das Ende als Indexposition ab. Hiermit werden die zuvor genannten Quasi-Zellen gebildet:

```
private static DataTable GetFieldTable(Dictionary<int, List<(string, int, int)>> indexedFields)
{
    // ! Return DataTable
    DataTable dataTable = new();
    // ! Rowindex
    dataTable.Columns.Add("Row", typeof(int));
    // ! Startindex
    dataTable.Columns.Add("Column", typeof(int));
    // ! Value
    dataTable.Columns.Add("Value", typeof(string));
    // ! Endindex
    dataTable.Columns.Add("End", typeof(int));
    // ! Iterate through rows
    for (int i = 0; i < indexedFields.Count; i++)
    {
        // ! Iterate through columns
        foreach ((string, int, int) entry in indexedFields[i]) dataTable.Rows.Add(i, entry.Item2, entry.Item1,
            entry.Item3);
    }
    // ! Convert DataTable to DataView
    DataView dataView = dataTable.DefaultView;
    // ! Sort in DataView
    dataView.Sort = "Row ASC, Column ASC";
    // ! Convert back to DataTable
    return dataView.ToTable();
}
```

Abb. 20: Quelltext der Methode GetFieldTable() aus PdfFileData

3.4.3 Implementierung der Assoziationsklasse und Methoden zu KeyValuePairPairs und Gruppen

Die Erzeugung von generischen Eigenschaften, welche als Attribut-Namen den Namen des Suchschlüssels erhalten, wurde hier mithilfe eines Expando-Objekts umgesetzt:

```

/// <summary>
/// The internal storage for the key-value pairs as an ExpandoObject.
/// Der interne Speicher für die Schlüssel-Wert-Paare als ExpandoObject.
/// </summary>
private readonly ExpandoObject _properties = new();
#endregion

// ! PROPERTIES
#region properties
/// <summary> Gets or sets the value as object associated with the specified key ...
4 Verweise
public object this[string name]...
#endregion

// ! METHODS
#region methods
/// <summary>
/// Adds a property with the specified name as string and value as object to the collection.
/// Fügt der Sammlung eine Eigenschaft mit dem angegebenen Namen als Zeichenkette und Wert als Objekt hinzu.
/// </summary>
/// <param name="name">The name of the property as string. Der Name der Eigenschaft als Zeichenkette.</param>
/// <param name="value">The value of the property as object. Der Wert der Eigenschaft als Objekt.</param>
21 Verweise
public void AddProperty(string name, object value) => (_properties as IDictionary<string, object>)[name] =
  value;

```

Abb. 21: Quelltextauszug der Klasse GroupedKeyValuePairPairs

3.4.4 Implementierung der Assoziationsklasse Config und Methode ReadConfig()

Parallel zur Entwicklung der Klasse Config wurde eine CFG-Datei als Muster im JSON-Format (JavaScript Objekt Notation) erstellt, welche in der Anwenderdokumentation genauer erläutert wird. Das Einlesen der Datei wurde mithilfe der populären Klassenbibliothek Newtonsoft.Json umgesetzt. Während der Implementierung stellte sich der Bedarf an weiteren Hilfsmethoden heraus, da die befüllten Member nicht mit ihren vorgesehenen Methoden angesprochen wurden:

```

public void ReadConfigFile(string filePath)
{
    string jsonContent = File.ReadAllText(filePath);
    JsonConvert.PopulateObject(jsonContent, this);
    // ! GenerateWrappedItems() & MaskAllMembers() are necessary because JsonConvert.PopulateObject() does not call members as
    IgnoreFields.GenerateWrappedItems();
    IgnoreRows.GenerateWrappedItems();
    BreakFields.GenerateWrappedItems();
    BreakRows.GenerateWrappedItems();
    CheckBoxes.GenerateWrappedItems();
    RadioButtons.GenerateWrappedItems();
    foreach (SearchKey searchKey in SearchKeys) searchKey.MaskAllMembers();
    foreach (KeyValuePair<string, GroupedSearchKeys> kvp in SearchKeyGroups)
    {
        foreach (SearchKey searchKey in kvp.Value) searchKey.MaskAllMembers();
    }
}

```

Abb. 22: Quelltext der Methode ReadConfigFile() aus Config

3.4.5 Implementierung der dynamischen Methode ReadFile()

Wie bereits vorweggegriffen, wurden weitere Dateitypen in grundlegenden Strukturen berücksichtigt, um zukünftige Weiterentwicklungen zu erleichtern. Ihre jeweiligen Verzweigungspfade wurden mit NotImplementedExceptions beendet:

```

public void ReadFile(string filePath)
{
    // ! Reset result objects before reading new file
    ClearFile();
    // ! Set to proof that necessary data is available
    _filePath = filePath;
    // ! Get file extension
    string fileExtension = Path.GetExtension(filePath).TrimStart('.');
    // ! Declare fileType, initialize fileType, switch-case by fileType
    FileType fileType = fileExtension.ToLower() switch
    {
        "pdf" => FileType.Pdf,
        "csv" => FileType.Csv,
        "xlsx" => FileType.Xlsx,
        "xml" => FileType.Xml,
        _ => FileType.Unknown,
    };
    // ! Dynamic instantiation of _fileData
    switch (fileType)
    {
        case FileType.Pdf:
            _fileData = new PdfFileData(Config, filePath);
            FieldTable = _fileData.FieldTable;
            GetKeyValuePairs();
            GetKeyValuePairGroups();
            break;
        case FileType.Csv:
            _fileData = new CsvFileData();
            FieldTable = _fileData.FieldTable;
            throw new NotImplementedException("CSV files not supported yet. " +
                "CSV-Dateien werden noch nicht unterstützt.");
    }
}
  
```

Abb. 23: Quelltextauszug der Methode ReadFile() aus KeyValuePairReader

3.5 Testphase

3.5.1 Black-Box-Test

Aufgrund des zu Beginn erwähnten Neukunden ergab sich die Möglichkeit, den Black-Box-Test in die ohnehin für diesen Kunden aufgesetzte Testroutine zu integrieren. Diese Integration erforderte dank dem Design, welches ich angestrebt hatte aufgrund eigener Erfahrungen mit anderen Klassenbibliotheken, erfreulicherweise nur einen Aufwand von ca. fünf Minuten für mich und den IT-Mitarbeiter aus der Personalplanung. Zudem stand aus dem begleitenden Debugging bereits eine Konfigurationsdatei hierfür bereit. Diese marginale Zeitspanne habe ich aufgrund der bisherigen Kalkulation mit einstündigen Werten final nicht gewichtet. Die folgende Abbildung zeigt die Oberflächen während dem Test in unserem ClassCockpit Ticketsystem mit PDF-Upload-Button und extrahierten Werten:

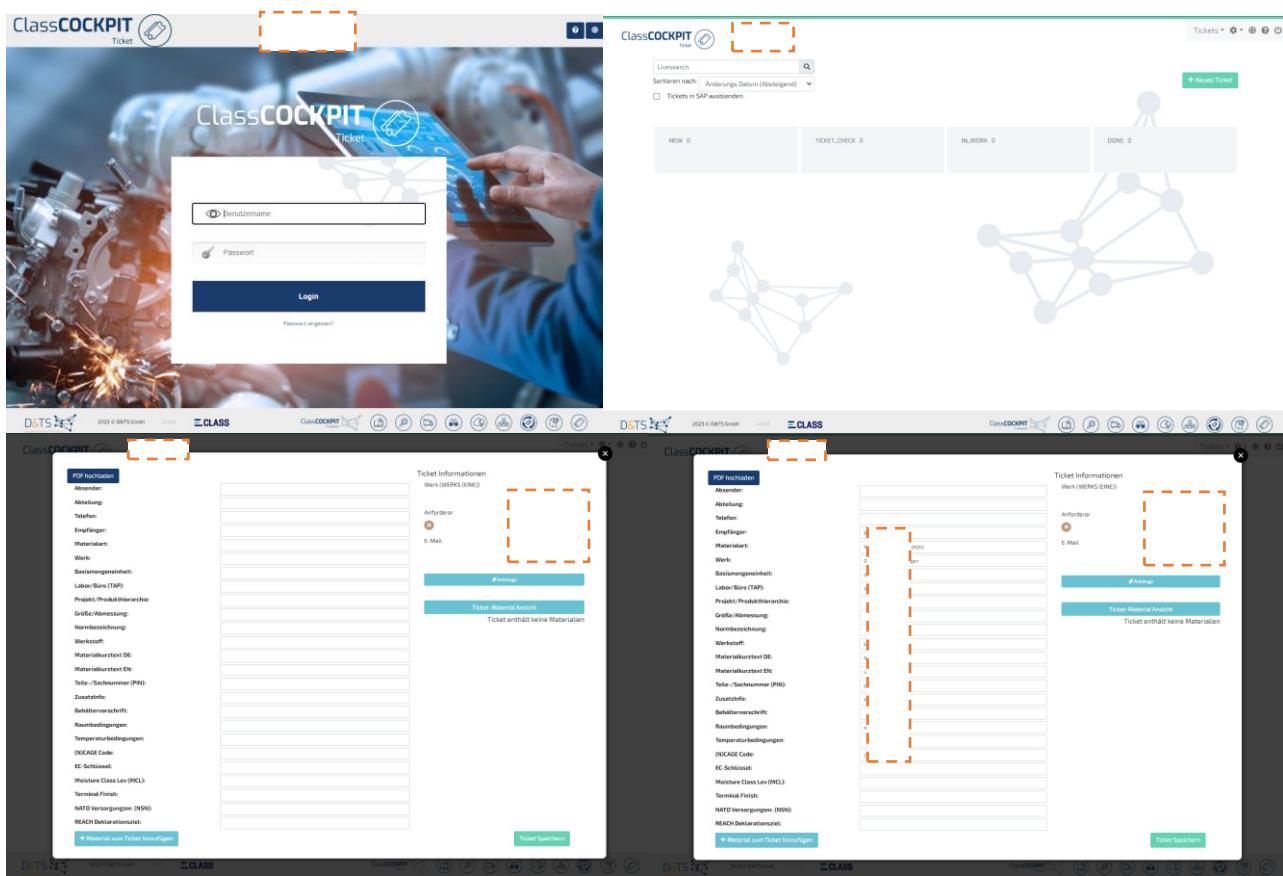


Abb. 24: Oberflächen des ClassCockpit Ticketsystems

3.5.2 White-Box-Text

Die finale Phase des Projektes war ein abschließender White-Box-Test, in dem ich noch einmal sicherstellte, dass kürzliche Entwicklungsschritte weder marginale, noch größere Veränderungen in den Ergebnissen der Extraktionen aus der verwendeten Palette an Testdateien verursacht hatten, um einen fehlerfreien Ablauf der Abnahme des Projekts zu gewährleisten. Hierbei fand noch ein letztes Mal eine detaillierte Sichtung aller Werte statt, da während der vorangegangenen Phasen, während dem begleitenden Debugging, der Fokus hauptsächlich auf die jeweils aktuelle Implementierung lag.

3.5.3 Begleitendes Debugging

Das begleitende Debugging war ein ständiger Routineprozess während der Entwicklung, was sich auch anhand der Vielzahl an sogenannten „CONSOLE_...()“-Methoden erkennen lässt, deren Ergebnisse in der CONSOLE()-Methode der Klasse KeyValuePairReader gebündelt werden. Diese Methode stellt ein Debugging- und Inspektions-Tool dar und gibt die Ergebnisse der jeweiligen Sub-Methoden in Reihenfolge der Verarbeitungsschritte in der Konsole aus. Ohne Parameter werden alle Methoden ausgegeben. Mithilfe von Integer-Werten als Parameter lässt sich diese Ausgabe zudem steuern.

Auch wenn es allgemeine Gültigkeit hat, dass direktes Debugging über die IDE effektiver ist als das Einsetzen von Konsolenausgaben, habe ich speziell für die Dokumentenverarbeitung die Erfahrung gemacht, dass strukturiert wiederverwendete Konsolenausgaben sehr viele Klicks einsparen und mehr Komfort bieten können.

Aufgrund der Beschaffenheit meines Projektes, welches untrennbar mit der direkten Verarbeitung von Kundendaten verbunden ist, sind meine Möglichkeiten zur Darstellung von Ergebnissen nachvollziehbar eingeschränkt. Um Ihnen dennoch ein Beispiel für eine solche Ausgabe bereit zu stellen und die Funktionalität der Klassenbibliothek zu demonstrieren, habe ich Ihre „Handreichung zur Abschlussprüfung in den Berufen Fachinformatiker/-in, IT-System-Elektroniker/-in, Kaufmann/-frau für IT-Systemmanagement & Kaufmann/-frau für Digitalisierungsmanagement“ nach Begriffen ausgelesen, welche die Charakteristik von Schlüssel-Wert-Paaren aufweisen:

```
CONSOLE_GroupedKeyValuePairs:
#####
°Umfang°
          °10 - 15 Seiten
          Fachinformatiker Systemintegration / Daten- und Prozessanalyse 20 - 25 Seiten
          Fachinformatiker Anwendungsentwicklung / Digitale Vernetzung 35 - 40 Seiten
          (Deckblatt, Inhaltsverzeichnis, Erklärung und Anhang zählen nicht zum Seitenumfang!)
          Anlage darf nicht mehr als 10 Seiten beinhalten!
°Gestaltung°
          °Schriftgröße 12 Punkt, / Zeilenabstand 1,5 Zeilen,
          Rand links 2,5 cm,
          Rand rechts 1,5 cm
          Seitennummerierung°

CONSOLE_Stand: Juni 2023:
CONSOLE_GroupedKeyValuePairs:
#####

°1. Vorbemerkung°
          °3°
°2. Zeitplan für die Abschlussprüfung°
          °3°
°Abbildung 1: Zeitlicher Ablauf der Sommer- bzw. Winterabschlussprüfung°
          °3°
°3. Projektantrag und Genehmigungsverfahren°
          °4°
°Abbildung 2: Online-Anmeldung°
          °4°
°4. Projektarbeit und deren Dokumentation°
          °7°
°Abbildung 3: Bewertungsmatrix Projektdokumentation°
          °9°
°5. Schriftliche Abschlussprüfung°
          °10°
°Abbildung 4: Prüfungsbreiche und deren Verteilung°
          °11°
°6. Präsentation und Fachgespräch°
          °12°
°Abbildung 5: Bewertungsmatrix Präsentation und Fachgespräch°
          °13°
°7. Bestehen der Abschlussprüfung°
          °14°
°Gilt ausschließlich für eine Wiederholungsprüfung°
          °14°
°Persönliche Erklärung°
          °15°
```

Abb. 25: Konsolenausgabe der Methode CONSOLE() aus KeyValuePairReader

4 Projektergebnis

4.1 SOLL-IST-Vergleich

Wie bereits unter Punkt 3.1 erläutert, zeichnete sich vor allem die praktische Umsetzung der Implementierungsphase durch Fluktuationen in der Zeitplanung der einzelnen Teilschritte aus. Dies lag beim vorliegenden Projekt vor allem daran, dass sich oftmals die Verlagerung von Funktionalitäten einer Klasse bzw. Methode in eine andere als sinnvoller erwies. Auch das Gebot von nachhaltiger Softwareentwicklung führte oftmals zur Bildung von neuen Assoziationsklassen. Zudem muss berücksichtigt werden, dass dem Projekt eine hohe Komplexität zugrunde lag, welche sich durch Testdateien eines Neukunden nochmals verstärkt hatte. Somit muss ich an diesem Punkt darauf hinweisen, dass die vorliegende Gegenüberstellung lediglich die verhältnismäßigen Verschiebungen innerhalb der Implementierungsphase versucht abzubilden. Eine exakte zeitliche Erfassung oder Abgrenzung auf jeweilige Klassen oder Methoden wäre mir in diesem Rahmen nicht möglich gewesen:

Anforderungsphase	6 h	4 h
Kick-off-Meeting	2 h	1 h
Prüfung SOLL-Konzept	2 h	1 h
Aufgabenanalyse	2 h	2 h

Planungsphase	4 h	4 h
Zeitplanung	2 h	1 h
Personalplanung	1 h	1 h
Kosten- und Amortisationsplanung	1 h	2 h

Entwurfsphase	17 h	18 h
Entwurf der Klasse KeyValuePairReader	2 h	2 h
Entwurf der Assoziationsklasse PdfFileData	1 h	1 h
Entwurf der Assoziationsklassen zu KeyValuePairs und Gruppen	1 h	1 h
Entwurf der Assoziationsklasse Config	1 h	1 h
Entwurf der dynamischen Methode ReadFile()	2 h	2 h
Entwurf der dynamischen Methode ExtractKeyValuePairs()	2 h	2 h
Entwurf der Methoden zu KeyValuePairs und Gruppen	1 h	1 h
Entwurf der Methode SetConfig()	1 h	1 h
Erstellen von Klassendiagrammen	1 h	1 h
Erstellen eines Klassendiagramms (verkürzt) zu Assoziationsbeziehungen	1 h	1 h
Erstellen eines Sequenzdiagramms zum Algorithmus der Datenextraktion	2 h	2 h
Erstellen eines Aktivitätsdiagramms zum Ablauf der Klassenfunktionalitäten	2 h	3 h

Implementierungsphase	26 h	29 h
Implementierung der Klasse KeyValuePairReader	4 h	5 h
Implementierung der Assoziationsklasse PdfFileData	2 h	5 h
Implementierung der Assoziationsklassen zu KeyValuePairs und Gruppen	3 h	1 h
Implementierung der Assoziationsklasse Config	2 h	5 h
Implementierung der dynamischen Methode ReadFile()	4 h	4 h
Implementierung der dynamischen Methode ExtractKeyValuePairs()	4 h	6 h
Implementierung der Methoden zu KeyValuePairs und Gruppen	3 h	1 h
Implementierung der Methode SetConfig()	4 h	2 h

Testphase	8 h	6 h
Black-Box-Test	2 h	0 h
White-Box-Test	2 h	2 h
Begleitendes Debugging	4 h	4 h

Projektergebnis	4 h	4 h
Projektabnahme	1 h	1 h
SOLL-IST-Vergleich	2 h	2 h
Erklärung der Abweichungen	1 h	1 h

Abschlussphase	15 h	15 h
Dokumentation Klassenbibliothek für administrative Zwecke	2 h	2 h
Release-Erstellung GitHub Repository	1 h	1 h
Projektdokumentation	12 h	12 h

Summe Projektphasen	80 h	80 h
----------------------------	-------------	-------------

Aufgrund der Integration des Black-Box-Tests in einen anderen Routineprozess verbesserten sich die Projektkosten wie folgt:

Verbesserte Projektkosten
Personaleinsatz in Stunden * durchschnittlicher interner Verrechnungssatz
73 h * 50,00 €/h = 3.650,00 €

Für das Amortisationsszenario eins ergab sich somit z.B. folgende Verbesserung:

Verbesserte Amortisationsdauer
(Verbesserte Projektkosten + Personaleinsatz für Konfigurationserstellung * d. i. V.) / d. i. V.
(3.650,00 € + 1 h * 50,00 €/h) / 50,00 €/h = 74 h

4.2 Erklärung der Abweichungen

Da ich bereits im Verlauf detaillierte Erklärungen geliefert habe, werde ich mich an dieser Stelle auf eine kurze Zusammenfassung beschränken:

- Das Kick-off-Meeting reduzierte sich um eine Stunde aufgrund vorangegangener großzügiger Zeitplanung.
- Die Prüfung des SOLL-Konzepts reduzierte sich um eine Stunde aufgrund vorangegangener großzügiger Zeitplanung.
- Die Zeitplanung als solches erforderte eine Stunde weniger aufgrund der Vorleistung aus dem Projektantrag.
- Die Kosten- und Amortisationsplanung wurde aufgrund vorangegangener Zeitgewinne und einer untypischen Amortisationsentwicklung mit einer zusätzlichen Stunde bearbeitet.
- Dem Aktivitätsdiagramm zum Ablauf der Klassenfunktionalitäten wurde aufgrund vorangegangener Zeitgewinne und Berücksichtigung zusätzlicher Komplexitäten mit einer zusätzlichen Stunde bearbeitet.
- Die Assoziationsklassen zu KeyValuePairPairs und Gruppen erforderten nur eine Basisklasse namens GroupedKeyValuePairPairs durch Verwendung von Dictionaries.
- Die dynamische Methode ExtractKeyValuePairPairs() wurde aufgespalten und in GetKeyValuePairPairs() und GetKeyValuePairGroups() umbenannt, um Veränderungen bezüglich der Klasse GroupedKeyValuePairPairs zu berücksichtigen.
- Die Methode SetConfig() wurde in ReadConfig() unbenannt, um ihrer finalen Funktionalität zu entsprechen.
- Die Parameter für die Konstruktoren der Klasse SearchKey wurden anstelle von simplen booleschen Werten mit der Assoziationsklasse SearchDirection umgesetzt, um präzisere Steuerung zu ermöglichen und einen Neukunden zu berücksichtigen.
- Die Kosten des Projektes und seiner Amortisation verbesserten sich aufgrund der Integration des Black-Box-Test in einen vorhandenen Routineprozess.

4.3 Fazit

Abgesehen von den Variationen in der Umsetzung erfüllt das Projekt alle geforderten Anforderungen und bietet darüber hinaus präzisere Steuerungsmöglichkeiten als von der Ausgangslage gefordert. Bis zum Abschluss dieser Dokumentation konnten keine Fehleranfälligkeitkeiten identifiziert werden und Umfang sowie bereitgestellte Dokumentationen erhielten durchweg positives Feedback ohne Beanstandungen. Das Projekt wurde vom Auftraggeber als vollumfänglich abgeschlossen beurteilt.

5 Abschlussphase

5.1 Erstellung der Anwenderdokumentation

Auf Wunsch des Unternehmens wurde die Anwenderdokumentation in Anlehnung zu Online-Repositories als Kurzbeschreibung verfasst. Sie umfasst zehn Seiten und enthält Versionshinweise, Abhängigkeitsauflistungen, eine allgemeine Erläuterung und einen Schnelleinstieg. Darüber hinaus bietet sie eine vollständige Auflistung der Konfigurationszeilen aus der CFG-Datei mit Erläuterungen. Aufgrund der Zeiteinsparungen, die z.B. durch die Integration des Black-Box-Tests in die Testroutine des Neukunden entstanden sind, konnte ich die XML-Kommentare so detailliert ausarbeiten, dass hieraus mit dem Tool Doxygen eine zusätzliche Schnittstellendokumentation automatisch generiert werden konnte. Diese wurden dem Unternehmen anschließend mit einem bereinigtem und geordnetem Repository bereit gestellt. Bei den folgend gezeigten Bildern handelt es sich um eine HTML-Variante mit Suchfunktion, Darkmode und Navigationsleiste. Das zweite Bild ist ein Call-Graph der Methode GetResultObject() hieraus, der anhand der XML-Informationen korrekt erzeugt wurde:

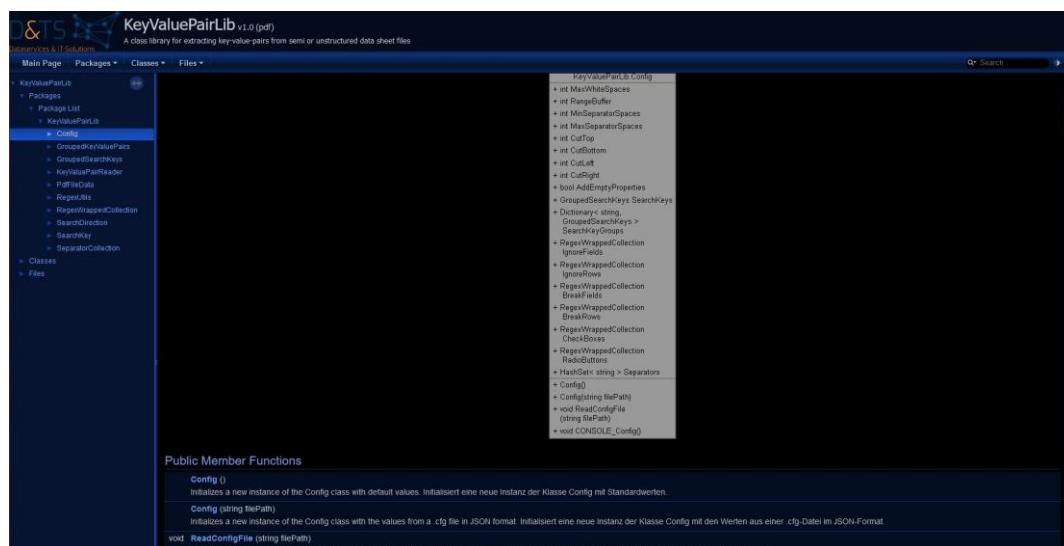


Abb. 26: Oberfläche der generierten Schnittstellendokumentation

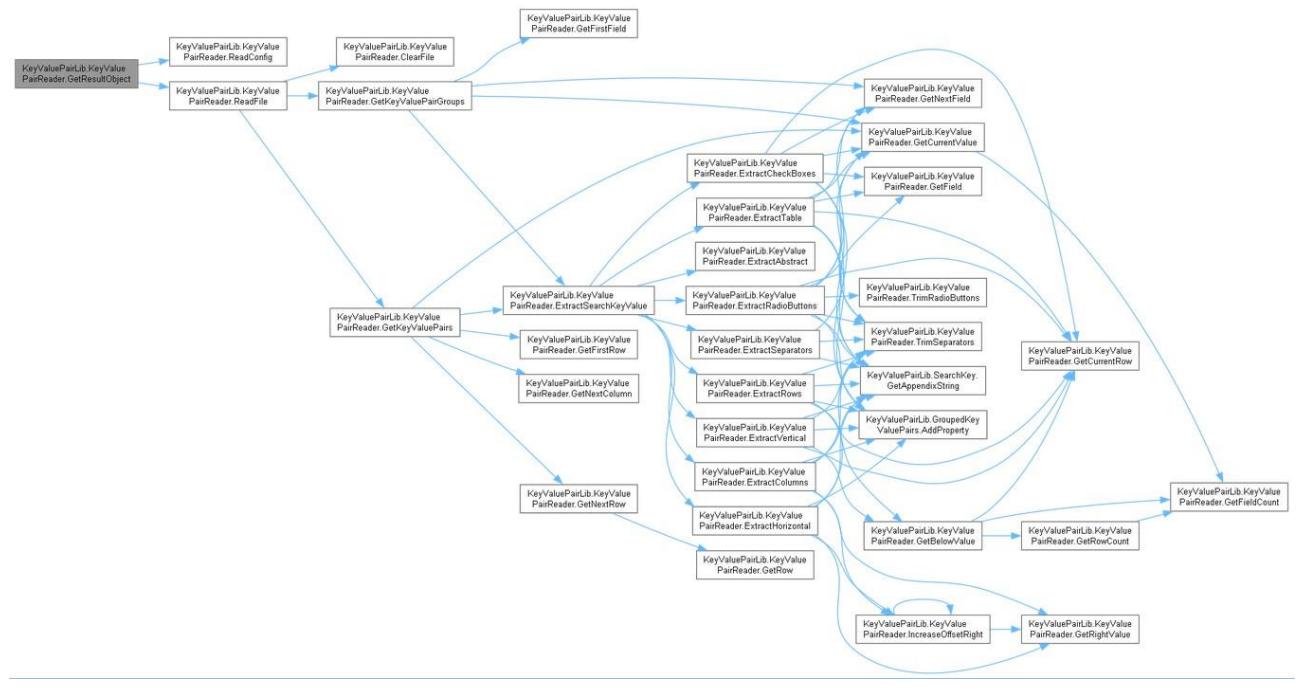


Abb.27: Call-Graph aus Schnittstellendokumentation

5.2 Abnahme des Projekts

Die Abnahme des Projekts wurde erfolgreich an dem geplanten Termin durchgeführt. Nach den Rückmeldungen aus dem kombinierten Black-Box-Test demonstrierte ich in einer Live-Demonstration mithilfe der CONSOLE()-Methode das Auslesen korrekt extrahierter Schlüssel-Wert-Paare aus einer Auswahl unterschiedlicher PDF-Dokumente. Weiterhin sichtete der Abteilungsleiter der IT-Abteilung die Anwender- und Schnittstellendokumentation bzw. das Repository des Projekts um einen vollen Eindruck zu gewinnen. Nach positivem Feedback wurde die Abnahme des Projekts ohne Beanstandungen beendet und es wurde mir umgehend eine Kopie des Protokolls ausgehändigt. Das unterschriebene Protokoll hierzu wurde dem Anhang beigefügt.

5.3 Selbsterklärung

Die unterschriebene Selbsterklärung über die selbstständige Durchführung des Projektes wurde dem Anhang beigefügt.

5.4 Datenschutz

Um der DSGVO zu entsprechen wurden einige Stellen geschwärzt, welche Rückschlüsse auf Personen, Unternehmen oder deren Daten ermöglicht hätten.

6 Anhang

6.1 Abbildungsverzeichnis

Abb. 1: Ursprünglicher Algorithmus	14
Abb. 2: UML-Klassendiagramm - KeyValuePairReader	17
Abb. 3: UML-Klassendiagramm - PdfFileData.....	18
Abb. 4: UML-Klassendiagramm - GroupedKeyValuePairs	19
Abb. 5: UML-Klassendiagramm - Config.....	20
Abb. 6: UML-Aktivitätsdiagramm - ReadFile()	21
Abb. 7: UML-Aktivitätsdiagramm - GetKeyValuePairs() / GetKeyValuePairGroups()	22
Abb. 8: UML-Aktivitätsdiagramm – Rückgabemöglichkeiten der Klasse GroupedKeyValuePairs	23
Abb. 9: UML-Aktivitätsdiagramm – ReadConfig()	23
Abb. 10: UML-Klassendiagramm – GroupedSearchKeys	24
Abb. 11: UML-Klassendiagramm – SearchKey.....	25
Abb. 12: UML-Klassendiagramm – SearchDirection	26
Abb. 13: UML-Klassendiagramm – RegexWrappedCollection.....	27
Abb. 14: UML-Klassendiagramm – SeparatorCollection.....	28
Abb. 15: UML-Klassendiagramm – RegexUtils	28
Abb. 16: UML-Klassendiagramm – Klassendiagramms (verkürzt) zu Assoziationsbeziehungen.....	29
Abb. 17: UML-Sequenzdiagramm zum Algorithmus der Datenextraktion.....	30
Abb. 18: UML-Aktivitätsdiagramm zum Ablauf der Klassenfunktionalitäten.....	31
Abb. 19: Quelltext der Methode ExtractSearchKeyValue() aus KeyValuePairReader	33
Abb. 20: Quelltext der Methode GetFieldTable() aus PdfFileData.....	33
Abb. 21: Quelltextauszug der Klasse GroupedKeyValuePairs	34
Abb. 22: Quelltext der Methode ReadConfigFile() aus Config.....	34
Abb. 23: Quelltextauszug der Methode ReadFile() aus KeyValuePairReader	35
Abb. 24: Oberflächen des ClassCockpit Ticketsystems	36
Abb. 25: Konsolenausgabe der Methode CONSOLE() aus KeyValuePairReader	37

6.2 Quellenverzeichnis

<https://chat.openai.com/> (aufgerufen zwischen dem 18.09.2023 und dem 27.09.2023)

<https://github.com/e-iceblue/Spire.PDF-for-.NET> (aufgerufen zwischen dem 18.09.2023 und dem 27.09.2023)

<https://github.com/JamesNK/Newtonsoft.Json> (aufgerufen zwischen dem 18.09.2023 und dem 27.09.2023)

<https://www.ihk.de/blueprint/servlet/resource/blob/5406028/9a381c7287b7c845d86488ba1c15862d/i-t-handreichungen-neue-vo-data.pdf> (aufgerufen am 20.10.2023)

6.3 Selbsterklärung

(Fortfolgende Seiten in Reihenfolge)

6.4 Abnahmeprotokoll

(Fortfolgende Seiten in Reihenfolge)

6.5 Anwenderdokumentation

(Fortfolgende Seiten in Reihenfolge)

6.6 Große Abbildungen

- UML-Aktivitätsdiagramm zum Ablauf der Klassenfunktionalitäten
- UML-Klassendiagramm – GroupedSearchKeys
- UML-Klassendiagramm - KeyValuePairReader
- UML-Klassendiagramm - PdfFileData
- UML-Klassendiagramm – SeparatorCollection
- UML-Sequenzdiagramm zum Algorithmus der Datenextraktion

(Fortfolgende Seiten in Reihenfolge)

Persönliche Erklärung

Erklärung des Prüfungsteilnehmers / der Prüfungsteilnehmerin:

Ich versichere durch meine Unterschrift, dass ich das betriebliche Projekt und die dazugehörige Dokumentation selbstständig und ohne fremde Hilfe angefertigt und alle Stellen, die ich wörtlich oder annähernd wörtlich aus Veröffentlichungen entnommen habe, als solche kenntlich gemacht habe. Die Arbeit hat in dieser Form keiner anderen Prüfungsinstitution vorgelegen.

Dortmund, 07.09.23

Ort und Datum



Unterschrift

Erklärung des Ausbildungsbetriebes / Praktikumsbetriebes:

Wir versichern, dass der betriebliche Auftrag wie in der Dokumentation dargestellt, in unserem Unternehmen realisiert worden ist.

02 336 42828-43

Telefon/Durchwahl

Jan-Uwe Ballauf

Ansprechpartner

Schwelln 19.03.2023

Ort und Datum



Unterschrift und Firmenstempel

D&TS GmbH

Data Management & Technical Services



Wilhelmstraße 41-43 • D-58332 Schwelm

Tel.: 02336 42828-0 * Fax: 02336 42828-28

E-Mail: info@dundts.com

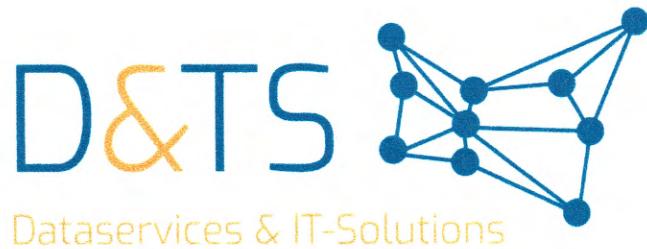
Website: www.dundts.com

Die unterschriebene „Persönliche Erklärung“ ist der Online-Version nur hinzuzufügen, wenn keine ausgedruckten Exemplare der Dokumentation angefordert worden sind!

Abnahmeprotokoll

Klassenbibliothek zur Extraktion von Daten aus PDF-Dateien in .NET 7.0

27.09.2023



1 Teilnehmer

Name	Firma	Funktion
Jan-Uwe Ballauf	D&TS GmbH	Projektbetreuer
Patrick Hildebrandt	D&TS GmbH	Softwareentwickler

2 Ergebnisse der Abnahmeprüfung

Das umgesetzte Projekt zur Programmierung einer Klassenbibliothek zum Parsen von PDF-Dateien als Key-Value-Pairs in .NET 7.0 wurde erfolgreich getestet und abgenommen.

Der Umgang und die Funktionalität konnten anhand von Kunden-Dokumenten getestet werden.

Die Daten aus den Dokumenten konnten erfolgreich ausgelesen und weiterverarbeitet werden.

Die zur Verfügung gestellte Schnittstellen-Dokumentation ist umfangreich und beinhaltet alle entsprechenden Funktionalitäten.

Eine zusätzliche Anwender-Dokumentation erleichtert einen schnellen Einstieg in die Benutzung der Bibliothek.

Das Projekt ist hiermit vollständig erfüllt worden.

3 Genehmigung und Freigabe

Titel	Funktion	Unterschrift
Auftragnehmer	Patrick Hildebrand	
Auftraggeber	D&TS GmbH	

Anwenderdokumentation

KeyValuePairLib

Version: 1.0 (pdf)



Inhaltsverzeichnis

Inhaltsverzeichnis	2
Historie der Versionen	2
1 Allgemeines	3
1.1 Anwendungsszenarien	3
1.2 Unterstützte Dateitypen	3
1.3 Weiterentwicklung	3
1.4 Abhängigkeiten	3
1.4.1 Backend	3
1.4.2 Frontend	3
2 Anwendung	4
2.1 Schnellstart	4
2.2 CONSOLE	4
2.3 Expliziter Zugriff auf Eigenschaften	5
2.4 Rückgabe	5
3 Konfiguration	7

Historie der Versionen

Version	Datum	Autor	Änderungsgrund / Bemerkungen
1.0	27.09.2020	Patrick Hildebrandt	PDF-Funktionalität implementiert
1.1			
1.2			
1.3			

1 Allgemeines

1.1 Anwendungsszenarien

Diese Klassenbibliothek extrahiert Schlüssel-Wert-Paare aus semi- oder unstrukturierten Datenblättern und stellt die Ergebnisse als generische Eigenschaften zur weiteren Verarbeitung bereit. Zusätzlich können diese Eigenschaften als Objekte zurückgegeben werden.

Generell sind die integrierten Typen der Klassenbibliothek öffentlich gestaltet, um einfache Refaktorisierung und Steuerung durch übergeordnete Projekte zu ermöglichen.

1.2 Unterstützte Dateitypen

In der ersten Version unterstützt die Klassenbibliothek den Dateityp PDF. Der Algorithmus basiert auf dem Prinzip, dass Felder in PDF-Dateien anhand ihrer räumlichen Trennung voneinander wie Zellen interpretiert und extrahiert werden.

Dieses Prinzip grenzt sich klar ab von den im PDF-Standard vorgesehenen Formularfeld-Attributen und ist somit auch nicht von diesen abhängig.

1.3 Weiterentwicklung

Die Dateitypen CSV, XLSX und XML sind bereits in grundlegenden Strukturen berücksichtigt worden. Da PDF-Dateien bei der Verarbeitung in eine tabellarische Form überführt und als DataTable gespeichert werden, ermöglicht dies für zukünftige Erweiterungen eine verhältnismäßig einfache Adaption des Extraktionsalgorithmus auf die oben genannten Dateitypen.

Dies hat nur Gültigkeit, sofern die entsprechenden Dateien einen typografischen Aufbau von der Art eines Datenblattes aufweisen und einzelne Materialien mit ihren Eigenschaften beschreiben.

1.4 Abhängigkeiten

1.4.1 Backend

Komponente:	Abhängigkeiten:
KeyValuePairLib.sln	.NET 7.0
Config.cs	Json.NET 13.0.3 (MIT License)
PdfFileData.cs	Spire.PDF 9.7.17 (free trial without time limitation: https://github.com/e-iceblue/Spire.PDF-for-.NET/blob/master/README.md)

1.4.2 Frontend

Komponente:	Abhängigkeiten:

2 Anwendung

2.1 Schnellstart

Konfigurationsdatei:

```
string configDefence = @"..\..\..\..\..\ConsoleTest\config\configDefence.cfg";
```

Quelldatei:

```
string defence1 = @"C:\Users\PatrickHildebrandt\Desktop\-----\pdf";
```

Instanziierung:

```
KeyValuePairReader kvprDefence1 = new(configDefence, defence1);
```

```
string defence1 = @"C:\Users\PatrickHildebrandt\Desktop\-----\pdf";
string configDefence = @"..\..\..\..\..\ConsoleTest\config\configDefence.cfg";
KeyValuePairReader kvprDefence1 = new(configDefence, defence1);
```

2.2 CONSOLE

Diese Methode stellt ein Debugging- und Inspektions-Tool dar und gibt die Ergebnisse der jeweiligen Sub-Methoden in Reihenfolge der Verarbeitungsschritte in der Konsole aus. Ohne Parameter werden alle Methoden ausgegeben:

```
kvprDefence1.CONSOLE();
kvprDefence1.CONSOLE(0, 11);
```

```
0 = Config.CONSOLE_Config()
1 = Separators.CONSOLE_SeparatorCollection()
2 = _fileData.CONSOLE_pdfDocString()
3 = _fileData.CONSOLE_pdfDocEmptyRowsRemoved()
4 = _fileData.CONSOLE_pdfDocFieldsDelimited()
5 = _fileData.CONSOLE_Rows()
6 = _fileData.CONSOLE_fields()
7 = _fileData.CONSOLE_indexedFields()
8 = _fileData.CONSOLE_FieldTable()
9 = CONSOLE_FieldTableMethods()
10 = CONSOLE_FieldTableMethodsExpand()
11 = KeyValuePairPairs.CONSOLE_GroupedKeyValuePairs()
12 = KeyValuePairGroups.CONSOLE_GroupedKeyValuePairs()
```

```
CONSOLE_Eingang / Input::
CONSOLE_GroupedKeyValuePairs:
#####
"Eingangsspannung (Ue) / Input voltage "Ein" / "On"°
"V
"Eingangsstrom / Input current°
"V
"Überspannungsschutz / Overvoltage protection°
"V
"Ansteueranzeige / Control display°
"n
"Polung / Polarity°
"p
```

2.3 Expliziter Zugriff auf Eigenschaften

Nach der Instanziierung können die Ergebnisse der Extraktion als Eigenschaften explizit angesprochen werden:

```
Console.WriteLine(kvprEnergie1.KeyValuePairs["Prozessauswahl *"]);
Console.WriteLine(kvprEnergie1.KeyValuePairs["Formular-ID"]);
Console.WriteLine(kvprEnergie1.KeyValuePairs["Materialnummer"]);
Console.WriteLine(kvprEnergie1.KeyValuePairs["Materialart *"]);
```

```
string energie1 = @"..\..\..\..\..\ConsoleTest\demoPDFs\energie1.Pdf";
string configEnergie = @"..\..\..\..\..\ConsoleTest\config\configEnergie.cfg";
KeyValuePairReader kvprEnergie1 = new(configEnergie, energie1);
Console.WriteLine(kvprEnergie1.KeyValuePairs["Prozessauswahl *"]);
Console.WriteLine(kvprEnergie1.KeyValuePairs["Formular-ID"]);
Console.WriteLine(kvprEnergie1.KeyValuePairs["Materialnummer"]);
Console.WriteLine(kvprEnergie1.KeyValuePairs["Materialart *"]);
```



2.4 Rückgabe

Über den expliziten Zugriff auf Eigenschaften hinaus stehen mehrere Möglichkeiten zur Rückgabe der Ergebnisse zur Verfügung. Zwecks Vollständigkeit werden hier alle Möglichkeiten anhand ihres Quelltextes aufgeführt:

```
public object this[string name]
{
    get => (_properties as IDictionary<string, object>)[name];
    set => (_properties as IDictionary<string, object>)[name] = value;
}
```

```
25 Verweise
public class GroupedKeyValuePairs
{
    // ! FIELDS
    fields

    // ! PROPERTIES
    #region properties
    /// <summary> Gets or sets the value as object associated with the specified key ... 12 Verweise
    public object this[string name]
    {
        get => (_properties as IDictionary<string, object>)[name];
        set => (_properties as IDictionary<string, object>)[name] = value;
    }
    #endregion

    public IDictionary<string, object> GetProperties() => _properties as IDictionary<string, object>;
    /// <summary> Gets a Dictionary of all key-value properties in the collection. G ...
    1 Verweis
    public IDictionary<string, object> GetProperties() => _properties as IDictionary<string, object>;
```

```
public dynamic GetDynamicObject() => _properties;
```

```
/// <summary> Gets the dynamic object representing the key-value properties. Gib ...
0 Verweise
public dynamic GetDynamicObject() => _properties;
```

```
public (GroupedKeyValuePairs, Dictionary<string, Dictionary<int,
GroupedKeyValuePairs>>) GetResultObject()
{
    if (_filePath == "") throw new FileNotFoundException("No file to
process set. Keine Datei zur Verarbeitung gesetzt.");
    return (KeyValuePairs, KeyValuePairGroups);
}
```

```
/// <summary> This class extracts key-value pairs from files with semi-structure ...
14 Verweise
public class KeyValuePairReader
```

```
/// <summary> This method returns the results of the extraction as a tuple of Ke ...
0 Verweise
public (GroupedKeyValuePairs, Dictionary<string, Dictionary<int, GroupedKeyValuePairs>>) GetResultObject()
{
    if (_filePath == "") throw new FileNotFoundException("No file to process set. " +
        "Keine Datei zur Verarbeitung gesetzt.");
    return (KeyValuePairs, KeyValuePairGroups);
}
```

```
public (GroupedKeyValuePairs, Dictionary<string, Dictionary<int,
GroupedKeyValuePairs>>) GetResultObject(string filePath)
{
    ReadFile(filePath);
    return (KeyValuePairs, KeyValuePairGroups);
}
```

```
/// <summary> This method returns the results of the extraction as a tuple of Ke ...
0 Verweise
public (GroupedKeyValuePairs, Dictionary<string, Dictionary<int, GroupedKeyValuePairs>>) GetResultObject(
    string filePath)
{
    ReadFile(filePath);
    return (KeyValuePairs, KeyValuePairGroups);
}
```

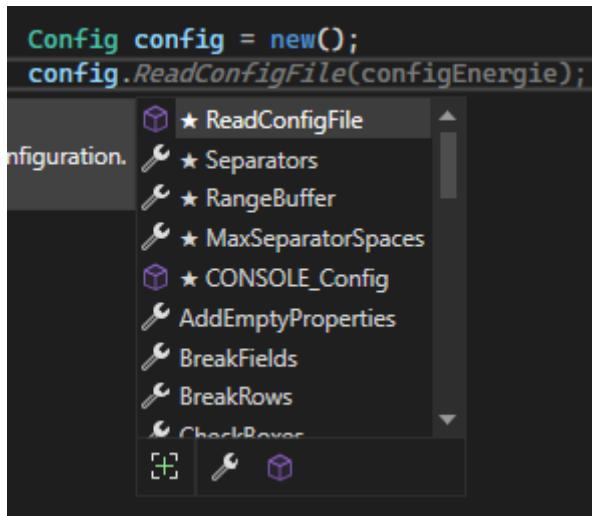
```
public (GroupedKeyValuePairs, Dictionary<string, Dictionary<int,
GroupedKeyValuePairs>>) GetResultObject(string filePath, string configPath)
{
    ReadConfig(configPath);
    ReadFile(filePath);
    return (KeyValuePairs, KeyValuePairGroups);
}
```

```
/// <summary> This method returns the results of the extraction as a tuple of Ke ...
0 Verweise
public (GroupedKeyValuePairs, Dictionary<string, Dictionary<int, GroupedKeyValuePairs>>) GetResultObject(
    string filePath, string configPath)
{
    ReadConfig(configPath);
    ReadFile(filePath);
    return (KeyValuePairs, KeyValuePairGroups);
}
```

3 Konfiguration

Im Folgenden wird die Bedeutung der einzelnen Zeilen der Konfigurationsdatei anhand des Templates „configTemplate.cfg“ erläutert. Hierbei handelt es sich um eine Konfigurationsdatei im JSON-Format, welche durch Konstruktoren und Methoden eingelesen werden kann. Die hierbei erläuterten Zeilen bilden die Eigenschaften der Klasse Config.cs ab. Diese Eigenschaften können äquivalent hierzu ebenfalls mit der Punktnotation von übergeordneten Projekten angesprochen werden:

```
Config config = new();
```



configTemplate.cfg:

```
{
```

```
  "MaxWhiteSpaces": 3,
```

Hier wird definiert ab wie vielen Leerzeichen zwischen Feldern diese als leerer Raum interpretiert werden. Standard ist 3, Minimum ist 2, sollte jedoch vermieden werden, da dies auch Eingabefehler von doppelten Leerzeichen berücksichtigt.

```
  "RangeBuffer": 3,
```

Hier wird ein Puffer definiert, wie viele Zeichen bei horizontalem Versatz zwischen vertikalen Schlüssel- und Wert-Feldern toleriert werden. Standard ist 3. Ein zu großer Puffer kann benachbarte Felder miteinbeziehen.

```
  "MinSeparatorSpaces": 2,
```

Hier wird die minimale Anzahl von Trennzeichen definiert für Schlüssel-Wert-Paare, welche durch eine Sequenz von Trennzeichen in einem Feld verbunden sind. Standard ist 2. Höherer Wert steigert die Performance.

```
  "MaxSeparatorSpaces": 30,
```

Hier wird die maximale Anzahl von Trennzeichen definiert für Schlüssel-Wert-Paare, welche durch eine Sequenz von Trennzeichen in einem Feld verbunden sind. Standard ist 30. Niedrigerer Wert steigert die Performance.

```
  "CutTop": 0,
```

Hier wird definiert, wie viele Pixel von PDF-Seiten oberhalb abgeschnitten werden. Standard ist 0.

```
  "CutBottom": 0,
```

Hier wird definiert, wie viele Pixel von PDF-Seiten unterhalb abgeschnitten werden. Standard ist 0.

```
  "CutLeft": 0,
```

Hier wird definiert, wie viele Pixel von PDF-Seiten links abgeschnitten werden. Standard ist 0.

```
"CutRight": 0,
```

Hier wird definiert, wie viele Pixel von PDF-Seiten rechts abgeschnitten werden. Standard ist 0.

```
"AddEmptyProperties": true,
```

Hier wird definiert, ob Eigenschaften ohne gefundene Werte zu den Ergebnissen hinzugefügt werden sollen oder nicht. Standard ist true.

```
"SearchKeys": [
```

Hier werden die SearchKey-Objekte definiert zu denen Werte aus Dateien extrahiert werden.

```
{"Key": "Schlüssel1", "Direction": 0, "Appendix": ["Anhang"]},
```

Die Suche wird horizontal durchgeführt. Zugewiesen dem Wert 0.

```
{"Key": "Schlüssel2", "Direction": 1, "Appendix": ["Anhang"]},
```

Die Suche wird vertikal durchgeführt. Zugewiesen dem Wert 1.

```
{"Key": "Schlüssel3", "Direction": 2, "Appendix": ["Anhang"]},
```

Die Suche wird über mehrere Spalten durchgeführt. Zugewiesen dem Wert 2.

```
{"Key": "Schlüssel4", "Direction": 4, "Appendix": ["Anhang"]},
```

Die Suche wird über mehrere Zeilen durchgeführt. Zugewiesen dem Wert 4.

```
{"Key": "Schlüssel5", "Direction": 8, "Appendix": ["Anhang"]},
```

Die Suche erfolgt über Spalten und Zeilen, ähnlich einer Tabelle. Zugewiesen dem Wert 8.

```
{"Key": "Schlüssel6", "Direction": 16, "Appendix": ["Anhang"]},
```

Die Suche erfolgt über die Erkennung von Checkboxen in und nach Schlüssel-Feldern. Zugewiesen dem Wert 16.

```
{"Key": "Schlüssel7", "Direction": 32, "Appendix": ["Anhang"]},
```

Die Suche erfolgt über die Erkennung von Radio-Buttons in Wert-Feldern. Zugewiesen dem Wert 32.

```
{"Key": "Schlüssel7", "Direction": 64, "Appendix": ["Anhang"]},
```

Die Suche erfolgt über die Erkennung von Trennzeichen-Sequenzen in Feldern. Zugewiesen dem Wert 64.

```
{"Key": "Schlüssel8", "Direction": 128, "Appendix": ["Anhang"]}
```

Reserviert für die abstrakte Implementierung von Sonderfällen in abgeleiteter Klasse, die nicht mit existierenden Konfigurationsregeln abgebildet werden können. Zugewiesen dem Wert 128.

```
"SearchKeyGroups": {
```

```
  "repeatingProperty1": [
```

Namen für SearchKeyGroups müssen aus Feldern im Dokument ausgewählt werden, die wie eine Überschrift für ihre Schlüssel-Wert-Kombinationen fungieren. Außerdem sind sie notwendig, um identische SearchKeys zu unterscheiden und die Ergebnisse entsprechend zu separieren. Weiterhin lassen sich hiermit auch Kategorien bzw. Abschnitte im Dokument voneinander abgrenzen.

```
{"Key": "Schlüssel1:", "Direction": 0, "Appendix": ["Anhang", "über 2
```

```
Zeilen"]},
```

Unter Anhängen sind Unterschriften von Schlüsseln zu verstehen (z.B. englische Bezeichnung unter deutschem Schlüssel). Anhänge sind notwendig, wenn sie in Ergebnissen benötigt werden. Weiterhin werden sie benötigt, um sich selbst im Dokumentenfluss zu überspringen.

```

    {"Key": "Schlüssel8:", "Direction": 128, "Appendix": ["Anhang", "über 2
Zeilen"]}
],
"repeatingProperty2": [
  {"Key": "Schlüssel1:", "Direction": 0, "Appendix": ["Anhang", "über", "3
Zeilen"]}],

```

Zeilenumbrüche in einem Anhang müssen mit separaten Einträgen für jede Zeile abgebildet werden.

```

  {"Key": "Schlüssel8:", "Direction": 128, "Appendix": ["Anhang", "über", "3
Zeilen"]}
]
},
```

HINWEIS: Ignorieren sollte vorrangig vor Unterbrechungen verwendet werden. Unterbrechungen beenden Operationen, während Ignorieren fortfährt und nach weiteren Werten sucht.

```

"IgnoreFields": [
  " are ignored by skipping "
```

Diese Felder werden dadurch ignoriert, dass ihre enthaltenden Zeilen bzw. ihre Zeichenkette enthaltende Felder übersprungen werden. Speziell für Felder gedacht, die nicht eindeutig identifiziert werden können bzw. ein Vorkommen der Zeichenkette enthalten.

```

],  

"IgnoreRows": [  

  "These rows are ignored by skipping them as a whole."
```

Diese Zeilen werden dadurch ignoriert, dass sie als Ganzes übersprungen werden. Speziell für Überschriften oder Felder gedacht, welche sich durch ihren Versatz zwischen Key- und Value-Zeilen schieben. Diese Zeichenketten müssen aus dem .CONSOLE(5)-Report ausgewählt werden um regulären Ausdrücken zu entsprechen.

```

],  

"BreakFields": [  

  " stop fields are defined here "
```

Hier werden absolute Stoppfelder definiert für Szenarien, bei denen der Algorithmus über mehrfache Zeilen- oder Spalten-Werte hinaus iteriert mangels einer anderen identifizierbaren Abgrenzung.

```

],  

"BreakRows": [  

  "Absolute stop rows are defined here for scenarios where the algorithm
iterates beyond multiple row values."
```

Hier werden absolute Stoppzeilen definiert für Szenarien, bei denen der Algorithmus über mehrfache Zeilenwerte hinaus iteriert mangels einer anderen identifizierbaren Abgrenzung. Diese Zeichenketten müssen aus dem .CONSOLE(5)-Report ausgewählt werden um regulären Ausdrücken zu entsprechen.

```
],  
"Checkboxes": [  
    "✓"  
,  
"RadioButtons": [  
    "○"  
,
```

Checkboxen und Radiobuttons können nur erkannt werden, wenn ihre Zeichen lesbar und hier definiert sind. Checkboxen sind für ein Vorkommen im oder nach dem Schlüssel gedacht, während Radio-Buttons für mehrere Werte gedacht sind, die das Radio-Button-Zeichen enthalten.

```
"Separators": [  
    "=",  
    ":" ,  
    ". "  
,
```

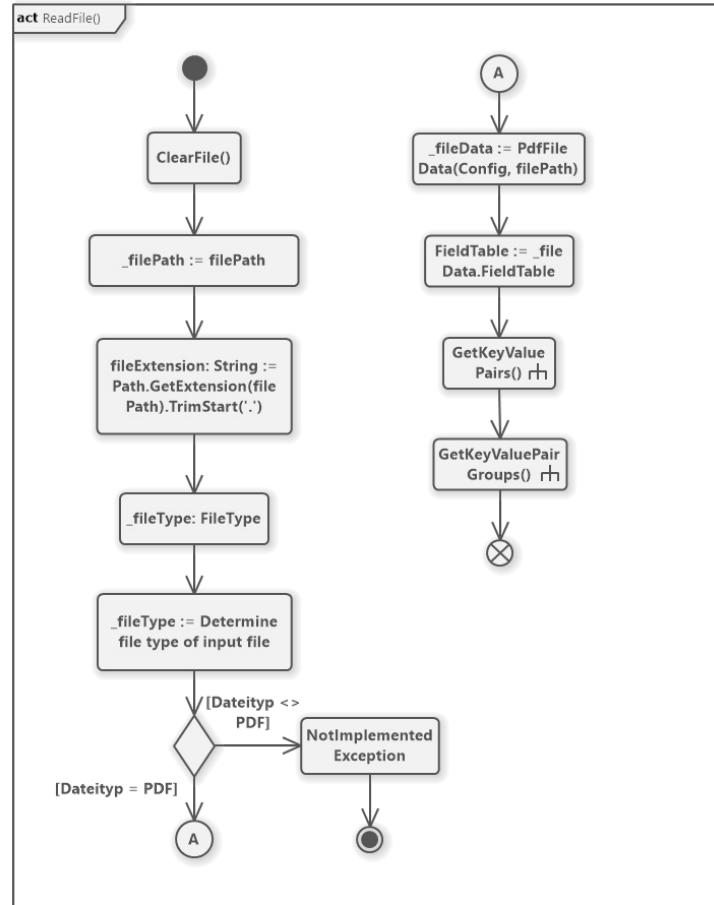
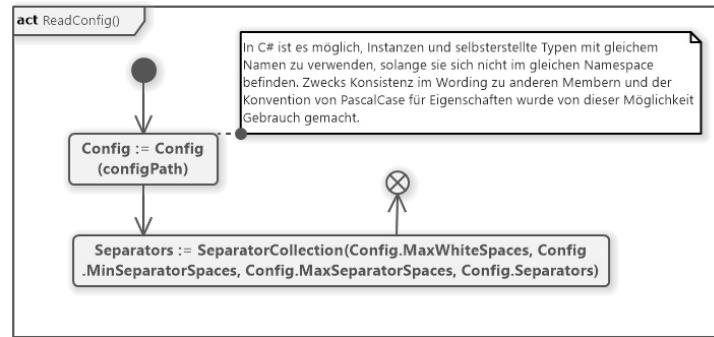
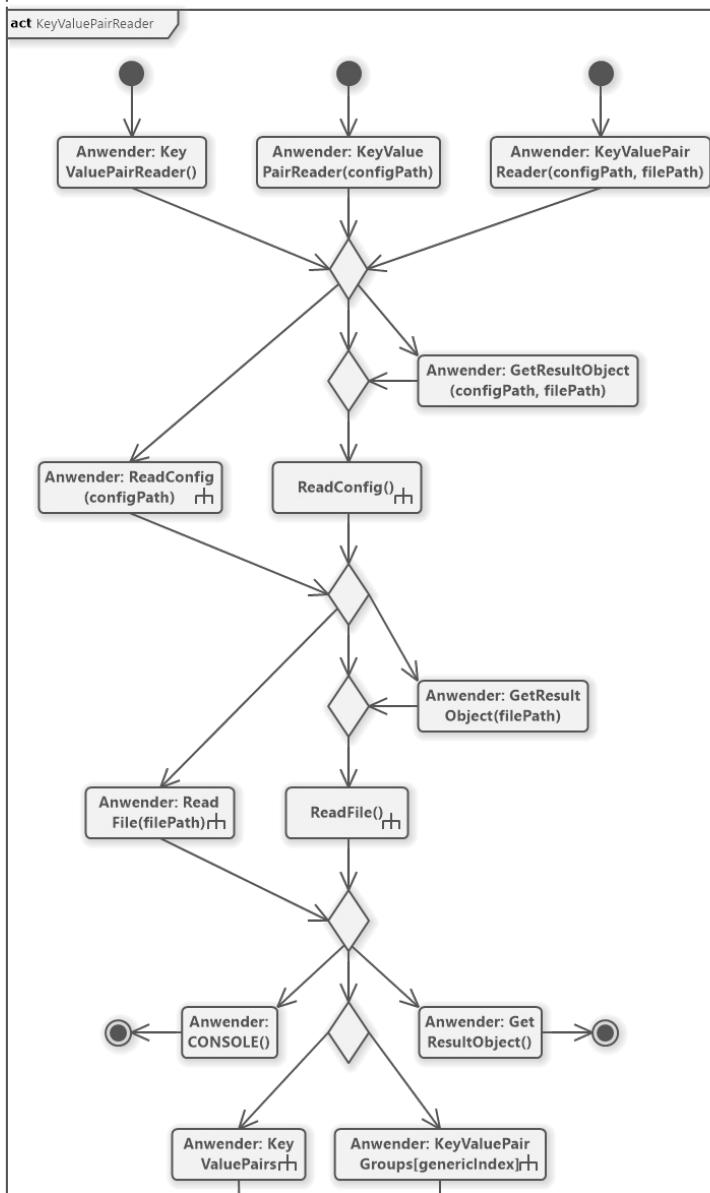
Separatoren stellen die Basis dar für die unter SearchKeys genannten Trennzeichen-Sequenzen. Diese werden automatisch generiert aus den Separatoren und den Konfigurationswerten „**MaxWhiteSpaces**“ „**MinSeparatorSpaces**“ und „**MaxSeparatorSpaces**“.

Beispiele für Trennzeichen-Sequenzen und Separatoren:

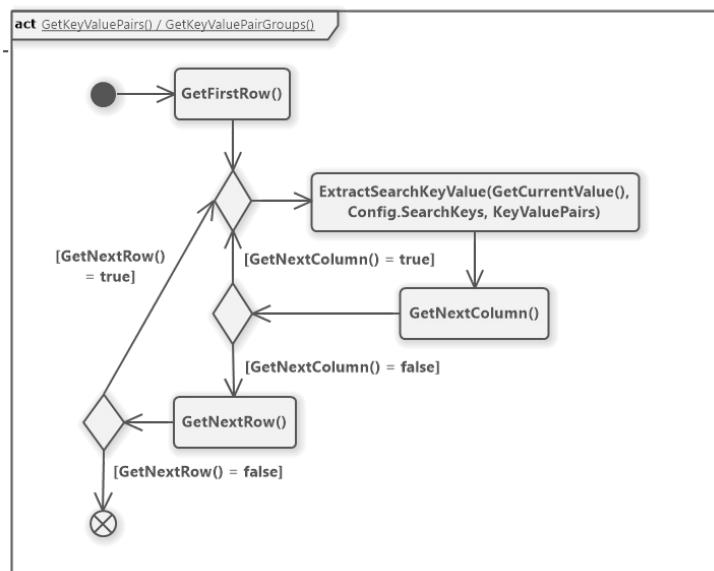
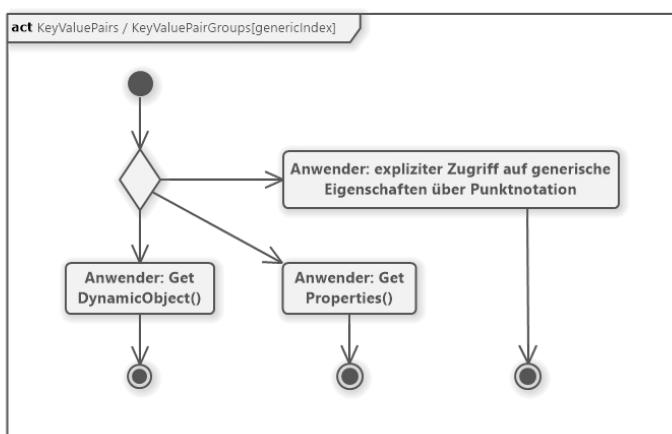
```
'Key: Value' => ':' = Separator  
  
'Key ..... Value' => '.' = Separator
```

```
}
```

Das Hauptdiagramm bildet die unterschiedlichen Pfade bei der Anwendung der Klassenbibliothek ab. Da Pseudocode nicht den Anwender als solchen abbilden kann, wird diesbezüglich von Pseudocode abweichen. Anwenderentscheidungen werden mit "Anwender: " dargestellt. Methodenaufrufe im regulären Algorithmus werden hingegen lediglich mit ihrem Methodennamen dargestellt. Dieser Logik folgend wird auf Bedingungen an Verzweigungspfaden verzichtet, da sie lediglich eine Doppelung der Beschreibung ihrer nachfolgenden Aktionen darstellen würden.



Beide Methoden basieren auf dem gleichen semantischen Prinzip. Deshalb wird hier zwecks einfacher Verständnis repräsentativ die Methode GetKeyValuePairs() abgebildet. Der Unterschied zur Methode GetKeyValuePairGroups() besteht darin, dass innerhalb dieser noch die variierten Sammlungen von Suchschlüsseln bzw. Konfigurationen berücksichtigt und zugeordnet werden.

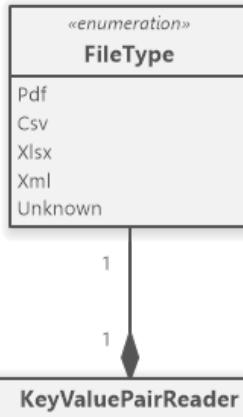


class KeyValuePairLib

HashSet<SearchKey>

GroupedSearchKeys

```
+ «create» GroupedSearchKeys ()  
+ «create» GroupedSearchKeys (keys : IEnumerable<String>, direction : SearchDirection)  
+ «create» GroupedSearchKeys (keys : IEnumerable<String>, direction : Integer)  
+ «create» GroupedSearchKeys (keys : IEnumerable<Tuple<String, String>>, direction : SearchDirection)  
+ «create» GroupedSearchKeys (keys : IEnumerable<Tuple<String, String>>, direction : Integer)  
+ «create» GroupedSearchKeys (keys : IEnumerable<Tuple<String, IEnumerable<String>>>, direction : SearchDirection)  
+ «create» GroupedSearchKeys (keys : IEnumerable<Tuple<String, IEnumerable<String>>>, direction : Integer)  
+ «create» GroupedSearchKeys (keys : IEnumerable<Tuple<String, SearchDirection>>)  
+ «create» GroupedSearchKeys (keys : IEnumerable<Tuple<String, Integer>>)  
+ Add (key : String, direction : SearchDirection): Void  
+ Add (key : String, direction : SearchDirection, appendix : String): Void  
+ Add (key : String, direction : SearchDirection, appendix : IEnumerable<String>): Void  
+ Add (key : String, direction : Integer): Void  
+ Add (key : String, direction : Integer, appendix : String): Void  
+ Add (key : String, direction : Integer, appendix : IEnumerable<String>): Void  
+ AddRange (keys : IEnumerable<String>, direction : SearchDirection): Void  
+ AddRange (keys : IEnumerable<String>, direction : Integer): Void  
+ AddRange (keys : IEnumerable<Tuple<String, String>>, direction : SearchDirection): Void  
+ AddRange (keys : IEnumerable<Tuple<String, String>>, direction : Integer): Void  
+ AddRange (keys : IEnumerable<Tuple<String, IEnumerable<String>>>, direction : SearchDirection): Void  
+ AddRange (keys : IEnumerable<Tuple<String, IEnumerable<String>>>, direction : Integer): Void  
+ AddRange (keys : IEnumerable<String, SearchDirection>): Void  
+ AddRange (keys : IEnumerable<String, Integer>): Void
```



```

- _currentRow: Integer
- _currentColumn: Integer
- _currentField: Integer
- _lookupRow: Integer
- lookupColumn: Integer
- _currentField: Integer
- _filePath: String
- _fileData: Dynamic
- _regexUtils: RegexUtils {readOnly}
+ FieldTable: DataTable
+ KeyValuePairPairs: GroupedKeyValuePairPairs
+ KeyValuePairGroups: Dictionary<String, Dictionary<Integer, GroupedKeyValuePairPairs>>
+ Config: Config
+ Separators: SeparatorCollection

+ «create» KeyValuePairReader ()
+ «create» KeyValuePairReader (configPath : String)
+ «create» KeyValuePairReader (configPath : String, filePath : String)
+ ReadConfig (configPath : String): Void
+ ReadFile (filePath : String): Void
- ClearFile (): Void
- GetKeyValuePairPairs (): Void
- GetKeyValuePairGroups (): Void
+ ExtractSearchKeyValue (currentField : String, searchKeys : GroupedSearchKeys, keyValuePairs : GroupedKeyValuePairPairs): Boolean
- ExtractHorizontal (searchKey : SearchKey, keyValuePairs : GroupedKeyValuePairPairs): Boolean
- ExtractVertical (searchKey : SearchKey, keyValuePairs : GroupedKeyValuePairPairs): Boolean
- ExtractColumns (searchKey : SearchKey, keyValuePairs : GroupedKeyValuePairPairs): Boolean
- ExtractRows (searchKey : SearchKey, keyValuePairs : GroupedKeyValuePairPairs): Boolean
- ExtractTable (searchKey : SearchKey, keyValuePairs : GroupedKeyValuePairPairs): Boolean
- ExtractCheckboxes (searchKey : SearchKey, keyValuePairs : GroupedKeyValuePairPairs): Boolean
- ExtractRadioButtons (searchKey : SearchKey, keyValuePairs : GroupedKeyValuePairPairs): Boolean
- ExtractSeparators (searchKey : SearchKey, keyValuePairs : GroupedKeyValuePairPairs): Boolean
- ExtractAbstract (searchKey : SearchKey, keyValuePairs : GroupedKeyValuePairPairs): Boolean {virtual}
- TrimRadioButtons (input : String): String
- TrimSeparators (input : String): String
- IncreaseOffsetRight (offset : Integer, searchKey : SearchKey): Void
+ GetFieldCount (): Integer
+ GetFirstField (): Boolean
+ GetField (field : Integer): Boolean
+ GetNextField (): Boolean
+ GetRowCount (): Integer
+ GetCurrentRow (): Integer
+ getFirstRow (): Boolean
+ GetRow (row : Integer): Boolean
+ GetNextRow (): Boolean
+ GetNextColumn (): Boolean
+ GetCurrentValue (): String
+ GetBelowValue (offset : Integer): String
+ GetRightValue (offset : Integer): String
+ GetResultObject (): Tuple<GroupedKeyValuePair, Dictionary<String, Dictionary<Integer, GroupedKeyValuePairPairs>>>
+ GetResultObject (filePath : String): Tuple<GroupedKeyValuePair, Dictionary<String, Dictionary<Integer, GroupedKeyValuePairPairs>>>
+ GetResultObject (filePath : String, configPath : String): Tuple<GroupedKeyValuePair, Dictionary<String, Dictionary<Integer, GroupedKeyValuePairPairs>>>
+ CONSOLE_FieldTableMethods (): Void
+ CONSOLE_FieldTableMethodsExpand (): Void
+ CONSOLE (methods : Params Integer [*]): Void
  
```

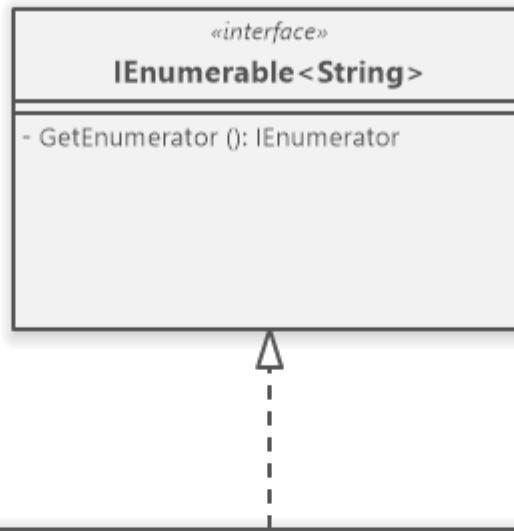
class KeyValuePairLib

PdfFileData

- _pdfDocString: String
- _pdfDocEmptyRowsRemoved: String
- _pdfDocFieldsDelimited: String
- _fields: Dictionary<Integer, List<String>>
- _indexedFields: Dictionary<Tuple<String, Integer, Integer>>
- _config: Config {readOnly}
- _regexUtils: RegexUtils {readOnly}
+ Rows: Dictionary<Integer, String>
+ FieldTable: DataTable

+ «create» PdfFileData ()
+ «create» PdfFileData (config : Config)
+ «create» PdfFileData (configPath : String)
+ «create» PdfFileData (config : Config, filePath : String)
+ «create» PdfFileData (configPath : String, filePath : String)
+ ReadFile (filePath : String): Void
- GetPdfPageString (pdfPage : PdfPageBase): String
- GetPdfDocString (pdfDoc : PdfDocument): String
- GetPdfDocEmptyRowsRemoved (pdfDocStr : String): String
- GetPdfDocFieldsDelimited (pdfDocEmptyRowsRemoved : String): String
- GetRows (pdfDocEmptyRowsRemoved : String): Dictionary<Integer, String>
- GetFields (pdfDocFieldsDelimited : String): Dictionary<Integer, List<String>>
- GetIndexedFields (rows : Dictionary<Integer, String>, fields : Dictionary<Integer, List<String>>): Dictionary<Integer, List<Tuple<String, Integer, Integer>>>
- GetFieldTable (indexedFields : Dictionary<Integer, List<Tuple<String, Integer, Integer>>): DataTable
+ CONSOLE_pdfDocString (): Void
+ CONSOLE_pdfDocEmptyRowsRemoved (): Void
+ CONSOLE_pdfDocFieldsDelimited (): Void
+ CONSOLE_Rows (): Void
+ CONSOLE_fields (): Void
+ CONSOLE_indexedFields (): Void
+ CONSOLE_FieldTable (): Void

class KeyValuePairLib



```
- _maxWhiteSpaces: Integer {readOnly}  
- _minSeparatorSpaces: Integer {readOnly}  
- _maxSeparatorSpaces: Integer {readOnly}  
- _separators: HashSet<String> {readOnly}  
- _separatorVariations: HashSet<String> {readOnly}  
- _regexUtils: RegexUtils {readOnly}  
+ Separators: HashSet<String>  
+ SeparatorVariations: HashSet<String>  
  
+ «create» SeparatorCollection ()  
+ «create» SeparatorCollection (separators : IEnumerable<String>)  
+ «create» SeparatorCollection (maxWhiteSpaces : Integer, minSeparatorSpaces : Integer, maxSeparatorSpaces : Integer)  
+ «create» SeparatorCollection (maxWhiteSpaces : Integer, minSeparatorSpaces : Integer, maxSeparatorSpaces : Integer, separators : IEnumerable<String>)  
+ Add (separator : String): Void  
+ AddRange (separators : IEnumerable<String>): Void  
- AddSeparatorVariations (separator : String): Void  
+ CONSOLE_SeparatorCollection (): Void  
+ GetEnumerator (): IEnumerator<String>
```

