Office Communication Server Adressbucherweiterung

Dokumentation OCSAddressExtension

0. Dokumentinformationen

0.1. Änderungsgeschichte

Datum	*Version*	*Änderung*	*Autor*	*Qualitätssicherung*
18.12.2008	1.0	Finalversion erstellt	Müller	Dietrich

0.2. Inhalt

0. Dokumentinformationen 2
0.1. Änderungsgeschichte 2
0.2. Inhalt 3
1. OCSAddrBookExtension 4
2. Dokumentation der Test 32
3. ABFileBeschreibung 47

Office Communication Server Adressbucherweiterung

Dokumentation

0. Dokumentinformationen

0.1. Änderungsgeschichte

Datum	*Version*	*Änderung*	*Autor*	*Qualitätssicherung*
30.9.2009	1.0	erstellt	Dietrich	
4.12.2009	1.1	Analyse begonnen	Dietrich	
2.12.2009	1.2	Analyse abgeschlossen	Dietrich	
2.12.2009	1.3	SAD begonnen	Dietrich	
2.12.2009	1.4	SAD abgeschlossen	Dietrich	Müller
16.12.2009	1.5	Dokument überarbeitet	Dietrich	
17.12.2009	2.0	Finale Version	Dietrich	

0.2. Inhalt

0. Dokumentinformationen 2
0.1. Änderungsgeschichte 2
0.2. Inhalt 3
1. Anforderungsspezifikation 5
1.1. Allgemeine Beschreibung 5
1.1.1. Produktfunktion 5
1.1.2. Benutzer Charakteristik 5
1.1.3. Einschränkungen 5
1.1.4. Abhängigkeiten 5
1.2. Spezifische Anforderungen 5
1.2.1. Funktionale Anforderungen 5
1.2.1.1. Richtigkeit 5
1.2.1.2. Interoperabilität 5
1.2.1.3. Sicherheit 5
1.2.2. Bedienbarkeit 5
1.2.3. Kurze Lernzeit 5
1.2.4. Zuverlässigkeit 5
1.2.4.1. Fehlertoleranz 5
1.2.5. Wartbarkeit 5
1.2.5.1. Logging 5
1.2.5.2. Installation 5
1.2.6. Schnittstellen 5
1.2.6.1. Benutzerschnittstelle 5
1.2.6.2. Softwareschnittstellen 6
1.2.7. Datenbankschnittstelle 6
1.2.8. Lizenzanforderungen 6
1.2.9. Verwendete Standards 6
1.3. Funktionale Anforderungen 6
1.3.1. Feature Liste 6
1.3.1.1. Einlesen der Daten 6
1.3.1.2. Dekomprimierung 6
1.3.1.3. Aufbereitung der Daten 6
1.3.1.4. Hinzufügen von neuen Kontaktinformationen 6
1.3.1.5. Generierung eines neuen Files 6
1.3.1.6. Komprimierung 6
2. Domainanalyse 7
2.1. Strukturdiagramm 7
2.2. Konzeptbeschreibung 7
2.3. Systemsequenzdiagramm 8
2.3.1. SSD Black Box Betrachtung 8
2.3.2. Beschreibung 8
2.3.3. SSD OCSFilegenerierung 9
2.3.3.1. Beschreibung 9
2.3.4. Systemoperationen 9
2.3.4.1. CO1: GetKontaktFile 9
2.3.4.2. CO2: AddNewContacts 9
2.3.4.3. CO3: WriteNewFilesBack 10
3. Softwarearchitekturdokument 10
3.1. Umgebung 10
3.2. Architektonische Ziele & Einschränkungen 10
3.2.1. Ziele 10
3.2.2. Einschränkungen 10
3.3. Architektonische Entscheidungen 11
3.3.1. Reader/Writer 11
3.3.1.1. Faktoren 11
3.3.1.2. Lösung 11
3.3.1.3. Erwogene Alternativen 11
3.3.1.4. Sequenz Diagramm Reader/Writer 11
3.3.2. Konstanten 12

3.3.2.1. Faktoren 12
3.3.2.2. Lösung 12
3.3.2.3. Erwogene Alternativen 12
3.3.3. Konfigurations-File 12
3.3.3.1. Faktoren 12
3.3.3.2. Lösung 12
3.3.3.3. Erwogene Alternativen 12
3.3.4. Logging 12
3.3.4.1. Faktoren 12
3.3.4.2. Lösung 12
3.3.4.3. Erwogene Alternativen 13
3.3.5. Auf-/Vorbereitung der Daten 13
3.3.5.1. Mögliche Lösungsansätze 13
3.3.5.2. Daten in Liste vorbereiten 13
3.3.5.2.1. Vorteil 13
3.3.5.2.2. Nachteil 13
3.3.5.2.3. Entscheid 13
3.3.5.3. Daten in Klassen abfüllen 13
3.3.5.3.1. Möglichkeit 1 13
3.3.5.3.2. Möglichkeit 2 13
3.3.5.3.3. Entscheid 14
3.4. Logische Architektur 14
3.4.1. Übersicht 14
3.4.2. Packagestruktur 15
3.4.3. Solutionübersicht 16
3.4.4. Package- und Klassendiagramm 17
3.5. Design Pakete 17
3.5.1. Package Common 17
3.5.1.1. Beschreibung des Package 17
3.5.2. Package Reader 17
3.5.2.1. Beschreibung des Package 17
3.5.2.2. Klassendiagramm 18
3.5.2.3. Objekt Graph 19
3.5.2.4. Klassenbeschreibung 19
3.5.2.5. Sequenz Diagramm 20
3.5.2.6. Operationen 20
3.5.3. Package Writer 21
3.5.3.1. Beschreibung des Package 21
3.5.3.2. Klassendiagramm 21
3.5.3.3. Klassenbeschreibung 22
3.5.3.4. Sequenz Diagramm 22
3.5.3.5. Operationen 22
4. Implementierung 23
4.1. Generieren von Bytes aus Uint 23
4.2. Logging 23
5. Ausblick 24
5.1. Erweiterungen 24
5.1.1. Ergänzen von Kontaktdetails 24
5.1.2. Erweitern der Kontaktliste 25
5.1.3. Einstiegspunkt für den Merger 26
5.2. Fertigstellung 26
5.2.1. Erkenntnisse aus dem ersten Versuch 26
5.2.2. Recyclebare Methoden 26
5.3. Optimierungen 26
5.3.1. Längenberechnung im Kontakt 26
6. Literaturverzeichnis 27
7. Abbildungsverzeichnis 27
8. Tabellenverzeichnis 27

1. Anforderungsspezifikation

1.1. Allgemeine Beschreibung

1.1.1. Produktfunktion

Der OCS berücksichtigt beim Erstellen seiner Kontaktfiles nur Kontaktinformationen, die im Active Directory abgelegt sind. Das Ziel ist, mit Hilfe des Directory Framework ein Tool zu implementieren, welches die OCS-Kontaktfiles einlesen und die darin enthaltenen Kontakte extrahieren kann. Diese Kontaktliste soll dann mit Hilfe des DF um Kontakteinträge aus z.B. Twix Tel, Exchange, etc. erweitert werden. Zum Schluss soll das Tool in der Lage sein, ein neues Kontaktfile aus der nun ergänzten Kontaktliste zu erstellen, welches dann dem OCS übergeben werden kann. Dieser verteilt dann dieses modifizierte Kontaktfile auf die Communicators.

1.1.2. Benutzer Charakteristik

Die Zielgruppe dieses Projektes sind Firmen, welche sich einen erhöhten Komfort im Umgang mit der IP – Telefonie wünschen.

1.1.3. Einschränkungen

Da das DF mittels C# entwickelt wurde und der OCS ebenfalls ein Microsoft spezifisches Produkt ist, ist das Projekt auf eine reine Microsoft- Umgebungen beschränkt.

1.1.4. Abhängigkeiten

Die OCSAddressExtension ist nur in einem IP – Telefon –Umgebung einsetzbar, welches von einem OCS verwaltet wird.

1.2. Spezifische Anforderungen

1.2.1. Funktionale Anforderungen

1.2.1.1. Richtigkeit

Das Kontaktfile des OCS (.lsabs) muss nach der Veränderung wieder vom OCS korrekt eingelesen werden können.
Die Daten, welche zum Schreiben in ein neues File freigegeben werden, dürfen bei diesem Vorgang keine Veränderungen erfahren.

1.2.1.2. Interoperabilität

Der Einsatz des Projekts ist nur in einer Microsoftumgebung die mit einem OCS ausgerüstet ist, vorgesehen.

1.2.1.3. Sicherheit

Durch die Verwendung der OCSAddressBookExtension sollen keine Sicherheitslöcher auf dem verwendeten System auftreten.

1.2.2. Bedienbarkeit

1.2.3. Kurze Lernzeit

Da das Einrichten der OCSAddressExtension als Service innerhalb einer OCS Umgebung, von einem OCS-Admin getätigt werden muss, kann ein gewisses Mass an technischem Verständnis vorausgesetzt werden.

1.2.4. Zuverlässigkeit

1.2.4.1. Fehlertoleranz

Das Projekt hat den Anspruch zu 99,9% fehlerfrei zu laufen.

1.2.5. Wartbarkeit

1.2.5.1. Logging

Da keine direkte Benutzerinteraktion existiert, muss der gesamte Exception- Output in einem LogFile mitgeschrieben werden, um für allfällige Analysen verfügbar zu sein.

1.2.5.2. Installation

Die Installation soll von einer Person ohne spezielle Vorkenntnisse innerhalb von weniger als 10 Minuten vollzogen werden können.

1.2.6. Schnittstellen

1.2.6.1. Benutzerschnittstelle

Da es sich hierbei um einen Service handelt, existiert keine Benutzerschnittstelle.

1.2.6.2. Softwareschnittstellen

- Das Directory Framework wird für die Absetzung von Requests verwendet um Kontaktinformationen anderer Quellen zu bekommen.
- Das Tool AbServer.exe wird benötigt, um die generierten Kontaktfiles validieren zu können.

1.2.7. Datenbankschnittstelle

Es wird keine Datenbankschnittstelle realisiert, da keine Daten in einer Datenbank persistiert werden.

1.2.8. Lizenzanforderungen

Es werden keine Lizenzen benötigt.

1.2.9. Verwendete Standards

Die Entwicklung des Projektes erfolgt mit dem .net Framework 3.5 und C#.

1.3. Funktionale Anforderungen

1.3.1. Feature Liste

1.3.1.1. Einlesen der Daten

Die Grundlage für die OCSAddressBookExtension bildet ein .lsabs File, welches vom OCS generiert wird. Um an die darin enthaltenen Kontaktinformationen zu gelangen, muss das File eingelesen werden können. Delta- wie auch Full- Files sollen eingelesen werden können.

1.3.1.2. Dekomprimierung

Die Kontaktfiles, welche der OCS generiert, sind immer mit einem LZ77 Algorithmus komprimiert. Um an die eigentlichen Daten zu gelangen muss die OCSAddressBookExtension über die Möglichkeit verfügen, besagte Files dekomprimieren zu können.
Um die Dekomprimierung sauber umsetzen zu können, empfiehlt es sich den unter Punkt 5.5.1 und 5.5.2[1] beschriebenen Pseudocode des Microsoft Dokumentes zu benutzen.

1.3.1.3. Aufbereitung der Daten

Die extrahierten Daten sollen für Vergleiche und Ergänzungen, ohne grossen Aufwand, nutzbar sein. Um dies zu gewährleisten, müssen die Daten in entsprechender Abstraktion aufbereitet werden.

1.3.1.4. Hinzufügen von neuen Kontaktinformationen

Mit Hilfe des DirectoryFrameworks können Kontaktinformationen von externen Datenquellen abgefragt werden (z.B. Twix Tel, Exchange, usw.) Die entpackten Kontakte sollen nun mit denjenigen, die das DirectoryFramework liefert, verglichen werden können. Finden sich nun Unterschiede, bzw. Kontakte, die im OCS- Kontaktfile nicht vorhanden sind, kann dieses um eine beliebige Anzahl an Kontakten ergänzt werden.

1.3.1.5. Generierung eines neuen Files

Um die angepassten Daten dem Server zur weiteren Verarbeitung übergeben zu können, muss ein neues (.lsabs) Kontaktfile daraus generiert werden. Dieses neue File hat folgende Anforderungen zu erfüllen:

- Es muss den selben strukturellen Aufbau wie das Original vorweisen
- Der OCS muss das File anerkennen / akzeptieren
- Die Struktur muss neu generiert werden können, unabhängig des Originals
- Delta- und Full-Files sollen generiert werden können

1.3.1.6. Komprimierung

Um Platz zu sparen und das Übertragen der Kontaktfiles vom OCS auf die angeschlossenen OC's nicht unnötig zu verlangsamen, sollen die generierten Files wieder mit einem LZ77 Algorithmus komprimiert werden. Für die Verpackung von Length und Offset von gefundenen Sequenzen mit Hilfe des LZ77 Algorithmus, muss der Pseudocode unter Punkt 5.4[2] aus der Microsoft Dokumentation verwendet werden.

[1] MS Address Book File Structure
[2] MS Address Book File Structure

2. Domainanalyse

2.1. Strukturdiagramm

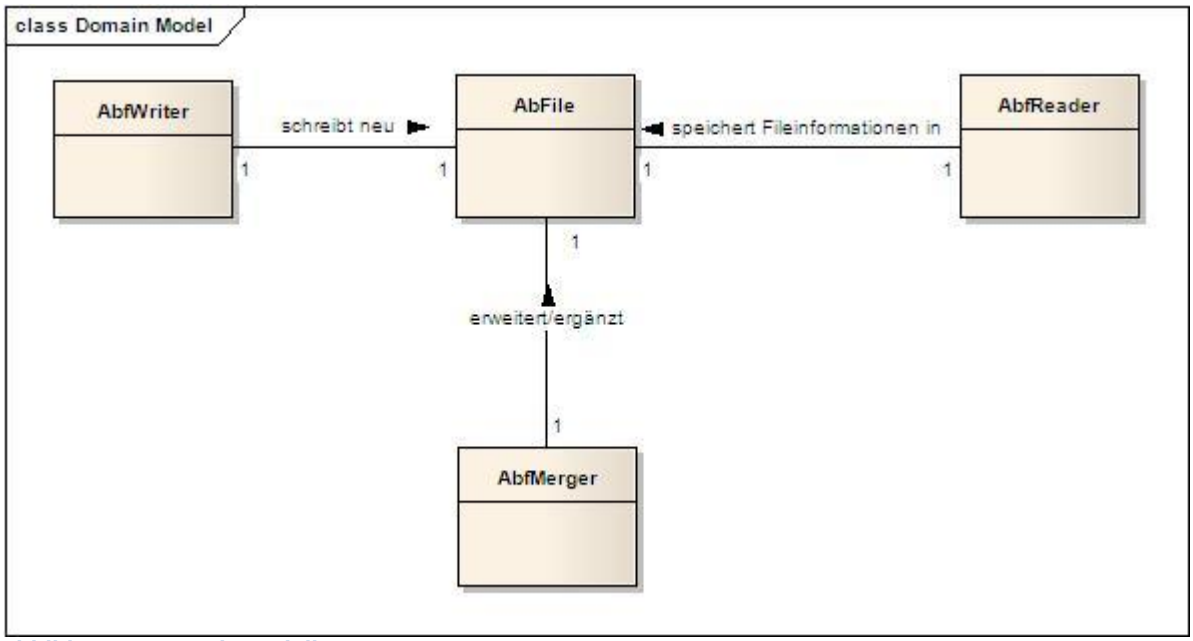


Abbildung 1: Domainmodell

Das Domainmodell stellt diejenigen Konzepte dar, die zur Kernproblemtatik der Software gehören. Diese liegen der Idee der Aufgabenstellung zugrunde. Nachfolgend sind die verschiedenen Konzepte und deren Verantwortlichkeiten genauer erläutert.

2.2. Konzeptbeschreibung

Konzepte	Beschreibung
AbfReader	Der *AbfReader* nimmt ein .lsabs Kontaktfile des OCS entgegen, dekomprimiert es und erstellt ein *AbFile*.
AbfWriter	Der *AbfWriter* übernimmt die Aufgabe des Erstellens eines neuen, OCS kompatiblen Kontaktfiles aus dem gelieferten *AbFile* . Ebenso übernimmt er die Aufgabe des Komprimierens der gegebenen Daten.
AbfMerger	Dem *AbfMerger* wird ein *AbFile* übergeben. Mit Hilfe des DirectoryFrameworks kann nun ein Request abgesetzt werden, welcher in allen gewünschten externen Datenquellen nach weiteren Kontakinformationen sucht. Die vom Request als Resultat zurückgegebenen Kontaktinformationen können nun mit den Vorhanden im *AbFile* verglichen und dort gleich ergänzt werden.
AbFile	Das *AbFile* entspricht einem digitalen Abbild eines .lsabs Files, welches vom *AbfReader* eingelesen wird. Es werden nebst den Kontaktinformationen alle relevanten Daten gespeichert, um später dem *AbfWriter* eine optimale Grundlage zur eigenständigen Generierung eines neuen Files zu geben. Die Kontaktinformationen, welche für den *AbfMerger* von speziellem Interesse sind, sollen derart vorbereitet werden, dass die weitere Verarbeitung keinerlei Mühe bereitet.

Tabelle 1: Konzepte

2.3. Systemsequenzdiagramm

2.3.1. SSD Black Box Betrachtung

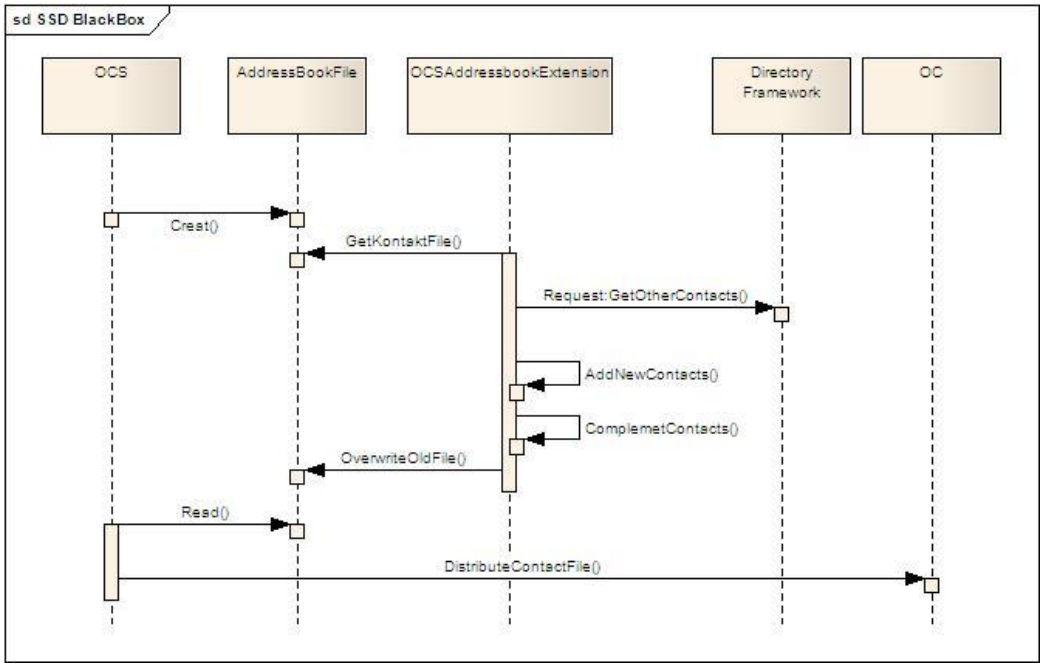


Abbildung 2: SSD BlackBox

2.3.2. Beschreibung

Methode	Beschreibung
Create()	Der OCS generiert alle 24h ein neues Full- und Delta- File aufgrund der im AD vorliegenden Informationen. Hier ist auch die Ursache zu suchen, warum das Tool Full- und Delta- Files lesen und schreiben können muss. Genaueres unter 2.3.3.
GetKontaktFile()	Die AddressBookExtension holt sich die generierten Files.
GetOtherContacts()	Ein Request an das Directory Framework wird abgesetzt. Dieser Liefert die angeforderten Kontaktinformationen zurück.
AddNewContacts()	Die Kontaktdaten werden verglichen und um diejenigen aus dem Directory Framework ergänzt, die noch nicht vorhanden sind.
ComplementContacts()	Die Kontaktdaten werden auf ihre Vollständigkeit geprüft und falls möglich durch die gelieferten Informationen aus dem Directory Framewrok ergänzt.
WriteNewFileBack()	Die AddressBookExtension schreibt das Neue Kontaktfile zurück auf OCS, wobei das originale Kontaktfile des OCS überschrieben wird.
Read()	Der OCS liest das neu generierte Kontaktfile ein.
DistributeContactFile()	Der OCS verteilt das neue Kontaktfile auf die OC's.

Tabelle 1: Beschreibung Operations

2.3.3. SSD OCSFilegenerierung

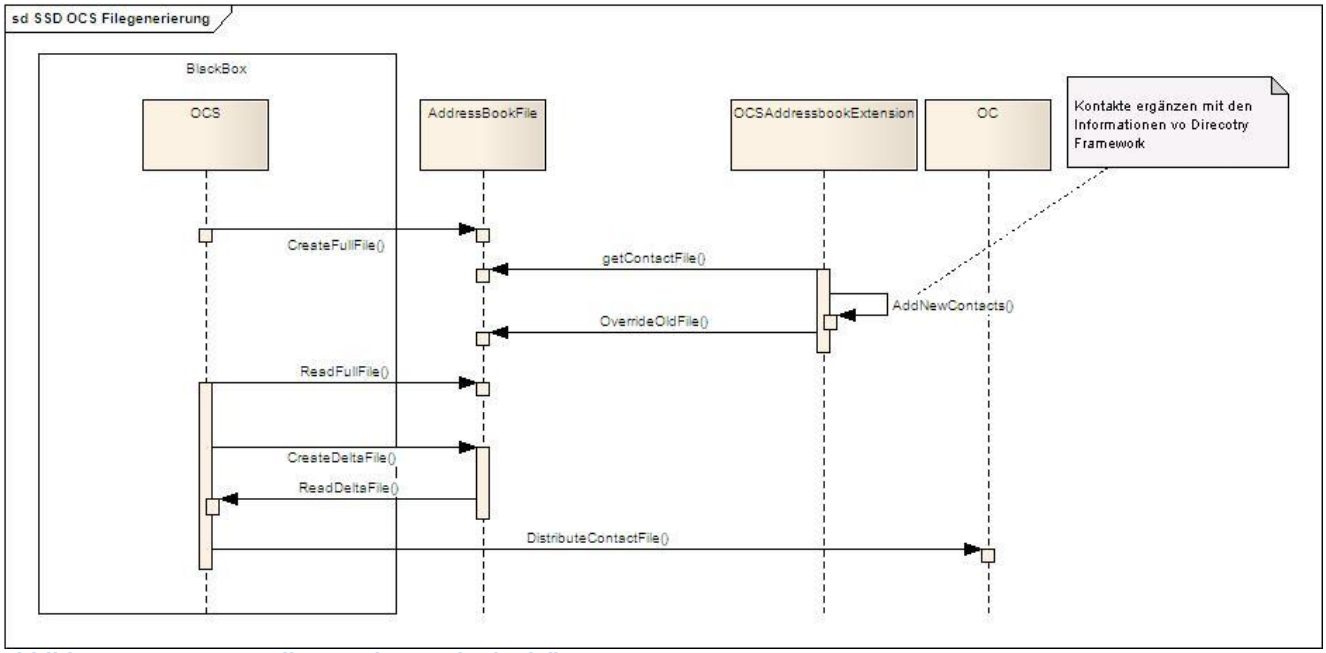


Abbildung 3: SSD OCS Filegenerierung (Beispiel)

2.3.3.1. Beschreibung

Die Ursache, warum Delta- Files und Full- Files von der OCSAddressBookExtension gelesen und geschrieben werden können müssen, ist in der Beschaffenheit des OCS zu suchen. Es ist nicht bekannt, in welcher Reihenfolge der OCS die Delta- und Full- Files generiert.
Das oben gezeigte Diagramm stellt die einzige Möglichkeit dar, in der nur das Full- File angepasst werden müsste. Da dann das Delta- File auf dem bereits modifizierten Full- File gebildet werden würde. Es ist nicht bekannt, ob das Generieren der Files auf dem OCS in einem oder zwei Prozessen abgearbeitet wird, geschweige denn, ob der Prozess unterbrochen werden könnte, um die Modifizierung vornehmen.
Da diese Faktoren nicht abgeklärt werden konnten bzw. können, wurde festgelegt, dass Delta- und Full- Files angepasst werden sollen, um jegliche Fehlerquellen in diesem Bereich ausschliessen zu können.

2.3.4. Systemoperationen

2.3.4.1. CO1: GetKontaktFile

Operation:	GetContactFile()
Querverweise:	Black Box Betrachtung
Vorbedingungen:	- OCS hat Kontaktfile erstellt
Nachbedingungen:	- OCSAddressBookExtension hat das File erfolgreich eingelesen - Das File wurde erfolgreich dekomprimiert - Das digitale Abbild des Kontaktfiles wurde erfolgreich erstellt.

Tabelle 2: GetKontaktFile- Operation

2.3.4.2. CO2: AddNewContacts

Operation:	AddNewContacts()
Querverweise:	Black Box Betrachtung
Vorbedingungen:	- Das Kontaktfile wurde erfolgreich eingelesen, dekomprimiert und ein digitales Abbild davon erstellt.
Nachbedingungen:	- Die neuen Kontakte wurden dem digitalen Abbild erfolgreich hinzugefügt. - Falls möglich, wurden bereits bestehende Kontakte mit neuen Informationen ergänzt

Tabelle 3: AddNewContacts- Operation

2.3.4.3. CO3: WriteNewFilesBack

Operation:	OverrideOldFile()
Querverweise:	Black Box Betrachtung
Vorbedingungen:	- Die neuen Kontakte wurden dem digitalen Abbild erfolgreich hinzugefügt.
Nachbedingungen:	- Das digitale Abbild wurde in Byteform zurück konvertiert - Das File wurde komprimiert - (optional) das File wurde nicht komprimiert - Das alte File auf dem Server wurde überschrieben

Tabelle 4: WriteNewFilesBack- Operation

3. Softwarearchitekturdokument

3.1. Umgebung

Die folgenden Bilder zeigen die physische Installation von OCSAddressBookExtension

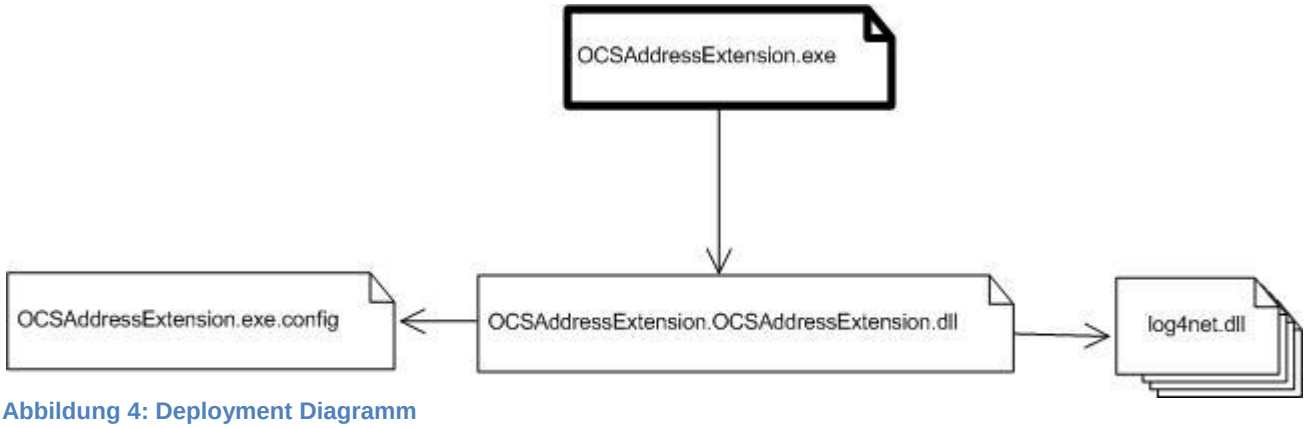


Abbildung 4: Deployment Diagramm

3.2. Architektonische Ziele & Einschränkungen

3.2.1. Ziele

- Die Daten eines AddressBook- Files sollen in möglichst strukturierter Form aufbereitet werden, damit deren weitere Verwendung möglichst einfach gehalten werden kann.

- Aufgrund von Übersichtlichkeit und Kapselung soll die Funktionalität von lesen, schreiben und ergänzen voneinander getrennt werden.

- Um das Vergleichen der Daten aus dem File mit denjenigen aus dem DF- Request möglichst einfach zu gestalten, sollte die Datenstruktur derjenigen der DF- Daten angelehnt werden.

- Alle Fehler die auftreten und den Ablauf der Applikation behindern, oder anderweitig zu Komplikationen führen, sollen in einem Log- File mitgeführt werden. Damit soll es möglich sein, die Fehlerquellen zu eruieren.

- Da viel mit Headern und Headerfeldern gearbeitet wird, gilt es alle Konstanten aus dem Code zu entfernen. Es sollen jegliche Magic Numbers zu verhindern.

3.2.2. Einschränkungen

- Um die Verständlichkeit des Codes und der Dokumente nicht zu strapazieren, wurden alle Namen und Bezeichnungen von Microsoft übernommen[3].

[3] MS Address Book File Structure

3.3. Architektonische Entscheidungen

3.3.1. Reader/Writer

Die Applikation wird zu Beginn zwei Hauptaufgaben zu bewältigen haben. Zum einen gilt es die Kontakt- Files, die der OCS erstellt einzulesen und die Daten derart aufzubereiten, dass sie für weitere Verwendungszwecke tauglich sind. Zum Andern muss aus den eingelesenen Daten ein komplett neues Kontaktfile generieren zu können, wobei dieses File auch noch vom OCS akzeptiert werden muss.

3.3.1.1. Faktoren

- Dekomprimieren der Daten
- Extraktion der Kontaktinformationen und allen andern Informationen, damit aus diesen ein komplettes, neues File generieret werden kann.
- Die extrahierten Kontaktinformationen sollen aufbereitet werden, um weiter verwendet werden zu können.
- Delta- und Full- Files sollen eingelesen werden können.
- Delta- und Full- Files sollen geschrieben werden können
- Generierte Files sollen komprimiert werden können, oder allenfalls unkomprimiert geschrieben werden können
- Soll fähig sein, aus den gelieferten Daten ein komplettes, vom OCS akzeptiertes Kontaktfile zu erzeugen

3.3.1.2. Lösung

Um die Lesende und die Schreibende Funktion sauber getrennt halten zu können, wurde die Applikation in einen *Abf_Reader* und einen *Abf_Writer* augetrennt.
Der Reader übernimmt ausschliesslich die Funktion des Auslesens und der Aufbereitung der Daten.
Der Writer übernimmt die Funktion des Generierens eines neuen Full- oder Delta- Files aus den Daten, die er bekommt.

3.3.1.3. Erwogene Alternativen

Es konnte keine Alternative gefunden werden, bei der sich eine klarere Trennung der Funktionalitäten ergeben hätte.

3.3.1.4. Sequenz Diagramm Reader/Writer

Nachfolgend ist in groben Zügen das Zusammenspiel zwischen Reader, Writer und dem Main-Rumpf aufgezeigt.

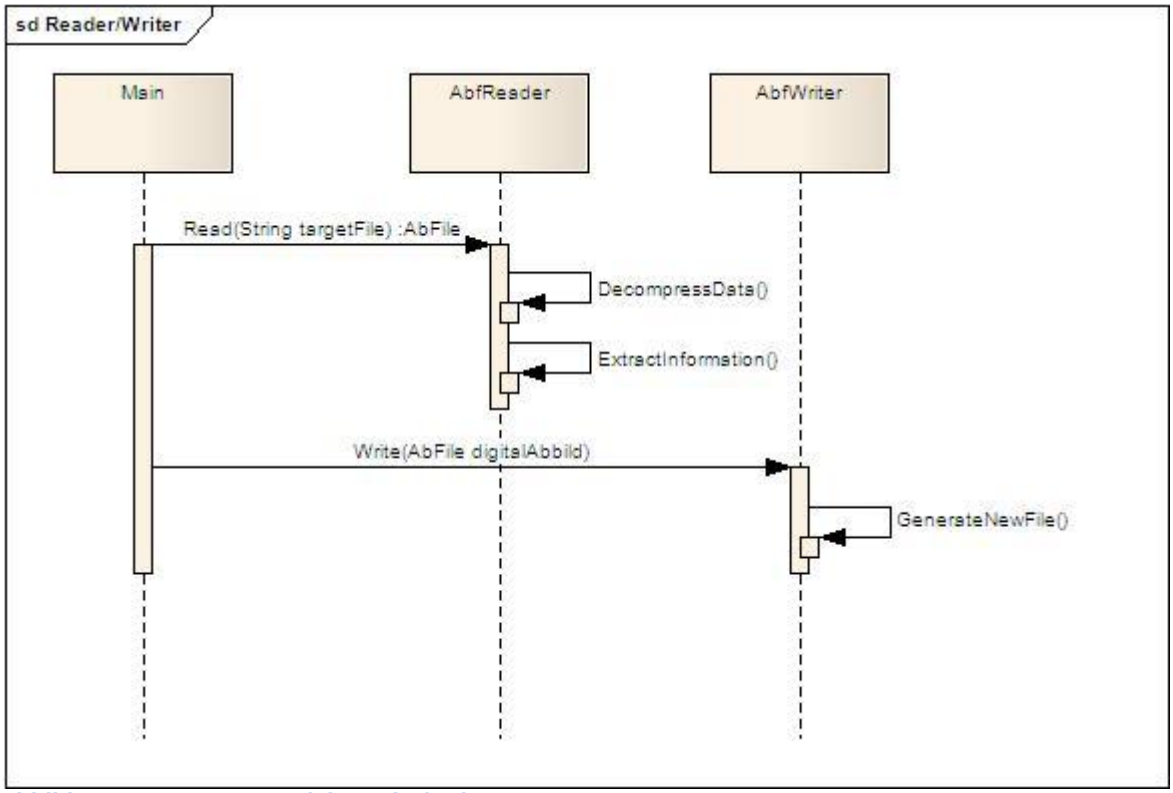


Abbildung 5: Zusammenspiel Reader/Writer

3.3.2. Konstanten

Da sehr oft Header wie auch Headerfelder ausgelesen oder generiert werden müssen, welche über eine fixe Grösse verfügen, existieren sehr viele Konstanten. Die Art und Weise wie diese Konstanten verwaltet werden sollen, wird hier geklärt.

3.3.2.1. Faktoren

- Einfacher Zugriff
- Speichert alle Konstanten um Magic Numbers zu eleminieren.
- Sinnvolle Gliederung
- Beschreibung um den Zweck der jeweiligen Konstanten klar zu machen.

3.3.2.2. Lösung

Alle Konstanten werden in einer Klasse *Constant.cs* abgelegt. Für ein gewisses Mass an Übersichtlichkeit sorgt die Verwendung der summary Tags, welche erlauben, die Konstante zu beschriften.

Beispiel:

```
/// <summary>
/// Exact amount of Bytes of a FullHeader
/// </summary>
public const int FULL_HEADER_SIZE = 152;
```

3.3.2.3. Erwogene Alternativen

Es galt abzuwägen, ob sich eine Constant Klasse für den *AbfReader* und eine zweite für den *AbfWriter* lohnen, bzw. für mehr Übersicht sorgen würde. Da allerdings die Headerfelder, welche ausgelesen werden, auch wieder geschrieben werden müssen. wären die beiden Constant Klassen beinahe identisch. Daher wurde die Lösung mit nur einer *Constan.cs* umgesetzt

3.3.3. Konfigurations-File

Absolute Pfadangaben im Code sind zu vermeiden. Daher stellte sich die Frage ob die in ein Constant File gehören oder doch besser in ein Konfigurations- File.

3.3.3.1. Faktoren

- Pfadangaben sollen angepasst werden können, ohne Änderungen am Code vornehmen zu müssen.

3.3.3.2. Lösung

Erstellen eines Konfiguration- Files, welches die Pfadangaben beinhaltet.

3.3.3.3. Erwogene Alternativen

Die Idee, die Pfadangaben starr in einem Konstanten- File niederzuschrieben, wurde nicht weiterverfolgt. Das Ändern der Pfadangaben hätte immer eine Manipulation des Codes zur Folge.

3.3.4. Logging

Da es sich hierbei um eine Applikation ohne UI oder sonstige Benutzerinteraktion handelt, scheint es durchaus am sinnvollsten, die ganzen Exceptions, Debugs, Informationen in einem LogFile zu persistieren. Es galt abzuwägen mit welchen Mitteln dies umgesetzt werden kann.

3.3.4.1. Faktoren

- Übersicht im Aufbau und der Gliederung des Logfiles
- Klassifizierung der geschriebenen Meldung ist wünschenswert
- Einfaches Konfigurieren
- Hohe Verständlichkeit der Einträge

3.3.4.2. Lösung

Der Entscheid viel zu Gunsten von Log4net aus. Zum Einen erfüllt diese dll alle wichtigen Faktoren. Zum Andern war die Hilfestellung von einem eingefleischten Log4net Benutzer sehr gut. Damit konnte der erforderliche Logging- Umfang für dieses Projekt schnell implementiert werden.

3.3.4.3. Erwogene Alternativen

Zu Beginn wurde eine Möglichkeit, die mit einem TraceListener gearbeitet hätte in Erwägung gezogen. Da diese jedoch erheblich mehr Aufwand gefordert hätte und ein noch wesentlich weniger professionelles Logging erlaubt hätte, wurde diese Variante zugunsten von Log4net fallen gelassen.

3.3.5. Auf-/Vorbereitung der Daten

Die Kontaktinformationen, welche mit Hilfe des AbfReaders ausgelesen werden, derart aufbereitet werden, damit sie mit den Daten, welche vom Directory Framework zusammengestellt werden, verglichen werden können. Nach dem Vergleich verbleiben eine Menge von Kontakten, welche nicht im AddressBook – File vorhanden sind, aber vom DirecotryFramework zur Verfügung gestellt werden und nun in das AddressBook – File eingefügt werden sollen.

3.3.5.1. Mögliche Lösungsansätze

Nachfolgend werden mehrere Ansätze analysiert und auf Umsetzbarkeit geprüft.

3.3.5.2. Daten in Liste vorbereiten

Die Kontaktinformationen, welche vom AddressBookFile stammen, sollen in einem Dictionary vorbereitet werden. Attributname fungiert als Key und das Value entspricht dem Kontaktinhalt zu diesem Attribut. Selbiges Verfahren wird auf die Daten angewandt, die das DirectoryFramework liefern, was schlussendlich dazuführt, dass zwei Listen mit Kontakten verglichen werden müssen. Die fehlenden Kontakte werden dann eruiert und am Schluss des AddressBookFiles, jedoch vor dem Trailer, angehängt.

3.3.5.2.1. Vorteil

Listen erstellen und abfüllen, wie auch vergleichen ist ohne grossen Aufwand umsetzbar.

3.3.5.2.2. Nachteil

Diese Lösung ist nur für das Hinzufügen neuer Kontakte geeignet. Möchte man in einer späteren Phase auch ein Erweitern bereits bestehender Kontakte in Betracht gezogen werden, ist dies mit dieser Variante nicht mehr möglich. Essentielle Informationen fehlen oder müssten neue berechnet werden, um alle Kontakte neu zu generieren, bzw. auch Header - und Trailerfelder müssten in diesem Fall neu gesetzt und/oder berechnet werden.

3.3.5.2.3. Entscheid

Diese Lösung wird abgelehnt, da von der Seite des Projektstellers explizit eine Erweiterung der Arbeit mit dem Ziel „Ergänzen von vorhanden Kontakten“ unterstützt werden soll.

3.3.5.3. Daten in Klassen abfüllen

Die Informationen werden seitens DirectoryFramework aufbereitet und in Klassen zur Verfügung gestellt. Somit wäre für einen Vergleich mit den Daten aus dem AddressBookFile, das Vorhanden sein derer, in einer ähnlichen Klassenstruktur wünschenswert. Pro Kontakt würde eine Instanz einer Klasse erzeugt werden, die pro Attribut ein Feld vorweist, welchem der Wert des dazugehörigen Attributes zugewiesen werden kann.
Das Problem bei diesem Ansatz ist das erstellen der Klasse bzw. die Definition der Attribute welche innerhalb der Klasse auch abgefüllt werden sollen.

3.3.5.3.1. Möglichkeit 1

Die Klasse und die dazugehörigen Felder werden dynamisch, während der Laufzeit per Reflection erzeugt. Der Aufwand für die Umsetzung und die Einarbeitungszeit in die Thematik, könnte hier viel Zeit in Anspruch nehmen. Ebenso gälte es abzuklären ob das geforderte Ziel überhaupt mittels Reflection erreichbar ist.

3.3.5.3.2. Möglichkeit 2

Die Klasse soll statisch definiert werden. Die Klassenfelder werden so spezifiziert, dass die häufigsten und wichtigsten Attribute vorhanden sind. So müssen lediglich diese Attribute überprüft werden und falls es AddressBookFiles gibt, welche Attribute beinhalten, die ausserhalb der Definierten anzusiedeln sind, werden diese ignoriert. Vorschlag für einen ersten Definitionsansatz sieht folgendermassen aus.
In der unten aufgeführten Tabelle werden die Attributeinträge zweier TestFiles und die Konfigurationsmöglichkeit des ABS Config Tools verglichen. Die Testfiles besitzen einen identischen Attribut-Pool und derjenige des ABS Config Tools ist sehr ähnlich. Der Name vereinzelter Attribute unterscheidet sich (z.B. LastName zu sn) und der Pool des Config Tools ist um drei Attribute kleiner.
Für einen ersten Prototyp der Klasse wird die Attribuierung, welche unter dem Tabellentitel „Vorschlag“ aufgeführt ist, eingesetzt.

ABS Config Tool	TestFile 0c89.lsabs	TestFile 0c7b.lsabs	Vorschlag:
ProxyAddresses	proxyAddresses	proxyAddresses	proxyAddresses
FirstName	givenName	givenName	givenName
LastName	sn	sn	sn
DisplayName	displayName	displayName	displayName
Title	title	title	title
Alias	mailNickName	mailNickName	mailNickName
Company	company	company	company
Location	physicalDeliveryOfficeName	physicalDeliveryOfficeName	physicalDeliveryOfficeNam e
ImAddress	msRTCSIP-PrimaryUserAddress	msRTCSIP-PrimaryUserAddress	msRTCSIP-PrimaryUserAddress
WorkPhoneNumber	telephoneNumber	telephoneNumber	telephoneNumber
HomePhoneNumber	homePhone	homePhone	homePhone
CelPhoneNumber	mobile	mobile	mobile
OtherPhoneNumber	otherTelephone	otherTelephone	otherTelephone
IPPhoneNumber	ipPhone	ipPhone	ipPhone
Email	mail	mail	mail
GroupType	groupType	groupType	groupType
Manager	manager	manager	manager
OuPathId	OuPathId	OuPathId	OuPathId
	otherHomePhone	otherHomePhone	otherHomePhone
	otherMobile	otherMobile	otherMobile
	MsExchHideFromAddressLists	MsExchHideFromAddressLists	MsExchHideFromAddress Lists

Tabelle 5: Attribut Vergleich

3.3.5.3.3. Entscheid

Möglichkeit 1 wurde aufgrund des Zeitaufwandes nicht weiter in Erwägung gezogen. Auch die Frage bezüglich der Umsetzbarkeit wurde noch nicht weiter abgeklärt.
Möglichkeit 2 wird mit dem vorgeschlagenen Attribut – Pool umgesetzt. Der Forderung nach Kontakterweiterbarkeit kann mit diesem Ansatz einfacher nachgekommen werden.

3.4. Logische Architektur

OCSAddressExtension besteht nur aus einer Logikschicht. Auf eine Persistenzschicht wurde verzichtet, da keine grösseren Datenmengen persistiert werden.

3.4.1. Übersicht

Im nachfolgenden Diagramm werden die zwei Hauptprojekte und die darunter implementierte Struktur aufgezeigt. Die Struktur in der OCSAddressBookExtension enthält die Solution OCSAddressBookExtension, in dieser Solution sind zwei „Haupt“-Projekte enthalten:

- TestProject_OCSAddressExtension
- OCSAddressExtension

In diesen Projekten sind dann diverse Packages und Subpackages enthalten, welche wiederum die einzelnen Klassen enthalten.

3.4.2. Packagestruktur

Das Projekt wurde folgendermassen gegliedert. Die Packagesturktur entspricht den Namespaces.

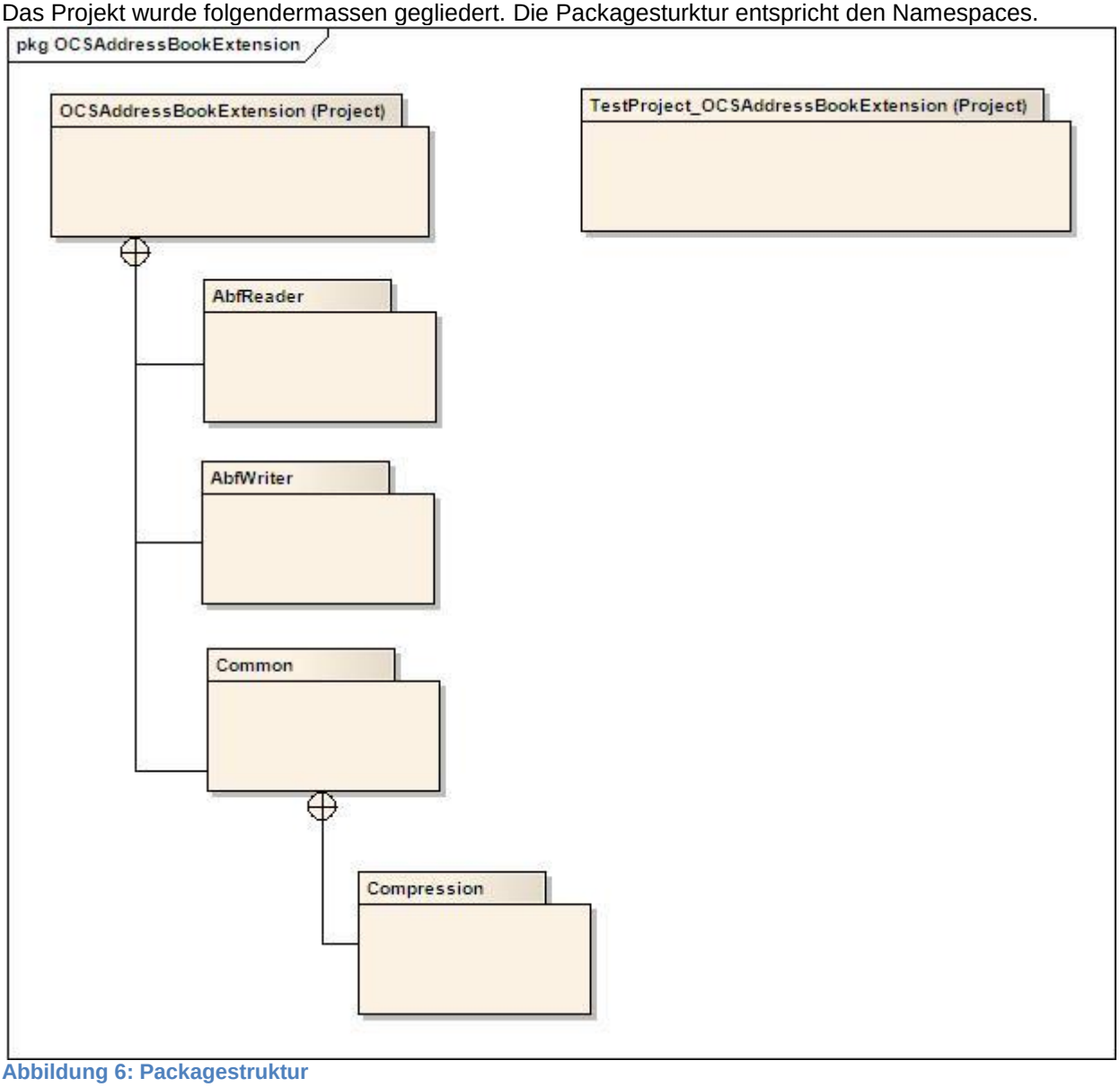


Abbildung 6: Packagestruktur

3.4.3. Solutionübersicht

Nachfolgendes Bild zeigt die Übersicht der ganzen Solution, mit allen Hauptprojekten.

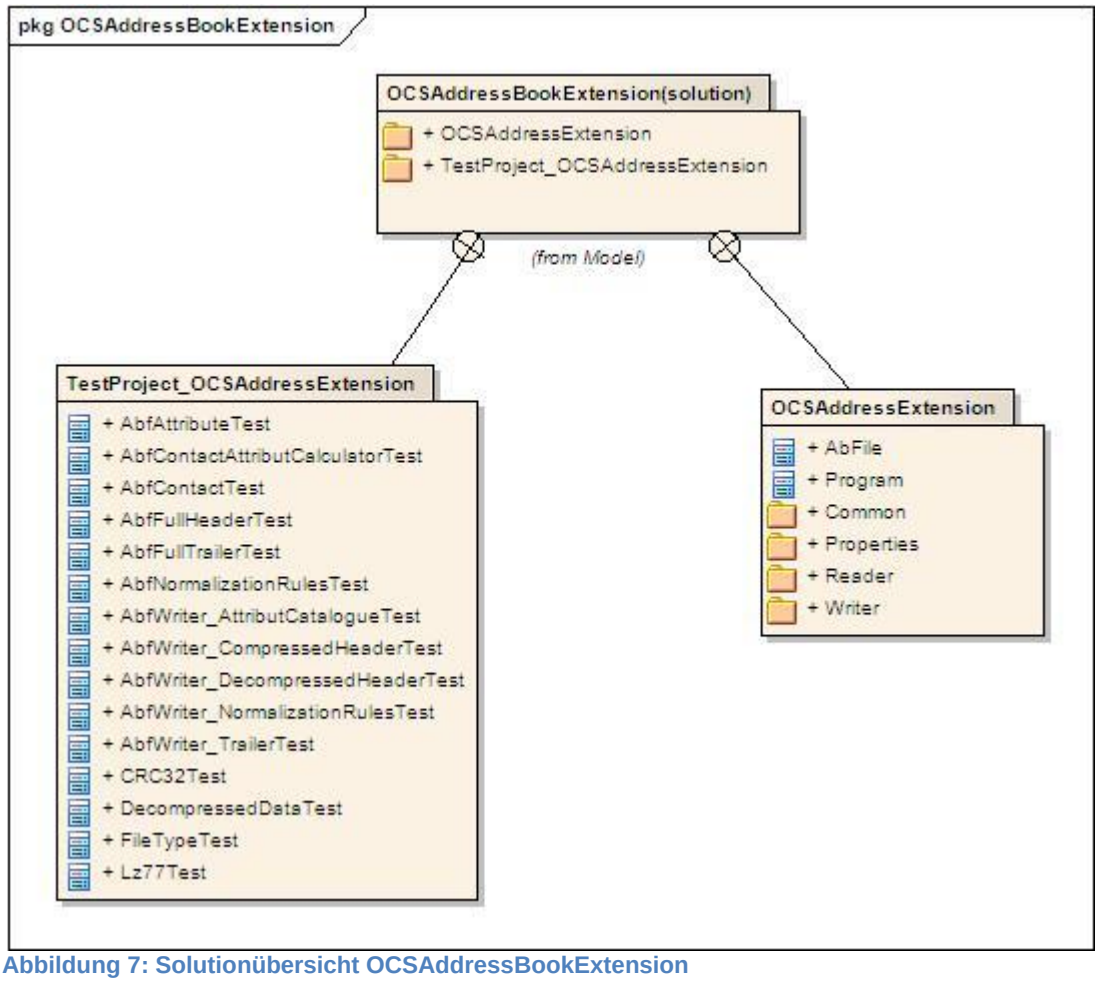


Abbildung 7: Solutionübersicht OCSAddressBookExtension

3.4.4. Package- und Klassendiagramm

Nachfolgendes Bild zeigt die Übersicht über das Projekt OCSAddressBookExtension.

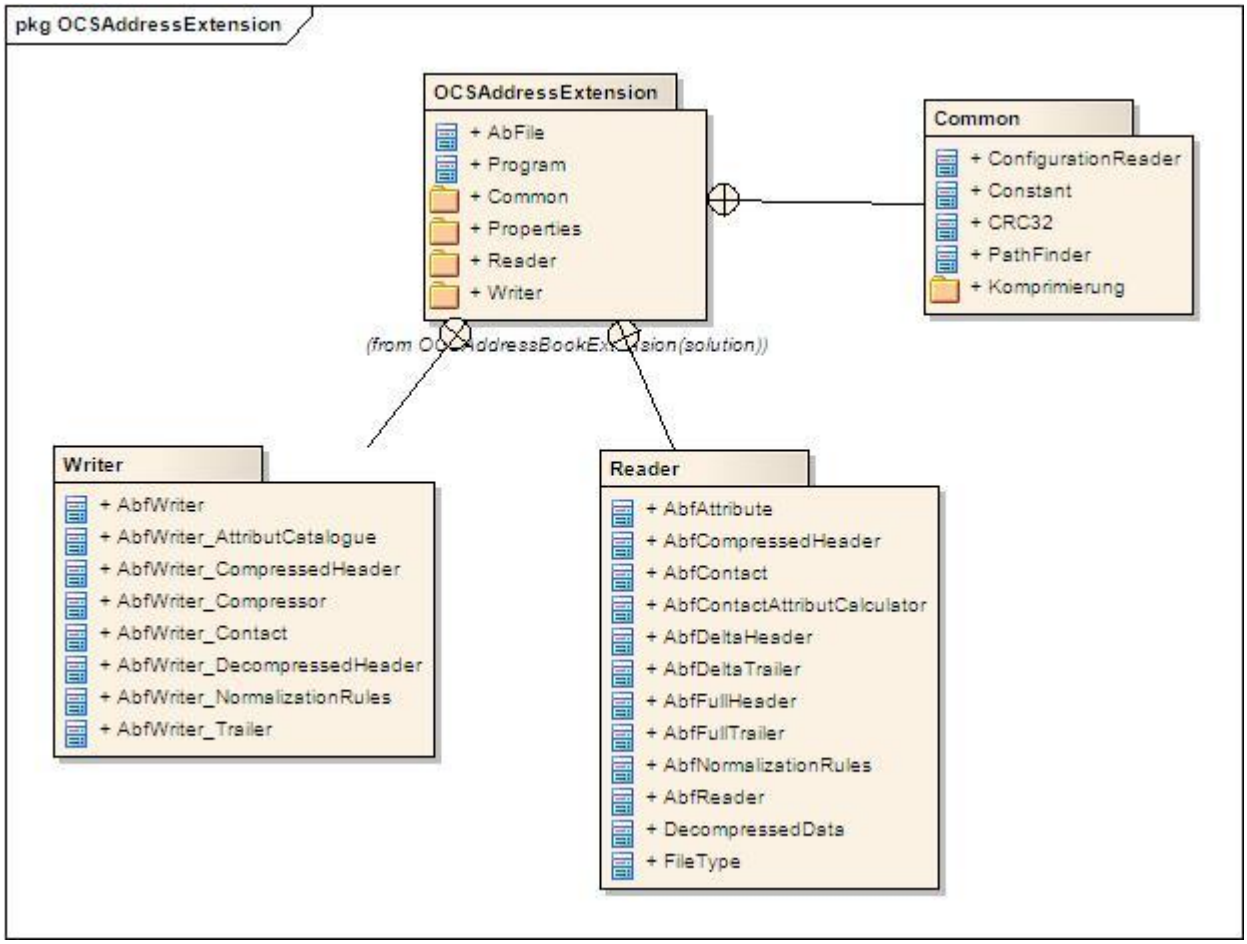


Abbildung 8: Package- und Klassendiagramm OCSAddressBookExtension

3.5. Design Pakete

3.5.1. Package Common

3.5.1.1. Beschreibung des Package

In diesem Package befinden sich diverse Hilfsklassen, die wichtigsten sind *CRC32*, *Constant* und ConfigurationReader.
Das Unterpackage Kompression dient lediglich zur logischen Strukturierung und beinhaltet die ersten Gehversuche zum Thema Komprimierung.

Klasse	Beschreibung
Crc32	Beinhaltet eine Funktion, die aus einem vorgegebenen Array über eine gewisse Länge den Crc32- Wert berechnen kann. Dies wird zur Validierung der eingelesenen Daten benötigt. Auch beim Generieren von neuen Kontaktfiles muss der Crc- Wert gebildet werden, da dieser im CompressedHeader eingetragen werden muss.
Constant	Alle einstigen Magic Numbers aus dem Code wurden hier her verlagert um dem Code mehr Leserlichkeit zu verschaffen und um die Werte zentrale verwalten zu können.
ConfigurationReader	Der ConfigurationReader erlaubt es, die Pfadangaben zu Input- und Output- File aus dem Konfiguration File auszulesen.

Tabelle 6: Klassenbeschreibung Commen

3.5.2. Package Reader

3.5.2.1. Beschreibung des Package

Dem Reader Package werden alle Entwicklungskomponenten zugeteilt, die in irgendeiner Form mit dem Einlesen, Dekomprimieren und Vorbereiten der Daten zu tun haben.

3.5.2.2. Klassendiagramm

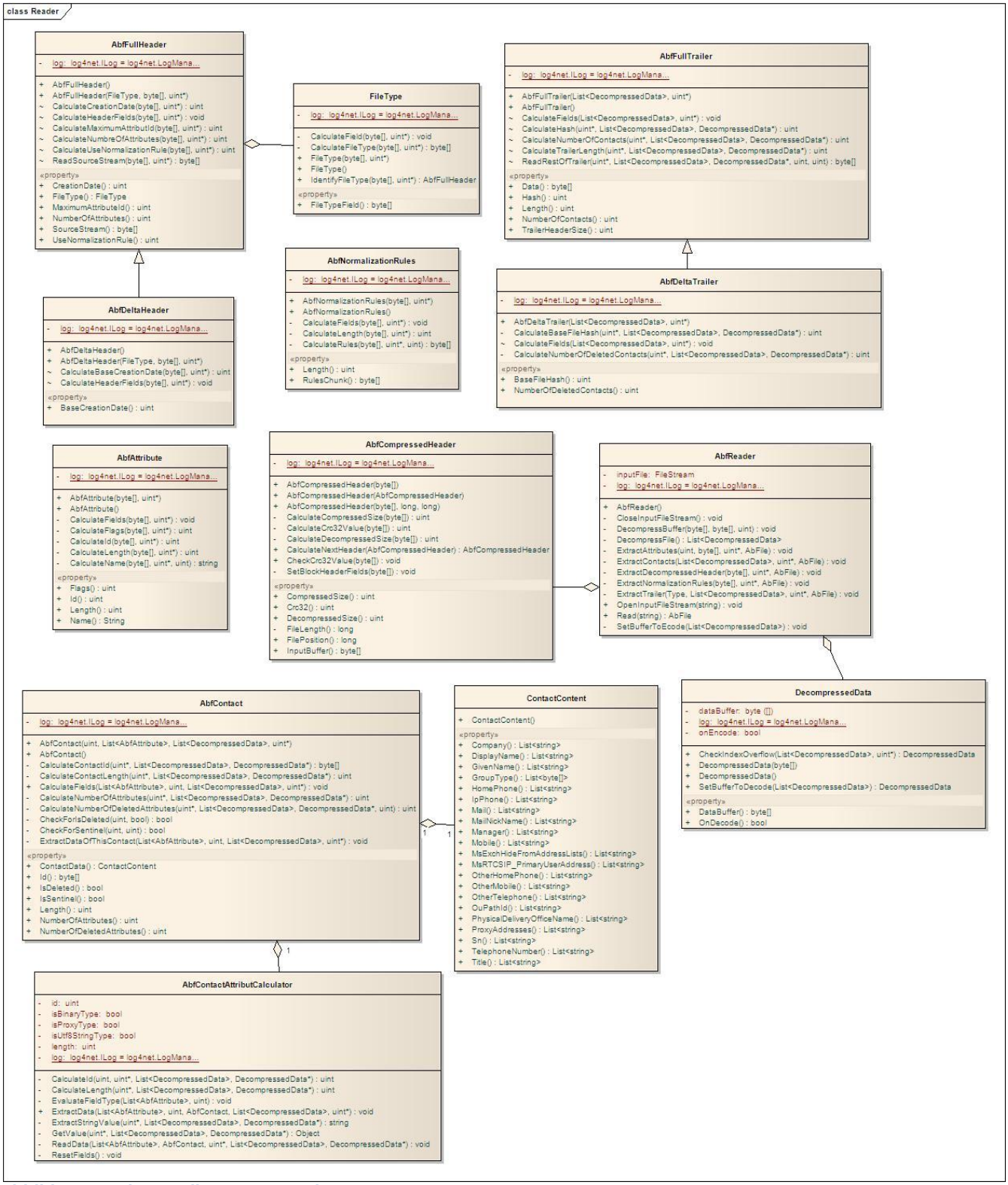


Abbildung 9: Klassendiagramm Reader

3.5.2.3. Objekt Graph

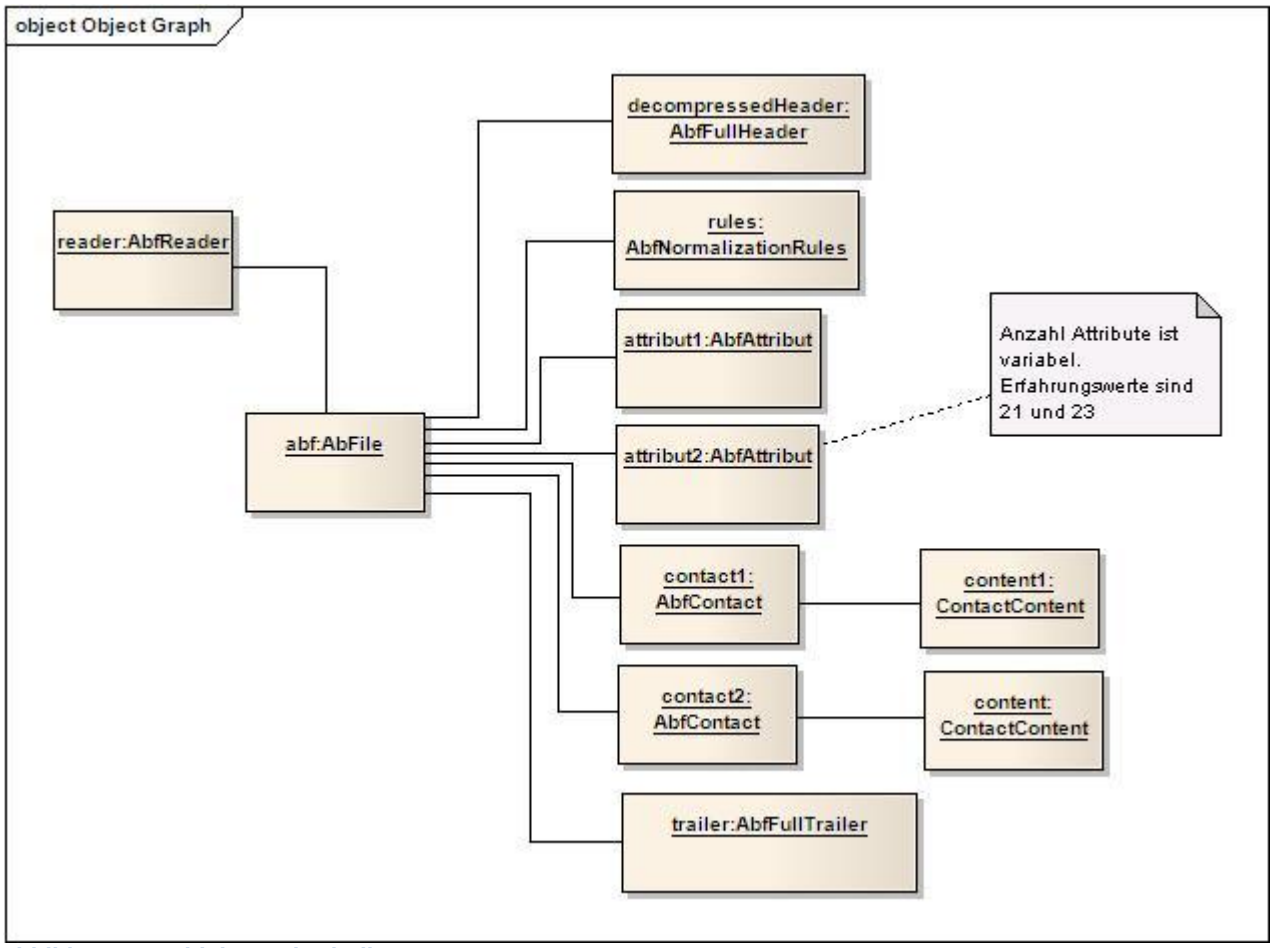


Abbildung 10: Objektgraph AbFile

Der Snapshot zu obiger Grafik wurde nach komplettem Einlesen eines Kontaktfiles geschossen. Sie zeigt auf, welche Instanzen alles in einem *AbFile* gespeichert werden.

3.5.2.4. Klassenbeschreibung

Klasse	Beschreibung
AbfFullHeader	Liest die vorgesehene Menge an Bytes, füllt damit seine Felder ab, um als digitale Abbildung eines FullHeaders fungieren zu können.
AbfDeltaHeader	Liest die vorgesehene Menge an Bytes, füllt damit seine Felder ab, um als digitale Abbildung eines DeltaHeaders fungieren zu können.
AbfAttribute	Repräsentiert ein Attribut und verfügt über die Methoden, um dieses auszulesen.
AbfContact	Repräsentiert einen Contact und verfügt über die Methoden, um dieses auszulesen.
AbfContactAttribute	Füllt die ContactContent-Daten ab, indem die einzelnen Attribute, die innerhalb eines Contacts vorkommen, korrekt ausgelesen und dem verantwortlichen Feld zugewiesen werden.
ContactContent	Speichert die extrahierten Kontaktdaten
AbfNormalizationRules	Repräsentiert die NormalizationRules und verfügt über die Methoden um dieses auszulesen.
AbfReader	Der Kern der gesamten Reader Funktionalität. Hier wird zuerst dekomprimiert, dann wird ein digitales AddressBookFile erzeugt welches zu den verantwortlichen Fraktionsklassen weitergereicht wird. Diese erzeugen eine Instanz mit den gelesenen Daten und speichern diese in das abf.
AbfFullTrailer	Repräsentiert einen Full Trailer und verfügt über die Methoden um dieses auszulesen
AbfDeltaTrailer	Repräsentiert einen Delta Trailer und verfügt über die Methoden um dieses auszulesen
FileType	Definiert aufgrund der ersten 16 Byte eines eingelesenen Files, ob dieses vom Typ Full, Delta oder ungültig ist.

DecompressedData	Pro 65536 Byte Block wird eine DecompressData Instanz erstellt, welche zusätzlich zu den Bytes über ein Flag verfügt, das aussagt, ob zur Zeit diese Bytes zum Extrahieren benutzt werden.

Tabelle 7: Klassenbeschreibung Reader

3.5.2.5. Sequenz Diagramm

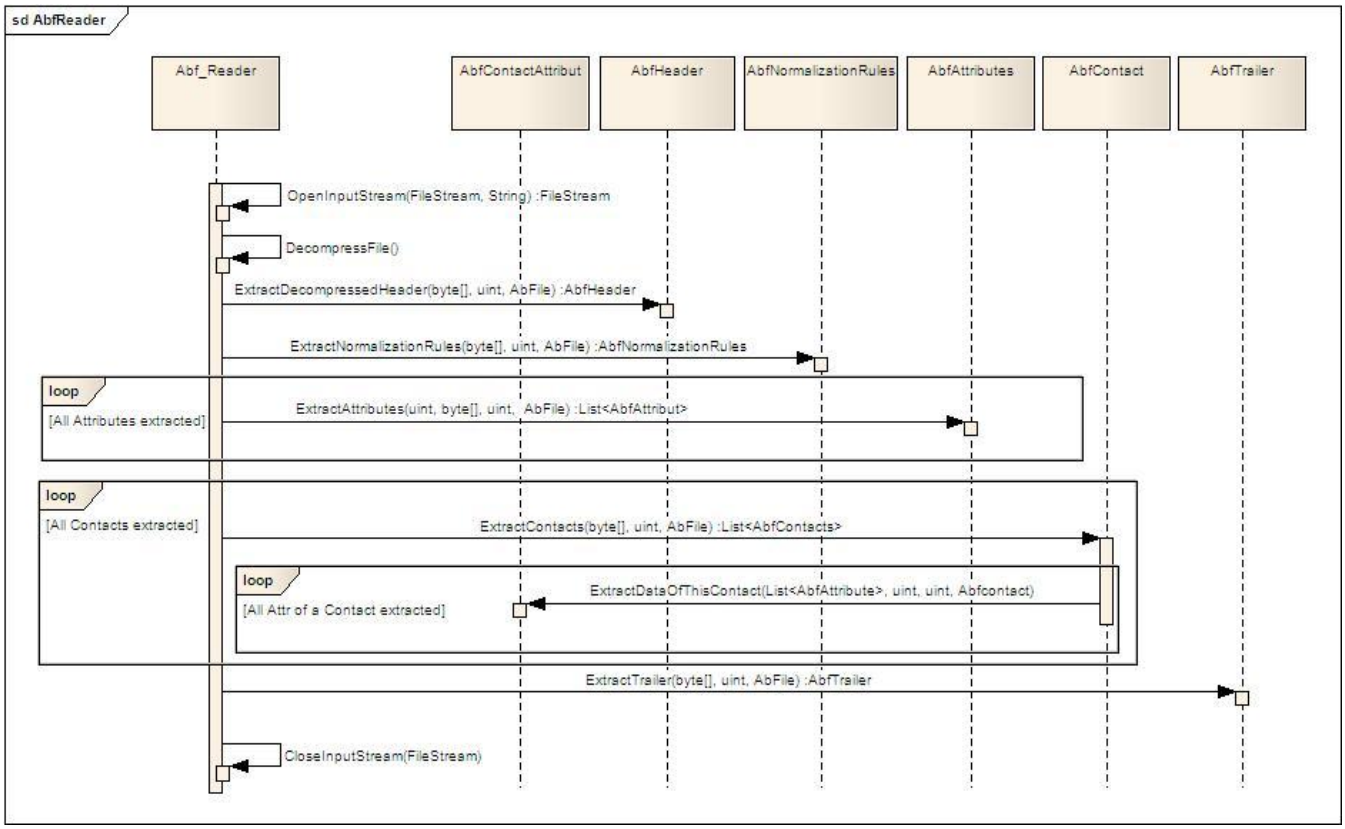


Abbildung 11: SD Reader

3.5.2.6. Operationen

- **OpenInputStream(***FileStream inputFile, String inputFileName***)**
  Öffnet den Stream zum File, das eingelesen werden soll.

- **DecompressFile()**
  Liest die Daten aus dem InputStream und dekomprimiert sie, sofern dies erforderlich ist.

- **ExtractDecompressedHeader(***byte[] dataBuffer, uint index, AbFile abf***)**
  Nimmt die dekomprimierten Daten in Form eines Byte[] entgegen und extrahiert daraus die Headerinformationen. Die so gewonnen Informationen werden in einem mit gereichten digitalen Abbild des AddressBookFiles (abf), in Form eines DecompressedHeaderobjektes, gespeichert.

- **ExtractNormalizationRules(***byte[] dataBuffer, uint index, AbFile abf***)**
  Extrahiert aus den dekomprimierten Daten die NormalizationRules. Diese werden als NormalizationRules-Objekt im mitgereichten abf gespeichert.

- **ExtractAttributes(***byte[] dataBuffer, uint index, AbFile abf, uint maxAttributId***)**
  Da die Anzahl Attribute variable ist, wird diese Methode in einer Schlaufe aufgerufen, die das Abbruchkriterium maxAttributId (DecompressedHeader- Feld) verwendet. Es wird eine Liste erzeugt welche über alle Attribute verfügt. Diese wird dann ebenfalls auf dem abf gespeichert.

- **ExtractContact(***byte[] dataBuffer, uint index, AbfFile abf***)**
  Extrahiert mittels Schlaufe alle Kontakte. Um dies sauber bewerkstelligen zu können, muss das Auslesen des KontaktAttribut mittels einer inneren Schlaufe geschehen.

- **ExtractDataOfThisContact(***List<AbfAttribute> attributeCatalogue, byte[] dataBuffer, uint index, AbfContact contact, uint maxAttributNr***)**
  Extrahiert die Values zu den Attributen innerhalb eines Kontakts. Dies geschieht ebnfalls mittels einer Schlaufe, da jeder Kontakt eine variable Anzahl von Attributen besitzen kann.

- **ExtractTrailer(**byte[] dataBuffer, uint index, AbfFile abf**)**
  Extrahiert die Trailerinformationen und speichert diese in Form eines Trailer- Objektes ins abf.

- **CloseInputStream(***FileStream inputStream***)**
  Schliesst den Inputstream, nachdem das File erfolgreich eingelesen werden konnte.

3.5.3. Package Writer

3.5.3.1. Beschreibung des Package

Dem Writer Package werden alle Entwicklungskomponenten zugeteilt, die in irgendeiner Form mit dem Generieren und der Vorbereitung dazu zu tun haben.

3.5.3.2. Klassendiagramm

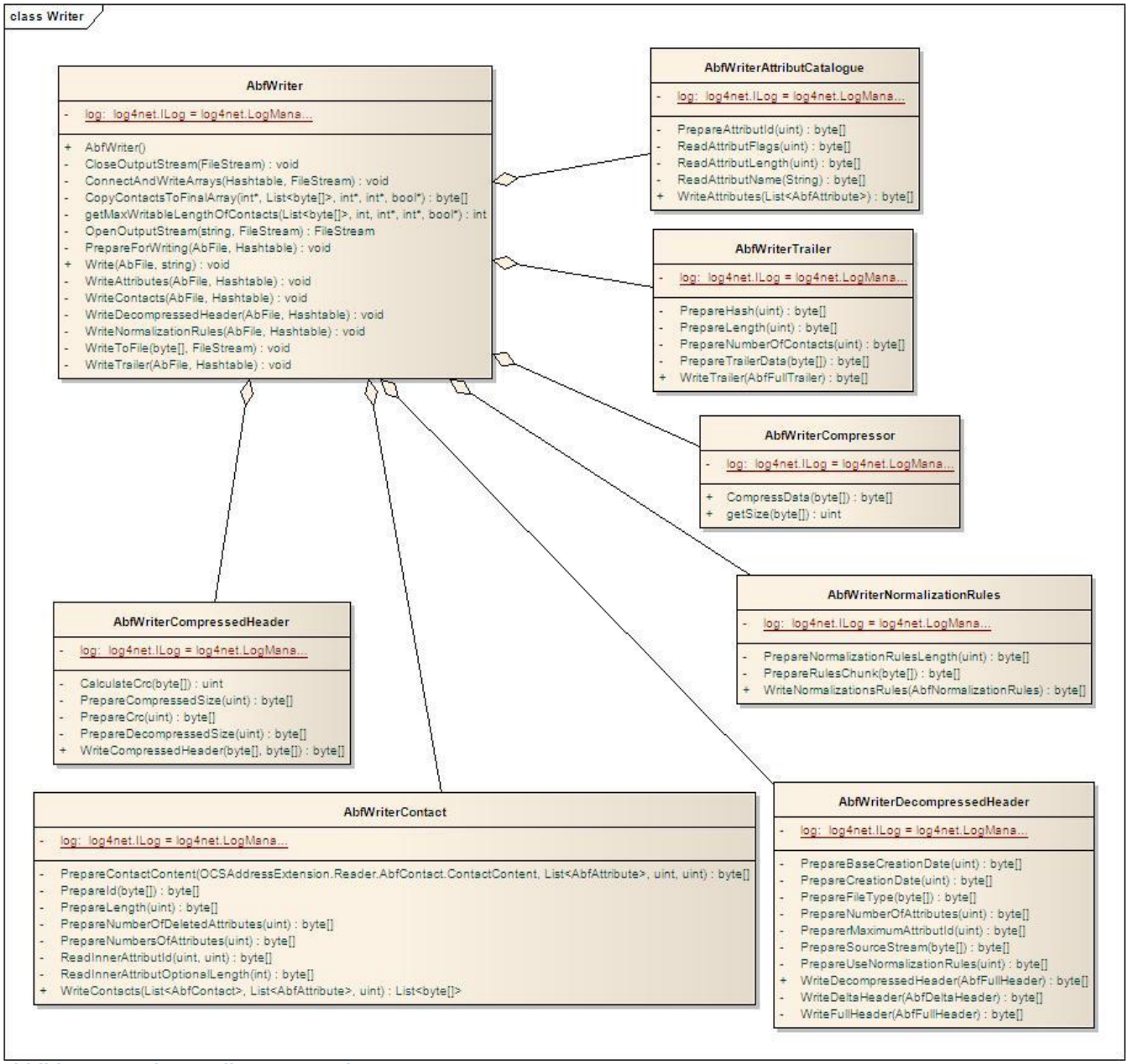


Abbildung 12: Klassendiagramm Writer

3.5.3.3. Klassenbeschreibung

Klasse	Beschreibung
AbfWriterContact	Diese Klasse transformiert alle Kontaktdaten, welche im digitalen Abbild des Kontaktfiles gespeichert sind, in Bytes.
AbfWriterDecompressedHeader	Erstellt einen neuen Header aufgrund der gelieferten Headerinformationen des digitalen AbFiles.
AbfWriterTrailer	Erzeugt einen neuen Trailer aus den gelieferten Trailerinformationen.
AbfWriterNomalizationRules	Erzeugt den NormalizationRules Byteblock
AbfWriterCompresser	Falls erwünscht, kann mit dieser Klasse die Kompression durchgeführt werden.
AbfWriterCompressedHeader	Erstellt aufgrund der vorhandenen Daten einen neuen CompressedHeader
AbfWriterAttributeCatalogue	Alle Attribute werden in Byte- Form konvertiert
AbfWriter	Die Hauptklasse der gesamten Schreibfunktion. Sie delegiert die Bytetransformationen der einzelnen Kontaktfilestrukturen an die zuständigen Klassen. Verbindet die Bytes der verschiedenen Strukturen und begrenzt die Grösse auf 65536 sofern notwendig. Auch das Schreiben der Daten in ein .lsabs File wird hier getätigt

Tabelle 8: Klassenbeschreibung Writer

3.5.3.4. Sequenz Diagramm

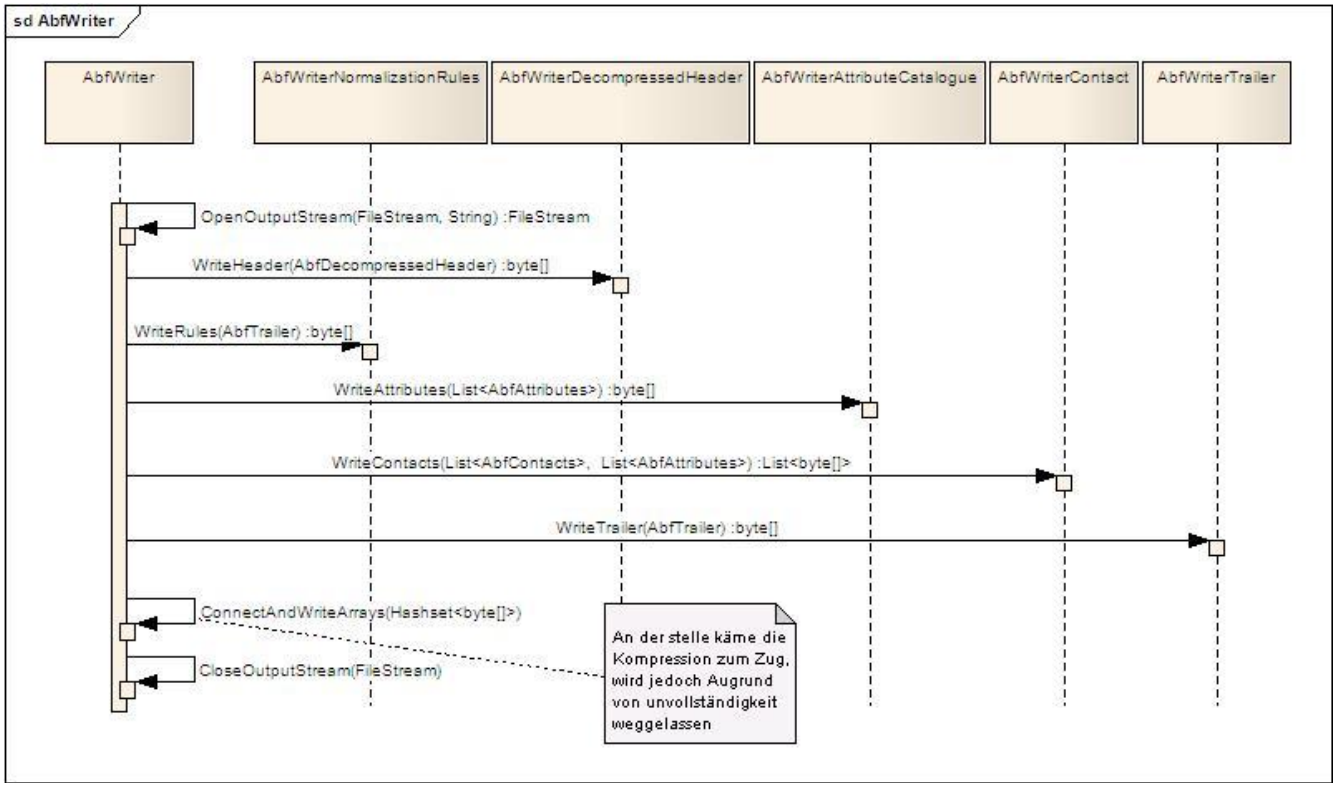


Abbildung 13: SD Writer

3.5.3.5. Operationen

- **OpenOutputStream(***FileStream outputstream, string target***)**
  Diese Operation öffnet den Stream und erstellt das Target- File.

- **WriteHeader(***AbfFullHeader decompressedHeader***)**
  Nimmt den Header des eingelesenen und/oder bearbeiteten AddressBook- Files entgegen und konvertiert die Header- Instanz zurück in ein byte[], welches auch gleich den ReturnTyp repräsentiert.

- **WriteRules(***AbfNormalizationRules rules***)**
  Nimmt die NormalizationRules des eingelesenen und/oder bearbeiteten AddressBook- Files entgegen und konvertiert die Rules- Instanz zurück in ein byte[], welches auch gleich den ReturnTyp repräsentiert.

- **WriteAttributes(**List<AbfAttributes> attributeCatalogue**)**

Nimmt den AttributeCatalogue, eine Liste die allen AbfAttributes beinhaltet, des eingelesenen und/oder bearbeiteten AddressBook- Files entgegen und konvertiert die komplette Liste zurück in ein byte[], welches dann returniert wird.

- **WriteContacts(***List<AbfContacts> contacts, List<AbfAttributes> attributeCatalogue***)**
  Mittels des attributeCatalogue kann festgestellt werden, welches ContactAttribut mit welcher Id verpackt werden muss. Jeder Kontakt wird in ein eigenes byte[] abgefüllt und einer Liste hinzugefügt. Diese Liste beinhaltet alle AbfContacts in byte[] – Form und stellt auch den Return-Wert dar.

- **WriteTrailer(***AbfTrailer trailer***)**
  Nimmt den Trailer des eingelesenen und/oder bearbeiteten AddressBook- Files entgegen und konvertiert die Trailer- Instanz zurück in ein byte[], welches auch returniert wird.

- **ConnectAndWriteArrays(***Hashset allInformationInByteArray***)**
  Alle generierten byte[] werden zusammengefügt und falls notwendig auf 65536 Blöcke Bschränkt, mit einem Compressed Header versehen und ins Output- File geschrieben. Hier wird auch die Komprimierung stattfinden.

- **CloseOutputStream(***Filestream outputStream***)**
  Schliesst den Stream, nachdem alles geschrieben wurde.

4. Implementierung

Dieses Kapitel beschreibt spezielle Implementierungsdetails, die für das Verständnis von OCSAddressExtension notwendig sind. Es werden nur diejenigen Aspekte herausgezogen, die möglicherweise nicht selbsterklärend sind.

4.1. Generieren von Bytes aus Uint

Im Writer müssen für die Generierung des neuen Files alle vorhandenen Daten in Byte- Form konvertiert werden. Um eine fehlerfreie Wandlung der Daten vom Typ Uint in ein Byte gewährleisten zu können wird die Systemfunktion BitConverter verwendet:

```
Byte[] byteValue = BitConverter.GetBytes(122345);
```

Die Headerfelder variieren jedoch in ihrer Grösse zwischen 1 und 4 Byte. Nun generiert der BitConverter aber immer vier Byte grosse Arrays aus einem Uint. Dies hat allerdings fatale Konsequenzen für das neu generierte File – es ist unbrauchbar. Um dieses Problem zu beheben mussten diverse Castoperationen eingeführt werden. Da in der gesamten OCSAddressExtension fast ausschliesslich mit Uint gearbeitet wird, muss bei der Erstellung von zwei Bytefeldern ein Cast in einen UShort getätigt werden. Damit liegt der BitConverter- Operation ein zwei Byte Datentyp zugrunde, und dementsprechend wird auch nur ein zwei Byte grosses Array erstellt, welches sich optimal in die Filegenerierung einfügt.

Type	Beschreibung
Uint	4 Byte
Ushort	2 Byte

Tabelle 9: Bytelänge der Datentypen

4.2. Logging

Das Log- Verhalten der log4net- Bibliothek wird durch die Konfiguration in OCSAddressExtension.exe.config beeinflusst. Zu beachten ist, dass die Sektion mit log4net gekennzeichnet ist und diese dem *System.configuration.IgnoreSectionhandler* bekannt gegeben wurde. Damit wird das Laden dieser Sektion durch das .net Framework unterbunden.

```
  <configSections>
    <section name="log4net" type="log4net.Config.Log4NetConfigurationSectionHandler, log4net"
/>
  </configSections>
```

Da die log4net Bibliothek bei Programmstart explizit geladen wird, verhindert die obige Massnahme, ein doppeltes Laden von log4net.

5. Ausblick

Aufgrund von diversen Schwierigkeiten und Problemen, die sich während der Dauer des Projektes immer wieder eingeschlichen haben, musste der ursprünglich geplante Funktionsumfang eingeschränkt werden. Die Priorität wurde auf eine Version festgelegt, bei der das Einlesen und das Schreiben inklusive Kompression einwandfrei funktionieren sollte. Somit lassen sich die verbleibenden Tasks in drei grobe Kategorien aufteilen:

- Erweiterungen
- Fertigstellung
- Optimierung

5.1. Erweiterungen

Dazu gehört der Merger. Dies entspricht der Funktionalität, welche die eingelesenen Daten aus dem Kontaktfile mit denjenigen aus dem DF Request vergleicht. Sofern sich diese zwei Datenmengen unterscheiden, käme dann noch die Aufgabe des Ergänzens von Kontaktdetails bzw. das Erweitern der Kontaktliste dazu.

5.1.1. Ergänzen von Kontaktdetails

Das Ziel dieser Implementation ist das Ergänzen von Kontaktdetails. Dazu werden die extrahierten Kontaktdaten des Kontaktfiles verglichen mit dem Resultat aus dem DF Request. Ist nun ein Kontakt in beiden Quellen vorhanden, wird untersucht, ob die beiden Kontakte auch inhaltlich übereinstimmen. Ist dies nicht der Fall und der Kontakt aus dem DF Request verfügt über mehr Details als der Kontakt aus dem Kontaktfile, sollen alle zusätzlichen Informationen aus dem DF Kontakt in den andern übernommen werden. Siehe Bsp. unten.
Tritt der Fall ein, dass der Kontakt aus dem Kontaktfile über mehr Informationen verfügt, hat dieser bereits die maximal mögliche Detailmenge und das Ergänzen wird hinfällig.
Es muss berücksichtigt werden, dass nur Kontaktdetails ins Kontaktfile aufgenommen werden können die einem der 23 definierten Attribute entsprechen. Definiert sind diese in der Klasse *ContactContent* oder zu finden unter Punkt 3.3.5.3.2.

Kontaktfile	
Attribut	Value
name	muster
vorname	hans
tel	055 123 45 45
tel mobile	076 589 89 89
tel geschäft	-
PLZ	7865

DF Request	
Attribut	Value
name	muster
vorname	hans
tel	055 123 45 45
tel mobile	076 589 89 89
tel geschäft	055 210 34 58
wohnort	Ortingen

Resultat Kontakt	
Attribut	Value
name	muster
vorname	hans
tel	055 123 45 45
tel mobile	076 589 89 89
tel geschäft	055 210 34 58
wohnort	Ortingen
PLZ	7865

Folgende Punkte sind hierbei zusätzlich zu berücksichtigen:

Feld	Beschreibung
NumberOfAttributes	Dies ist ein Feld des Kontaktes. Dieses beschreibt wie viele Attribute für diesen Kontakt definiert sind. Werden nun neue Details hinzugefügt, egal ob das Attribut schon verwendet wurde oder neu dazu kommt, muss dieser Feldwert angepasst werden.
Length	Dies ist ein weiteres Feld des Kontaktes. Es enthält die komplette Länge des Kontaktfiles – 2 Byte (LängenHeaderFeld). Da der Writer die Arraygrösse des Kontaktes aufgrund dieser Information vorberechnet, ist es zwingend notwendig, die Anzahl an hinzugefügten Bytes hier festzuhalten.

Tabelle 10: Feldanpassungen

5.1.2. Erweitern der Kontaktliste

Um die Kontaktliste des Kontaktfiles erweitern zu können, bedarf es erneut eines DF- Requests. Dieser liefert eine gewisse Menge an Kontaktinformationen, welche mit denjenigen aus dem Kontaktfile verglichen werden müssen.
Die Kontakte, welche nur im DF- Request und nicht auch im Kontaktfile zu finden sind, bilden die Menge jener Kontakte die nun der Liste des Kontaktfiles hinzugefügt werden müssen.
Soll dies erfolgreich von statten gehen sollte, folgenden Punkten besondere Aufmerksamkeit geschenkt werden.

1) Es muss pro Kontakt, der hinzugefügt werden soll, eine AbfContact- Instance erstellt und abgefüllt werden.

Feld	Value	Beschreibung
NumberOfAttributes(uint)	calculating	Beschreibt die Anzahl an Attributen für diesen Kontakt
Length(uint)	calculating	Repräsentiert die gesamte Menge an Bytes des Kontaktes. Muss mit dem Momentanen Stand eingegeben werden, da der Writer die Arraygrösse für einen Kontakt damit kalkuliert.
ContactData(class)	abfüllen	Die eigentlichen Kontaktdaten werden hier einem Attribut zugeordnet. Jedes Feld entspricht einer Liste, um auch z.B. mehrere Mobieltelefon-Einträge für einen Kontakt möglich zu machen.
Id(16 Byte[])	caculating	Es ist noch nicht bekannt, ob dieser Wert nach einem Muster generiert wird, oder ob er lediglich innerhalb der Kontaktliste unikat sein muss.
IsDeleted (bool)	false (default)	Ist der Kontakt noch gültig
IsSentinel	false (default)	der Letzte Kontakteintrag der Liste MUSS Sentinell sein, um den Schluss zu signalisieren. Folgende Felder müssen den genannten Werten entsprechen. - NumberOfAttributes = 0 - Length = 0x12
NumberOfDeletedAttributes	0 (default)	Feld ist spezifiziert, der eigentliche Sinn konnte bis anhin noch nicht eruiert werden.

Tabelle 11: Feldanpassungen

2) Nachfolgende Felder ausserhalb des eigentlichen Kontaktes müssen angepasst werden, damit das Kontaktfile als Ganzes nicht inkonsistent wird.

Feld	Klasse	Beschreibung
NumberOfContacts	Trailer	Dieses Feld beinhaltet die gesamte Anzahl an Kontakten OHNE den Sentinel

Tabelle 12: Feldanpassungen

5.1.3. Einstiegspunkt für den Merger

```
static void Main()
  {
     //load log4net
     log4net.Config.XmlConfigurator.Configure();

     //load config
     ConfigurationReader config ConfigurationReader.ReadConfiguration(Constant.CONFIG_FILE);

     //read file
     AbFile abf = new AbfReader().Read(config.ReadFile);

     // -> merge contacts
     // -> complement contacts

     //write file
     if (abf != null)
         new AbfWriter().Write(abf, config.WriteFile);
  }
```

5.2. Fertigstellung

Der Komprimierungsalgorithmus hat beim Implementieren mehr Probleme aufgeworfen als ursprünglich kalkuliert. Dazu kommt noch der eigentlich erfolgreiche Test, nachzulesen in der Testdokumentation[4], und dennoch akzeptiert der OCS bzw. AbServer.exe das komprimierte File nicht. Hierzu wären folgende Abklärungen sinnvoll:

1) Warum akzeptiert AbServer.exe das korrekt komprimierte File nicht?
   a. Bleibt abzuklären, ob der Komprimierungsalgorithmus, entgegen der im Microsoftdokument[5] vertretenen Meinung, doch nicht egal ist.
2) Der zurzeit verwendete Algorithmus hat mit LZ77 nicht viel zu tun, dazu kommen noch die versteckten Bugs. Daher würde sich empfehlen, einen neuen zu implementieren.

Genauere Informationen, den Aufbau und die Struktur eines Komprimierten Files betreffend, können der Technischen Daten Dokumentation entnommen werden.

5.2.1. Erkenntnisse aus dem ersten Versuch

Die Erkenntnisse sind der Technischen Dokumentation[6] zu entnehmen.

5.2.2. Recyclebare Methoden

Nachfolgend aufgeführte Methoden stammen aus der Klasse *Commen.Komprimieren.LZ77*

- `EncodeOffsetLength`
  Diese Methode muss verwendet werden, um die von Microsoft geforderte Struktur erstellen zu können. Wird immer dann eingesetzt, um bei einer gefundenen Sequenz Offset und Länge in der richtigen Form ins Outputarray schreiben zu können.

- `UpdateTokenIndex`
  Diese Methode ist mit einem Unit Test versehen und erfüllt ihre Aufgabe zuverlässig. Sollte immer dann verwendet werden, um nach gefundenen Komprimierungssequenzen den TokenIndex zu aktualisieren.

5.3. Optimierungen

5.3.1. Längenberechnung im Kontakt

Der AbfWriter_Contact benutz das Längenfeld des Kontaktes, um das Byte[] für den Kontakt erstellen zu können. Soll nun der Merger implementiert werden, müsste beim Hinzufügen eines neuen Kontaktes die genaue Anzahl an Bytes berechnet werden.
Um den Komfort beim Mergen zu erhöhen, könnte eine alternative Lösung für den AbfWriter_Contact gesucht werden. Z.B. wäre das kontinuierliche Anpassen der Arraygrösse ein möglicher Ansatz, die Performance würde allerdings darunter leiden.

[4] Dokumentation der Tests.docx
[5] MS Address Book File Structure
[6] OCS-AddressBook-File Beschreibung.docx

6. Literaturverzeichnis

Dokumentationsliteratur
Microsoft, Address Book File Structure, 2009
Craig Larma, UML 2 und Patterns angewendet – Objektorientierte Softwareentwicklung, 1. Auflage 2005

Entwicklungsliteratur
Microsoft, Address Book File Structure, 2009
http://msdn.microsoft.com
http://de.wikipedia.org
Jesse Liberty und Donald Xie, Programmieren mit C# 3.0, 3. Auflage 2008
Gespräch mit A. Kobler bezüglich der Nutzung von log4net
https://wiki.ins/index.php/.NET_Programming

7. Abbildungsverzeichnis

Abbildung 1: Domainmodell 7
Abbildung 2: SSD BlackBox 8
Abbildung 3: SSD OCS Filegenerierung (Beispiel) 9
Abbildung 4: Zusammenspiel Reader/Writer 11
Abbildung 5: Packagestruktur 15
Abbildung 6: Solutionübersicht OCSAddressBookExtension 16
Abbildung 7: Package- und Klassendiagram OCSAddressBookExtension 17
Abbildung 8: Klassendiagramm Reader 18
Abbildung 9: SSD Reader 20
Abbildung 10: Klassendiagramm Writer 21
Abbildung 11: SSD Writer 22

8. Tabellenverzeichnis

Tabelle 1: Beschreibung Operations 8
Tabelle 2: GetKontaktFile- Operation 9
Tabelle 3: AddNewContacts- Operation 9
Tabelle 4: WriteNewFilesBack- Operation 10
Tabelle 5: Attribut Vergleich 14
Tabelle 6: Klassenbeschreibung Commen 17
Tabelle 7: Klassenbeschreibung Reader 20
Tabelle 8: Klassenbeschreibung Writer 22
Tabelle 9: Bytelänge der Datentypen 23
Tabelle 10: Feldanpassungen 25
Tabelle 11: Feldanpassungen 25
Tabelle 12: Feldanpassungen 25

Office Communication Server Adressbucherweiterung

Dokumentation der Tests

0. Dokumentinformationen

0.1. Änderungsgeschichte

Datum	*Version*	*Änderung*	*Autor*	*Qualitätssicherung*
4.11.2009	1.0	erstellt	Dietrich	
18.11.2009	1.1	Beschreibung der Tests hinzugefügt	Dietrich	
6.12.2009	1.2	Validierungs-und Kompressionstest hinzugefügt	Dietrich	
7.12.2009	1.3	Weiter Validierungstests hinzugefügt	Dietrich	
11.12.2009	1.4	Beschreibung UnitTests hinzugefügt	Dietrich	Müller
11.12.2009	1.5	Korrekturen vorgenommen	Dietrich	Müller
17.12.2009	2.0	Finale Version	Dietrich	

0.2. Inhalt

0. Dokumentinformationen 2
0.1. Änderungsgeschichte 2
0.2. Inhalt 3
1. Systemtests / Validierung generierter Files 4
1.1. Einführung 4
1.1.1. Ziel 4
1.1.2. Ablauf 4
1.1.3. Einsatz der Testmittel 4
1.1.3.1. Original Files 4
1.1.3.2. AbServer.exe 4
1.1.3.3. OCSAddressBookExtension 4
1.2. ValidierungsTest Sw9 5
1.2.1. Vorbereitung 5
1.2.2. Auswertung 5
1.2.3. Erkenntnis 5
1.3. ValidierungsTest Sw11 5
1.3.1. Vorbereitung 5
1.3.2. Auswertung 6
1.3.3. Erkenntnis 6
1.4. ValidierungsTest Sw12 6
1.4.1. Vorbereitung 6
1.4.2. Auswertung 6
1.4.3. Erkenntnis 7
1.5. Systemtest \ Funktionalität des Komprimierens 7
1.6. KomprimierungsTest Sw11 7
1.6.1. Vorbereitung 7
1.6.2. Vorgehen 7
1.6.3. Auswertung 7
1.6.4. Erkentnis 7
2. UnitTests 8
2.1. AbfAttributeTest 8
2.2. AbfContactAttributCalculatorTest 8
2.3. AbfContactTest 9
2.4. AbfFullHeaderTest 9
2.5. AbfFullTrailerTest 10
2.6. AbfNormalizationRulesTest 10
2.7. AbfWriterAttributeCatalogue 10
2.8. AbfWriterCompressedHeaderTest 11
2.9. AbfWriterDecompressedHeaderTest 11
2.10. DecompressDataTest 12
2.11. CrcTest 12
2.12. AbfWriterNormalizationRulesTest 12
2.13. AbfWriterTrailerTest 13
2.14. FileTypeTest 13
2.15. LZ77Test 13
3. Tabellenverzeichnis 14

1. Systemtests / Validierung generierter Files

1.1. Einführung

1.1.1. Ziel

Der Test soll zweierlei aufzeigen. Zum einen soll überprüft werden, ob die vom OCSAddressBookExtension eingelesenen und neu geschriebenen Files mit dem vom OCS erwarteten Format kompatibel sind. Dies wird mit Hilfe des OCS-Tools AbServer.exe überprüft.
Zum andern soll die Kompatibilität zwischen dem vom Writer generierten Output-File und dem Reader geprüft werden. Der Test soll zeigen, ob es dem *AbfReader* möglich ist die vom *AbfWriter* generierten Files einzulesen.

1.1.2. Ablauf

Die originalen Files vom OCS werden vom Reader eingelesen und vom Writer neu geschrieben. Die neu generierten Files werden nun auf den OCS kopiert und dort mittels des AbServer.exe validiert. Danach werden die Files vom OCSAddressBookExtension-Tool eingelesen um sicherzustellen, dass die vom Writer generierten Dateien mit dem Reader kompatibel sind.

1.1.3. Einsatz der Testmittel

1.1.3.1. Original Files

Folgende OCS-Kontaktfiles bilden die Grundlage für die Tests. Die generierten Testfiles sind immer aus einem der folgenden Originale[1] erstellt worden.

- F-0c7b.lsabs (8 Kontakte)
- F-0c89.lsabs (20009 Kontakte)
- D-0c79-0c7b.lsabs (1 Kontakte)
- ca500.lsabs (446 Kontakte)
- ca600.lsabs(555 Kontakte)
- ca700.lsabs(669 Kontakte)

1.1.3.2. AbServer.exe

Zu finden ist AbServer.exe auf dem OCS unter folgendem Pfad:
c:\Program Files\Microsoft Communications Server 2007 R2\Server\Core
AbServer.exe in der Konsole starten und mittels des Befehls „ AbServer.exe –dumpFile inputFile“ das gewünschte File einlesen. Kann der Server daraus eine .txt mit demselben Namen wie das inputFile generieren, war der Test erfolgreich.
Konsolenausgabe bei Erfolg:

```
Text representation of „target.lsabs“ “target.txt”
```

Konsolenausgabe bei Fehler:

```
Error reading file – “target.lsabs”
Exception: Error HRESULT E_FAIL has been returned from a call to a CIM component.
```

1.1.3.3. OCSAddressBookExtension

Da der Pfad für das Input-und das Output-File in der Konfiguration festgehalten ist, muss dieser nach der Generierung der Testfiles neu definiert werden. Das Write-File wird nun zum Read-File. Das Konfigurations-XML, OCSAddressExtension.exe.config findet sich im Arbeitsverzeichnis von OCSAddressBookExtension.

```
<FilePath>
  <ReadFile>…\OCSAddressExtension\Contact Files\F-0c89.lsabs</ReadFile>
  <WriteFile>…\OCSAddressExtension\Contact Files\testling.lsabs</WriteFile>
</FilePath>
```

[1] Zu finden auf der CD unter: *doc\06_TestDokumentation\OCSAddressBookExtension\TestFiles*

1.2. ValidierungsTest Sw9

1.2.1. Vorbereitung

Für diesen Test wurden drei Test Files erstellt:

FullFileKleinNichtKomp.lsabs	FullFileGrossNichtKomp.lsabs	DeltaFileKleinNichtKomp.lsabs
Decompressed Size: 2552 Byte Compressed Size: 2552 Byte 65536 Byte Blocks: 1 Attributes: 21 Contacts(incl. Sentinel): 8 compressed: no Original: F-0c7b.lsabs	Decompressed Size: ca. 9100000 Byte Compressed Size: ca. 9100000 Byte 65536 Byte Blocks: 143 Attributes: 21 Contacts(incl. Sentinel): 20009 compressed: no Original: F-0c89.lsabs	Decompressed Size: 1364 Byte Compressed Size: 1364 Byte 65536 Byte Blocks: 1 Attributes: 21 Contacts(incl. Sentinel): 1 compressed: no Original: D-0c79-0c7b.lsabs

Tabelle 1: Testfiles Validierungstes Sw9

1.2.2. Auswertung

File	AbServer.exe	OCSAddressBookExtension
FullFileKleinNichtKomp.lsabs	ok	ok
FullFileGrossNichtKomp.lsabs	fail	ok
DeltaFileKleinNichtKomp.lsabs	fail	ok

Tabelle 2: Auswertung Validierungstest Sw9

1.2.3. Erkenntnis

- Delta-Files können aufgrund ihrer Beschaffenheit nicht mit dem AbServer.exe validiert werden. Um ein Delta-File sinnvoll einsetzen zu können, bräuchte man zusätzlich die zwei Full-Files, die zur Generierung des Delta verwendet wurden.
- Der Writer scheint bei Files, die aus mehreren 65536 Byteblöcken bestehen, noch Übergangsprobleme zu haben.
- Aufgrund der Auswertung der CRC-Werte ist ersichtlich, dass ca. 1/5 der 143 CRC-Berechnungen nach dem neuen Schreiben durch den Writer nicht mehr stimmen. Es gilt abzuklären, dass die Konvertierung der Daten in Bytearrays sauber vonstattengeht.

1.3. ValidierungsTest Sw11

1.3.1. Vorbereitung

Die Konvertierungen der Byte im Writer wurden neu implementiert. Die ab und zu erzeugten Rundungsfehler sollten damit ausgemerzt sein.
DeltaFiles wurden aus der Testliste gestrichen, da diese nicht sinnvoll getestet oder validiert werden können.
Für diesen Test wurden drei Test Files erstellt:

FullFileKleinNichtKomp.lsabs	FullFileGrossNichtKomp.lsabs	FullFileKleinKomp.lsabs
Decompressed Size: 2552 Byte Compressed Size: 2552 Byte 65536 Byte Blocks: 1 Attributes: 21 Contacts(incl. Sentinel): 8 compressed: no Original: F-0c7b.lsabs	Decompressed Size: ca. 9100000 Byte Compressed Size: ca. 9100000 Byte 65536 Byte Blocks: 143 Attributes: 21 Contacts(incl. Sentinel): 20009 compressed: no Original: F-0c89.lsabs	Decompressed Size: 2552 Byte Compressed Size: 1562 Byte 65536 Byte Blocks: 1 Attributes: 21 Contacts(incl. Sentinel): 8 compressed: yes Original: F-0c7b.lsabs

Tabelle 3: Testfiles Validierungstest Sw11

1.3.2. Auswertung

File	AbServer.exe	OCSAddressBookExtension
FullFileKleinNichtKomprimiert.lsabs	ok	ok
FullFileKleinKomprimiert.lsabs	fail	ok
FullFileGrossNichtKomprimiert.lsabs	ok	ok

Tabelle 4: Auswertung Validierungstest Sw11

1.3.3. Erkenntnis

- Alle bis Sw 11 verfügbaren Full-Files können nun, ohne Kompression, vom OCS erkannt werden.
- Komprimierung funktioniert nicht, wenn angewendet auf dem F-0c89.lsabs (20009 Kontakte)
- Wird die Komprimierung auf dem F-0c7b.lsabs (8 Kontakte) angewandt, ist es dem AbServer.exe nicht möglich dieses zu entpacken/validieren.
- Das Entpacken des FullFileKleinKomprimiert.lsabs mittels des OCSAddressBookExtension Reader's, erzeugt eine Bytefolge welche identisch ist mit dem Original. Dennoch wird das File nicht akzeptiert von AbServer.exe! Der Fehler ist beim Komprimieren zu suchen.

1.4. ValidierungsTest Sw12

1.4.1. Vorbereitung

Für die Abgabe soll die Komprimierung nicht weiter beachtet werden, dafür soll noch mit ein paar zusätzlichen Files der Test gemacht werden, ob sie vom OCS anerkannt werden. Um dies zu bewerkstelligen, kommen zusätzlich drei weitere Kontaktfiles dazu.
Mittels der OCSAddressBookExtension wurden die folgenden fünf Kontaktfiles generiert.

FullFileKleinNichtKomp.lsabs	FullFileGrossNichtKomp.lsabs	Fullca500NichtKomp.lsabs
Decompressed Size: 2552 Byte Compressed Size: 2552 Byte 65536 Byte Blocks: 1 Attributes: 21 Contacts(incl. Sentinel): 8 compressed: no Original: F-0c7b.lsabs	Decompressed Size: ca. 9100000 Byte Compressed Size: ca. 9100000 Byte 65536 Byte Blocks: 143 Attributes: 21 Contacts(incl. Sentinel): 20009 compressed: no Original: F-0c89.lsabs	Decompressed Size: 201123 Byte Compressed Size: 201123 Byte 446 Byte Blocks: 4 Attributes: 21 Contacts(incl. Sentinel): 446 compressed: no Original: ca500.lsab
Fullca600NichtKomp.lsabs	**Fullca700NichtKomp.lsabs**	
Decompressed Size: 250598 Byte Compressed Size: 250598 Byte 65536 Byte Blocks: 4 Attributes: 21 Contacts(incl. Sentinel): 555 compressed: no Original: ca600.lsabs	Decompressed Size: ca. 302398 Byte Compressed Size: ca. 302398 Byte 65536 Byte Blocks: 5 Attributes: 21 Contacts(incl. Sentinel): 669 compressed: no Original: ca700.lsabs	

Tabelle 5: Testfiles Validierungstest Sw12

1.4.2. Auswertung

File	AbServer.exe	OCSAddressBookExtension
FullFileKleinNichtKomp.lsabs	ok	ok
FullFileGrossNichtKomp.lsabs	ok	ok
Fullca500NichtKomp.lsabs	ok	ok
Fullca600NichtKomp.lsabs	ok	ok
Fullca700NichtKomp.lsabs	ok	ok

Tabelle 6: Auswertung Validierungstest Sw12

1.4.3. Erkenntnis

- Alle verfügbaren Testfiles werden vom OCS akzeptiert.
- Alle durch die OCSAddressBookExtension generierten Kontaktfiles werden von dieser akzeptiert und können erneut eingelesen und bearbeitet werden.

1.5. Systemtest \ Funktionalität des Komprimierens

1.6. KomprimierungsTest Sw11

1.6.1. Vorbereitung

Für diesen Test wird die Bytestruktur eines originalen Kontaktfiles und die eines durch die OCSAddressBookExtension generierten Kontaktfiles verglichen. Die Auswertung soll aufzeigen, ob beide Strukturen identisch sind. Sind sie es nicht, hat der Komprimierungsalgorithmus noch Bugs, die ausgemerzt werden müssen.

1.6.2. Vorgehen

Wird im *AbfReader* bei Zeile 35 ein BreakPoint gesetzt kann mittels der Variable abfRawData auf alle Dekomprimierten Daten in byte[]-Form zugegriffen werden.
Eine Kopie der Bytefolge beider Files wird in ein Excel eingefügt und dort verglichen. Problematisch an dieser Methode ist die Grösse der Files, selbst die Kleinen, welche bloss acht Kontakte beinhalten, sind schnell über 2500 Bytes gross. Daher ist unten aufgezeigtes Vorgehen nur empfehlenswert bis zu einer Grösse von ca. 4000 Bytes. Danach wird die Datenmenge einfach zu gross, um zuverlässige Auswertungen machen zu können.

Original File				Generiertes File				Diff
[0]	118	byte		[0]	118	byte		118-118 = 0
[1]	108	byte		[1]	108	byte		0
[2]	225	byte		[2]	225	byte		0
[3]	68	byte		[3]	68	byte		0
[4]	253	byte		[4]	253	byte		0
[5]	10	byte		[5]	10	byte		0
[6]	169	byte		[6]	169	byte		0
[7]	64	byte		[7]	64	byte		0
[8]	139	byte		[8]	39	byte		100

Tabelle 7: Vorgehensbsp. Komprimierungstest

Die beiden Files werden wie oben gezeigt, gegenüber gestellt. Unter „Diff“ wird die Differenz der Bytes, auf dieser Zeile gebildet. Ist die Differenz Null, stimmt die Bytefolge, weicht die Differenz jedoch von null ab, ist beim komprimieren ein Fehler aufgetreten.

1.6.3. Auswertung

Die Auswertung ist dem „KomprimierenAuswertungen.xlsx“ zu entnehmen.

1.6.4. Erkentnis

- Die Bytestruktur ist identisch
- Bleibt zu klären warum das komprimierte File nicht mit AbServer.exe validierbar ist (siehe Testauswertung unter 0)

2. UnitTests

Alle Unit-Tests werden im Projekt *TestProject_OCSAddressExtension* implementiert und ausgeführt.

2.1. AbfAttributeTest

CalculateFlagsTest
Prüft die Konvertierung der Rohdaten des Flags-Feldes in einen Uint.
CalculateIdTest
Prüft die Konvertierung der Rohdaten des Id-Feldes in einen Uint.
CalculateLengthTest
Prüft die Konvertierung der Rohdaten des Length-Feldes in einen Uint.
CalculateNameTest
Testet das korrekte Extrahieren und anschliessende UTF8 Entschlüsseln der Attributvalues

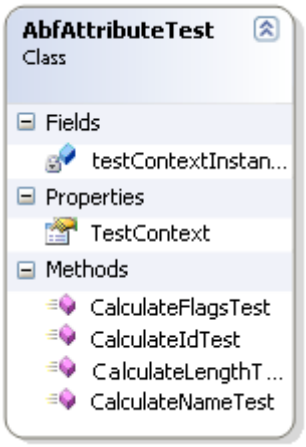


Tabelle 8: AbfAttributeTest

2.2. AbfContactAttributCalculatorTest

CalculateIdTest
Prüft die Konvertierung der Rohdaten des Id-Feldes in einen Uint.
CalculateLengthTest
Prüft die Konvertierung der Rohdaten des Length-Feldes in einen Uint.
EvaluateFieldTypeTest
Prüft ob anhand der Attribut-Id, der korrekte FeldTyp (String, ProxyString, Binary) zugewiesen wird.
GetValueTest
Derselbe Test wie *ExtractStringValue*. Allerdings wird hier auch noch geprüft ob diejenigen Values, die nicht in Strings konvertiert werden müssen, richtig ausgelesen werden.
ExtractStringValueTest
Prüft die korrekte Umwandlung der gelesenen Bytes mittels UTF8-Decodierung in ein Char[] bzw. zu einem String.

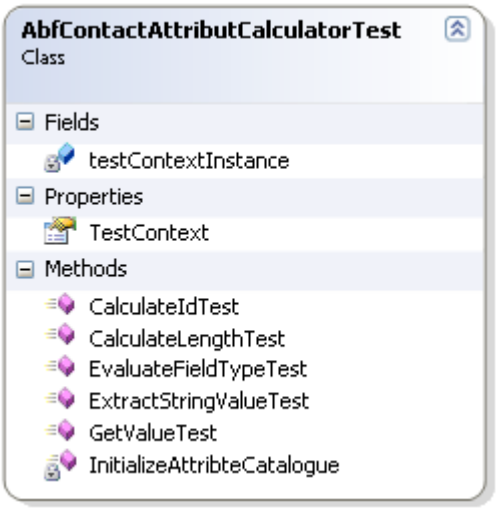


Tabelle 9: AbfContactAttributCalculatorTest

2.3. AbfContactTest

CalculateContactIdTest
Prüft die Konvertierung der Rohdaten des Id-Feldes in einen Uint.
CalculateContactLengthTest
Prüft die Konvertierung der Rohdaten des Length-Feldes in einen Uint.
CalculateNumberOfAttributesInThisContactTest
Prüft die Konvertierung der Rohdaten des NumberOfAttributes-Feldes in einen Uint.
CheckForIsSentinelTest
Prüft ob der Sentineleintrag richtig erkannt wird, bzw. nur kein anderer Kontakteintrag ausversehen als Sentinel interpretiert wird
CheckForIsDeletedTest
Prüft, ob gelöschte Kontakte – und nur gelöschte – als solche markiert werden.

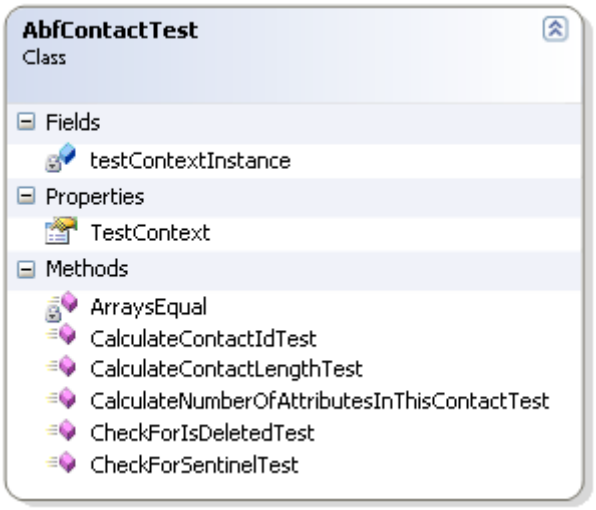


Tabelle 10: AbfContactTest

2.4. AbfFullHeaderTest

CalculateCreationDateTest
Prüft die Konvertierung der Rohdaten des CreationDate-Feldes in einen Uint.
CalculateMaximumAttributIdTest
Prüft die Konvertierung der Rohdaten des MaximumAttributId-Feldes in einen Uint.
CalculateNumberOfAttributes
Prüft die Konvertierung der Rohdaten des NumberOfAttributes-Feldes in einen Uint.
CalculateUseNormalizationRuleTest
Prüft die Konvertierung der Rohdaten des UseNormalizationRules-Feldes in einen Uint.

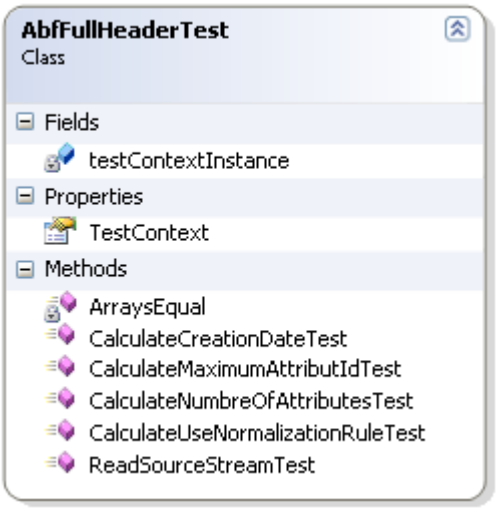


Tabelle 11: AbfFullHeaderTest

2.5. AbfFullTrailerTest

CalculateHashTest
Prüft die Konvertierung der Rohdaten des Hash-Feldes in einen Uint.
CalculateNumberOfContactsTest
Prüft die Konvertierung der Rohdaten des NumberOfContacts-Feldes in einen Uint.
CalculateTrailerLenghtTest
Prüft die Konvertierung der Rohdaten des Length-Feldes in einen Uint.
ReadRestOfTrailerTest
Prüft, ob die Daten zwischen Trailer-Header und dem Trailer-Length-Feld verlustfrei ausgelesen und in ein Array gespeichert werden konnten.

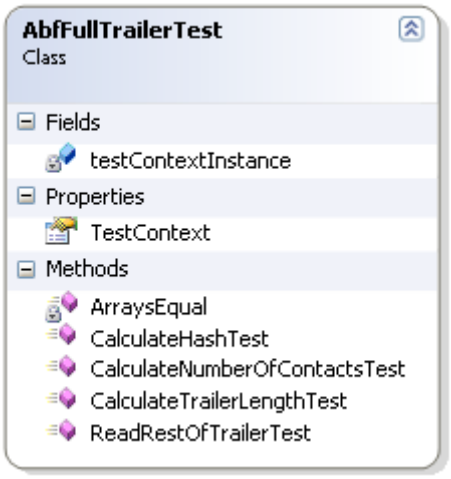


Tabelle 12: AbfFullTrailerTest

2.6. AbfNormalizationRulesTest

CalculateLengthTest
Prüft die Konvertierung der Rohdaten des Length-Feldes in einen Uint.
CalculateRulesTest
Prüft ob die gesamten Rules nach dem Lengenfeld sauber in einem byte[] gespeichert werden.

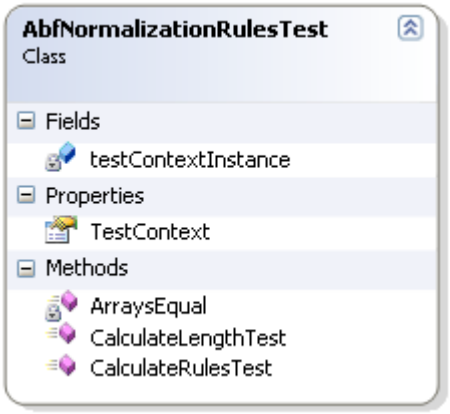
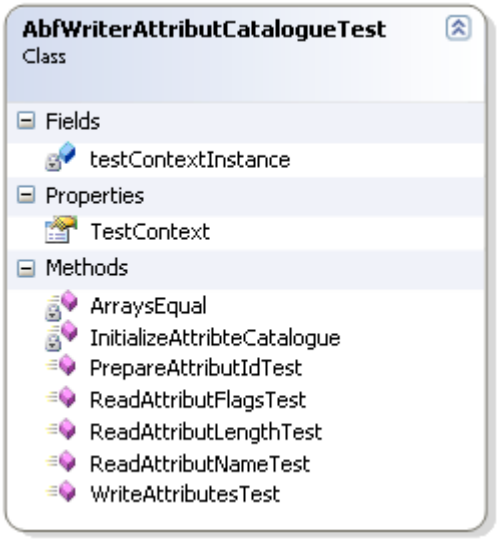


Tabelle 13: AbfNormalizationRulesTest

2.7. AbfWriterAttributeCatalogue

PrepareAttributIdTest
Testet, ob die Rückkonvertierung vom Id Feld in ein Byte[] sauber funktioniert.
ReadAttributFlagsTest
Testet, ob die Rückkonvertierung vom Flags-Feld in ein Byte[] sauber funktioniert.
ReadAttributLengthTest
Testet, ob die Rückkonvertierung vom Length-Feld in ein Byte[] sauber funktioniert.
ReadAttributNameTest
Testet, ob die Rückkonvertierung des strings in ein Byte[] sauber funktioniert.
WriteAttributesTest
Überprüft die Konvertierung eines kompletten AttributeCatalogues, mit allen Inhalten, in ein Byte[]



2.8. AbfWriterCompressedHeaderTest

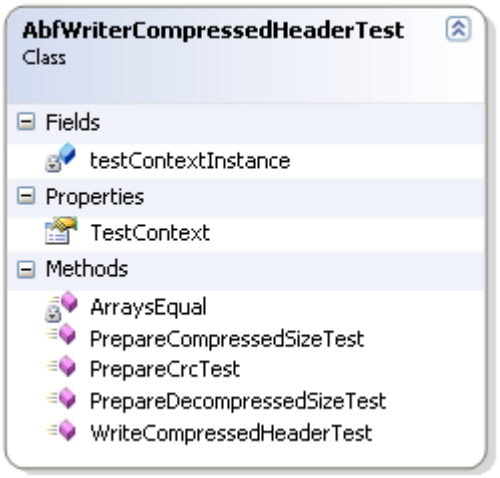
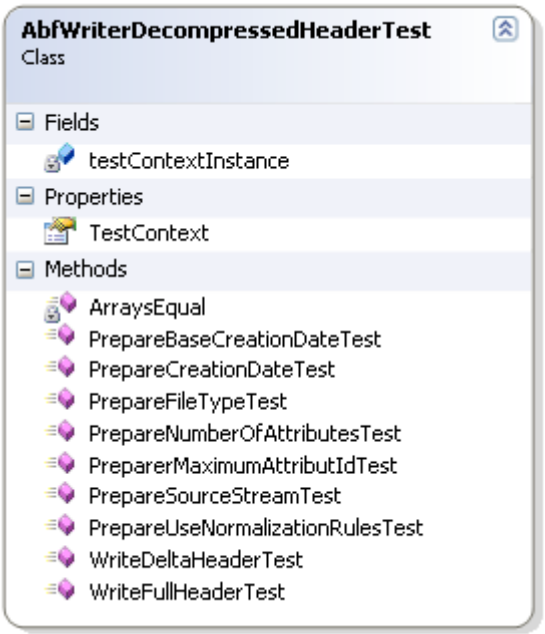
PrepareCompressedSizeTest Testet, ob die Rückkonvertierung vom CompressedSize-Feld in ein Byte[] sauber funktioniert. **PrepareCrcTest** Testet, ob die Rückkonvertierung vom Crc-Feld in ein Byte[] sauber funktioniert. **PrepareDecompressedSizeTest** Testet, ob die Rückkonvertierung vom DecompressedSize-Feld in ein Byte[] sauber funktioniert. **WriteCompressedHeaderTest** Überprüft die Konvertierung eines kompletten CompressedHeaders in ein Byte[]	

Tabelle 14: AbfWriterCompressedHeaderTest

2.9. AbfWriterDecompressedHeaderTest

PrepareBaseCreationDateTest Testet, ob die Rückkonvertierung vom BaseCreationDate-Feld in ein Byte[] sauber funktioniert. **PrepareCreationDateTest** Testet, ob die Rückkonvertierung vom CreationDate-Feld in ein Byte[] sauber funktioniert. **PrepareFileTypeTest** Testet, ob die Rückkonvertierung vom FileType-Feld in ein Byte[] sauber funktioniert. **PrepareNumberOfAttributesTest** Testet, ob die Rückkonvertierung des NumberOfAttributes-Feld in ein Byte[] sauber funktioniert. **PrepareMaximumAttributIdTest** Testet, ob die Rückkonvertierung des MaximumAttributId-Feld in ein Byte[] sauber funktioniert. **PrepareUseNormalizationRulesTest** Testet, ob die Rückkonvertierung des UseNormalizationRules-Feld in ein Byte[] sauber funktioniert. **PrepareSourceStream** Prüft, ob das Source Stream byte[] sauber in den Header integriert wird. **WriteDeltaHeaderTest** Prüft die Generierung eines kompletten DeltaHeaders, mit allen Feldern **WriteFullHeaderTest** Prüft die Generierung eines kompletten FullHeaders,	

mit allen Feldern	

Tabelle 15: AbfWriterDecompressedHeaderTest

2.10. DecompressDataTest

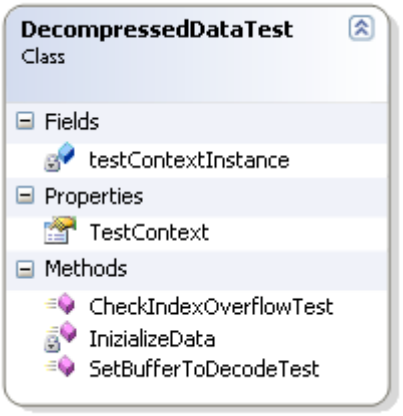
CheckIndexOverflowTest Testet den saubern Austausch der RohdatenByteArrays, wenn der Index die 65536 Grenze überschreiten würde. **SetBuffertoDecodeTest** Prüft das Neusetzten der Markierung, wenn ein Byte[] aufgrund der 65536 Indexüberschreitung ausgewechselt werden muss.	

Tabelle 16: DecompressDataTest

2.11. CrcTest

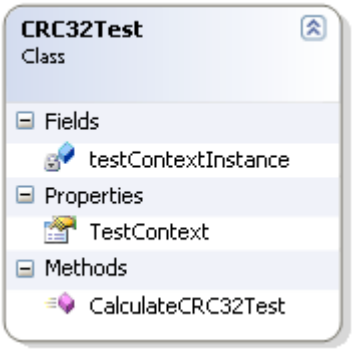
CalculateCRC32Test Prüft die korrekte Berechnung von Crc32-Werten mit Arrays verschiedener Grösse und Länge.	

Tabelle 17: CrcTest

2.12. AbfWriterNormalizationRulesTest

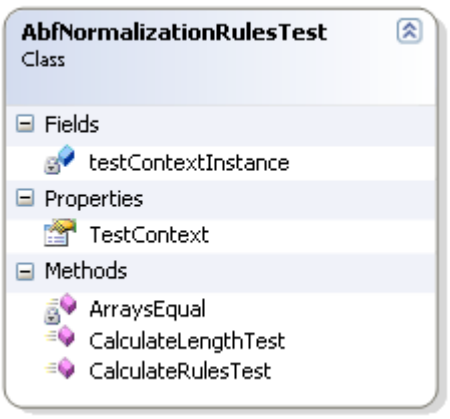
PrepareNormalizationRulesLengthTest Testet, ob die Rückkonvertierung vom Length-Feld in ein Byte[] sauber funktioniert. **PrepareRulesChunkTest** Prüft, ob das RulesChunk byte[] sauber integriert wird. **WriteNormalizationRulesTest** Setzt eine komplette NormalizationRule zusammen und prüft deren Korrektheit.	

Tabelle 18: AbfWriterNormalizationRulesTest

2.13. AbfWriterTrailerTest

PrepareHashTest
Testet, ob die Rückkonvertierung vom Hash-Feld in ein Byte[] sauber funktioniert.
PrepareLengthTest
Testet, ob die Rückkonvertierung vom Length-Feld in ein Byte[] sauber funktioniert.
PrepareNumberOfContactsTest
Testet, ob die Rückkonvertierung vom NumberOfContacts-Feld in ein Byte[] sauber funktioniert.
PrepareTrailerDataTest
Prüft, ob die Trailerdaten sauber in den Trailer integriert werden
WriteTrailerTest
Setzt einen kompletten Trailer zusammen und prüft dessen Bytestruktur

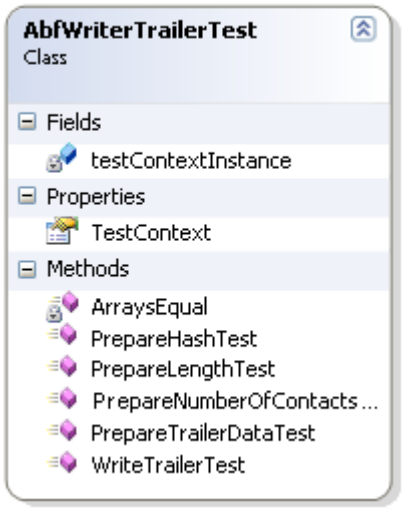


Tabelle 19: AbfWriterTrailerTest

2.14. FileTypeTest

CalculateFileTypeTest
Prüft das korrekte Auslesen der 16 Byte, die den FileType bestimmen.

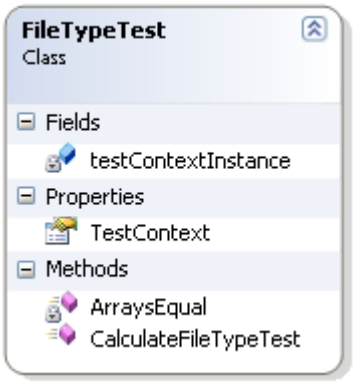


Tabelle 20: FileTypeTest

2.15. LZ77Test

SetTokenIndexTest
Prüft, ob der TokenIndex korrekt aktualisiert wird.

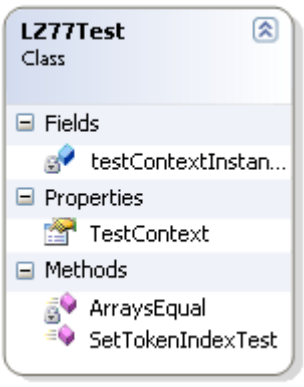


Tabelle 21: Lz77Test

3. Tabellenverzeichnis

Tabelle 1: Testfiles Validierungstes Sw9 5
Tabelle 2: Auswertung Validierungstest Sw9 5
Tabelle 3: Testfiles Validierungstest Sw11 5
Tabelle 4: Auswertung Validierungstest Sw11 6
Tabelle 5: Testfiles Validierungstest Sw12 6
Tabelle 6: Auswertung Validierungstest Sw12 6
Tabelle 7: Vorgehensbsp. Komprimierungstest 7
Tabelle 8: AbfAttributeTest 8
Tabelle 9: AbfContactAttributCalculatorTest 8
Tabelle 10: AbfContactTest 9
Tabelle 11: AbfFullHeaderTest 9
Tabelle 12: AbfFullTrailerTest 10
Tabelle 13: AbfNormalizationRulesTest 10
Tabelle 14: AbfWriterCompressedHeaderTest 11
Tabelle 15: AbfWriterDecompressedHeaderTest 12
Tabelle 16: DecompressDataTest 12
Tabelle 17: CrcTest 12
Tabelle 18: AbfWriterNormalizationRulesTest 12
Tabelle 19: AbfWriterTrailerTest 13
Tabelle 20: FileTypeTest 13
Tabelle 21: Lz77Test 13

Office Communication Server Adressbucherweiterung

AbFile-Beschreibung

Dokumentstruktur basiert auf „Address Book File Structure“ von Microsoft

0. Dokumentinformationen

0.1. Änderungsgeschichte

Datum	*Version*	*Änderung*	*Autor*	*Qualitätssicherung*
8.8.2009	1.0	erstellt	Dietrich	
8.8.2009	1.1	Aufbau eines AbFiles hinzugefügt	Dietrich	
19.11.2009	1.2	Dekomprimierung hinzugefügt	Dietrich	
22.11.2009	1.3	Komprimierung hinzugefügt	Dietrich	Müller
17.12.2009	2.0	Finale Version	Dietrich	

0.2. Inhalt

0. Dokumentinformationen 2
0.1. Änderungsgeschichte 2
0.2. Inhalt 3
1. Einführung 4
1.1. Zweck 4
2. Aufbau eines AddressBook-Files 4
2.1. Compressed AbFile 4
2.1.1. Compressed Header 4
2.1.1.1. Beschreibung der Felder 4
2.1.1.2. Besonderheit: 5
2.2. Decompressed Full-File 5
2.3. Decompressed Delta-File 6
2.4. Beschreibung der Header 6
2.4.1. Decompressed Header 6
2.4.2. Normalization Rules 7
2.4.3. Attribute 7
2.4.4. Contact 7
2.4.4.1. Besonderheiten 8
2.4.4.1.1. Sentinel Contact 8
2.4.4.1.2. Deleted Contact 8
2.4.5. ContactAttributes 8
2.4.5.1. ContactAttributes Besonderheiten 8
2.4.6. Trailer 8
2.4.7. TrailerLength 9
3. Komprimierung / Dekomprimierung 9
3.1. Grundlagen 9
3.1.1. Gliederung eines Compressed Blocks 9
3.1.2. Gliederung eines Tokens 9
3.1.2.1. Offset 10
3.1.2.2. Length 10
3.2. Der Dekomprimierungsalgorithmus 10
3.2.1. Dekomprimierung eines Ab-Files 10
3.3. Komprimierung eines Ab-Files 10
3.3.1.1. LZ77 10
3.3.1.1.1. Beispiel 11
3.3.1.2. Verpacken von Token 11
3.3.1.3. Implementierung von Token/ Tokenindexes 12
3.3.1.4. Ablauf der Komprimierung 13

1. Einführung

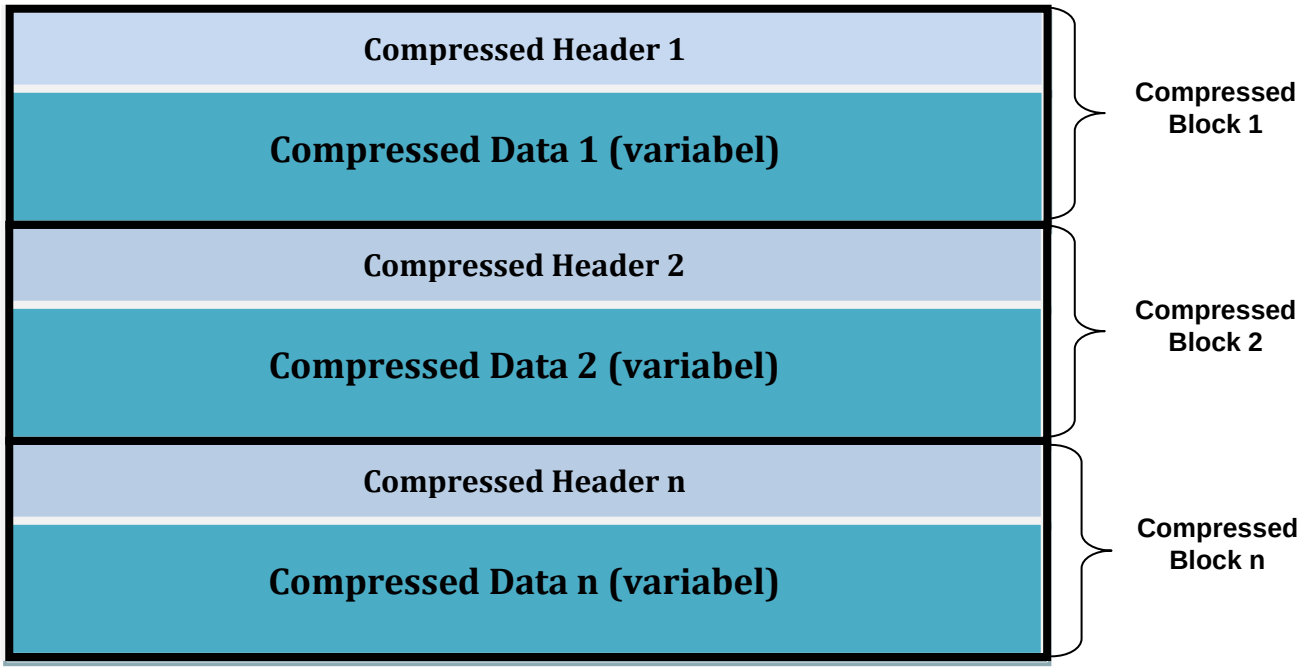
1.1. Zweck

Die Grundlagen für die nachfolgenden Erkenntnisse basieren auf der Dokumentation von Microsoft[1], siehe Anhang B . Hier wird nicht auf den gesamten Inhalt eingegangen, sondern nur auf Punkte, welche in der von Microsoft zur Verfügung gestellten Dokumentation umständlich oder ungenau erklärt werden oder für das Verständnis der OCSAddresExtension-Arbeit besonders wichtig sind.

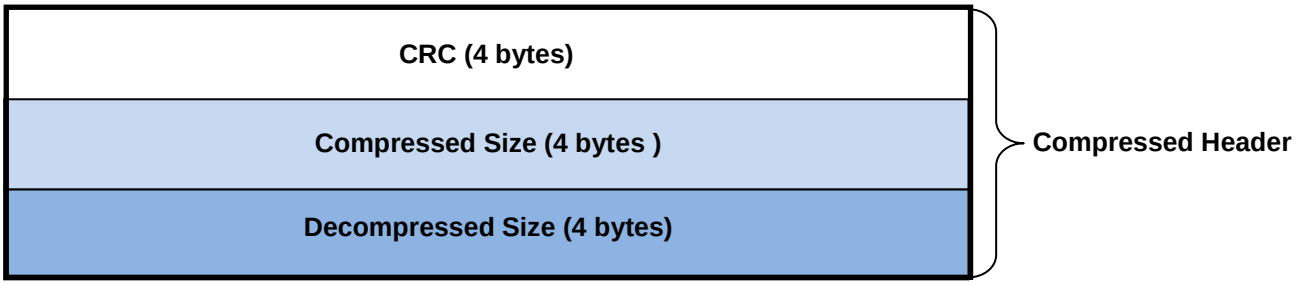
2. Aufbau eines AddressBook-Files

2.1. Compressed AbFile

En komprimiertes .lsabs File, welches vom OC Server generiert wird, ist wie folgt aufgebaut. Es besteht aus 1-n Compressed Blocks, wobei jeder Block aus einem Compressed Header und den dazugehörigen Compressed Data besteht.



2.1.1. Compressed Header



2.1.1.1. Beschreibung der Felder

- **CRC (4 Bytes)**
  Die ersten 4 bytes representieren einen 32 Bit unsigned Integer, welcher den CRC -Wert der original dekomprimierten Daten darstellt.

[1] Microsoft, Address Book File Structure, 2009

- **Compressed Size (4 Bytes)**
  Ein 32 Bit Feld, das die Länge der komprimierten Daten beinhaltet, die dem Compressed Header folgen. Das Feld muss kleiner oder gleich 64 KB (0x10000) sein.

- **Decompressed Size (4 Bytes)**
  Ein 32 Bit Feld, das die Länge der Daten nach der Dekomprimierung mittels des im Kapitel 3.2 beschriebenen Algorithmus beinhaltet.

2.1.1.2. Besonderheit:

Falls die Felder Compressed Size und Decompressed Size gleich gross sind, enthält der Block keine komprimierten Daten.

2.2. Decompressed Full-File

Wurden die komprimierten Daten dekomprimiert, muss mit diesen weitergearbeitet werden, um die Kontaktinformationen extrahieren zu können. Nachfolgend ein Überblick über die dekomprimierte Datenstruktur, eines Full -Files.
Das Full-File ist ein eigenständiges Kontaktfile, es kann vom OCS an angehängte Geräte geschickt werden, und diesen ist es möglich, alle Kontaktdaten zu extrahieren.

Decompressed Header	
File Type (16 Bytes)	Creation Date (2 Bytes)
Number of Attributes (2 Bytes)	Maximum Attribut Id (2 Bytes)
Use NormalizationRules (2 Bytes)	Source Stream (128 Bytes)
Normalization Rules	
Length (4 Bytes)	Rules (variable)
1-n Attribute	
Length (2 Bytes)	Id (2 Bytes)
Flags (4 Bytes)	Name (variable)
1-m Contact	
Length (2 Bytes)	Id (16 Bytes)
Number of Attributes (2 Bytes)	Number of deleted Attributes (optional)
Data (variabel)	
1 –n Contact Attributes	
Id (1 or 2 Bytes)	Length (2 Bytes, optional)
Data (variable)	
Trailer	
Hash (2 Bytes)	Number of Contacts (4 Bytes)
Trailer Length	
Trailer length (4 Bytes)	

2.3. Decompressed Delta-File

Das Delta-File wird aufgrund zweier Full-Files erstellt. Das Delta vergleicht die beiden Full-Files und übernimmt nur die Änderungen, die zwischen den zwei Full-Files zu finden sind. Meistens sind dies gelöschte oder neu hinzugefügte Kontakte.
Damit ein Delta-File von einem System ausgewertet werden kann, müssen auf diesem auch die beiden Full-Files vorhanden sein, aufgrund deren Basisinformationen die Delta-Daten generiert wurden.

Decompressed Header	
File Type (16 Bytes)	Creation Date (2 Bytes)
Base Creation Date (2 Bytes)	Number of Attributes (2 Bytes)
Maximum Attribut Id (2 Bytes)	Use NormalizationRules (2 Bytes)
Source Stream (128 Bytes)	
Normalization Rules	
Length (4 Bytes)	Rules (variable)
1-n Attribute	
Length (2 Bytes)	Id (2 Bytes)
Flags (4 Bytes)	Name (variable)
1-m Contact	
Length (2 Bytes)	Id (16 Bytes)
Number of Attributes (2 Bytes)	Number of deleted Attributes (optional)
Data (variabel)	
1 –n Contact Attributes	
Id (1 or 2 Bytes)	Length (2 Bytes, optional)
Data (variable)	
Trailer	
Hash (2 Bytes)	Base File Hash (2 Bytes)
Number of Contacts (4 Bytes)	Number of deleted Contacts (4 Bytes)
Trailer Length	
Trailer length (4 Bytes)	

2.4. Beschreibung der Header

2.4.1. Decompressed Header

Der Decompressed Header ist das Erste, was ausgelesen werden kann und auch muss, um ein AddressBookFile sauber entpacken zu können. Der Full-Header ist immer 152 Byte gross und der Delta-Header immer 154 Byte. Der Unterschied kommt durch das nur im Delta –Header vorhandene Feld „BaseCreationDate“ zustande.

- **File Type (16 Byte)**
  Mit Hilfe dieses Feldes kann festgestellt werden, ob es sich beim File um ein Full-File oder Delta-File handelt.

- **Creation Data (2 Byte)**
  Das Erstellungsdatum ist gespeichert in Anzahl Tagen ab dem 01.01.2001. Oder bei Delta-Files der Name des zweiten verwendeten Full-Files.

- **Base Creation Data (2 Byte) (Nur Delta-File)**
  Das Erstellungsdatum des Basis Full-Files.

- **Number of Attributes (2 Byte)**
  Die Anzahl Attribute welche verwendet werden, um einen Kontakt zu gliedern. Wobei nicht alle Attribute in jedem Kontakt verwendet werden müssen.

- **Maximum Attribut Id (2 Byte)**
  Die höchste Id, welche einem Attribut vergeben wurde.

- **UseNormalizationRules (2 Byte)**
  Gibt Auskunft darüber, ob der Server die Normalization Rules (siehe 2.4.2Normalization Rules) verwendet oder nicht.

- **Source Stream (128 Byte)**
  128 Bytes die alle 0 sein sollten, Verwendungszweck ist nicht von Belang für diese Arbeit

2.4.2. Normalization Rules

Die Rules folgen dem Decompressed Header und sind nicht von Interesse für die Gewinnung der Kontaktinformationen. Sie müssen allerdings ausgelesen und abgespeichert werden, um später wieder ein voll funktionsfähiges Full-File erstellen zu können.

- **Length(4 Byte)**
  Sagt aus, wie viele Bytes das Rulesfeld enthält

- **Rules (variabel)**
  Die eigentlichen Rulesdaten. Werden in dieser Arbeit lediglich in einem Byte-Array gespeichert.

2.4.3. Attribute

Hier sind alle möglichen Attribute serialisiert, welche in Kontakten vorkommen können. Attribute können sein: Telefonnummer, Mail, Surname usw.
Da alle Attribute hintereinander gespeichert sind, muss hier das Decompressed-Header-Feld „Number of Attributes“ hinzugezogen werden, um definieren zu können, wie viele Attribute ausgelesen werden müssen.

- **Length(2 Byte)**
  Spezifiziert die verbleibende Anzahl Bytes für das auszulesende Attribut. Das Längenfeld selber wird dabei nicht mitgerechnet.

- **Id (2 Byte)**
  Id-Wert, um das Attribut eindeutig identifizieren zu können.

- **Flags (4 Byte)**
  Identifiziert den Value-Type, welcher dann innerhalb eines Kontaktes diesem Attribut zugewiesen werden soll.
  Die drei möglichen Typen sind: String, ProxyAddressString und Binary.

- **Name (variabel)**
  Null terminierter UTF8-String, der den Name des Attributes darstellt.

2.4.4. Contact

Alle Kontakte sind hier hintereinander serialisiert. Die Kontaktreihe wird durch einen Sentinel Kontakt beendet.

- **Id (16 Byte)**
  16 Byte Wert, der den Kontakt eindeutig identifiziert.

- **Length (2 Byte)**
  Spezifiziert die verbleibende Anzahl Bytes für den Kontakt. Das Längenfeld selber wird dabei nicht mitgerechnet.

- **Number of Attributes (2 Byte)**
  Anzahl Attribute, die in diesem Kontakt Verwendung finden.

- **Number of deleted Attributes (2 Byte, optional)**
  Der Verwendungszweck bzw. die Notwendigkeit dieses Feldes ist nicht klar ersichtlich[2]. Durch diverse Tests mit verschiedenen Files hat sich allerdings ergeben, dass bei keinem der Files dieses Feld vorhanden war und somit immer ignoriert werden konnte.

Data (variabel)
- Siehe *2.4.5*

2.4.4.1. Besonderheiten

2.4.4.1.1. Sentinel Contact

Der SentinelContact ist ein Kontakt, welcher signalisiert, dass nach ihm keine weiteren Kontakte mehr folgen. Die Erfüllung von drei Eigenschaften charakterisiert den Sentinel:

1. Number of Attributes = 0
2. Length = 0x12
3. Id = 0

2.4.4.1.2. Deleted Contact

Der Kontakt wurde gelöscht, falls für den Kontakt gilt:

1. Number of Attributes = 0;

2.4.5. ContactAttributes

ContactAttributes sind ausgewertete Werte, innerhalb eines Kontakts, welche mittels einer Attribut-Id einem Attribut zugeordnet werden. Die Anzahl dieser Kontaktattribute kann aus dem Contactheaderfeld „NumberOfAttributes“ ausgelesen werden.

- **Id (1 or 2 Byte)**
  Id, welche mit einem Attribute übereinstimmt. Falls das Feld „MaximumAttributeId“ des DecompressedHeader’s kleiner 256 ist, ist die Id 1 Byte, sonst 2 Byte.

- **Length (2 Byte)**
  Spezifiziert die verbleibende Anzahl Bytes für dieses ContactAttribute. Das Längenfeld selber wird dabei nicht mitgerechnet

- **Data (variabel)**
  Die effektiven Daten

2.4.5.1. ContactAttributes Besonderheiten

Die Daten können vom Type String(utf8 verschlüsselt) oder Byte(direkt übernehmen aus dem Speicher) sein. Um diese zwei Möglichkeiten unterscheiden zu können, muss die Id des ContactAttributes ausgewertet werden. Diese stimmt mit der Id eines Attributes überein, dieses hat wiederum ein Flag, welches nun ausgewertet werden muss, um herauszufinden, vom welchem Typ die Daten sind.

2.4.6. Trailer

Verfügt über ein spezielles System zur Sicherstellung der Kompatibilität mit älteren Protokollen. Für die Arbeit sind folgende Felder relevant.

- **Number of Contacts (4 Bytes)**
  Anzahl aller Kontakte in diesem File. Ohne Sentinel.

- **Hash (2 Bytes)**
  Hashwert, generiert über dem Inhalt des Files. Der benutze Algorithmus kann irgendeiner sein, muss jedoch immer derselbe für alle Files sein. Ein möglicher Hashalgorithmus ist unter 6[3] beschrieben

[2] Microsoft, Address Book File Structure, 2009
[3] Ebd

- **BaseFileHash (2 Bytes)**
  Beinhaltet den Hash-Wert des Base-File, das verwendet wurde, um das Delta-File zu generieren.

2.4.7. TrailerLength

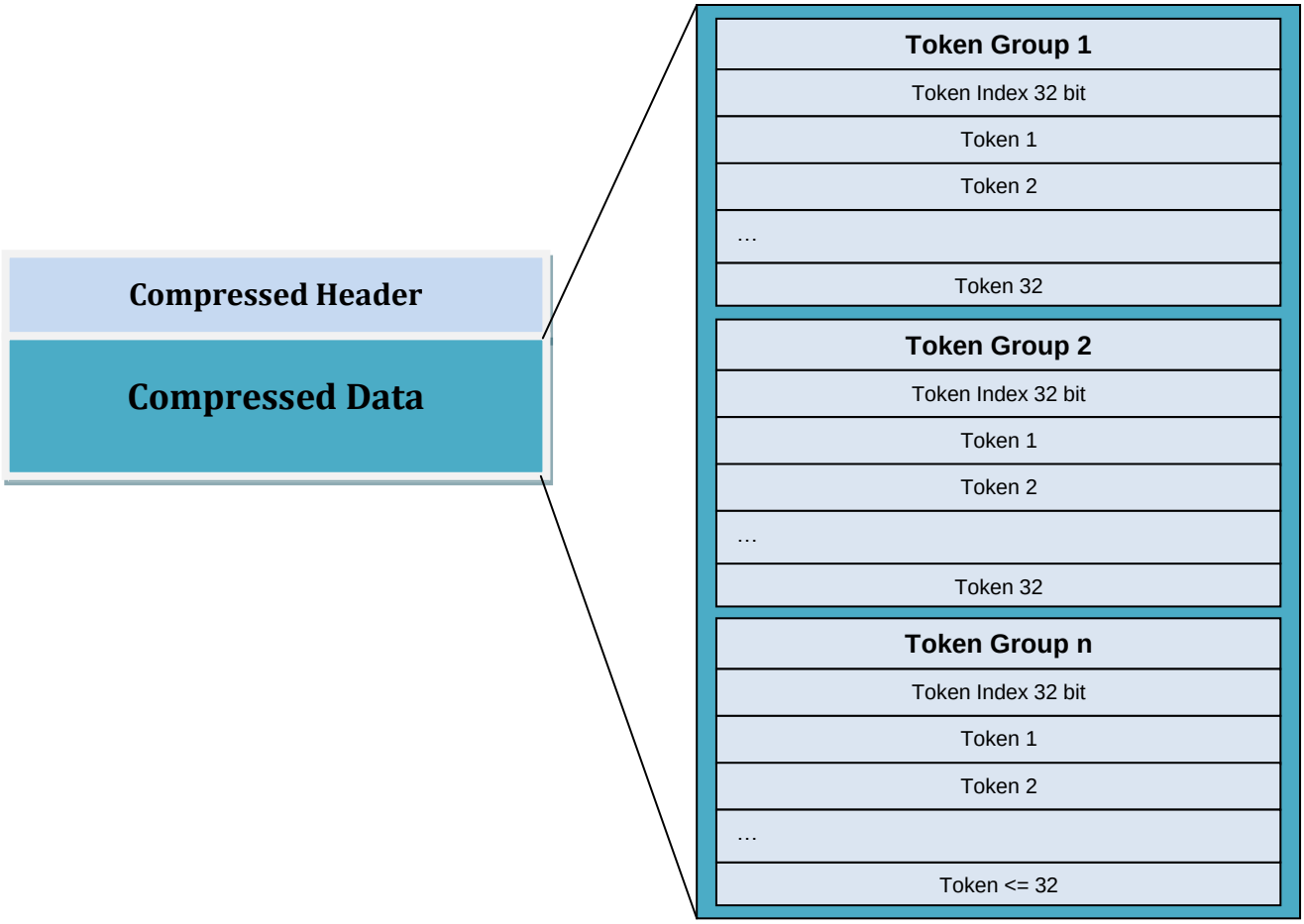
Die letzten vier Bytes des gesamten Files. Darin wird die Länge des gesamten Trailers gespeichert. Die Längenangabe in Bytes beinhaltet nicht die vier Bytes des TrailerLength-Feldes.

3. Komprimierung/ Dekomprimierung

3.1. Grundlagen

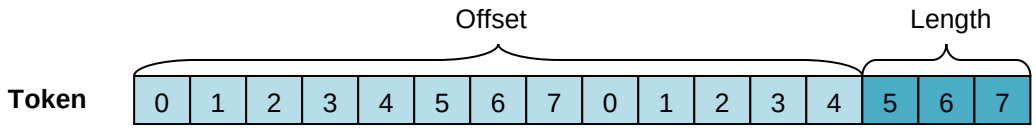
3.1.1. Gliederung eines Compressed Blocks

Um das Prinzip erklären zu können, beginnen wir auf der obersten Ebene und analysieren den Aufbau eines Compressed-Data Teils. Auf dem obersten Level besteht jeder Compressed Data-Teil aus einem oder mehreren Token Groups. Jede dieser Gruppen, ausser der letzten beinhalten genau 32 Token. Die einzelnen Token Groups beginnen mit einem 32 Bit unsigned Integer, welcher als Index für die Gruppe fungiert. Bit 0 repräsentiert Token 1, Bit 1 repräsentiert Token 2 usw. Die Daten der Token sind gleich nach dem Index aufgeführt.
Ist ein Bit im Token Index auf 0 gesetzt, bedeutet dies für den Token, dass dieser direkt in den Outputbuffer übernommen werden kann (Literal Byte). Steht eine 1 im Token Index, ist der Token eine Referenz auf eine Sequenz von 3 oder mehr Bytes, welche bereits in den Outputbuffer geschrieben wurde.



3.1.2. Gliederung eines Tokens

Das Verfahren benötigt Tokengrössen von mindestens 2 Bytes bis zu einem Maximum von 6 Bytes, abhängig von der Länge der zu kopierenden Daten.
Die ersten zwei Bytes eines Tokens werden als unsigned Integer behandelt. Wobei die 13 hochwertigen Bits den Offset repräsentieren und die niederwertigen 3 Bit die Länge.

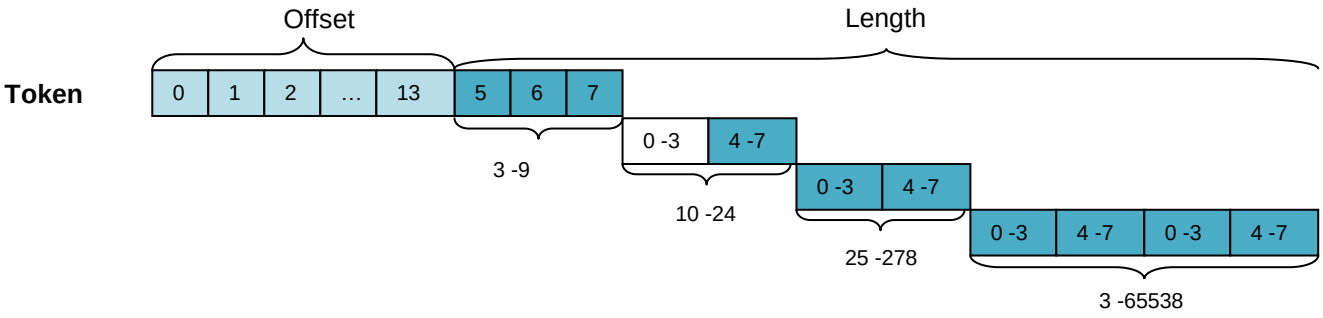


3.1.2.1. Offset

Ist der Token eine Referenz auf eine Bytefolge, welche es nun zu kopieren gilt, muss die Anfangsposition dieser zu kopierenden Sequenz festgestellt werden können. Dazu benötigt man den Offset. Da dieses System jedoch einen Backwardoffset erwartet, muss der errechnete Offset von der Indexposition des Inputbuffers abgezogen werden.

3.1.2.2. Length

Wurde die Startposition einer zu kopierenden Sequenz gefunden, wird noch die Länge der Sequenz benötigt. Diese zu spezifizieren ist nun die Aufgabe des Längenfeldes.
Die Länge wird mit -3 initialisiert. Dies liegt dem Verpackungssystem zugrunde. Da ein Token, das eine Referenz speichert mindestens, 2 Bytes braucht um eine Sequenz wiederspiegeln zu können, machen Sequenz-Längen die kleiner als 3 sind keinen Sinn, da keine Komprimierung erreicht würde.
Konkret bedeutet dies, mit den drei Bits des Längenfeldes, können Längen von 3 bis 10 codiert werden (nicht nur 0 bis 7).
Da die Länge einer Sequenz häufig grösser als 10 ist, kann dieses Feld „vergrössert“ werden. Wenn alle 3 Bit gesetzt sind (0x7), wird das nächste Byte hinzugezogen. Allerdings werden nur die vier niederwertigen Bit als Länge ausgewertet. Somit können Längen von 10 bis 25 dargestellt werden. (Initial ist die Länge für diesen Nibbel -(3+7) = -10).
Falls die Länge grösser als die nun erreichten 25 sein sollte zu erkennen daran, dass alle 4 Bit gesetzt sind (0xF) - wird das nächste Byte hinzugezogen. Mit dieser 4 Byte Adressierung ist es nun möglich Längen von 25 bis 279 darzustellen. (Initial ist die Länge für dieses Byte -(3 + 7 + 15) = -25).
Entspricht die Wertigkeit von Byte vier allerdings 0xFF (alle Bits gesetzt), steht die Länge in den Bytes fünf und sechs, welche nun hinzugezogen werden. Der Initialwert der Länge für diese zwei Bytes beträgt -3.



3.2. Der Dekomprimierungsalgorithmus

3.2.1. Dekomprimierung eines Ab-Files

Die Komprimierung der Daten eines Kontaktfiles wird mittels eines LZ77 Algorithmus erzielt. Um die Komprimierung optimal speichern zu können hat Microsoft die oben beschriebene Tokenstruktur entwickelt. Diese Struktur ist dann auch die Grundlage um Kontakt-Files sauber dekomprimieren zu können. Microsoft stellt hierfür zwei Methoden zur Verfügung. Beschrieben sind diese unter 5.5.1 und 5.5.2[4]. Sie können wie dort beschrieben übernommen werden, es eröffneten sich keinerlei Probleme bei der Implementierung.

3.3. Komprimierung eines Ab-Files

Der Algorithmus stammt aus der Familie LZ77. Die Aufgabe des Algorithmus ist es, Mehrfachvorkommen von ByteSequenzen innerhalb des InputArrays zu erkennen und diese mit Hilfe von Offset und Länge festzuhalten. Grundsätzlich kann hier irgendein Algorithmus verwendet werden, sofern er der LZ77 Familie angehört. Er muss jedoch in der unter 3.2.1 beschriebenen Art gespeichert werden, damit es später möglich ist, die Komprimierte Struktur mit dem Dekomprimierungsalgorithmus auspacken zu können.

3.3.1.1. Beschreibung LZ77

LZ77 ist ein Komprimierungsverfahren, das von Abraham Lempel und Jacob Ziv entwickelt wurde[5].
Um die Erklärung einfacher gestalten zu können, verwenden wir vier Buffer.

- Lookahead-Buffer (definiert den Bereich um den vorausgeschaut werden soll)
- Input-Buffer (beinhaltet den Originaltext, der komprimiert werden soll)

[4] Microsoft, Address Book File Structure, 2009
[5] http://de.wikipedia.org/wiki/LZ77

- Readdata-Buffer (beinhaltet alle gelesenen Bytes, kopiert aus dem Input-Buffer)
- Output-Buffer (beinhaltet die komprimierten Daten)

Mit dem Lookahead-Buffer wird nun eine Sequenz aus dem Input-Buffer geladen. Normalerweise wird mit der Länge eins gestartet. Diese Sequenz wird nun mit den Daten im Readdata-Buffer verglichen. Ist die Sequenz noch inexistent, ist es ein sogenanntes Literal-Byte und wird direkt in den Output-Buffer und den Readdata-Buffer kopiert. Existiert die Sequenz bereits, kann der Lookahead-Buffer um eins vergrössert werden und einer erneuten Prüfung unterzogen werden. Dies wird solange gemacht bis die grösste vorhandene Sequenz gefunden werden konnte. Diese wird dann als Tuppel in der Form (Offset, Length) im Ouput-Buffer gespeichert. Im Readdata-Buffer wird jedoch nicht die komprimierte, sondern die orginale Squenz gespeichert.
Der Offset bestimmt den Starpunkt der zu komprimierenden Sequenz, während die Länge die Anzahl Bytes der Sequenz spezifiziert.

3.3.1.1.1. Beispiel

Der Einfachheit halber wird in diesem Bsp.mit einem String „ANANAS“ gearbeitet.
Der Offset ist ein Backwardoffset.

0	1	2	3	4	5	
A	N	A	N	A	S	
A						A wird mit dem Inhalt des ReadData-Buffers verglichen. Da dieser noch leer ist, wird A in den Ouput-Buffer kopiert
	N					N wird mit dem Inhalt des ReadData-Buffers verglichen. Der Vergleich mit A liefert keinen Match, somit ist N ebenfalls ein Literal Byte
		A				A wird mit dem Inhalt des ReadData-Buffers verglichen und liefert einen Match zurück. Somit kann der Loockahead-Buffer um eins vergrössert werden
		A	N			AN wird mit dem Inhalt des ReadData-Buffers verglich und liefert einen Match zurück. Der Lookahead-Buffer wird ein weiteres mal vergössert
		A	N	A		ANA wird mit dem Inhalt des ReadData-Buffers verglichen und liefert einen Match zurück. Der Loockahead-Buffer wird um eins vergrössert.
		A	N	A	S	ANAS liefert nach dem Vergleich mit dem Readdata-Buffer keinen Match. Somit ist die längste Sequenz ANA und wird als Referenz gespeichert (2,3).
					S	S wird mit dem Inhalt des ReadData-Buffers verglichen und liefert keinen Match, somit ist S Literal.
						Ende, ReadDataBuffer und InputBuffer sind gleich gross, bzw. Lookahead ist nicht mehr möglich.

Tabelle 1: LZ77 Auswertung

Buffer	Schritt 1	Schritt 2	Schritt 3	Schritt 4	Schritt 5	Schritt 6	Schritt 7	Schritt 8
ReadData-Buffer	-	A	AN	ANA	ANAN	ANANA	ANANA	ANANAS
Input-Buffer	ANANAS	ANANAS	ANANAS	ANANAS	ANANAS	ANANAS	ANANAS	ANANAS
Output-Buffer	-	A	AN	AN	AN	AN	AN(2, 3)	AN(2, 3)S
Lookahead-Buffer	A	N	A	AN	ANA	ANAS	S	-

Tabelle 2: LZ77 Buffer-Inhalte

.

3.3.1.2. Verpacken von Token

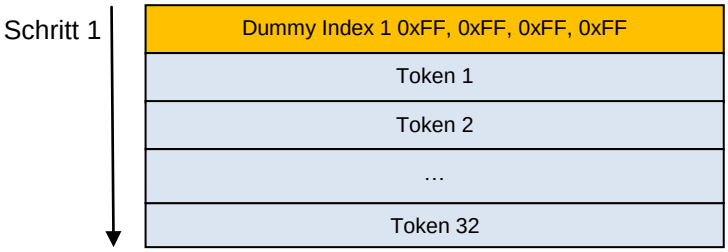
Die Tokens müssen mit einer von Microsoft unter 5.4[6] spezifizierten Methodik verpackt werden. Die Verwendung dieser ist nicht weiter anspruchsvoll. Sie erwartet den Backwardoffset, die Länge, den Outputbuffer und den Index des Outputbuffers, sprich die Momentane Position darin.

[6] Microsoft, Address Book File Structure, 2009

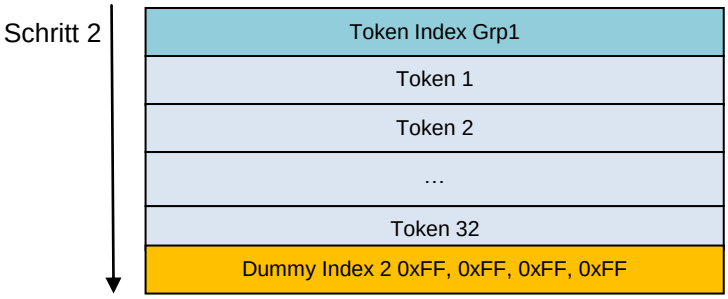
3.3.1.3. Implementierung von Token / Tokenindizes

Was bis anhin noch nicht berücksichtigt wurde, ist die Generierung der Tokens und der TokenIndizes. Die Eigenschaften sind unter 3.1.1 erläutert, sie müssen lediglich noch implementiert werden. In dieser Arbeit wurde der folgende Ansatz verfolgt.

Da der Tokenindex immer erst bekannt ist, nachdem die 32 Tokens verpackt wurden, wird zu Beginn jeder Tokengroupe ein Dummy-Tokenindex erstellt. Dieser fungiert als Platzhalter, bis der berechnete Tokenindex bekannt ist und den Dummy überschreibt.



Alternativ wäre es auch möglich, alle Tokenindexes mit der dazugehörigen Indexposition in einer Liste zwischen zu speichern und nach vollendeter Komprimierung die Tokenindexes einzufügen.



3.3.1.4. Ablauf der Komprimierung

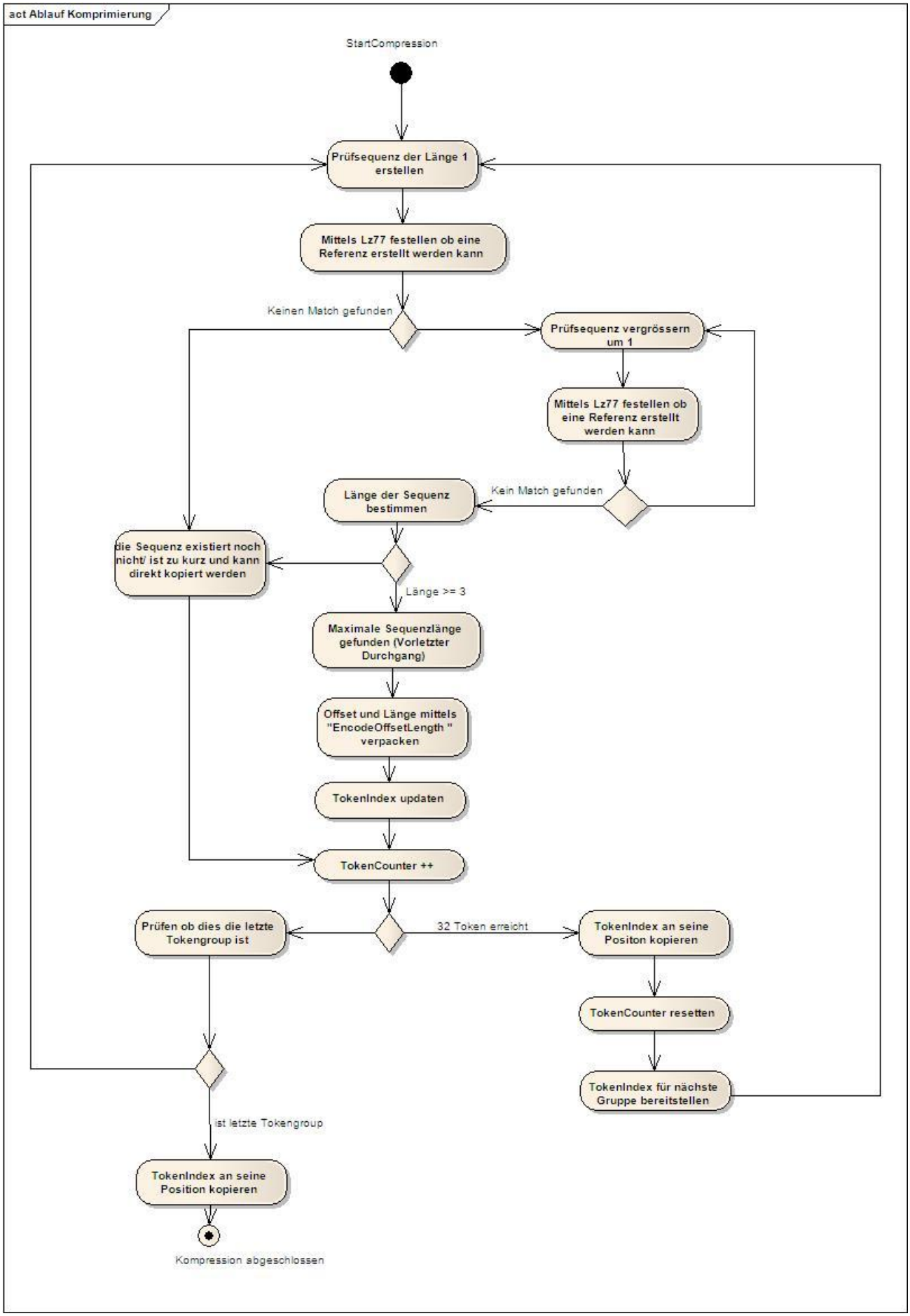


Abbildung 1: Ablauf der Komprimierung

4. Tabellenverzeichnis

Tabelle 1: LZ77 Auswertung 11
Tabelle 2: LZ77 Buffer-Inhalte 11

5. Abildungsverzeichnis

Abbildung 1: Ablauf der Komprimierung 14