HOCHSCHULE
MITTWEIDA
University of Applied Sciences

BACHELORARBEIT

Frau
Linda Becker

Entwicklung einer X-Tension für die Untersuchung von ESE Datenbanken in X-Ways

Mittweida, September 2023

Fakultät **Angewandte Computer- und Biowissenschaften**

BACHELORARBEIT

Entwicklung einer X-Tension für die Untersuchung von ESE Datenbanken in X-Ways

Autorin:
Linda Becker

Studiengang:
Allgemeine und digitale Forensik

Seminargruppe:
FO20w1-B

Erstprüfer:
Prof. Ronny Bodach

Zweitprüfer:
Stefan Schildbach, M.Sc.

Einreichung:
Mittweida, 11.09.2023

Verteidigung/Bewertung:
Mittweida, 2023

Faculty of **Applied Computer Sciences and Biosciences**

BACHELOR THESIS

Development of an X-Tension for the investigation of ESE Databases in X-Ways

Author:
Linda Becker

Course of Study:
General and digital forensics

Seminar Group:
FO20w1-B

First Examiner:
Prof. Ronny Bodach

Second Examiner:
Stefan Schildbach, M.Sc.

Submission:
Mittweida, 11.09.2023

Defense/Evaluation:
Mittweida, 2023

Bibliografische Beschreibung:

Becker, Linda:
Entwicklung einer X-Tension für die Untersuchung von ESE Datenbanken in X-Ways. – 2023. – 47 S.
Mittweida, Hochschule Mittweida – University of Applied Sciences, Fakultät Angewandte Computer- und Biowissenschaften, Bachelorarbeit, 2023.

Referat:

Die vorliegende Arbeit beschäftigt sich mit der Umsetzung einer Viewer X-Tension für die Darstellung von ESE Datenbanken in X-Ways Forensics. Dazu wird vor allem der konkrete Aufbau und die Struktur einer ESE Datenbank analysiert und vorgestellt. Weiterhin werden die technischen Grundlagen der Software X-Ways Forensics und der X-Tensions-API zur Erstellung einer Erweiterung gelegt. Zudem werden die Schritte der Umsetzung aufgezeigt und die konkrete Programmimplementierung an Code-Beispielen erläutert.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

API . . . . . Application Programming Interface

ASCII . . . . . American Standard Code for Information Interchange

BOM . . . . . Byte Order Mark

DB . . . . . Datenbank

DLL . . . . . Dynamic Link Library

ECC . . . . . Error Correcting Code

ESE . . . . . Extensible Storage Engine

FDP . . . . . Father Data Page

GUID . . . . . Globally Unique Identifier

HTML . . . . . Hypertext Markup Language

ISAM . . . . . Indexed Sequential Access Method

JET . . . . . Joint Engine Technology

KB . . . . . Kilobyte

LV . . . . . Long Value

MB . . . . . Megabyte

SP . . . . . Service Pack

TB . . . . . Terabyte

USB . . . . . Universal Serial Bus

WAL . . . . . Write-Ahead-Logging

WWW . . . . . World Wide Web

XWF . . . . . X-Ways Forensics

XWI . . . . . X-Ways Investigator

1 Einleitung

„Die zunehmende Digitalisierung lässt sich auch im Anstieg der weltweiten Datenmengen ablesen. [...] Würde man [die im Jahr 2020 erreichte] Datenmenge auf DVDs speichern, wäre der Stapel 2,6 Millionen Kilometer hoch.“ [1]
Mit der fortlaufenden Digitalisierung ergeben sich für digitale Forensiker:innen sowohl Chancen, als auch Herausforderungen. Einerseits werden mehr Daten gespeichert, welche potenziell forensisch relevante Informationen enthalten können. Andererseits ist die Suche nach wertvollen Informationen weitaus umfangreicher. Eine wichtige Voraussetzung ist deshalb das Wissen über die Speicherorte forensisch interessanter Daten. Beachtung sollte in diesem Zusammenhang auch der Extensible Storage Engine (ESE) Datenbank (DB) gegeben werden, da eine steigende Anzahl an Anwendungen diese Datenbank bereits zur Speicherung ihrer Daten verwendet [2]. Beispielsweise werden Informationen wie der Suchverlauf oder der Inhalt eines Postfachs in der Datenbank-Datei gespeichert.
Die Auswertung digitaler Daten erfolgt meistens mit einer forensischen Software. Eine bekannte Anwendung ist X-Ways Forensics (XWF). Diese bietet die Möglichkeit etliche Dateiformate zu öffnen und zu untersuchen. Eine Darstellung des Inhalts einer ESEDB-Datei wird hingegen aktuell nicht unterstützt. Darin begründet sich die Motivation für diese Arbeit, eine entsprechende Erweiterung für die Software zu entwickeln.

1.1 Zielstellung

Das Ziel dieses Bachelorprojektes ist die Implementierung und erfolgreiche Einbindung einer Erweiterung für die forensische Software XWF, welche die Untersuchung von ESEDB-Dateien ermöglicht. Diese Arbeit soll dabei die Grundlagen für die Umsetzung einer X-Tension legen, welche die Möglichkeit bietet, viele weitere forensisch interessante Probleme zu lösen. Weiterhin soll die Struktur einer ESE Datenbank analysiert und aufgezeigt werden, sowie anhand von Beispielen, mögliche in der Datenbank gespeicherte Daten vorgestellt werden. Schließlich soll die praktische Programmumsetzung aufgezeigt werden.

1.2 Aufbau der Arbeit

Im nachfolgenden Kapitel werden die erforderlichen Grundlagen für die Erstellung und Einbindung einer X-Tension in XWF gelegt. Dafür wird einerseits die Software XWF kurz vorgestellt und weiterhin die X-Tensions Application Programming Interface (API) genauer beleuchtet. Außerdem wird in diesem Kapitel die Struktur einer ESEDB analysiert und erläutert. Dies ist entscheidend, um die Datei später auslesen zu können. Zusätzlich wird auf einige Anwendungen, welche zur Datenspeicherung eine ESE Datenbank verwenden, eingegangen.

Kapitel 3 widmet sich daraufhin der praktischen Umsetzung der Erstellung der X-Tension. Dafür wird auf die Schritte zur Erstellung einer Dynamic Link Library (DLL) eingegangen und anschließend die konkrete Programmumsetzung an Code-Beispielen erläutert.

In Kapitel 4 wird das Ergebnis der Implementierung der X-Tension präsentiert und kritisch diskutiert. Abschließend wird die Arbeit zusammengefasst, sowie ein Ausblick auf mögliche Verbesserungen und Erweiterungen der X-Tension gegeben.

2 Grundlagen

2.1 X-Ways Forensics

X-Ways Forensics von der X-Ways Software Technology AG ist ein Programm für die forensische Untersuchung von Datenträgern. Die von Stefan Fleischmann entwickelte Software ist 2004 aus dem Hex-Editor WinHex entstanden. Sie ist portabel und kann zum Beispiel bei der Live-Analyse direkt von einem Universal Serial Bus (USB)-Stick aus gestartet werden. Die Software bietet viele Funktionen und Möglichkeiten digitale Daten zu untersuchen. Daher soll nach Fleischmann auch der Name des Programms stammen, denn es gäbe „‚X' ways", also „viele Wege", um Daten in XWF zu untersuchen. Zu den Features der Software gehören unter anderem die physikalische und logische Datenträgersicherung, das Einlesen von Images im EnCase-Format[1], die Datenrettung mittels Carven, die Berechnung von Hash-Werten für Dateien und Datenträger, eine Dateivorschau und integrierte Viewer für mehr als 270 Dateitypen, sowie unzählige Filtermöglichkeiten. Weitere Vorteile des Tools sind eine gute Performance und die Möglichkeit, die vorhandenen Funktionen der Software mittels der X-Tensions API zu erweitern. [4, 5]

Neben XWF gibt es den X-Ways Investigator (XWI), eine vereinfachte Version von XWF, die sich vor allem an kriminalpolizeiliche Ermittler richtet, die keine speziellen IT-Kenntnisse haben. Das Programm hat einen geringeren Funktionsumfang, sowie eine vereinfachte Benutzeroberfläche mit weniger technischen Features. Außerdem wird die Arbeit bei der Verwendung des XWI in zwei Schritte aufgeteilt: Im ersten Schritt erfolgt die Datensicherung und Datenaufbereitung mit XWF. Anschließend kann die Analyse des Sachbearbeiters mit XWI durchgeführt werden. [6]

2.1.1 Viewer

Ein Viewer ist eine Bibliothek, welche das Einsehen eines speziellen Dateityps ermöglicht. Um eine Datei in XWF über einen internen Viewer zu öffnen, kann der Befehl „Einsehen" im Kontextmenü des Verzeichnis-Browsers verwendet werden. Mit separaten, externen Viewer-Komponenten lassen sich außerdem die internen Viewer erweitern. Damit sind Dateien einsehbar, deren Format nicht unterstützt wird. Die separaten Viewer-Komponenten müssen in dem selben Verzeichnis wie XWF abgelegt werden. Bei einer Live-Untersuchung sollte jedoch auf externe Viewer verzichtet werden, weil diese ihre Einstellungen „in Form von Dateien im Windows-Profilverzeichnis des ausführenden Benutzers" ablegen. Dies führt zu einer Veränderung des zu untersuchenden Systems und sollte vermieden werden. [7]

2.1.2 ESE Datenbanken in X-Ways Forensics

Für die Betrachtung und Auswertung von ESE-Datenbanken gibt es aktuell keine Viewer-Komponente in XWF.
Die Filtermöglichkeiten lassen sich auf den Inhalt des Verzeichnis-Browsers anwenden. Darunter

[1] Das Encase (E01)-Format wird verwendet, um digitale Beweise zu speichern. Dabei werden mehrere Dateien mit einer einheitlichen Größe (Standard: 640 Megabyte (MB)) erstellt. [3]

gibt es, wie in Abbildung 1 zu sehen ist, die Möglichkeit unter `Typ | Windows Internals` nach `edb - Extensible Storage Engine` zu filtern. Auch wenn eine Betrachtung des Dateiinhalts einer ESEDB nicht möglich ist, so können unter Details einige Metainformationen zu der Datei gewonnen werden. Dies umfasst „Daten aus dem Datei-Überblick“, „Berechtigungen“, sowie „Aus dem Datei-Inhalt gewonnene interne Metadaten“. Unter Letzterem werden grundlegende Informationen zur ESEDB-Datei, welche dem Datenbank-Header entnommen wurden, aufgelistet. Dies umfasst folgende Daten:

- Erstellungszeit
- Consistent-Zeit
- Attach-Zeit
- Detach-Zeit
- Zustand der Datenbank
- Dateiformat Revision
- Betriebssystem-Version
- Betriebssystem-Build-Nummer
- Service Pack
- Repair Count

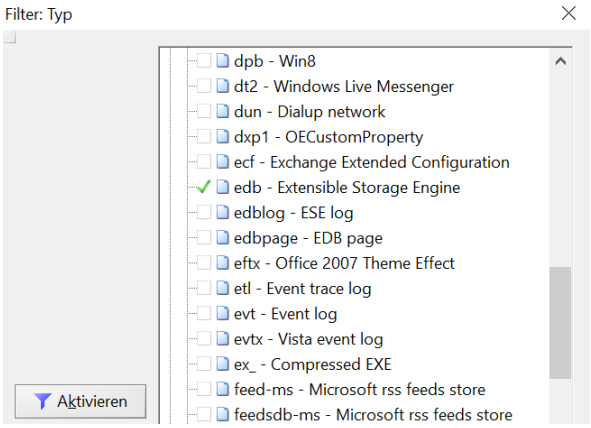

Abbildung 1: Filtermöglichkeit in XWF nach Extensible Storage Engine Dateien

2.2 X-Ways Forensics X-Tensions API

Um den Funktionsumfang von XWF zu erweitern oder Aufgaben zu automatisieren, stellt die Software verschiedene Möglichkeiten zur Verfügung. Dazu gehören beispielsweise die Verwendung von Skripten und X-Tensions [5].
„X-Tensions sind DLLs“, die vom Benutzenden der Anwendung selbst programmiert werden können, „um die Funktionalität von X-Ways Forensics zu ergänzen oder für [...] [eigene] Zwecke automatisiert zu nutzen“ [7]. Eine X-Tension ist kein interpretiertes Skript, sondern regulär kompilierter Code. Dieser wird im Adressraum von XWF selbst ausgeführt und läuft daher mit gleicher Performance wie interne Funktionen. [8]
Um eigene Erweiterungen zu implementieren, wird ein entsprechender Download der X-Tensions API, das heißt der notwendigen Dateien `X-Tension.h` und `X-Tension.cpp`, auf der Herstellerwebsite unter `https://www.x-ways.net/forensics/x-tensions/api.html` bereitgestellt. Eine API ist eine Programmierschnittstelle, „die definiert, welche Betriebssystem-Aufrufe einem Anwendungsprogramm zur Verfügung stehen“ [9]. Die API ist in XWF ab Version 16.4 in 32 und 64 Bit verfügbar [8].

Weiterhin wird eine Sammlung von X-Tensions durch XWF unter folgendem Link veröffentlicht: `https://www.x-ways.net/forensics/x-tensions/index.html`. Diese können frei heruntergeladen und genutzt werden.

Die Programmierschnittstelle bietet viele Implementierungsmöglichkeiten, beispielsweise spezielles Carving für bestimmte Dateitypen oder das automatische Herausfiltern unerwünschter Suchtreffer. Grundsätzlich ist es zum Beispiel möglich, einen Datenträger oder Image auszulesen, Informationen über jede Datei und jedes Verzeichnis aus dem Datei-Überblick abzurufen, neue Objekte im Datei-Überblick anzulegen, Vermerke oder Kommentare zu erstellen, Suchtreffer zu verarbeiten und, aufgrund der Windows-API, im Allgemeinen alles weitere, was mit einem Windows-Programm möglich ist. [8]
Laut der Dokumentation der X-Tensions API ist zu beachten, dass der Name der X-Tension mit „XT“ beginnen muss und weniger als 32 American Standard Code for Information Interchange (ASCII) Zeichen lang ist. Die kompilierte DLL muss außerdem im selben Verzeichnis, wo auch XWF ausgeführt wird, gespeichert werden (bei 64 Bit DLL: im Unterverzeichnis `\x64`). [8]
Für die Programmierung der X-Tension sind sowohl Compiler als auch Programmiersprache frei wählbar. Voraussetzung ist lediglich, dass die Erstellung einer Windows-DLL möglich ist. Zu den Compilern gehören zum Beispiel Visual Studio, Free Pascal, Lazarus und lcc-win. Bei den Programmiersprachen stehen unter anderem C, C++, Delphi/Pascal, Visual Basic und Python zur Verfügung, außerdem gibt es auch einen C# Port. [8]

X-Tensions lassen sich an verschiedenen Stellen im Programm aufrufen [8]:

- bei der Erweiterung des Datei-Überblicks
- bei der Ausführung einer parallelen Suche, um Suchtreffer direkt zu verarbeiten
- über das Kontext-Menü des Verzeichnisbrowsers zur Verarbeitung der ausgewählten Objekte im Datei-Überblick
- über `Optionen | Viewer-Programme`, um eine Viewer X-Tension aufzurufen (siehe Abbildung 2)

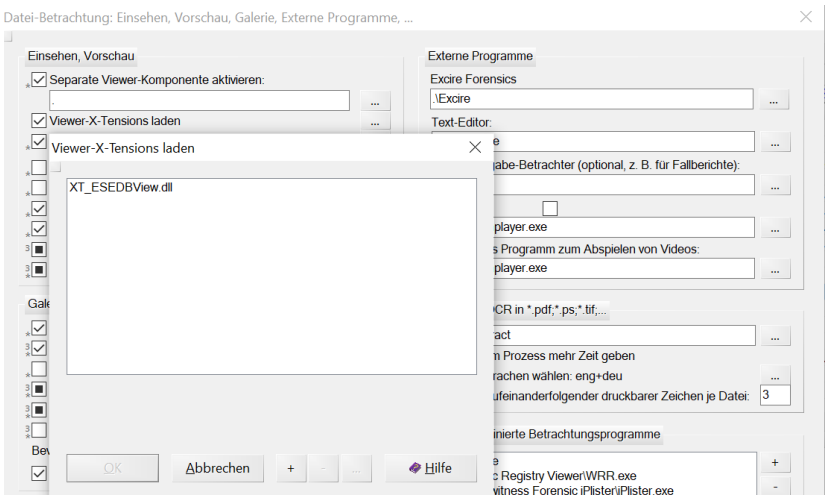

Abbildung 2: Aufruf einer Viewer X-Tension

Neben „normalen“ und Viewer X-Tensions gibt es außerdem Disk I/O X-Tensions. Diese erlauben es, Daten abzufangen, die XWF von Geräten oder Images liest und sie im laufenden Betrieb zu verändern, bevor sie geparst werden oder die Anzeige in der Benutzeroberfläche erreichen (zum Beispiel für die Entschlüsselung von Daten). [8]

In jeder DLL muss mindestens eine Funktion, die `XT_Init()`-Funktion, implementiert und exportiert werden. Welche weiteren Funktionen verwendet werden hängt davon ab, wann genau und durch wen diese aufgerufen werden sollen. Grundsätzlich wird zwischen XT_*-Funktionen und XWF_*-Funktionen, in Abhängigkeit davon, von wem die Funktionen aufgerufen werden, unterschieden. [8]

XT_* Funktionen werden von XWF aufgerufen. Diese Funktionen können für „normale“ X-Tensions, Viewer-X-Tensions und Disk I/O-X-Tensions verwendet werden. Damit die Funktionen von XWF gefunden werden können, müssen sie mit einem bestimmten Namen exportiert werden. [8]
Die obligatorische Funktion `XT_Init()` gehört zu den XT_*-Funktionen und wird, neben den weiteren verwendeten Funktionen, in Unterabschnitt 3.3.1 genauer vorgestellt.

Die XWF_*-Funktionen werden vom Entwickelnden aus den XT_*-Funktionen aufgerufen. XWF_*-Funktionen werden zum Beispiel für das Abfragen oder Setzen von Werten verwendet. [8] Mithilfe der Funktion `XWF_GetProp()` kann beispielsweise die Größe einer Datei abgefragt werden. `XWF_OutputMessage()` erlaubt es, eine benutzerspezifische Nachricht im Nachrichten-Fenster von XWF auszugeben. [10]

Alle exportierten X-Tensions Funktionen und internen XWF-API-Funktionen verwenden die `stdcall` Aufrufkonvention unter 32 Bit [8]. Eine Aufrufkonvention teilt dem Compiler die Regeln für die Einrichtung des Stacks, das Übergeben von Argumenten und das Abrufen des Rückgabewertes mit [11].
Bei der Implementierung der `__stdcall` Aufrufkonvention bereinigt der Aufgerufene den Stack und die Argumente werden von rechts nach links übergeben. Außerdem wird Namen ein Unterstrich (_) vorangestellt und auf den Namen folgt das At-Zeichen (@), sowie die Anzahl der Bytes in der Argumentliste. Die allgemeine Syntax lautet [11]:

```
return-type __stdcall function-name[( argument-list )]
```

2.3 Dynamic Link Library

Wie bereits erklärt wurde, ist eine X-Tension im Grunde eine DLL. „Eine DLL ist eine Bibliothek, die Code und Daten enthält [...] [und] von mehreren Programmen gleichzeitig verwendet werden [kann]“ [12]. Sie wird, wie auch bei XWF, verwendet, um die Funktionalität einer Anwendung zu erweitern [13].
Die gemeinsam genutzte Bibliothek ist Teil des Shared Memory Konzepts: gemeinsam genutzte Speicherbereiche werden nur einmal in den Hauptspeicher geladen und können so von mehreren Prozessen genutzt werden. Dies ermöglicht das Sparen von Arbeitsspeicher, da nicht jeder Prozess separat den gleichen Code in den Speicher lädt. [14]

Durch die Verwendung von DLLs ergeben sich neben einer effizienteren Speicherplatznutzung weitere Vorteile. Mithilfe von DLLs lassen sich Programme in einzelne separate Komponenten, sogenannte Module aufteilen. Da diese vom Hauptprogramm getrennt sind und nur nach Bedarf geladen werden,

kann die Ladezeit eines Programms sowie des Betriebssystems verkürzt werden. [12] Da mehrere Anwendungen auf die gleiche DLL zugreifen, muss die Aktualisierung dieser auch nur einmal durchgeführt werden [14]. Außerdem hat eine Veränderung von Funktionen in einer DLL nicht zur Folge, dass Anwendungen, die diese nutzen, neu kompiliert oder verknüpft werden müssen, solange Funktionsargumente, Aufrufkonventionen und Rückgabewerte sich nicht ändern. Weiterhin können sogar Anwendungen, die in verschiedenen Programmiersprachen geschrieben wurden, die gleiche DLL verwenden, wenn die Programme den gleichen Aufrufkonventionen folgen. [15]

Durch die gemeinsame Nutzung von Systembibliotheken entstehen aber auch Nachteile. Verwendet ein Programm eine DLL, so bildet sich eine Abhängigkeit. Das Programm ist nicht mehr eigenständig. Fehlt ein Objekt (z.B. aufgrund von Löschen oder Verschieben), so ist möglicherweise die komplette Anwendung nicht mehr lauffähig. Wird bei der Installation einer älteren Software eine DLL mit einer älteren Version überschrieben, so kann dies zu Kompatibilitätsproblemen bei der Ausführung anderer Programme führen. Außerdem können DLLs für Windows-Systeme ein Sicherheitsrisiko darstellen. Deswegen unterbindet die Windows-Sicherheit bei der Installation von nicht autorisierten Anwendungen das Überschreiben von Systembibliotheken. [12]

2.4 ESE Datenbanken

Die ESE Datenbank ist eine von Microsoft entwickelte Datenspeicherungstechnologie, welche mit Windows NT 3.51 und Exchange 4.0 eingeführt wurde [16]. ESE ist auch als Joint Engine Technology (JET) Blue bekannt, die nicht mit der zweiten Implementierung der JET-API, JET Red, verwechselt werden sollte [17].
Die Datenbank wird in Form einer Datei gespeichert, welche eine maximale Größe von 16 Terabyte (TB) besitzen kann. Sie wird in Windows Systemen vielfältig eingesetzt, vor allem in Anwendungen, die eine schnelle oder leicht strukturierte Datenspeicherung benötigen. Dazu gehören beispielsweise die Windows Suche, Windows Mail und Microsoft Exchange. [18]
Die ESEDB beruht auf der Indexed Sequential Access Method (ISAM)-Speichertechnologie. Diese Zugriffsmethode für Datensätze in einer Datei ermöglicht Anwendungen, Daten sequentiell oder index-basiert in Tabellen zu speichern und abzurufen. [17, 19] Weiterhin wendet die Engine das Write-Ahead-Logging (WAL)-Prinzip an: vor der Änderung der Datenbank-Datei werden die Modifikationen protokolliert, sodass im Falle eines Systemabsturzes die Datenkonsistenz gewahrt werden kann. [17, 20]

Es werden drei Typen der ESE Datenbank unterschieden [21]:

- ESENT: Datenbank für Active Directory und viele weitere Microsoft Windows Komponenten
- ESE97: Datenbank in Exchange Server 5.5
- ESE98: Datenbank in Exchange Server 2000, 2003 und 2007

2.4.1 Anwendungen

ESE Datenbanken werden von vielen Anwendungen für die Datenspeicherung verwendet. Dazu zählen unter anderem die Browser Microsoft Edge und Internet Explorer, sowie Windows Mail, Windows Desktop Suche, Microsoft Exchange Server, das Active Directory, Windows Live Messenger, Cortana Suche und viele weitere Windows-Komponenten. In den folgenden Unterabschnitten wird auf

einige ausgewählte Anwendungen eingegangen, um die forensische Bedeutung der Untersuchung von ESEDB-Dateien zu erläutern. Dafür werden jeweils die in der zugrundeliegenden ESE Datenbank gespeicherten Informationen kurz vorgestellt.

2.4.1.1 Internet Explorer und Microsoft Edge

Das World Wide Web (WWW) gehört zu den Hauptinformationskanälen, den Nutzer:innen zum Austausch von Daten verwenden. Die gängigsten Tools dafür sind Browser. Eine Untersuchung dieser ist also wichtig, um die Aktivitäten eines Nutzenden zu prüfen. [22]
Sowohl Microsoft Edge als auch der Internet Explorer 10 und 11 verwenden für die Speicherung von Browsing-Artefakten, wie zum Beispiel dem Suchverlauf, eine ESE Datenbank [18]. Auch wenn der Support für den Internet Explorer eingestellt wurde, ist es möglich, in Fällen, welche bereits in der Vergangenheit liegen, relevante Informationen aus Dateien des Internet Explorers zu extrahieren, vorausgesetzt diese sind auf einem System noch vorhanden. Weiterhin seien laut Chivers die Erkenntnisse aus Studien des Internet Explorers auch auf den Browser Microsoft Edge übertragbar. [2]
Die Speicherung von Browsing-Artefakten mittels ESE Datenbank wurde mit dem Internet Explorer 10 eingeführt. Zu den gespeicherten Informationen zählen unter anderem der Suchverlauf, Cookies und der Web Cache. In beiden Browsern werden diese Daten in der ESEDB-Datei `WebCacheV01.dat` gespeichert. [23]
Der Standard-Speicherort der Datei ist:

```
%LOCALAPPDATA%\Microsoft\Windows\WebCache
```

Ein weiteres Feature beider Anwendungen ist das InPrivate-Browsing. In diesem Modus werden laut Microsoft keine Daten lokal auf dem Computer gespeichert, weshalb diese besonders von forensischem Interesse sein könnten. Chivers untersuchte ob Spuren von InPrivate-Browsing Sessions in der ESEDB-Datei zu finden sind und konnte unter bestimmten Umständen Artefakte identifizieren und wiederherstellen. [23]

2.4.1.2 Windows Mail

Eine weitere forensisch interessante Anwendung, welche für die Speicherung von Daten auf eine ESE Datenbank zurückgreift, ist Windows Mail. Ein wichtiges Merkmal dieser Anwendung, ist die Fähigkeit, Zugriff auf viele verschiedene E-Mail Konten in einer einzigen Benutzeroberfläche zu ermöglichen. Fügt ein Benutzer ein Konto zur Mail-Anwendung hinzu, so werden E-Mails, Kontakte und Termine lokal auf dem Computer zwischengespeichert. In der Datenbank-Datei `store.vol` werden neben Nachrichten und Kontoinformationen auch Termine und zugehörige Daten gespeichert. [2]
Die Datei wird an folgender Stelle gespeichert:

```
%LOCALAPPDATA%\Comms\UnistoreDB
```

In der Datenbank-Datei gibt es unter anderem die Tabellen Message, Contact, Appointment, Attachment, Recipient, Folders und Store. Die Message-Tabelle enthält zum Beispiel den Nachrichtenteil der E-Mail und die Attachment-Tabelle die Anhänge. Nachteilig ist jedoch, dass der vollständige E-Mail Header nicht gespeichert wird. Somit lässt sich die Kette der Server, über die die E-Mail zugestellt wurde nicht zurück verfolgen. [2]

2.4.1.3 Windows Suche

Die Windows Suche nutzt eine ESE Datenbank, für die Verwaltung aller Dateien und Programme aller Nutzer eines Computers. Die Datei wird vom Windows Indizierungsdienst erstellt, um Suchprozesse im Windows Betriebssystem zu beschleunigen. Neben vorhandenen Daten eines Systems werden außerdem Informationen über Dateien und Ordner gespeichert, die nicht verfügbar sind. Dazu gehören zum Beispiel externe oder verschlüsselte Laufwerke oder gelöschte Dateien. Neben diesen potenziell wichtigen Informationen für einen forensischen Ermittler werden auch Metadaten zu jeder Datei gespeichert. Dazu gehören der Speicherpfad oder der Zeitstempel. Alle Daten werden in der Datei `Windows.edb` abgelegt, welche sich in folgendem Verzeichnis befindet [22, 24]:

```
%SYSTEMDRIVE%\ProgramData\Microsoft\Search\Data\Applications\
    Windows
```

2.4.1.4 Active Directory

Das Active Directory ist ein Verzeichnisdienst und „speichert beispielsweise Informationen über Benutzerkonten wie Namen, Kennwörter [und] Telefonnummern“ eines Netzwerks [25]. Diese Informationen werden in der ESEDB-Datei `ntds.dit` gespeichert. Dabei beinhaltet die Datei drei Haupttabellen: die Schema-Tabelle, die Link-Tabelle und die Daten-Tabelle. Die Schema-Tabelle enthält Informationen zu Objektklassen, Attributen und deren Beziehungen untereinander. In der Link-Tabelle werden zum Beispiel Angaben zu Gruppenmitgliedschaften gespeichert. Die Daten-Tabelle enthält Daten über Benutzer, Gruppen und allen weiteren Daten die in das Active Directory integriert sind. [26]
Der Standard-Speicherpfad der Datei ist:

```
%SYSTEMDRIVE%\Windows\NTDS
```

2.4.1.5 Microsoft Exchange

Die Messaging- und Groupware-Plattform von Microsoft speichert den Inhalt eines Postfachs in einer sogenannten Postfachdatenbank. Diese Datenbank-Datei ist eine ESEDB und wird seit dem Exchange Server 4.0 für die Datenspeicherung verwendet. Dabei ist jedem Postfach eine Datenbank-Datei (`MDB01.edb`) zugeordnet. Die Daten werden in verschiedene Tabellen gespeichert, dazu zählen Postfach-, Ordner-, Nachrichten- und Anhänge-Tabellen. Allgemein enthalten die Tabellen alle Informationen die im Posteingang zu sehen sind. Neben den Postfachdatenbanken gibt es öffentliche Ordner. Dort können Nachrichten, Termine und Kontakte gespeichert werden, welche für mehrere Personen sichtbar sind. Diese öffentlichen Ordner wurden in früheren Exchange Server Versionen in einer eigenen ESE Datenbank-Datei gespeichert. Seit dem Exchange Server 2013 werden sie hingegen in die Postfachdatenbank integriert. [27]

2.4.2 Allgemeiner Aufbau einer ESE Datenbank

Die ESE Datenbank wird in einer einzigen Datei gespeichert und kann aus mehreren Tabellen bestehen, in denen die Datensätze gespeichert werden. Die Tabellen bestehen aus mehreren benutzerdefinierten Spalten. Indizes ermöglichen die Organisation der Datensätze. [28]

Eine Tabelle besteht aus den folgenden Elementen [28]:

- Spalten: Eine Spalte ist ein Feld in der Tabelle, welches eine bestimmte Art von Information in fester oder variabler Länge, in Abhängigkeit des Datentyps, speichert.
- Datensätze: „Ein Datensatz ist eine Sammlung von Spaltenwerten, die eine eindeutige Identität haben, die durch den Primärschlüssel definiert ist.“ (Übers. der Verfasserin) [28]
- Index: Der (primäre) Index ist eine Sammlung von Spalten. Diese definieren in welcher Reihenfolge die Datensätze in der Tabelle gespeichert werden. Neben dem primären Index können weitere Indizes definiert werden.
- Cursor: Der Cursor zeigt den aktuellen Datensatz in der Tabelle an und navigiert mithilfe des Indexes zu Datensätzen in der Tabelle.

Sowohl Spalten als auch Indizes können jederzeit zu einer Tabelle hinzugefügt oder gelöscht werden. [28]

Die Daten in der Tabelle werden entsprechend der Definition des Primärindexes in einem B^+-Baum gespeichert. Jeder weitere Sekundärindex wird in einem separaten B^+-Baum gespeichert und enthält lediglich logische Zeiger auf die tatsächlichen Daten, die in der Primärtabelle gespeichert sind. Wenn kein Index definiert ist, so werden die Daten in der Reihenfolge, in der sie eingefügt wurden im B^+-Baum gespeichert und als sequentieller Index bezeichnet. [28]

Aus physischer Sicht werden die Daten in Seiten fester Größe gespeichert. Die erste Seite der ESEDB-Datei enthält den Datenbank-Header. In der zweiten Seite wird eine Kopie des Headers gespeichert. Anschließend folgen die Seiten, welche die Daten der Tabellen enthalten. Dieser Aufbau wird in Abbildung 3 veranschaulicht. Wie zu sehen ist, startet die Nummerierung der Seiten erst bei der dritten Seite mit der Nummer 1. [18] Die Daten werden in Little-Endian[2] gespeichert. Alle Datums- und Zeitangaben sind in UTC. [21]

Datei

H	K	1	2	3	4	5	6
7	8	9	10	11	12	13	14
15	16	17	18	19	20	21	22
23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38
39	40	41	42	43	44	45	...

Abbildung 3: Aufbau der Datenbank-Datei: Header (H), Kopie des Headers (K) und Seiten; in Anlehnung an [30]

[2] Im Little-Endian Format wird das niederwertigste Byte zuerst gespeichert [29].

2.4.3 B⁺-Bäume

Um einen schnellen Zugriff auf die Daten zu ermöglichen, werden diese meistens in Form von Bäumen, insbesondere B⁺-Bäumen organisiert [31]. Auch in der ESEDB werden die Daten in B⁺-Bäumen strukturiert [28].

Ein Baum ist „eine Datenstruktur, die aus einer Anzahl von Knoten besteht, die so durch Kanten verbunden sind, dass keine Kreise auftreten [...]. Die Kanten verbinden die Knoten [...] [und] der oberste Knoten des Baumes wird als Wurzel bezeichnet.“ [32]
Hat ein Knoten mindestens einen Nachfolger, so wird er als innerer Knoten bezeichnet. Knoten die keine Nachfolger haben, heißen Blätter. [32]

Die von Bayer und McCreight 1972 für die Speicherung von Daten auf externen Speichermedien entwickelten B-Bäume, sind eine spezielle Variante des allgemeinen Baums [33]. Das „B“ steht dabei für „balanciert“ und bedeutet, dass „alle Pfade von der Wurzel zu den Blättern des Baumes gleich lang sind“ [34]. Demzufolge müssen alle Blattknoten auf der gleichen Ebene liegen. Die Länge des Pfades wird auch als Höhe h des Baums bezeichnet. [33]
Weiterhin müssen nach Bayer und McCreight [33] folgende Eigenschaften erfüllt sein:

1. Alle Knoten außer die Wurzel und Blätter haben mindestens $k+1$ Nachfolger. Wenn die Wurzel kein Blatt ist, dann hat sie mindestens zwei Nachfolger.
2. Jeder Knoten hat maximal $2k + 1$ Nachfolger.
3. Jeder Knoten außer der Wurzel enthält zwischen k und $2k$ Elemente. Die Wurzel enthält zwischen 1 und $2k$ Elemente.
4. Ein Knoten mit d Elementen, der kein Blattknoten ist, besitzt $d + 1$ Nachfolger.
5. Die Elemente innerhalb eines Knotens sind aufsteigend sortiert.

k wird auch als Ordnung bezeichnet und gibt damit die minimale Anzahl an Elementen eines Knotens, außer der Wurzel an. Ein Knoten eines B-Baums der Ordnung $k = 3$ enthält damit zwischen 3 und 6 Elementen. [34] Die Knoten eines B-Baums werden auch als Seiten bezeichnet [31].

Die in der Praxis am häufigsten verwendete Variante des B-Baums ist der B⁺-Baum [34]. Im Unterschied zum klassischen B-Baum sind die Datensätze oder die Verweise auf die Datensätze in einem B⁺-Baum lediglich in den Blattknoten enthalten. Die oberen Ebenen sind hingegen wie ein normaler B-Baum organisiert und enthalten Zeiger, welche eine schnelle Suche in den Daten ermöglichen. Die Blattknoten sind außerdem in der Regel von links nach rechts miteinander verknüpft. So entsteht eine verkettete Liste, welche eine einfache sequentielle Verarbeitung ermöglicht. Die sich ergebende Struktur in einer ESE Datenbank wird vereinfacht in Abbildung 4 dargestellt. [35, 31]
Der B⁺-Baum wird auch als „hohler Baum [bezeichnet], da die inneren Knoten keine Information tragen und nur zum Verweis auf die Blattseiten dienen“ [34].

In einer ESE Datenbank wird, in Abhängigkeit von ihrer Funktion, zwischen verschiedenen B⁺-Bäumen unterschieden. Die eigentlichen Daten der Tabellen werden in Daten-Bäumen gespeichert. Da die Seiten eine feste Größe besitzen, gibt es für größere Daten (die größer als die Seiten-Größe sind) Long Value (LV) Seiten. Diese werden in einem separaten LV Baum gespeichert. Die Indizies werden in Index-Bäumen gespeichert. Weiterhin gibt es Space-Bäume, welche für die Verwaltung der freien und belegten Seiten verantwortlich sind. [21]

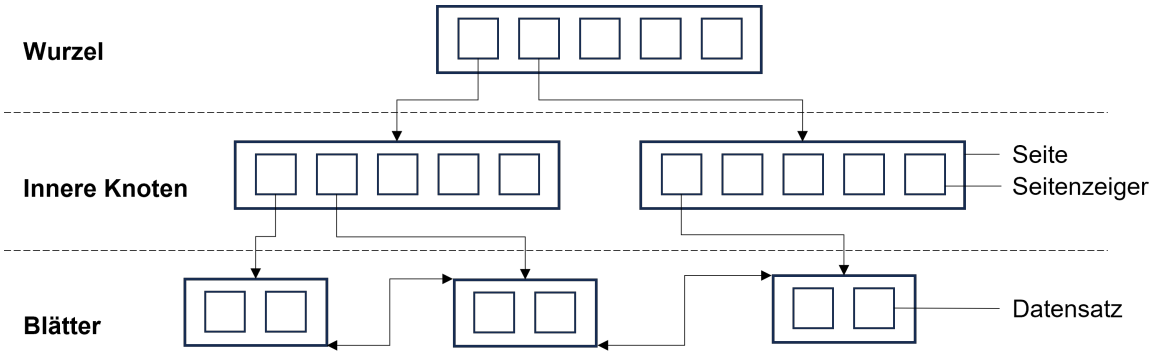

Abbildung 4: B⁺-Baum in einer ESE Datenbank: nur die Blattseiten enthalten die Datensätze, jede Blattseite hat einen Seitenzeiger auf die vorherige und die nächste benachbarte Seite; in Anlehnung an [36]

2.4.4 Tabellen

In einer ESEDB-Datei werden immer mehrere Tabellen gespeichert. Die Informationen über die Tabellen werden in der Tabelle „MSysObjects“ verwaltet, welche deshalb auch als „Katalog“ bezeichnet wird. [21] Tabellen können Untertabellen enthalten. Dabei handelt es sich um sogenannte LVs, die für die Speicherung einer größeren Menge an Daten existieren. [18]
Des Weiteren haben alle Tabellen eine eindeutige Identifikationsnummer, welche in allen zugehörigen Seiten angegeben wird. Somit wird eine eindeutige Zuordnung ermöglicht. Auch LVs nutzen eindeutige IDs und werden auf gleiche Weise verwaltet. [18]

Eine ESEDB-Datei beinhaltet mehrere Metadaten-Tabellen. Diese werden für die Verwaltung der Datenbank benötigt. Die Metadaten-Tabellen sind [21]:

- MSysObjects
- MSysObjectsShadow
- MSysObjids
- MSysLocales

Die Tabelle MSysObjects enthält die Definition aller Tabellen und Indizes, die in der Datenbank gespeichert sind. Als Backup existiert die Tabelle MSysObjectsShadow. Die Datensätze in den Blattseiten der Tabelle enthalten folgende Informationen für jede Tabelle in der Datenbank:

- Die Tabellendefinition, das heißt den Tabellennamen, sowie wo der zugeordnete B⁺-Baum startet.
- Eine oder mehrere Spaltendefinitionen, welche die Spaltennamen und die darin gespeicherten Datentypen enthalten.
- Eine oder mehrere Indexdefinitionen, sowie den Start der entsprechenden B⁺-Bäume.

Auch die eigene Tabellendefinition wird in MSysObjects gespeichert. Die vierte Seite der Datenbank enthält die Wurzelseite der Tabelle. [21, 30]

2.4.5 Datenbank-Header

Der Header der Datenbank ist der erste Eintrag und damit die erste Seite in einer ESEDB. In der zweiten Seite wird als Backup eine Kopie des Datenbank-Headers gespeichert. Der Datenbank-Header ist mindestens 668 Bytes groß. Abbildung 5 zeigt den Header in hexadezimaler Darstellung. Außerdem ist für den exakten Aufbau in Anhang A die Offset-Tabelle des Datenbank-Headers angegeben. [21]

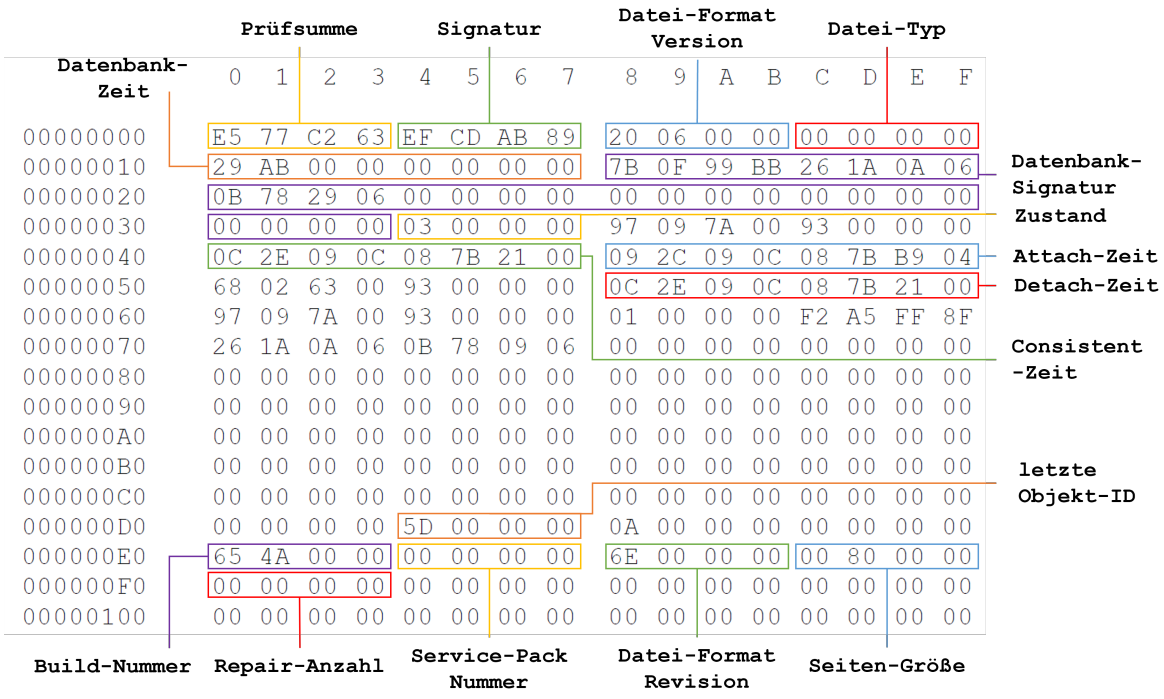

Abbildung 5: Hexadezimale Darstellung des Datenbank-Headers

Die ersten 4 Byte des Headers bilden eine Prüfsumme. Anschließend folgt an Offset `0x04` die 4 Byte lange Signatur einer ESEDB-Datei: `0x89ABCDEF`. Die Dateiformat-Version (Offset `0x08`) in Zusammenhang mit der Dateiformat-Revision (Offset `0xE8`) sind weitere wichtige Informationen, welche die Struktur der Datei bestimmen können. Der Dateityp (Offset `0x0C`) gibt an, ob es sich um eine Datenbank, also eine hierarchische auf Seiten basierende Speicherung oder eine Streaming Datei mit gestreamten Daten handelt.[3] Die vorliegende Arbeit, sowie die Auswertung in XWF beziehen sich lediglich auf Variante 1 - Speicherung in einer Datenbank. [21]
Im Header werden mehrere verschiedene Zeitstempel gespeichert. Die 28 Byte lange Datenbank-Signatur an Offset `0x18` enthält die Erstellungszeit der Datei (siehe Tabelle 8). Weiterhin werden im Header die Consistent, Attach und Detach Zeiten gespeichert. Ein Zeitstempel hat jeweils eine Länge von 8 Bytes. Die zwei höchstwertigen Bytes sind Füllbytes. Abbildung 6 zeigt den Aufbau der niederwertigen 6 Bytes. [21]

Durch die Umwandlung der hexadezimalen Werte in Dezimalzahlen, erhält man aus dem Zeitstempel `78 0B 06 0A 1A 26` die Erstellungszeit 06.11.2022 10:26:38 Uhr.

[3] `0x00`: Datenbank, `0x01`: Streaming-Datei

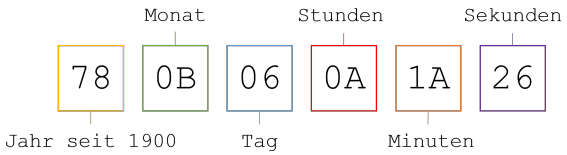

Abbildung 6: Hexadezimaler Aufbau des Zeitstempels (Darstellung im Little Endian-Format)

Ein weiteres wichtiges Feld ist der Zustand der Datenbank (Offset `0x34`): dieser gibt beispielsweise an, ob die Datenbank in einem sauberen Zustand ist. Tabelle 1 zeigt mögliche Werte, sowie deren Bedeutung. [21]

Tabelle 1: Mögliche Werte des Feldes Datenbank-Zustand [21]

Wert	Bedeutung
`0x01`	Datenbank wurde gerade erstellt
`0x02`	Dirty Shutdown
`0x03`	Clean Shutdown
`0x04`	Datenbank wird aktualisiert
`0x05`	Intern

Häufig ist die Datenbank nicht im Zustand „Clean Shutdown“, sondern im Zustand „Dirty Shutdown“. Das bedeutet, dass die Datenbank-Datei nicht richtig aktualisiert wurde. Dies entsteht dadurch, dass die Engine das WAL-Prinzip verwendet. Die Logdateien enthalten alle verschiedene Datenbankoperationen, bevor sie in die Datenbankdatei geschrieben werden. Wird eine Datei untersucht, welche sich im Zustand "Dirty Shutdown“ befindet, sollte also beachtet werden, dass diese sich nicht auf dem aktuellen Stand befindet, weil nicht alle Seiten aus dem Speicher auf die Festplatte geschrieben wurden. [37] Die Consistent-Zeit (Offset `0x40`) gibt an, wann die Datenbank das letzte mal in einem sauberen Zustand war, wenn sie aktuell im Zustand „Dirty Shutdown“ ist [21].
Es gibt die Möglichkeit mithilfe des in Windows integrierten Kommandozeilen-Tools `esentutl` die Datenbank wiederherzustellen. Dabei kommt es jedoch zu einer Veränderung der Datei. Es können Daten sowohl geschrieben, als auch gelöscht werden. In Abschnitt 3.1 wird auf das Tool, sowie die entsprechenden Kommandos zur Wiederherstellung eingegangen.

Die letzte Objekt ID an Offset `0xD4` gibt die aktuelle Anzahl an B^+-Bäumen in der Datenbank an [38]. An Offset `0xEC` findet sich die Seiten-Größe, welche auch der Größe des Headers entspricht. Ist diese größer als 668 Bytes, so werden die restlichen Bytes des Headers mit 0 aufgefüllt. [21]

2.4.6 Seiten

Eine Seite ist eine logische Einheit, die zum Speichern und Verwalten der Datensätze in einer ESE Datenbank verwendet wird. Eine Seite hat eine festgelegte Größe, welche wie im vorherigen Abschnitt erwähnt, dem Datenbank-Header entnommen werden kann. Die Seiten sind in einem B^+-Baum angeordnet und können andere Seiten oder Daten im Baum referenzieren. Jede Seite referenziert außerdem eine Father Data Page (FDP)-Objekt-ID, welche die eindeutige Identifikationsnummer des zugehörigen B^+-Baums ist. Abbildung 7 zeigt den allgemeinen Aufbau einer Seite. Sie besteht aus einem Header, den eigentlichen Daten im Datenbereich und den Seiten-Tags. [21]

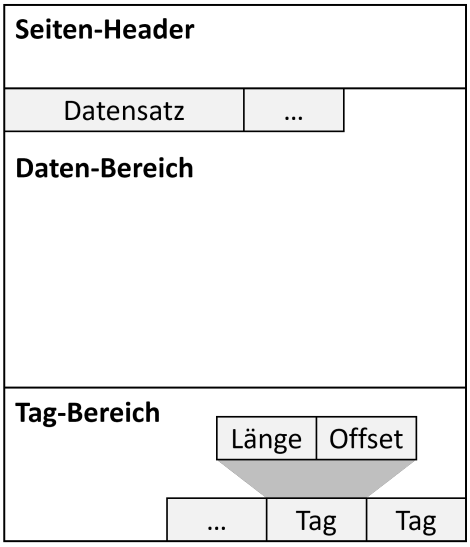

Abbildung 7: Aufbau einer Seite aus dem Seiten-Header, dem Daten- und dem Tag-Bereich; in Anlehnung an [18]

Der Offset einer Seite lässt sich wie folgt berechnen [21]:

$$\begin{aligned}\text{Seiten-Offset} &= (\text{Seiten-Nummer} * \text{Seiten-Größe}) + \text{Seiten-Größe} \\ &= (\text{Seiten-Nummer} + 1) * \text{Seiten-Größe}\end{aligned}$$

Umgekehrt lässt sich die Seiten-Nummer aus dem Seiten-Offset berechnen [21]:

$$\begin{aligned}\text{Seiten-Nummer} &= (\text{Seiten-Offset} - \text{Seiten-Größe}) \,/\text{Seiten-Größe} \\ &= (\text{Seiten-Offset}/\text{Seiten-Größe}) - 1\end{aligned}$$

Folgendes Beispiel zeigt die Berechnung des Offsets der Seite 3086 (`0xC0E`), welche eine Größe von `0x8000` Bytes hat. Der Header dieser Seite ist in Abbildung 8 dargestellt.

$$\text{Offset (Seite 3086)} = (\texttt{0xC0E} + \texttt{0x1}) * \texttt{0x8000} = \texttt{0x6078000}$$

2.4.6.1 Seiten-Arten

Die verschiedenen Seiten lassen sich einerseits anhand der Position der Seite im Baum und andererseits anhand der Art des Baums unterscheiden. Betrachtet man die Position, so differenziert man zwischen Wurzel-Seiten, internen Seiten und Blatt-Seiten. Unter Beachtung der Baum-Art unterscheidet man zwischen LV-Seiten, Space-Seiten, Index-Seiten, Daten-Seiten und leeren Seiten. Da in einem B^+-Baum die eigentlichen Daten lediglich in den Blatt-Seiten gespeichert werden, unterscheiden sich diese Seiten-Arten auch hauptsächlich in den Blatt-Seiten. Die verschiedenen Seiten-Typen können anhand der Seiten-Flags unterschieden werden. [18]

Wurzel-Seiten Eine Wurzel-Seite kann durch das Flag „is root“ identifiziert werden. Sie enthält interne Seiten- oder Blatt-Seiten-Einträge. [21]

Interne Seiten Wenn das Level des Baumes größer wird, dann werden einem Baum interne Seiten hinzugefügt. Diese werden für die Speicherung der Seiten-Nummern der Blatt-Seiten verwendet. [18] Eine interne Seite kann nicht durch ein bestimmtes Flag identifiziert werden. Jedoch sollte das „is leaf“ Flag nicht gesetzt sein und das „is parent“ Flag kann gesetzt sein. [21]

Blatt-Seiten Eine Blatt-Seite wird durch das Flag „is leaf“ identifiziert und enthält Blatt-Seiten-Einträge. Es wird zwischen Space, Index, LV und Daten-Blatt-Seiten unterschieden, welche jeweils durch das entsprechend gesetzte Flag bestimmt werden können. Dabei hat jede Seite, je nach Typ, unterschiedliche Einträge. [21]
Für die Einträge der Blatt-Seiten wird eine ähnliche Struktur verwendet. Die Einträge haben dabei grundsätzlich eine variable Größe und setzen sich aus einem Schlüssel und den eigentlichen Daten zusammen. [21]

Leere Seiten Eine leere Seite kann Daten enthalten, diese werden bei der Erstellung des B^+-Baumes jedoch ignoriert. [21]

2.4.6.2 Seiten-Header

Der Seiten-Header hat in Abhängigkeit der Dateiformat-Version, -Revision und Seiten-Größe eine Größe von 40 oder 80 Bytes. Mit Windows 7 wurden bei 16 und 32 Kilobyte (KB) Seiten die Header erweitert, hauptsächlich um zusätzliche Error Recovery Prüfsummen. [21]
Abbildung 8 zeigt die hexadezimale Darstellung eines Seiten-Headers. Weiterhin kann Anhang A die Offset-Tabelle des Seiten-Headers entnommen werden.

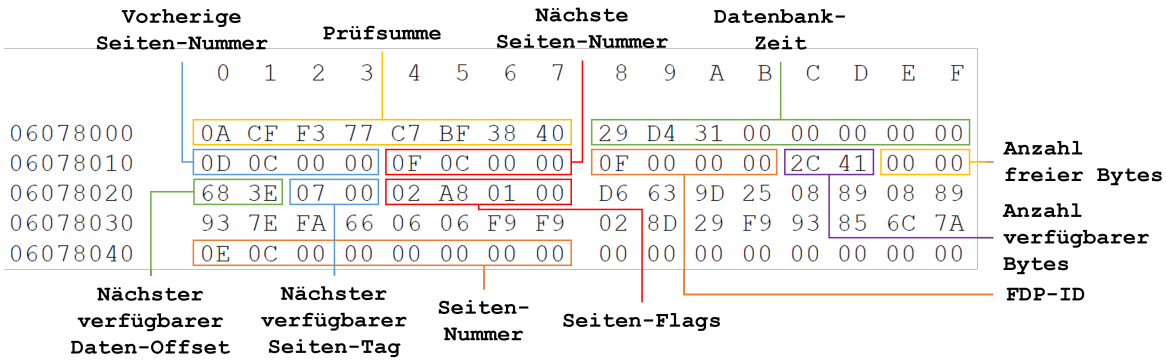

Abbildung 8: Hexadezimaler Aufbau eines erweiterten Seiten-Headers

Der Seiten-Header startet mit einer 8 Byte langen Error Correcting Code (ECC) Prüfsumme, mit der die Integrität überprüft wird. Diese wurde mit dem Exchange Server 2003 Service Pack (SP)1 eingeführt. Dieses Prüfsummen-Format ermöglicht die Korrektur von Single-Bit Errors in den Datenbank-Seiten. Dafür wird anstatt des 32-Bit ein 64-Bit Format verwendet. Im alten Format folgt auf die 32-Bit Prüfsumme die 32-Bit Seiten-Nummer für eine Verifizierung. Im neuen Format wird anstelle der 32-Bit Seiten-Nummer eine 8-Byte Prüfsumme verwendet, für welche die Seiten-Nummer als Eingabeparameter für die Berechnung dient. Wird eine falsche Seiten-Nummer gelesen, so stimmt die Prüfsumme nicht überein. Somit besteht das aktuelle Prüfsummen-Format eigentlich aus zwei 32-Bit Prüfsummen. [21] Nach van Dongen u. a. ist es deshalb unwahrscheinlich, dass die Seiten einer Offline-Datenbank manipuliert werden können, ohne diese zu beschädigen [38].

Das darauffolgende Feld, die Datenbank-Zeit, gibt laut van Dongen u. a. an, wie oft eine Seite geändert wurde [38]. Hingegen soll laut Metz dieses Feld anzeigen, wann die Seite zuletzt geändert wurde [21].
Die nächsten beiden 4 Byte langen Felder geben die vorherige und die folgende Seiten-Nummer an. Darauf folgt die FDP-ID, durch welche eine eindeutige Zuordnung zu einem B^+-Baum möglich ist. Außerdem werden im Seiten-Header die Anzahl der verfügbaren Bytes im Daten-Bereich, sowie die Anzahl wie viel davon frei sind, gespeichert. Anschließend folgt der nächste verfügbare Daten-Offset und Seiten-Tag. Die Seiten-Flags definieren um welche Art Seite es sich handelt (z.B. Wurzel oder Blatt-Seite). Eine Auflistung der möglichen Flags sowie deren Bedeutung kann Tabelle 2 entnommen werden. Der Seiten-Header in Abbildung 8 hat die Seiten-Flags `0x01A802`. Demzufolge handelt es sich um eine Seite mit den folgenden Eigenschaften: Blatt, Primary und Neues Datensatz-Format / Prüfsummen-Format. [21]
Handelt es sich um einen erweiterten Header mit einer Länge von 80 Byte, so folgen auf die Flags drei 8 Byte lange erweiterte Prüfsummen. An Offset `0x40` ist die Seiten-Nummer zu finden. [21]

Tabelle 2: Mögliche Seiten-Flags und ihre Bedeutung [21]

Flag	Bedeutung
0x00000001	Wurzel-Seite
0x00000002	Blatt-Seite
0x00000004	Eltern- oder interne Seite
0x00000008	Leere Seite
0x00000020	Space-Baum-Seite
0x00000040	Index-Seite
0x00000080	LV-Seite
0x00000400	Unbekannt
0x00000800	Unbekannt, vermutlich Primary-Seite
0x00002000	Neues Datensatz-Format / Prüfsummen-Format
0x00004000	Auf Null gesetzt
0x00008000	Unbekannt
0x00010000	Unbekannt

2.4.6.3 Daten- und Tag-Bereich

Nach dem Header folgt der Datenbereich. Für die Interpretation der Daten werden die Informationen aus dem Tag-Bereich benötigt, da die Datensätze nicht die gleiche Länge haben. Offset und Länge eines jeden Datensatzes, der in einer Seite gespeichert ist, wird im Tag-Bereich abgelegt. Der Tag-Bereich liegt am Ende einer Seite und wird von hinten nach vorne gespeichert. Ein Tag ist 4 Byte lang und besteht aus einem 2 Byte langen Offset und der 2 Byte langen Größe eines Datensatzes (siehe Abbildung 7). Im Seiten-Header wird das erste ungenutzte Seiten-Tag angegeben. [18, 21]

2.4.6.4 Datensatz in einer Blatt-Seite

Je nach Art der Seite ist der Aufbau eines Datensatzes unterschiedlich. Allgemein besteht ein Datensatz aber jeweils aus einem Schlüssel, gefolgt von den eigentlichen Daten. [18]
In den ersten drei Bit des zweiten Bytes befinden sich die Flags des Schlüssels, welche den genauen Aufbau des Schlüssels bestimmen. Mögliche Werte, sowie deren Bedeutung können Tabelle 10 in

Anhang A entnommen werden. Weiterhin ist es möglich, dass sich die Flags nicht im Schlüssel, sondern in den drei höchstwertigen Bits im Offset des entsprechenden Tags befinden [21].
Je nach gesetzten Flags, ist die variable Sprunggröße in den ersten zwei Byte, oder in Byte 3 und 4 enthalten. Nach der Sprunggröße folgt der Datendefinitions-Header, welcher 4 Byte lang ist. Eine Auswertung dieses Headers ist essentiell, um die danach folgenden Daten lesen zu können. Dort ist die ID des letzten Datentyps fester Größe und variabler Größe enthalten, sowie der Offset zu den Datentypen mit variabler Größe. Dieser Offset ist relativ zum Start des Datendefinitions-Headers. (Der Aufbau ist auch im Anhang A in der Tabelle 11 angegeben.) Anschließend folgen zuerst die Spalten mit fester Größe. Am angegebenen Offset folgen danach die Spalten mit variabler Größe. Sofern vorhanden, folgen schließlich die getaggten Spalten. Abbildung 9 veranschaulicht den konkreten Aufbau des Schlüssels. [18, 21]

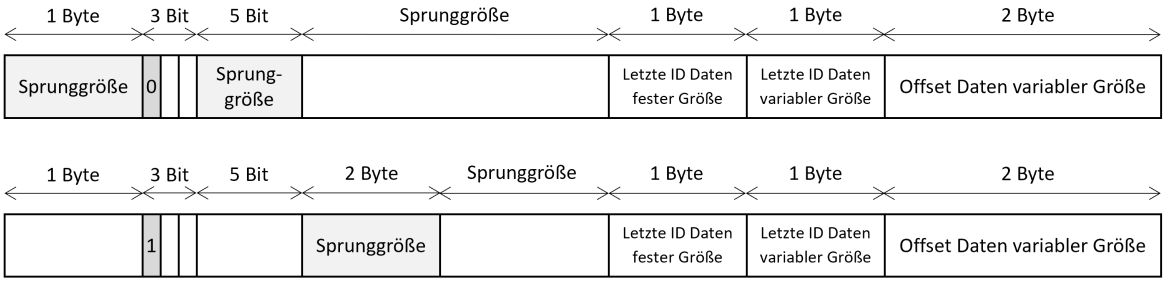

Abbildung 9: Aufbau des Schlüssels in einer Daten-Seite; in Anlehnung an [18]

2.4.6.5 Datensatz in einer internen Seite

Der erste Teil des Schlüssels in einem Datensatz in einer internen Seite ist äquivalent aufgebaut wie in einer Datenseite: Die ersten drei Bit des zweiten Bytes enthalten auch hier Flags, welche die Position der Sprunggröße im Schlüssel bestimmen. Alternativ ist auch hier die Position der Flags im Tag möglich. Nach der Sprunggröße folgt jedoch kein Datendefinitions-Header, da interne Seiten keine eigentlichen Daten, sondern lediglich Verweise auf Kind-Seiten enthalten. Deswegen folgt nach der Sprunggröße eine 4 Byte lange Seitennummer. Der Aufbau eines Datensatzes wird in Abbildung 10 gezeigt. [18]

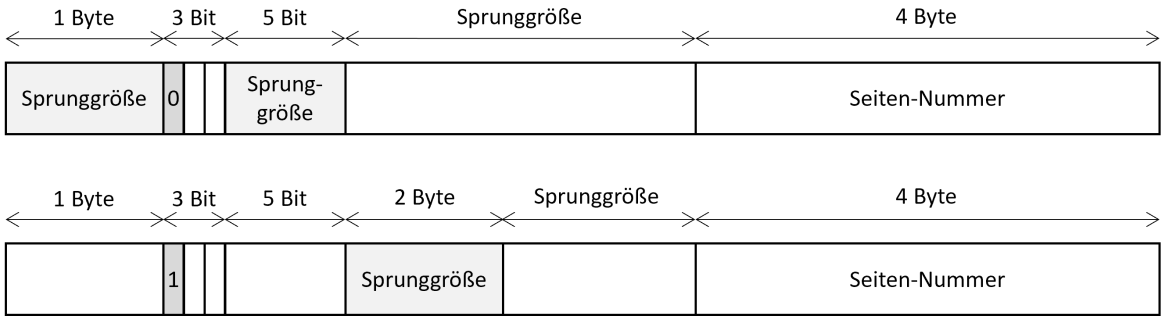

Abbildung 10: Aufbau eines Datensatzes in einer internen Seite; in Anlehnung an [18]

2.4.7 Datendefinitionen

Um die Datensätze richtig interpretieren zu können, muss bekannt sein, aus welchen Spalten eine Tabelle besteht, sowie welche Größe eine Spalte hat und welcher Datentyp in einer Spalte gespeichert wird. Diese Datendefinitionen sind in der Katalog-Tabelle MSysObjects enthalten. Auch die

eigene Definition wird in der Tabelle gespeichert. Diese kann im Anhang A in Tabelle 12 eingesehen werden. [21]
An Offset 0x04 eines Datensatzes in der MSysObjects-Tabelle wird der Typ des Eintrags spezifiziert. Dieser legt zum Beispiel fest, ob es sich um eine Spalten- oder Tabellendefinition handelt. Mögliche Werte sowie deren Bedeutung können der folgenden Tabelle 3 entnommen werden. [21]

Tabelle 3: Katalog-Typen [21]

Wert	Bedeutung
0x0001	Tabelle
0x0002	Spalte
0x0003	Index
0x0004	LV
0x0005	Callback

An Offset 0x06 ist eine 4 Byte lange ID des Eintrags angegeben. Diese spielt vor allem bei Spaltendefinitionen eine wichtige Rolle, da durch sie eine Zuordnung der Spalten in einem Datensatz erfolgt. Bei einer Spalte wird an Offset 0x0A außerdem der Spaltentyp definiert. Mögliche Spaltentypen sind in Anhang A in Tabelle 13 enthalten. Offset 0x0E enthält den Platzbedarf, welcher in Abhängigkeit des Typs sich auf die Anzahl Bytes (z.B. Spalte) oder die Anzahl Seiten (z.B. Tabelle) bezieht. [21]

2.4.8 Datentypen

Grundsätzlich wird zwischen drei verschiedenen Spalten-Arten unterschieden: Spalten mit Daten fester Größe (Fixed Size), mit Daten variabler Größe (Variable Size) und getaggte Spaltenwerte. Letztere nehmen keinen Speicherplatz ein, wenn sie keine Daten enthalten. Der entsprechende Datentyp einer Spalte kann der ID in der Definition in der Tabelle MSysObjects entnommen werden. Wie der Wert interpretiert werden muss zeigt Tabelle 4. [21]

Tabelle 4: Datentypen-IDs und ihre Bedeutung [21]

Wert	Kategorie	Beschreibung
0x0001 - 0x007F	Fixed Size	Spalten mit Daten fester Größe nutzen eine definierte Größe an Platz, auch wenn kein Wert gespeichert ist.
0x0080 - 0x00FF	Variable Size	Spalten mit Daten variabler Größe können bis zu 256 Bytes an Daten enthalten. Ein Array mit den Längen der Variable Size Datentypen ist im Datensatz gespeichert, jeder Eintrag belegt dabei 2 Bytes.
0x0100 - 0xFFFF	Tagged	Spalten mit Tagged Datentypen haben eine unbegrenzte Größe. Wenn keine Daten enthalten sind, werden keine Informationen gespeichert.

Informationen zu Daten mit fester Größe können aus der MSysObjects-Tabelle, aus den Datensätzen mit Typ 2 (Spaltendefinition) und ID $\leq$ 127 herausgefunden werden.
Daten mit variabler Größe können bis zu 255 Zeichen speichern. Informationen zu diesem Feld können aus der MSysObjects-Tabelle, aus den Datensätzen mit Typ 2 und einer ID $\geq$ 128 und $\leq$ 255 entnommen werden. [18] Die Anzahl sowie der Start der Daten mit variabler Größe wird im

Datendefinitions-Header angegeben. An diesem Offset wird in einem Array die Länge jeder Spalte definiert. Jeder Eintrag ist dabei 2 Byte lang. Außerdem muss die Größe der vorherigen Einträge von der aktuellen Größe subtrahiert werden. Weiterhin wird im höchstwertigen Bit angegeben, ob die Spalte leer ist. Anschließend folgen die Spaltenwerte in der zuvor angegebenen Länge. [21]
Eine getaggte Spalte kann bis zu 65.535 Daten speichern. Sie besitzt eine eindeutige ID, welche zwischen 256 und 65.535 liegt. Diese, sowie der Offset wird für jedes Objekt am Anfang vor den eigentlichen Daten in einem Array gespeichert. Jeder Eintrag hat eine Länge von 4 Byte und besteht aus einer 2 Byte langen ID und einem 2 Byte großem Offset. Dieser ist relativ zum Start des Tagged Daten-Arrays. [21]
Der Aufbau eines Eintrags im Array der Datentypen variabler Größe und der getaggten Datentypen kann den Tabellen 14 und 15 in Anhang A entnommen werden. [18, 21]

2.4.9 Löschen von Datensätzen

Wird ein Datensatz aus der Datenbank gelöscht, so verursacht diese Aktion eine Veränderung der Baum-Struktur und Änderungen im Tag-Bereich. Entscheidend ist, dass die Daten bei einer Löschung nicht aus dem Daten- und dem Tag-Bereich gelöscht werden, sondern lediglich der Wert für die vorhandenen Tag-Einträge im Header verändert wird, welcher für die Anzahl der Datensätze steht. Wird eine Seite nicht verwendet, wird außerdem das Flag für leere Seiten gesetzt. [18]
Neben dem Löschen von Datensätzen kommt es durch das Balancing im B^+-Baum zum Verschieben der Datensätze und Einträge. Der zu löschende Platz wird dabei von der Datenbank jedoch nicht überschrieben. Die Wahrscheinlichkeit ist groß, dass ältere Kopien von Datensätzen noch im nicht zugewiesenen Speicherplatz vorhanden sind, solange sie noch nicht von anderen Daten überschrieben wurden. [37]

3 Methoden

Dieses Kapitel widmet sich der praktischen Umsetzung der Erstellung und Implementierung der X-Tension. Dazu werden die Schritte zur Erstellung einer DLL erläutert und das umgesetzte Programmkonzept an ausgewählten Code-Beispielen vorgestellt. Eingangs wird außerdem ein Überblick über das im Entwicklungsprozess verwendete Tool `esentutl` gegeben.
Wie in Abschnitt 2.2 erläutert, ist die Wahl der Programmierumgebung und Programmiersprache bei der Entwicklung einer X-Tension frei. Für die Umsetzung dieser Arbeit wurde Microsoft Visual Studio 2022 gewählt. Außerdem ist die DLL, wie viele weitere X-Tensions, in C geschrieben.

Zusammenfassend wurde mit folgender Systemkonfiguration gearbeitet:

- Betriebssystem: Windows 10 Home, Version 22H2, 64 Bit
- Programmierumgebung: Microsoft Visual Studio Community 2022, Version 17.6.4
- Microsoft.NET Framework: Version 4.8.1
- X-Ways Forensics: Version 20.4 SR-1 x64

3.1 Esentutl

`Esentutl` ist ein Windows enthaltenes Kommandozeilen-Tool. Neben der Wiederherstellung von ESE Datenbanken kann es auch für die Anzeige der in den Tabellen gespeicherten Daten, sowie den Metadaten der Datenbank verwendet werden. Damit ist dieses Tool sehr hilfreich, um einen Überblick über eine ESEDB-Datei zu bekommen, sowie den Aufbau der Speicherung der Datensätze zu verstehen. [39]

`Esentutl` ist in folgendem Ordner gespeichert:

```
%SYSTEMDRIVE%\Windows\System32\
```

Um zu überprüfen in welchem Zustand sich die Datenbank-Datei befindet, können mit folgendem Befehl die Header-Informationen ausgeben werden [39]:

```
esentutl /mh <ESE DATABASE>
```

Im Eintrag „State“ lässt sich der Zustand der Datenbank-Datei ablesen (siehe Abbildung 11).

Im Falle des Zustands „Dirty Shutdown“ lässt sich die Datenbank mit der Option `/r` wiederherstellen. Die Option `/d` sorgt dafür, dass die Logdatei aus dem aktuellen Verzeichnis genutzt wird [39]:

```
esentutl /r <LOGFILE BASE NAME> /d
```

Wenn der Header anschließend erneut ausgegeben wird, sollte der Zustand „Clean Shutdown“ eingetragen sein.

Mit der Option `/mm` können alle in der Datenbank-Datei enthaltenen Tabellen aufgelistet werden [39]:

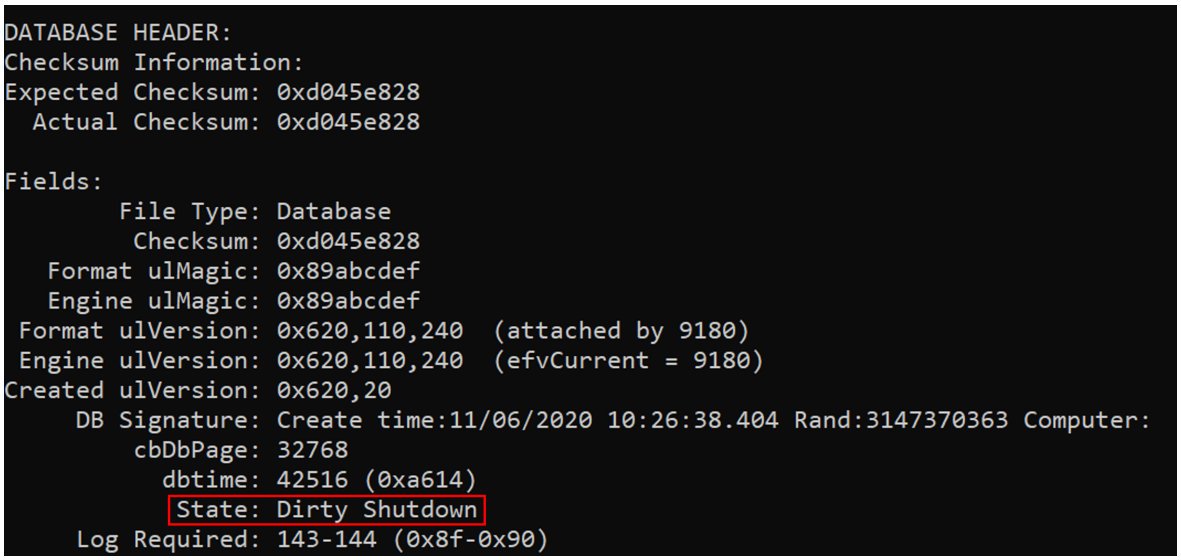

Abbildung 11: Ausgabe der Header-Informationen einer ESE Datenbank mit dem Tool `esentutl`: im Eintrag „State“ lässt sich der Zustand der Datenbank-Datei ablesen

```
esentutl /mm <ESE DATABASE>
```

Dabei werden außerdem dazugehörige Indizes und LV-Tabellen ausgeben. Weiterhin werden die entsprechenden Objekt-IDs der B^+-Bäume, sowie die FDP-Nummer des Baums angegeben.

Um den Inhalt einer bestimmt Seite auszugeben, müssen die Optionen `/ms` und `/p` verwendet werden [39]:

```
esentutl /ms <ESE DATABASE> /p<PAGE NUMBER>
```

Neben den Informationen aus dem Seiten-Header werden auch alle auf der Seite vorhanden Tags aufgelistet.

Ein bestimmter Eintrag auf einer Seite kann mit den Optionen `/ms` und `/n` aufgerufen werden. Dabei muss beachtet werden, dass die Zählung bei Tag 1 mit Null startet. Die allgemeine Syntax lautet [30]:

```
esentutl /ms <ESE DATABASE> /n<PAGE NUMBER>:<TAG NUMBER>
```

Bei internen Seiten-Einträgen wird die referenzierte Seitennummer bzw. bei Blatt-Seiten-Einträgen werden alle gespeicherten Spaltenwerte ausgegeben.

3.2 Erstellung einer X-Tension

Für die Erstellung einer X-Tension mit Microsoft Visual Studio sind folgende Schritte notwendig:

1. Microsoft Visual Studio starten, ein neues Projekt erstellen und die Vorlage `Dynamic Link Library (DLL)` auswählen wie in Abbildung 12 zu sehen ist. Anschließend Projektname, sowie Speicherort festlegen. [40]

2. Die Verwendung von vorkompilierten Headern unter `Projekt | Eigenschaften | C/C++ | Vorkompilierte Header` deaktivieren (siehe Abbildung 13). Der Zweck vorkompilierter Header besteht darin, den Build-Prozess zu beschleunigen. Der vorkompilierte Header wird nur bei Änderungen des Headers selbst oder darin enthaltenen Dateien kompiliert. Bei Änderungen des Quellcodes findet im Build-Prozess keine Kompilierung statt. [41]

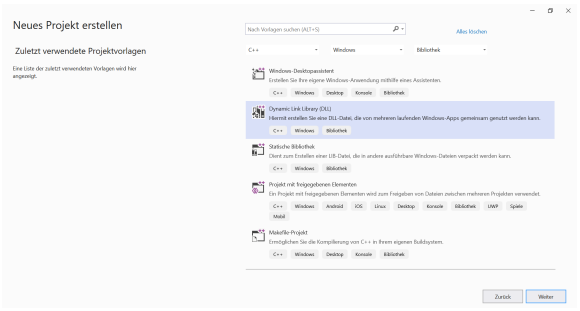

Abbildung 12: Erstellung einer X-Tension: Schritt 1

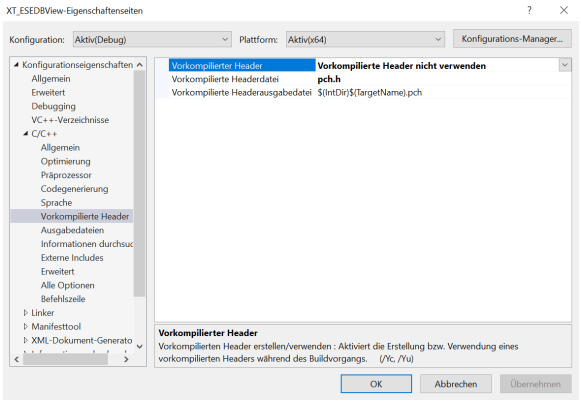

Abbildung 13: Erstellung einer X-Tension: Schritt 2

3. Die Dateien `pch.h`, `framework.h` und `pch.cpp`, die für die Verwendung vorkompilierter Header verwendet werden, sowie die Datei `dllmain.cpp` löschen. Die Datei `dllmain.cpp` definiert den Einstiegspunkt für die DLL-Anwendung und wird bei der Verwendung mit XWF nicht benötigt. [42]
4. Im Explorer die Dateien `X-Tension.h`, `X-Tension.cpp` und `New.cpp` in den Projektordner der DLL kopieren. Anschließend in Microsoft Visual Studio unter `Projekt | Vorhandenes Element hinzufügen` die entsprechenden Dateien hinzufügen. `X-Tension.h` ist die Headerdatei, `X-Tension.cpp` enthält die Implementation der Funktion `XT_RetrieveFunctionPointers()` und `New.cpp` ist die Vorlage für eigene X-Tensions, welche den dynamischen Code enthält.
5. Eine Moduldefinitionsdatei hinzufügen. Dazu unter `Projekt | Neues Element hinzufügen | Visual C++ | Hilfsprogramm` die Option `Textdatei (.txt)` auswählen, diese entsprechend benennen und die Dateiendung in `.def` ändern (siehe Abbildung 14). Die Moduldefinitionsdatei stellt „dem Linker[4] Informationen zu Exporten, Attributen und anderen Informationen über das zu verknüpfende Programm bereit“ [44]. In ihr müssen unter „Exports“ alle aufgerufenen XT_*-Funktionen definiert werden.
6. Die Moduldefinitionsdatei unter `Projekt | Eigenschaften | Linker | Eingabe | Moduldefinitionsdatei` definieren (siehe Abbildung 15).
7. Die eigene X-Tension implementieren. Die Programmumsetzung dieser Arbeit wird in Abschnitt 3.3 vorgestellt.
8. Über `Erstellen | Projekt erstellen` oder `Strg + B` die DLL erstellen. Diese wird standardmäßig im Projektordner im Unterverzeichnis `\x64\Debug\` gespeichert. Nach Abschluss der Programmierung sollte beim Erstellen der fertigen DLL von Debug auf Release umgestellt werden, damit diese vollständig optimiert wird. Die DLL wird äquivalent im Unterverzeichnis `\x64\Release\` gespeichert. [45]

[4] Ein Linker dient zum Verknüpfen von Objektdateien und Bibliotheken, um eine ausführbare Datei oder DLL zu erstellen [43].

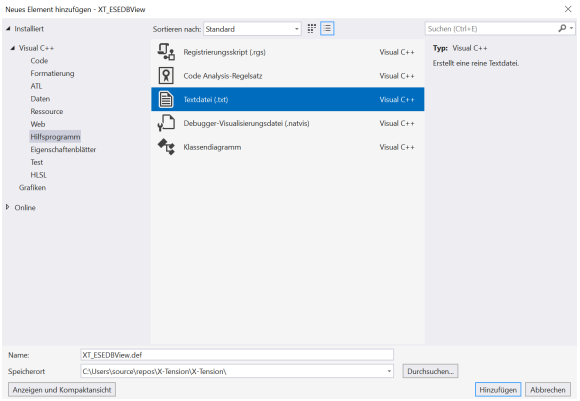

Abbildung 14: Erstellung einer X-Tension: Schritt 5

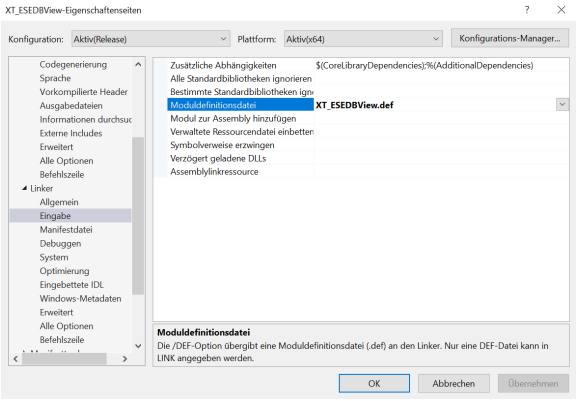

Abbildung 15: Erstellung einer X-Tension: Schritt 6

9. Zum Schluss muss die fertige X-Tension lediglich in den Programmordner von XWF kopiert werden, im Falle einer 64-Bit DLL in das Unterverzeichnis `\x64` [8]. Nun kann die X-Tension in XWF unter `Optionen | Viewer-Programme` wie in Abschnitt 2.2 erläutert geladen werden.

3.3 Programmumsetzung

Vor der Implementierung der X-Tension wurde ein Programmkonzept erstellt. In diesem Abschnitt der Arbeit werden zuerst die in der DLL exportierten XT_*-Funktionen vorgestellt, welche für eine erfolgreiche Einbindung in XWF essentiell sind. Anschließend wird das umgesetzte Programmkonzept an ausgewählten Code-Beispielen erläutert.

3.3.1 Aufruf der XT_*-Funktionen

Das Implementieren und Exportieren der Funktion `XT_Init()` ist für jede X-Tension obligatorisch. Die Funktion gehört zu den XT_*-Funktionen, welche von XWF aufgerufen werden. Sie ist auch die erste Funktion die aufgerufen wird, bevor die eigentlichen Aufgaben der X-Tension ausgeführt werden. In ihr wird die Funktion `XT_RetrieveFunctionPointers()` aufgerufen, welche die XWF_*-Funktionen enthält. Diese können im weiteren Programmablauf durch den Entwickelnden aufgerufen werden. Mit dem Rückgabewert wird das weitere Ausführen der X-Tension angezeigt. [46]
Der Aufruf der Funktion ist wie folgt:

```
LONG __stdcall XT_Init(DWORD nVersion, DWORD nFlags, HANDLE hMainWnd,
    void* lpReserved)
{
        XT_RetrieveFunctionPointers();
        XWF_OutputMessage (L"XT_ESEDBViewer initialized", 0);
        return 1;
}
```

Um eine Viewer X-Tension zu implementieren, muss die Funktion `XT_View()` exportiert werden. Eine Viewer X-Tension wird für jede Datei aufgerufen und der in der `XT_View()`-Funktion enthaltene Code wird interpretiert und im Vorschaufenster der Datei angezeigt. [46]
Folgendes Listing zeigt den Aufruf ohne den Funktionskörper der Funktion:

```
PVOID __stdcall XT_View(HANDLE hItem, LONG nItemID, HANDLE hVolume,
    HANDLE hEvidence, PVOID lpReserved, PINT64 lpResSize)
```

Die Variable `lpResSize` muss durch den Entwickelnden während der Laufzeit, auf eine der vier in Tabelle 5 genannten Optionen gesetzt werden [46].

Tabelle 5: Mögliche Werte des Parameters `lpResSize` der `XT_View()`-Funktion [46]

Wert	Bedeutung
-2	Fehler
-1	X-Tension ist nicht für die Darstellung der Datei zuständig
0	Keine Daten werden dargestellt
> 0	Angabe der Größe der Daten im Puffer

Um von der aktuellen Datei zu lesen oder weitere Informationen zu erlangen, kann die Handle-Variable `hItem` verwendet werden. Der Parameter `nItemID` wird für Funktionen des Datei-Überblicks genutzt. Die Variablen `hVolume` und `hEvidence` können verwendet werden, um weitere Informationen über den Datenträger bzw. das Asservat zu bekommen. Der Parameter `lpReserved` sollte laut der aktuellen Dokumentation ignoriert werden. [46]
Der Rückgabewert der Funktion ist die Adresse eines Puffers[5], welcher in der X-Tension definiert werden muss. Der adressierte Puffer wird interpretiert und dann im Vorschaufenster der ausgewählten Datei entsprechend angezeigt. [46]

Wird die Funktion `XT_View()` in einer X-Tension implementiert, dann ist die Funktion `XT_ReleaseMem()` obligatorisch [46]:

```
BOOL __stdcall XT_ReleaseMem(PVOID lpBuffer)
```

Die Funktion wird von XWF aufgerufen, um den Speicher freizugeben, der durch den Aufruf von `XT_View()` belegt wurde. Der Rückgabewert der Funktion wird aktuell ignoriert, jedoch sollte der Erfolg bzw. Misserfolg der Speicherfreigabe angezeigt werden. [46]

3.3.2 Prüfen der Zuständigkeit

Bevor es zum Auslesen und Interpretieren einer ESEDB kommt, muss zuerst überprüft werden, ob die X-Tension für die Datei zuständig ist. Wie in Unterabschnitt 2.4.5 erläutert, beginnt jede ESEDB-Datei mit einem Header, welcher die Signatur `0x89ABCDEF` an Offset `0x04` trägt. Darüber lässt sich eine ESEDB-Datei identifizieren. Um zu kontrollieren, ob die X-Tension für die Interpretation einer Datei zuständig ist, muss demzufolge das Vorhandensein dieser Signatur überprüft werden. Andernfalls würde die Datei von der X-Tension verworfen werden.
Die Überprüfung findet in der Funktion `XT_View()` statt. Das Ergebnis wird XWF durch das Setzen des Parameters `lpResSize` mitgeteilt (siehe Tabelle 5). Für das Prüfen werden die Funktionen `XWF_Read()` und `strcmp()` verwendet werden. Mit `XWF_Read()` kann eine bestimmte Anzahl Bytes von der Datei am angegebenen Offset gelesen werden. Anschließend können mit `strcmp()` die

[5] Ein Puffer ist ein Speicherbereich, welcher zum zwischenspeichern von Daten verwendet wird [29].

gelesenen Bytes mit der Signatur verglichen werden. Wenn diese nicht übereinstimmen, dann wird die `return`-Anweisung ausgeführt. Durch das vorherige Setzen von `lpResSize` auf -1, weiß XWF, dass die X-Tension nicht für die Darstellung dieser Datei zuständig ist. [10, 47]

```
*lpResSize = -1; // nicht zustaendig
char sig_buffer[5] = {};
char signatur[5] = {0xef, 0xcd, 0xab, 0x89, 0x00}; // Signatur

XWF_Read(hItem, 4, (BYTE*)sig_buffer, 4); // ab Offset 4 lesen, 4 Bytes

if (strcmp(signatur, sig_buffer) != 0)
{
        return NULL;
}
```

3.3.3 Auslesen des Headers

Handelt es sich um eine ESEDB-Datei, so wird zunächst der Header ausgelesen. Wie in Unterabschnitt 2.4.5 erläutert, enthält dieser wichtige Informationen, wie zum Beispiel die Seitengröße und den Dateityp, welche den weiteren Aufbau der Datei bestimmen. Dafür wird mithilfe einer Heap[6]-Funktion Speicher reserviert, welcher anschließend mit der Funktion `XWF_Read()` byteweise mit dem entsprechenden Datei-Inhalt befüllt wird.

```
HANDLE heap = GetProcessHeap();
BYTE* header = (BYTE*)HeapAlloc(heap, 0, file_size);
XWF_Read(hItem, 0, header, 668);
```

Um Werte die größer als ein Byte sind auslesen zu können, wird außerdem ein Zeiger deklariert. Folgendes Listing zeigt beispielsweise das Auslesen des Datei-Typs. Äquivalent dazu wird die Seitengröße bestimmt.

```
void* ptr;
ptr = &header[12];
file_type = *((uint32_t*)ptr);
```

Der Dateityp kann die Werte `0x00` und `0x01` annehmen. Da die X-Tension lediglich den ersten Fall (Datenbank-Dateien) unterstützt, folgt daraufhin eine Überprüfung des Datei-Typs:

```
if (file_type != 0)
{
        *lpResSize = 46;
        wsprintf(buffer, L"ERROR: Streaming Datei");
        HeapFree(heap, 0, header);
        return buffer;
}
```

[6] Ein Heap „ist ein Speicherbereich für Daten, die erst zur Laufzeit des Programms bei Bedarf angelegt werden“ [9].

Mit der Seiten-Größe kann außerdem der Start Offset der Seiten in der Datei nach der in Unterabschnitt 2.4.6 genannten Formel, berechnet werden.

```
page_offset = (page_number + 1) * page_size;
```

Weiterhin wird in diesem Schritt die Größe des Seiten-Headers bestimmt, welcher sich nach der Größe einer Seite richtet. Die Größe des Seiten-Headers stellt gleichzeitig den Offset des Datenbereichs dar, welcher relativ zum Offset einer Seite ist.

```
if (page_size >= 0x4000)
{
        data_offset = 0x50;
}
else
{
        data_offset = 0x28;
}
```

Abschließend wird der reservierte Speicherbereich wieder freigegeben:

```
HeapFree(heap, 0, header);
```

3.3.4 Auslesen der Tabelle MSysObjects

Als Grundlage zum Auslesen der Tabellen muss zuerst die Tabelle MSysObjects ausgewertet werden, die die Defintionen aller Tabellen, Spalten und Indizes der Datei enthält. Die Wurzel-Seite des zugehörigen B^+-Baums ist die Seite 4. Je nach Datenbank-Größe ist die Wurzel gleichzeitig auch eine Blatt- oder Eltern-Seite. Ist die Seite kein Blatt, enthält sie Verweise auf nachfolgende Seiten. Andernfalls sind die Daten-Definitionen bereits in Seite 4 enthalten.
Äquivalent zum Auslesen des Headers wird zuerst ein Heap erstellt und mit dem Dateiinhalt befüllt. Um die Struktur des Baums auszuwerten und alle zugehörigen Seiten auszulesen, wird daraufhin ein Array angelegt, welches alle Seiten-Nummern des Baums speichert. Dieses wird mit einer `while`-Schleife durchlaufen, bis alle Seiten des Baums ausgewertet wurden.

Unabhängig vom Seiten-Typ muss als erstes der Tag-Bereich ausgelesen werden, bevor der Datenbereich ausgewertet werden kann. Nur mit den Offsets und Längenangaben der Einträge kann der Datenbereich korrekt interpretiert werden. Die Anzahl der Tags wird mithilfe des Seiten-Headers bestimmt, dort wird an Offset `0x22` der erste verfügbare Seiten-Tag gespeichert. Der 2 Byte lange Wert wird, wie in Unterabschnitt 3.3.3 vorgestellt, ausgelesen und in der Variable `page_tags` gespeichert. Die in den Tags gespeicherten Offsets und Längenangaben sollen in einem Array abgelegt werden. Dieses wird aus Performance-Gründen auf 500 Tags beschränkt, weshalb eine Überprüfung der Anzahl der vorhandenen Tags stattfinden muss. Übersteigt die Anzahl die definierte Grenze wird eine entsprechende Fehlermeldung in XWF ausgegeben:

```
if (page_tags > 500)
{
```

```
        XWF_OutputMessage(L"ERROR: Seite hat mehr als 500 Eintraege", 0)
            ;
        return NULL;
}
```

Anschließend kann das Tag-Array erstellt und befüllt werden. Der Start-Offset des Tag-Bereichs ist das Ende der Seite, da dieser von hinten noch vorne aufgefüllt wird. Davon ausgehend werden schrittweise 2 Byte gelesen und gespeichert. Die geraden Positionen des Arrays enthalten deshalb immer einen Offset, die ungeraden die zugehörige Längenangabe.

```
uint16_t tag_array[1000] = {};
tag_offset = page_offset + page_size - 2;
for (int i = 0; i < (page_tags * 2); i++)
{
        ptr = &msysobjects_buffer[tag_offset];
        tag_array[i] = *((uint16_t*)ptr);
        tag_offset = tag_offset - 2;
}
```

Die Seiten-Schlüssel-Flags können sich im Schlüssel eines Datensatzes, oder alternativ in den drei höchstwertigen Bits des im Tag-Bereich gespeicherten Offsets befinden. Deshalb folgt im nächsten Schritt eine Überprüfung des Speicherortes. Metz [21] schlägt zur Unterscheidung beider Versionen den Revision-Wert aus dem Datenbank-Header vor. Bei der Umsetzung des Projekts konnte jedoch eine Differenzierung anhand dieses Wertes nicht bestätigt werden, weil beide Varianten bei gleichem Revision-Wert beobachtet wurden. Deswegen wird eine Entscheidung anhand des Vorhandenseins von Flags im zweiten gespeicherten Offset getroffen. Das Ergebnis der Überprüfung wird in der Variable `key_flags` gespeichert.

```
offset = tag_array[2];
if ((offset & 0xE000) == 0)
{
        key_flags = 1; // Flags in Schluessel
}
else
{
        key_flags = 0; // Flags in Tag
}
```

Nun wird unterschieden, ob die auszuwertende Seite eine Blatt- oder Eltern-Seite ist. Eine Blatt-Seite lässt sich anhand des gesetzten Flags `0x02` in den Seiten-Flags, welche an Offset `0x24` gespeichert sind, identifizieren. Das Vorhandensein dieses Flags, lässt sich durch eine UND-Operation überprüfen:

```
if ((page_flags & 2) != 2)
{
        // ist keine Blatt-Seite
}
```

Im nächsten Schritt werden die gespeicherten Einträge der Seite ausgewertet. Dafür werden die im Tag-Array gespeicherten Informationen verwendet. Der erste gespeicherte Tag kann dabei übersprungen werden, da dieser den Header referenziert. Anschließend folgen die Datensätze. Für jeden Datensatz wird der Start-Offset aus dem Seiten-Offset, dem Offset des Datenbereichs und dem Offset des entsprechenden Tags berechnet. Dabei muss beachtet werden, dass im Falle der Speicherung der Seiten-Schlüssel-Flags im Tag, die drei höchstwertigen Bits im entsprechenden Offset für die Berechnung ignoriert werden:

```
key_offset = page_offset + data_offset + (tag_array[j * 2] & 0x1FFF);
```

Jeder Datensatz beginnt mit einem Schlüssel, welcher zunächst mit der Funktion `ddHeaderOffset()` ausgewertet wird. Ihr werden unter anderem beim Aufruf die Seiten-Schlüssel-Flags übergeben, welche den weiteren Aufbau bestimmen. Um eine einheitliche Überprüfung zu ermöglichen, müssen die im Offset gespeicherten Flags zuvor an die gleiche Stelle geschoben werden, wie die im Schlüssel gespeicherten Flags:

```
page_key_flags = (int)tag_array[j * 2] >> 8;
```

Mithilfe der Funktion `ddHeaderOffset()` wird die Sprunggröße, sowie der Start des Datendefinitions-Headers bzw. der folgenden Seiten-Nummer bestimmt. Dieser Offset ist der Rückgabewert der Funktion. Die Funktion ist wie folgt aufgebaut:

```
size_t ddHeaderOffset(int page_key_flags, size_t key_offset, BYTE*
    file_buf)
{
        if ((page_key_flags & 128) != 128) // flags: 0xx
        {
                jump_size = file_buf[key_offset];
                dd_header_offset = key_offset + 2 + jump_size;
        }
        else // flags: 1xx
        {
                ptr = &file_buf[key_offset + 2];
                jump_size = *((uint16_t*)ptr);
                dd_header_offset = key_offset + 4 + jump_size;
        }
        return dd_header_offset;
}
```

Im Falle von internen Seiten-Einträgen kann nun mit dem berechneten Offset die 4 Byte lange Seiten-Nummer ausgelesen und im Array für die weitere Auswertung gespeichert werden. Andernfalls folgt anstelle der Seitennummer der Datendefinitions-Header, welcher im nächsten Schritt ausgewertet werden muss. Dort werden die ID des letzten Fixed Size und Variable Size Datentyps, sowie der relative Offset zu den variablen Spaltenwerten gespeichert. Für die Auswertung wird die Funktion `ddHeader()` aufgerufen. In ihr werden die eben genannten Werte bestimmt, zudem der Offset der Spaltenwerte nach dem Datendefinitions-Header berechnet und in der Variable `record_offset` gespeichert:

```
void ddHeader(BYTE* file_buf, size_t dd_header_offset)
{
        last_fixed_id = file_buf[dd_header_offset];
        last_variable_id = file_buf[dd_header_offset + 1];
        ptr = &file_buf[dd_header_offset + 2];
        relative_variable_offset = *((uint16_t*)ptr);
        variable_offset = dd_header_offset + relative_variable_offset;
        record_offset = dd_header_offset + 4;
}
```

Weiterhin werden nun die relevanten Spaltenwerte ausgelesen und in einem Array gespeichert. Wichtig für die spätere Dateninterpretation sind dabei vor allem folgende Felder:

- FDP-ID
- Katalog-Typ
- ID
- Spaltentyp oder FDP
- Platzbedarf
- Flags

Die Werte werden am entsprechenden Offset (siehe Tabelle 12) ausgelesen und in der nächsten freien Zeile des Datendefinitions-Arrays gespeichert. Diese wird in der Variable `data_definition_count` gespeichert. Außerdem wird das zweidimensionale Array mit einer zusätzlichen Spalte initialisiert, in welcher die Länge des Namen der Datendefinition an späterer Stelle im Programmablauf gespeichert werden kann.

```
INT32 data_definitions[1000][7] = {};
int data_definition_count = 0;

data_definitions[data_definition_count][0] = fdp_id;
data_definitions[data_definition_count][1] = catalog_type;
data_definitions[data_definition_count][2] = id;
data_definitions[data_definition_count][3] = column_type_or_fdp;
data_definitions[data_definition_count][4] = space;
data_definitions[data_definition_count][5] = record_flags;
```

Neben diesen Werten wird außerdem der Name der Definition in einem Namensarray gespeichert. Der Name ist der erste Datentyp variabler Länge. Um die Position und Länge zu bestimmen werden folgende Schritte durchgeführt:

1. Anhand der ID des letzten Variable Size Datentyps die Anzahl der gespeicherten Werte berechnen.

   ```
   int variable_count = 0;
   variable_count = last_variable_id - 0x7f;
   ```

2. Den Offset des ersten Variable Size Datentyps nach dem Array berechnen.

   ```
   variable_offset_data = real_variable_offset + 2 * variable_count;
   ```

3. Die Länge des Namens bestimmen, welcher als erster Wert im Array abgelegt ist und diesen im Datendefintions-Array speichern.

```
ptr = &msysobjects_buffer[variable_offset];
variable_size = *((uint16_t*)ptr);
data_definitions[data_definition_count][6] = variable_size;
```

Anschließend kann am berechneten Offset der Name gelesen und im Namensarray gespeichert werden.

```
char data_definitions_name[1000][256] = {};

for (int k = 0; k < variable_size; k++)
{
        data_definitions_name[data_definition_count][k] =
           msysobjects_buffer[variable_data_offset + k];
}
```

3.3.5 Formatieren der Ausgabe

Für die Ausgabe aller Daten wird ein Puffer definiert und verwendet:

```
wchar_t* buffer = (wchar_t*)calloc(SpeicherZugeordnet, sizeof(wchar_t));
```

Damit der Puffer mit den entsprechenden Informationen befüllt und bei Bedarf erweitert werden kann, wurde die Funktion `Ausgabe()`, welche von Kittan [48] für eine X-Tension für Apple Konfigurationsdateien geschrieben wurde, integriert. Bei jedem Aufruf der Funktion wird überprüft, ob die Anzahl der zu hinzuzufügenden Bytes, den bereits reservierten Speicherplatz überschreiten würden. In diesem Fall wird der reservierte Speicher um den Faktor zwei erweitert. Anschließend werden mit der Funktion `wmemcpy()` die neuen Daten dem Puffer hinzugefügt und die aktuelle Speichernutzung wird aktualisiert. [48, 49]

```
wchar_t* Ausgabe(wchar_t* buffer, const wchar_t* Zeile, wchar_t*
   Ausgabebuf, int zeichencount)
{
        size_t Zeilenlaenge = wcslen(Zeile) + zeichencount;

        while (Speichernutzung + Zeilenlaenge >= SpeicherZugeordnet)
        {
                SpeicherZugeordnet *= 2;
                buffer = (wchar_t*)realloc(buffer, SpeicherZugeordnet *
                   sizeof(wchar_t));
                if ((buffer == NULL))
                {
                        XWF_OutputMessage(L"ERROR: Fehler beim Zuordnen
                           des groesseren Speichers", 0);
                        return NULL;
```

```
                }
        }

        wmemcpy(buffer + Speichernutzung, Ausgabebuf, Zeilenlaenge);
        Speichernutzung += Zeilenlaenge;
        return buffer;
}
```

Um die in der ESEDB-Datei gespeicherten Tabellen im Viewer-Fenster von XWF auszugeben, wurde die Auszeichnungssprache Hypertext Markup Language (HTML) verwendet. Damit der Code mit der entsprechenden internen Viewer-Komponente korrekt dargestellt werden kann, müssen die ersten zwei Byte, welche dem Puffer übergeben werden, ein Byte Order Mark (BOM) sein. [46] Ein BOM ist eine spezielle Bytefolge am Anfang einer Datei, die das Unicode[7]-Zeichen U+FEFF darstellt und dient zur Definition der Byte-Reihenfolge. Das BOM für die UTF-16[8] Zeichenkodierung ist `0xFEFF`. [50] Anschließend kann der Puffer mit den Daten befüllt werden, dabei muss die W3C[9]-Syntax beachtet werden [48].

3.3.6 Auslesen der Tabellen

Nach dem aus Auslesen der Datendefinitionen und der Vorbereitung der Ausgabe der Daten können die Daten, welche in den Tabellen gespeichert sind, ausgelesen werden. Dafür werden alle Datendefinitionen mit einer `for`-Schleife durchlaufen. Ist der Typ einer Definition gleich 1, so wird eine neue Tabelle initiiert und der entsprechende Name ausgegeben. Dazu werden alle im Namensarray abgelegten Zeichen mithilfe der gespeicherten Länge des Namens ausgelesen. Die Variable `i` ist die Laufvariable der `for`-Schleife, womit eine korrekte Zuordnung im Array sichergestellt wird:

```
for (int j = 0; j < data_definitions[i][6]; j++)
{
        zeichen = data_definitions_name[i][j];
        wsprintf(Ausgabebuf, L"%c", zeichen);
        buffer = Ausgabe(buffer, L"", Ausgabebuf, 1);
}
```

Weiterhin werden alle Spaltendefinitionen, die zur entsprechenden Tabelle gehören, in einem Array gespeichert, um einen leichteren Zugriff darauf zu ermöglichen. Die Spalten können anhand der Übereinstimmung der FDP-ID identifiziert werden, welche eine Zugehörigkeit zum gleichen B^+-Baum anzeigt. Der Typ einer Spaltendefinition ist gleich 2. Diese beiden Bedingungen müssen vor der Speicherung im Array überprüft werden:

```
fdp_id = data_definitions[i][0]; // FDP-ID der Tabelle
if ((data_definitions[x][0] == fdp_id) && data_definitions[x][1] == 2)
{
        ...
```

[7] Zeichensatz, der die Zeichen aller Sprachen abbildet [29]

[8] Universal Multiple-Octet Coded Character Set (UCS) Transformation Format-16; eine mögliche Codierung des Unicodes [29]

[9] World Wide Web Consortium: „Konsortium zur Vergabe von Normen im WWW“ [29]

```
}
```

In der `if`-Anweisung werden außerdem die Spaltennamen der Tabelle ausgegeben. Das Vorgehen ist hierbei identisch zur Ausgabe des Tabellen-Namen.

Äquivalent zum Auslesen der Tabelle MSysObjects werden im nächsten Schritt alle Seiten des B^+Baums der aktuellen Tabelle ausgewertet. Zuerst wird die Wurzel-Seite (FDP) der Tabelle bestimmt und in ein Array für alle dem Baum zugehörigen Seiten gespeichert. Dieses wird anschließend mit einer `while`-Schleife durchlaufen. Für jede auszuwertende Seite wird zunächst der Offset berechnet. Weiterhin werden die Anzahl der Seiten-Tags, sowie der Tag-Bereich ausgelesen. Für jeden Eintrag werden die Flags bestimmt und der Schlüssel ausgewertet. Handelt es sich um eine Seite mit internen Seiten-Einträgen, so wird als nächstes die referenzierte Seiten-Nummer bestimmt und dem Array hinzugefügt. Andernfalls wird anstatt der Seitennummer der Datendefinitions-Header ausgewertet. Anschließend folgen die Spaltenwerte.

3.3.6.1 Spalten mit Fixed Size Daten

Die Anzahl der Spalten mit Daten fester Größe kann dem Datendefintions-Header entnommen werden. Für jeden Wert wird mit Hilfe der gespeicherten Datendefintion der Spaltentyp und Platzbedarf bestimmt. Der Offset der ersten Spalte ergibt sich durch das Ende des Datendefinitions-Headers. Für jede weitere Spalte muss die Länge des vorherigen Spaltenwertes addiert werden. Dies wird durch die Variable `sum_fixed_size` realisiert.

```
column_type_or_fdp = data_definitions_table[f][3];
space = data_definitions_table[f][4];
offset_data = record_offset + sum_fixed_size;
```

In einer `switch-case`-Anweisung wird daraufhin der Spaltentyp ausgewertet und die Daten werden entsprechenden ausgegeben. Der Aufbau der Anweisung wird in Unterunterabschnitt 3.3.6.4 genauer erläutert.
Abschließend müssen in diesem Schritt die Variablen `sum_fixed_size` und `printed_columns` aktualisiert werden. Der Parameter `printed_columns` ist ein Zähler, welcher die Anzahl der bereits ausgegeben Spalten speichert und im späteren Programmablauf relevant wird.

```
sum_fixed_size += space;
printed_columns++;
```

3.3.6.2 Spalten mit Variable Size Daten

Für die Spaltenwerte mit variabler Größe wird äquivalent zur Auswertung der MSysObjects zuerst die Anzahl der Spaltenwerte berechnet. Damit kann die Größe des Variable Size Daten-Arrays und damit der Offset der variablen Spaltenwerte bestimmt werden.
Zu beachten ist, dass in einer Tabelle leere Zellen existieren können. Deshalb muss vor der Ausgabe der Variable Size Daten überprüft werden, ob leere Zellen nach den Fixed Size Spalten ausgegeben werden müssen. Um das zu realisieren wird, solange die nächste ID in den Spaltendefinitionen der Tabelle ungleich `0x80` ist, eine leere Zelle ausgegeben:

```
while (data_definitions_table[printed_columns][2] != 0x80)
{
        wsprintf(Ausgabebuf, L"<td></td>");
        buffer = Ausgabe(buffer, L"<td></td>", Ausgabebuf, 0);
        printed_columns++;
}
```

Mithilfe einer `for`-Schleife können nun alle variablen Spaltenwerte interpretiert und ausgegeben werden. Dafür wird zuerst die Größe des Wertes an der entsprechenden Stelle des Variable Size Daten-Arrays bestimmt. Weiterhin muss das höchstwertige Bit ignoriert und die Summe der Größe aller vorherigen Werte abgezogen werden.

```
ptr = &file_buf[real_variable_offset + 2 * v];
variable_size = *((uint16_t*)ptr);
variable_size = (variable_size & 0x7FFF) - sum_variable_size;
```

Ist die resultierende Länge der Daten größer Null, so wird der Spaltentyp bestimmt und dieser äquivalent zu den Fixed Size Daten interpretiert und ausgegeben.
Abschließend müssen der Offset für die nächsten Daten, sowie die Gesamtgröße und die Anzahl der ausgegebenen Spalten aktualisiert werden.

```
variable_offset_data += variable_size;
sum_variable_size += variable_size;
printed_columns++;
```

3.3.6.3 Spalten mit Tagged Daten

Zuletzt werden die getaggten Spaltenwerte interpretiert und ausgegeben. Da zu diesem Datentyp keine Informationen im Datendefinitions-Header hinterlegt sind, muss anhand der Größe des Datensatzes das Vorhandensein geprüft, sowie ggf. der Offset berechnet werden. Hier werden zwei Fälle unterschieden:

1. Es sind Spaltenwerte variabler Größe in dem Datensatz vorhanden.
2. Es sind keine Spaltenwerte variabler Größe in dem Datensatz vorhanden.

Im ersten Fall muss überprüft werden, ob das Ende des Datensatzes dem Ende der variablen Spaltenwerte entspricht. Das Ende des Datensatzes lässt sich mit dem im Tag-Array gespeicherten Offset und der Länge berechnen. Stimmen beide Werte überein, so sind keine getaggten Spaltenwerte vorhanden. In diesem Fall werden lediglich alle verbleibenden leeren Zellen ausgegeben. Stimmen die Werte nicht überein, so wird der Offset der getaggten Daten auf das Ende der variablen Spaltenwerte gesetzt.

```
if (page_offset + data_offset + tag_array[2 * j] + tag_array[2 * j + 1]
   == variable_data_offset)
{
        ...     // leere Zellen ausgeben
```

```
}
else
{
        tagged_offset = variable_data_offset;
}
```

Im zweiten Fall, wenn keine Variable Size Spalten vorhanden sind, wird überprüft, ob das Ende des Datensatzes mit dem im Datendefinitions-Header angegebenen Offset der variablen Spaltenwerte übereinstimmt. Ist dies der Fall, sind keine Spalten mit getaggten Daten vorhanden und es werden, wie im ersten Fall, lediglich alle verbleibenden leeren Zellen ausgegeben. Andernfalls ist der Offset der getaggten Daten, der angegebene Offset der variablen Spaltenwerte.

```
if (page_offset + data_offset + tag_array[2 * j] + tag_array[2 * j + 1]
   == variable_offset)
{
        ...     // leere Spalten ausgeben
}
else
{
        tagged_offset = variable_offset;
}
```

Wurde das Vorhandensein getaggter Spaltenwerte festgestellt, so muss im nächsten Schritt das Array der getaggten Daten, welches vor den eigentlichen Daten gespeichert wird, ausgelesen und interpretiert werden. Jeder Eintrag des Arrays ist 4 Byte lang und besteht aus einer 2 Byte langen ID der Spalte und einem 2 Byte langen Offset. Dieser ist relativ zum Start des Arrays. Zuerst wird der erste Offset ausgelesen und damit die Anzahl getaggter Spaltenwerte berechnet:

```
ptr = &file_buf[tagged_offset + 2];
tagged_data_offset = *((uint16_t*)ptr);
tagged_count = tagged_data_offset / 4;
```

Für jeden Spaltenwert kann nun jeweils die ID und der Offset ausgelesen und in einem Array zwischengespeichert werden.

```
INT16 tagged_data_array[50][2] = {};

for (int t = 0; t < tagged_count; t++)
{
        ptr = &file_buf[tagged_offset + 4 * t];
        tagged_id = *((uint16_t*)ptr);

        ptr = &file_buf[tagged_offset + 2 + 4 * t];
        tagged_data_offset = *((uint16_t*)ptr);

        tagged_data_array[t][0] = tagged_id;
        tagged_data_array[t][1] = tagged_data_offset;
```

```
}
```

Die Länge des Spaltenwertes ergibt sich aus der Subtraktion des Offsets des nächsten getaggten Spaltenwertes bzw. des Ende des Datensatzes und dem Start-Offset des aktuellen Wertes:

```
if (a + 1 < tagged_count)
{
        tagged_data_size = tagged_data_array[a + 1][1] -
           tagged_data_array[a][1];
}
else
{
        tagged_data_size = (page_offset + data_offset + tag_array[2 * j]
            + tag_array[2 * j + 1]) - (tagged_offset + tagged_data_array
           [a][1] + 1);
}
```

Für jede ID muss nun die Zuordnung in den Datendefinitionen stattfinden um den Spaltentyp und die Position der Spalte in der Tabelle zu bestimmen. Dies wird über eine `for`-Schleife realisiert, mit welcher nacheinander alle ID's der Spalten der Tabelle überprüft werden. Gegebenenfalls müssen vor der Ausgabe des Spaltenwertes leere Zellen ausgegeben werden.

```
for (int b = 0; b < column_count; b++) // ID suchen
{
        if (data_definitions_table[b][2] == tagged_id)
        {
                ...
        }
}
```

Äquivalent zu den Fixed und Variable Size Datentypen wird auch hier als nächstes der Datentyp der Spalte bestimmt und dieser mithilfe der `switch-case`-Anweisung entsprechend ausgegeben. Der Offset der Daten ergibt sich jeweils aus der Addition des Start-Offsets der getaggten Spaltenwerte und dem relativen Offset des aktuellen Spaltenwertes. Außerdem muss das erste Byte, die Tagged Daten Flags, übersprungen werden.

```
offset_data = tagged_offset + tagged_data_array[a][1] + 1;
```

Abschließend müssen alle verbleibenden leeren Zellen des aktuellen Datensatzes ausgegeben werden, bevor die Zeile geschlossen und der nächste Eintrag ausgewertet und ausgegeben wird.

3.3.6.4 Auswerten des Spaltentyps

Im Programmablauf muss an mehreren Stellen der Spaltentyp, welcher einen Wert von 0 bis 17 annehmen kann (siehe Tabelle 13), ausgewertet und entsprechend ausgegeben werden. Dies wird jeweils mit einer `switch-case`-Struktur realisiert. Aufgrund des Wiederkehrens der gleichen Aufga-

be, wurde versucht die Anweisung in eine separate Funktion auszulagern. Dieser wurden bei einem Aufruf der konkrete Offset, Spaltentyp und die Länge des entsprechenden Wertes übergeben. Da im Testversuch der X-Tension festgestellt wurde, dass diese Umsetzung zu Problemen führte, wurde sich gegen die Variante entschieden. Der Aufbau der Anweisung ist jedoch für Fixed Size, Variable Size und Tagged Spalten identisch und soll an dieser Stelle kurz vorgestellt werden.

Der Spaltentyp Null ist ungültig und wird ignoriert. Außerdem wird auch der Spaltentyp 13, welcher überholt ist, nicht weiter interpretiert.
Die Spaltentypen 1 und 2 sind beide 8 Bit lang und werden als vorzeichenloser Integer ausgegeben. Die Spaltentypen 3 und 17 stellen 16 Bit Integer Werte dar. Im Falle des Typs 17, wird dieser vorzeichenlos interpretiert. 32 Bit und 64 Bit Integer Werte werden durch die Spaltentypen 4 (32 Bit mit Vorzeichen), 14 (32 Bit ohne Vorzeichen) und 15 (64 Bit mit Vorzeichen) dargestellt und entsprechend ausgegeben. Auch der Spaltentyp 5 ist ein 8 Byte langer Integer. Im Unterschied zu Typ 15 stellt dieser jedoch eine Währung dar und wird aber auf gleiche Weise formatiert ausgegeben. Folgendes Listing zeigt beispielsweise die Verarbeitung des Spaltentyps 3:

```
switch (column_type)
{
        ...
        case 3:
                ptr = &file_buf[offset_data];
                data_16bit = *((uint16_t*)ptr);
                wsprintf(Ausgabebuf, L"%6d", data_16bit);
                buffer = Ausgabe(buffer, L"", Ausgabebuf, 6);
                break;
        ...
}
```

Im Falle der Spaltentypen 6 und 7 werden Gleitkommazahlen gespeichert. Spaltentyp 6 ist dabei 32 Bit lang und wird als Float interpretiert. Typ 7 ist 64 Bit groß und repräsentiert ein Double. Der Wert wird in beiden Fällen als Integer ausgelesen und anschließend mit der Funktion `memcpy()` in den Speicherbereich der Float- bzw. Double-Variable kopiert und entsprechend interpretiert [49]. Anschließend würde die formatierte Ausgabe erfolgen, welche jedoch zu Darstellungsproblemen führte. Deshalb wird an dieser Stelle, um eine vollständige Ausgabe der Tabelle zu ermöglichen, der Wert in hexadezimaler Darstellung ausgegeben:

```
switch (column_type)
{
        ...
        case 6:
                ptr = &file_buf[offset_data];
                data_32bit = *((uint32_t*)ptr);
                memcpy(&data_float, &data_32bit, sizeof(float));
                wsprintf(Ausgabebuf, L"%08X", data_float);
                buffer = Ausgabe(buffer, L"", Ausgabebuf, 8);
                break;
        ...
```

```
}
```

Der Spaltentyp 8 ist 64 Bit lang und repräsentiert einen Zeitstempel, welcher als FILETIME gespeichert wird. In diesem Format stellt der Wert die Anzahl an 100-Nanosekunden-Intervallen seit dem 01.01.1601 00:00:00 Uhr dar [51]. Für die Umwandlung in einen lesbaren Zeitstempel liegt folgende Idee zugrunde: Zuerst werden die Nanosekunden in Sekunden umgerechnet. Diese können dann in einen UNIX-Zeitstempel[10] umgewandelt werden. Mithilfe der Funktion `strftime()` kann dieser anschließend ausgegeben werden [53].

```
uint64_t filetime_seconds = (filetime / 10000000);
time_t unix_timestamp = (time_t) filetime_seconds - 11644473600;
struct tm t = *localtime(&unix_timestamp);
strftime(buf, sizeof(buf), "%Y-%m-%d %H:%M:%S %Z", &t);
printf("%s\n", buf);
```

Leider führte die Ausführung diese Konzepts zu Problemen in XWF, weshalb der Zeitstempel aktuell als Hexadezimalzahl ausgegeben wird. Diese kann dann zum Beispiel mithilfe eines Online-Tools[11] in einen lesbaren Zeitstempel umgewandelt werden.

Die Spaltentypen 9 und 11 enthalten Binärdaten, welche unterschiedlich lang sein können. Die Verarbeitung und Ausgabe ist in beiden Fällen gleich: Für die Gesamtgröße wird jedes Byte gelesen und als hexadezimaler Wert ausgegeben:

```
switch (column_type)
{
        ...
        case 9:
                for (int j = 0; j < size; j++)
                {
                        data_8bit = file_buf[offset_data + j];
                        wsprintf(Ausgabebuf, L"%02X", data_8bit);
                        buffer = Ausgabe(buffer, L"", Ausgabebuf, 2);
                }
                break;
        ...
}
```

Enthält eine Spalte Text-Daten, so hat sie den Typ 10 oder 12. Diese unterscheiden sich äquivalent zu den Binärdaten lediglich in der maximalen Länge der gespeicherten Daten. Auch hier werden die Daten in beiden Fällen gleich verarbeitet: Jedes Byte wird gelesen und als Zeichen ausgegeben.

```
switch (column_type)
{
        ...
```

[10] Sekunden seit dem 01.01.1970 00:00 Uhr [52]
[11] Beispielsweise `https://www.epochconverter.com/ldap`

```
        case 10:
                for (int j = 0; j < size; j++)
                {
                        zeichen = file_buf[offset_data + j];
                        wsprintf(Ausgabebuf, L"%c", zeichen);
                        buffer = Ausgabe(buffer, L"", Ausgabebuf, 1);
                }
                break;
        ...
}
```

Der Spaltentyp 16 ist 128 Bit lang und repräsentiert eine Globally Unique Identifier (GUID). Eine GUID wird normalerweise im 8-4-4-4-12 Format dargestellt. Die Zeichen sind hexadezimale Werte. [54] Um die gespeicherte GUID ihrem Format entsprechend auszugeben werden byteweise nacheinander 4 Byte, 3 mal 2 Byte und abschließend 6 Byte gelesen und als Hexadezimalzahlen ausgegeben. Dazwischen wird jeweils ein Bindestrich ausgegeben, welcher die Gruppen von Bytes trennt. Da die Werte in Little Endian gespeichert sind, wird das niederwertigste Byte zuerst ausgelesen.

```
switch (column_type)
{
        ...
        case 16:
                for (int j = 1; j <= 4; j++)
                {
                        data_8bit = file_buf[offset_data + 16 - j];
                        wsprintf(Ausgabebuf, L"%02X", data_8bit);
                        buffer = Ausgabe(buffer, L"", Ausgabebuf, 2);
                }
                wsprintf(Ausgabebuf, L"-");
                buffer = Ausgabe(buffer, L"-", Ausgabebuf, 0);
                for (int j = 1; j <= 3; j++)
                {
                        for (int k = 1; k <= 2; k++)
                        {
                                data_8bit = file_buf[offset_data + 12 -
                                    x];
                                wsprintf(Ausgabebuf, L"%02X", data_8bit)
                                    ;
                                buffer = Ausgabe(buffer, L"", Ausgabebuf
                                    , 2);
                                x++;
                        }
                        wsprintf(Ausgabebuf, L"-");
                        buffer = Ausgabe(buffer, L"-", Ausgabebuf, 0);
                }
                for (int j = 1; j <= 6; j++)
                {
```

```
                    data_8bit = file_buf[offset_data + 6 - j];
                    wsprintf(Ausgabebuf, L"%02X", data_8bit);
                    buffer = Ausgabe(buffer, L"", Ausgabebuf, 2);
                }
                break;
        ...
}
```

Abschließend werden mit der `default`-Option alle sonstigen Werte abgefangen, welche lediglich in hexadezimaler Darstellung ausgegeben werden.

4 Ergebnisse und Diskussion

Nach der Vorstellung des Programmkonzepts werden in diesem Kapitel die zur Evaluation verwendeten Testdaten aufgelistet. Anschließend werden die Ergebnisse anhand von Beispiel-Darstellungen in XWF aufgezeigt und kritisch diskutiert.

4.1 Testdaten

Nach dem Fertigstellen der X-Tension kann diese in XWF eingebunden werden und ESEDB-Dateien können im Viewer-Fenster betrachtet werden. Dort werden alle Tabellen, die in einer Datenbank enthalten sind, ausgegeben. Um dies zu testen, wurden verschiedene ESE-Datenbanken verwendet. Diese wurden in einem zum Test erstellten Fall in XWF hinzugefügt und anschließend zur Evaluation der Ergebnisse im Vorschaufenster betrachtet. Tabelle 6 gibt eine Übersicht aller verwendeten Dateien. Diese stammen aus verschiedenen Anwendungen und Umgebungen. Die vollständigen Pfadangaben der Dateien können der Tabelle 16 in Anhang A entnommen werden.

Tabelle 6: Testdaten für die Evaluation der X-Tension

Dateiname	Betriebssystem-Version	Programm
HCdata.edb	Windows 7	Windows Health Check
ntds.dit	Windows Server 2003 SP1	Active Directory
Windows.edb	Windows 10	Windows Suche
WebCacheV01.dat	Windows 10	Internet Explorer
meta.edb	Windows 10	Windows Store
IndexedDB.edb	Windows 10	Microsoft Edge
store.vol	Windows 10	Windows Mail

4.2 Ergebnisse

Die nachfolgenden Abbildungen zeigen das Ergebnis der Ausgabe der Tabellen im Viewer-Fenster von ESEDB-Dateien. An ausgewählten Beispielen soll auch die Darstellung der verschiedenen Spaltentypen gezeigt werden.
Abbildung 16 zeigt einen Ausschnitt der Ausgabe der Tabelle MSysObjects, welche die Datendefinitionen aller Tabellen enthält.

MSysObjects

ObjidTable	Type	Id	ColtypOrPgnoFDP	SpaceUsage	Flags	PagesOrLocale	RootFlag	RecordOffset	LCMapFlags	KeyMost	LVChunkMax	Name
2	1	2	4	80	-1073741824	20	255					MSysObjects
2	2	1	4	4	1	1252	42	4				ObjidTable
2	2	2	3	2	1	1252	42	4				Type
2	2	3	4	4	1	1252	42	4				Id
2	2	4	4	4	1	1252	42	4				ColtypOrPgnoFDP
2	2	5	4	4	1	1252	42	4				SpaceUsage
2	2	6	4	4	1	1252	42	4				Flags
2	2	7	4	4	1	1252	42	4				PagesOrLocale
2	2	8	1	1	0	1252	42	4				RootFlag
2	2	9	3	2	0	1252	42	4				RecordOffset
2	2	10	4	4	0	1252	42	4				LCMapFlags
2	2	11	17	2	0	1252	42	4				KeyMost
2	2	12	4	4	0	1252	42	4				LVChunkMax
2	2	128	10	255	1	1252						Name

Abbildung 16: Darstellung der Tabelle MSysObjects in XWF

Der dazugehörige generierte HTML-Code im Rohformat wird in Abbildung 17 dargestellt.

```
<HTML><HEAD><style>table, th, td {border: 1px solid black; border-collapse: collapse;}</style></HEAD><BODY><center><b>ESEDB File: IndexedDB.edb</b></center><br><table style="width:100%"><caption
style="text-
align:left">   MSysObjects</caption><tr><th>ObjidTable</th><th>Type</th><th>Id</th><th>ColtypOrPgnoFDP</th><th>SpaceUsage</th><th>Flags</th><th>PagesOrLocale</th><th>RootFlag</th><th>RecordOf
fset</th><th>LCMapFlags</th><th>KeyMost</th><th>LVChunkMax</th><th>Name</th><th>Stats</th><th>TemplateTable</th><th>DefaultValue</th><th>KeyFldIDs</th><th>VarSegMac</th><th>ConditionalCol
umns</th><th>TupleLimits</th><th>Version</th><th>SortID</th><th>CallbackData</th><th>CallbackDependencies</th><th>SeparateLV</th><th>SpaceHints</th><th>SpaceDeferredLVHints</th><th>LocaleNam
e</th></tr><tr><td>      2</td><td>   1</td><td>      2</td><td>      4</td><td>      80</td><td>-1073741824</td><td>
20</td><td>255</td><td></td><td></td><td></td><td></td><td>MSysObjects</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></
td><td></td><td></td></tr><tr><td>      2</td><td>   2</td><td>      1</td><td>      4</td><td>      4</td><td>      1</td><td>    1252</td><td> 42</td><td>
4</td><td></td><td></td><td></td><td>ObjidTable</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr
><tr><td>      2</td><td>   2</td><td>      2</td><td>      3</td><td>      2</td><td>      1</td><td>    1252</td><td> 42</td><td>
4</td><td></td><td></td><td></td><td>Type</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr><tr>
<td>      2</td><td>   2</td><td>      3</td><td>      4</td><td>      4</td><td>      1</td><td>    1252</td><td> 42</td><td>
4</td><td></td><td></td><td></td><td>Id</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr><tr><td
>      2</td><td>   2</td><td>      4</td><td>      4</td><td>      4</td><td>      1</td><td>    1252</td><td> 42</td><td>
4</td><td></td><td></td><td></td><td>ColtypOrPgnoFDP</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></
td></tr><tr><td>      2</td><td>   2</td><td>      5</td><td>      4</td><td>      4</td><td>      1</td><td>    1252</td><td> 42</td><td>
4</td><td></td><td></td><td></td><td>SpaceUsage</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></
tr><tr><td>      2</td><td>   2</td><td>      6</td><td>      4</td><td>      4</td><td>      1</td><td>    1252</td><td> 42</td><td>
4</td><td></td><td></td><td></td><td>Flags</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr><tr>
<td>      2</td><td>   2</td><td>      7</td><td>      4</td><td>      4</td><td>      1</td><td>    1252</td><td> 42</td><td>
4</td><td></td><td></td><td></td><td>PagesOrLocale</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td
></tr><tr><td>      2</td><td>   2</td><td>      8</td><td>      1</td><td>      1</td><td>      0</td><td>    1252</td><td> 42</td><td>
4</td><td></td><td></td><td></td><td>RootFlag</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr><
tr><td>      2</td><td>   2</td><td>      9</td><td>      3</td><td>      2</td><td>      0</td><td>    1252</td><td> 42</td><td>
4</td><td></td><td></td><td></td><td>RecordOffset</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><
/tr><tr><td>      2</td><td>   2</td><td>      10</td><td>      4</td><td>      4</td><td>      0</td><td>    1252</td><td> 42</td><td>
4</td><td></td><td></td><td></td><td>LCMapFlags</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></
tr><tr><td>      2</td><td>   2</td><td>      11</td><td>      17</td><td>      2</td><td>      0</td><td>    1252</td><td> 42</td><td>
```

Abbildung 17: Ausschnitt des generierten HTML-Codes für die Erstellung der Tabelle MSysObjects in XWF

Die formatierte Ausgabe einer GUID wird in Abbildung 18 gezeigt.

schema

schemaId
D036B887-6C9C-68B3-45EA-4FB868015B93

Abbildung 18: Ausgabe einer GUID

Abbildung 19 zeigt die Darstellung von Textdaten und binären Daten.

collectionIdMapping

collectionIdHash	replicaIdHash	collectionId
132312C5FF6A24406DAF27E8616098343C7C162B	2D38E85256EA83FCF210D90B2637732F97FCBE0D	windows-deviceinterest
1A4B2309927C24EEAC50BC5784607303A4D05E8F	D8B353C1875DD468964EBB3816FD960F5A950F75	windows-spellingdictionary
1A71FB28678677B701AD843DA429B1B139F38EA0	3387D578109E05AD45C69372D31CD676D0A01D86	windows-rsimechs
1B2E6C4DB507110AE7CACC6CEA51B3AD6C9B4CE4	4E8D0A0B1296F21721C9316355D78AE9E8CD508A	browsersettings-tabbedbrowsing-internet-explorer
1D8097F22535754E16FB73579FF79655E06DE03D	2B47F199F7BD3D55339E08A616FA3FF92AB5C56F	browsersettings-wininet-internet-explorer
23AEEF4FD01C0A3C78CC6D222153D4CE54416CCE	4E4379323EDD12A2FFF046C2816963507355A9C1	windows-narrator
2444CD1DA81EB605F6E5933D249CBA283298A326	1FB1F3A10CA50DE927C4F704CA1DAF366CA44DCF	browsersettings-formsuggest-internet-explorer
2CF8E5C6D8B3E5F2BCE086963BFDBEB0C48B0629	5AB4647FF737FF1A37E3FE64B81F57AD41EC1F2C	browsersettings-tabroaming-internet-explorer
3125930ABFF2646FADAE10181C7EFA7D99B49514	AA359314373CEF9B8CA7E66C1FFE73F59021BAAB	packagestate-microsoft.microsoftedge_8wekyb3d8bbwe-0
37C97560ADE30891771A6042C432E546188A1989	B27B4B3969FCEF9B7A03D08445BFF37509EBC6A1	packagestate-microsoft.aad.brokerplugin_cw5n1h2txyewy-0
398778C8E716591794B36B160E95758ED3B3F916	C08B247DEE00B390FE6FFCF4C61075F002D1D664	browsersettings-flipahead-internet-explorer
3B45D0BCBAAD0FCD5128EC15507B26A0552061F7	68E933A3E5268D6B9E5D0221C1E445977028E615	browsersettings-thirdpartycookies-internet-explorer
4AFDF6A61851C18E6B03FAF83C78EBE79FC898F8	7A25C5FD21A964C8EBBFA293AC3AC4F7561C8FAA	packagestate-ad2f1837.hpsystemeventutility_v10z8vjag6ke6-0
52325C532911A0301A613AD71578FAB8496FE2ED	2967E3CA0A5AEBCE4CDC299398B595DBDF94ACE8	packagestate-microsoft.windows.cortana_cw5n1h2txyewy-0

Abbildung 19: Darstellung von Text- und Binärdaten

Die aktuelle Ausgabe von Zeitstempeln und Gleitkommazahlen im hexadezimalen Format wird in Abbildung 20 dargestellt.

T-2

autoIncObjectCount	I-1-Date	I-1-Number	I-1-VariantType	dataBlob	I-1-String
25	2A2A2A2A2A2A2A2A	0000000000000000	4	19000000	6100640075006C0074007C00640065007C00640065007C006400680070007C006500640067006
32	2A2A2A2A2A2A2A2A	0000000000000000	4	20000000	6100640075006C0074007C00640065007C00640065007C006E00740070007C006500640067006

T-4

autoIncObjectCount	I-2-Date	I-2-Number	I-2-VariantType	dataBlob	I-2-String
1	2A2A2A2A2A2A2A2A	0000000000000000	4	01000000	66006F006C006C006F007700650064007400 6F007000690063007300 7C006100640075006C007

T-6

autoIncObjectCount	I-3-Date	I-3-Number	I-3-VariantType	dataBlob	I-3-String
16	2A2A2A2A2A2A2A2A	0000000000000000	4	11000000	640065002D00640065007C00730070006F00720074007300

Abbildung 20: Darstellung von Zeitstempeln und Gleitkommazahlen

Weiterhin soll an dieser Stelle eine Fehlermeldung erwähnt werden, welche zum Beispiel bei der Datei `ntds.dit` auftritt. In der Logdatei von XWF `error.log` ist dazu lediglich folgende Meldung zu lesen:
„`Eine Ausnahmesituation vom Typ 216 (page protection fault, high or unknown impact, with an X-Tension in control) ist bei Offset 7FFD0D1583B4 aufgetreten als ich [bitte ergänzen]...`"
Außerdem existiert aktuell noch ein Fehler, der die Software zum Abstürzen bringt, wenn versucht wird die Datei `Windows.edb` zu interpretieren.

4.3 Diskussion

In den vorherigen Kapiteln wurden die Grundlagen für das Erstellen einer X-Tension, sowie der Aufbau der Struktur einer ESEDB-Datei gelegt. Weiterhin wurde ein Programmkonzept vorgestellt, mit welchem es möglich ist, ESE Datenbanken in XWF darzustellen.

Für die Implementierung einer X-Tension ist eine ausführliche Auseinandersetzung mit der X-Tensions API unerlässlich. Dies ist wegen der kurz gehaltenen Dokumentation sehr zeitaufwändig. Weiterhin ist auch das Debugging des Programmcodes aufgrund der API-Funktionen kompliziert. Zur Fehlersuche ist es deswegen notwendig, im Code Informationen der Ausgabe hinzuzufügen, anschließend die DLL zu erstellen und diese in XWF einzubinden. An dieser Stelle könnte für die weitere Entwicklung die Möglichkeit in Betracht gezogen werden, ein eigenes Logfile anzulegen.
Außerdem konnte für die beobachtete Fehlermeldung, welche in Abschnitt 4.2 erwähnt wurde, keine Dokumentation gefunden werden. Dies stellte eine zusätzliche Herausforderung für die Fehlersuche dar und konnte abschließend nicht geklärt werden.

Neben der X-Tensions API musste sich auch intensiv mit dem Aufbau einer ESEDB-Datei beschäftigt werden. Diese hat eine komplexe Struktur, welche aufgrund der verschiedenen Datentypen sehr variabel sein kann. Auch an dieser Stelle fehlte eine ausführliche Dokumentation um den Prozess zu beschleunigen. Besonders hilfreich war vor allem die von Metz [21] erstellte Dokumentation. Jedoch zeigte sich im Laufe der Entwicklung, dass diese teilweise unvollständig oder nicht aktuell ist.

Das Ergebnis dieser Arbeit ist eine funktionsfähige X-Tension, welche die Grundlage für die Darstellung von ESEDB-Dateien in XWF bildet. Wie bereits in Abschnitt 3.3 erwähnt, existieren aktuell Einschränkungen in der Verarbeitung und Darstellung, welche aufgrund der eingeschränkten Zeitspanne nicht gelöst werden konnten. Deswegen wird hier auch auf die Möglichkeit der Verwendung anderer Programme für den Zugriff auf ESE Datenbanken, wie beispielsweise ESEDatabaseView[12] von Nirsoft, verwiesen.

[12] `https://www.nirsoft.net/utils/ese_database_view.html`

5 Zusammenfassung

Das Ziel der vorliegenden Arbeit war, eine funktionsfähige X-Tension für die Untersuchung von ESEDB-Dateien in XWF zu erstellen. Dafür wurde sich zuerst mit den Möglichkeiten der Einbindung einer Erweiterung in die Software beschäftigt. Weiterhin wurde sich mit der API auseinandergesetzt. Somit konnte schließlich eine Anleitung für die Erstellung einer DLL erarbeitet und umgesetzt werden.
Weiterhin wurde die Struktur einer ESE Datenbank analysiert und umfassend erläutert. Ein Verständnis des Aufbaus der Datei ist zwingend notwendig um die in der Datenbank gespeichert Daten zu extrahieren.

Nach der theoretischen Auseinandersetzung wurde ein Programmkonzept für die Implementierung der X-Tension erarbeitet. Die Programmierung der Erweiterung nahm, aufgrund der in Abschnitt 4.3 eingegangenen Probleme und Schwierigkeiten am meisten Zeit in Anspruch. Das umgesetzte Programmkonzept wurde außerdem an ausgewählten Code-Beispielen vorgestellt und erklärt.

Im Ergebnis ist eine lauffähige X-Tension für XWF entstanden. Diese kann alle in der Datenbank enthaltenen Tabellen extrahieren und die meisten Spaltentypen korrekt interpretieren. Sowohl Fixed Size, Variable Size und Tagged Spaltenwerte können festgestellt und ausgegeben werden.
Bei einigen Dateien kommt es aktuell zu einer Fehlermeldung und eine Darstellung des Dateiinhalts ist nicht möglich. Aufgrund der fehlenden Dokumentation konnte dieser Fehler abschließend nicht gelöst werden.

6 Ausblick

Abschließend werden in diesem Kapitel der Arbeit mögliche Verbesserungen der X-Tension vorgeschlagen.
Zuerst müssen dabei die aktuellen Einschränkungen der Erweiterungen erwähnt werden. So wurde beispielsweise die maximale Anzahl der Tags auf einer Seite auf 500 beschränkt. Diese und weitere Beschränkungen müssten für eine Verbesserung aufgehoben werden.
Auch die aktuelle Fehlermeldung muss weiter untersucht werden, um eine Darstellung aller ESEDB-Dateien zu ermöglichen.

Weiterhin bietet auch der komplexe Aufbau einer ESEDB-Datei viele Möglichkeiten das Programm weiter auszubauen: Bei der Auswertung der Tagged Spaltenwerte wird aktuell das erste Byte übersprungen. In diesem Byte werden Flags gespeichert, welche ausgewertet werden können. Diese geben unter anderem an, ob der gespeicherte Spaltenwert komprimiert ist, in einem LV gespeichert ist oder ein Multi Value enthält.
Bei einer Komprimierung wird zwischen der 7-Bit-Komprimierung und der LZXPRESS-Komprimierung unterschieden. Für beide Varianten kann eine Dekomprimierung implementiert werden um eine Interpretation der gespeicherten Daten zu ermöglichen.
Werden die Daten in einem LV gespeichert, so müssen in dem Zusammenhang auch die LV Bäume ausgewertet werden. Diese haben den Katalog-Typ 4 und werden bei der aktuellen Auswertung der Tabellen nicht beachtet. Jedoch könnten hier weitere wichtige Daten gespeichert sein. Zudem wird auch die Interpretation von Multi Values noch nicht unterstützt.

ESE Datenbanken werden für die Datenspeicherung vieler verschiedener Anwendungen verwendet. Die Tabellen und Spalten sind dabei anwendungsspezifisch nutzbar. Dies bietet die Möglichkeit einer anwendungsspezifischen Interpretation der Daten, welche aber auch eine Auseinandersetzung mit dem spezifischen Aufbau der in der entsprechenden Anwendung verwendeten Datei und den darin vorhanden Tabellen und Spaltenwerten erfordert.

Eine Umsetzung der oben genannten Erweiterungsmöglichkeiten hat das Potenzial weitere forensisch relevante Informationen aus einer ESEDB-Datei zu extrahieren und interpretieren zu können.

Anhang A: Tabellen

Tabelle 7: Aufbau des Datenbank-Headers [21]

Offset	Länge	Beschreibung
0x00	4	Prüfsumme: XOR über 32-Bit Little-Endian Werte im Header beginnend am Offset 8 bis mindestens Offset 668, aber wahrscheinlich Seitengröße
0x04	4	Signatur: 0xEFCDAB89
0x08	4	Datei-Format Version
0x0C	4	Dateityp
0x10	6	Datenbank-Zeit
0x18	28	Datenbank-Signatur
0x34	4	Datenbank-Zustand
0x38	8	Consistent Position: Position, welche genutzt wurde als die Datenbank zuletzt den Zustand Clean Shutdown erreicht hat, wenn Datenbank im Zustand Dirty Shutdown ist
0x40	8	Consistent Datum und Zeit: Zeitpunkt, wann Datenbank zuletzt den Zustand Clean Shutdown erreicht hat, wenn Datenbank im Zustand Dirty Shutdown ist
0x48	8	Attach Datum und Zeit: Zeitpunkt, wann Datenbank zuletzt verbunden wurde
0x50	8	Attach Position: Log Posititon, welche genutzt wurde als Datenbank zuletzt verbunden wurde
0x58	8	Detach Datum und Zeit: Zeitpunkt, wann Datenbank zuletzt getrennt wurde
0x60	8	Detach Position: Posititon, welche genutzt wurde als Datenbank zuletzt getrennt wurde
0x68	4	Unbekannt
0x6C	28	Log-Signatur
0x88	24	Vorheriges vollständiges Backup
0xA0	24	Vorheriges inkrementelles Backup
0xB8	24	Aktuelles vollständiges Backup
0xD0	4	Shadowing
0xD4	4	Letzte Objekt-ID
0xD8	4	Windows NT Major Version
0xDC	4	Windows NT Minor Version
0xE0	4	Windows NT Build Nummer
0xE4	4	Windows NT Service Pack Nummer
0xE8	4	Datei-Format Revision
0xEC	4	Seitengröße in Bytes
0xF0	4	Repair Count
0xF4	8	Repair Datum und Zeit
0xFC	28	Unbekannt
0x118	8	Scrub Datenbank Zeit
0x120	8	Scrub Datum und Zeit
0x128	8	Erforderliche Log-Datei
0x130	4	Upgrade Exchange 5.5 Format
0x134	4	Upgrade freie Seiten

Tabelle 7: Aufbau des Datenbank-Headers (Fortsetzung)

Offset	Länge	Beschreibung
0x138	4	Upgrade Space-Seiten
0x13C	24	Aktuelles Shadow Copy Backup
0x154	4	Erstellungs-Datei-Format Version
0x158	4	Erstellungs-Datei-Format Revision
0x15C	16	Unbekannt
0x16C	4	Old Repair Count
0x170	4	ECC Fix Success Count
0x174	8	Last ECC Fix Success Datum und Zeit
0x17C	4	Old ECC Fix Success Count
0x180	4	ECC Fix Error Count
0x184	8	Last ECC Fix Error Datum und Zeit
0x18C	4	Old ECC Fix Error Count
0x190	4	Bad Prüfsummen Error Count
0x194	8	Letze Bad Prüfsummen Error Datum und Zeit
0x19C	4	Old Bad Prüfsummen Error Count
0x1A0	4	Committed Log: niederen 32-Bit Werte
0x1A4	24	Vorheriges (Shadow) Copy Backup
0x1BC	24	Vorheriges differentielles Backup
0x1D4	40	Unbekannt
0x1FC	4	NLS Major Version
0x200	4	NLS Minor Version
0x204	148	Unbekannt
0x298	4	Unbekannte Flags

Tabelle 8: Aufbau des Feldes Datenbank-Signatur [21]

Offset	Länge	Beschreibung
0x00	4	Zufällige Nummer
0x04	8	Erstellungsdatum und -zeit
0x0C	16	NETBIOS Computer Name: ASCII Zeichnenkette, abgeschlossen durch End-of-String Zeichen, ungenutzte Bytes werden mit 0 aufgefüllt

Tabelle 9: Aufbau des Seiten-Headers [21]

Offset	Länge	Beschreibung
vor Exchange 2003 SP1 und Windows Vista		
0x00	4	XOR Prüfsumme
0x04	4	Seitennummer
Exchange 2003 SP1 und Windows Vista und später (Version 0x620, Revision 0x0B, New Record Format Flag muss gesetzt sein)		
0x00	4	XOR Prüfsumme
0x04	4	ECC Prüfsumme
Windows 7 und später (Version 0x620, Revision 0x11)		
0x00	8	Prüfsumme
Allgemein		
0x08	8	Zuletzt geändert
0x10	4	Vorherige Seitennummer
0x14	4	Nächste Seitennummer
0x18	4	FDP Objekt-ID: ID des B^+-Baums zu dem die Seite gehört
0x1C	2	Verfügbare Daten-Größe: verfügbare Anzahl Bytes in Seite
0x1E	2	Verfügbare freie Daten-Größe: Anzahl freie Bytes in der Seite
0x20	2	Erster verfügbarer Daten-Offset: relativer Offset vom Ende des Seiten-Headers
0x22	2	Erster verfügbarer Seiten-Tag
0x24	4	Seiten-Flags
Erweiterter Seiten-Header Windows 7 und später (Version 0x620, Revision 0x11)		
0x28	8	Erweiterte Prüfsumme 1
0x30	8	Erweiterte Prüfsumme 2
0x38	8	Erweiterte Prüfsumme 3
0x40	8	Seitennummer
0x48	8	Unbekannt

Tabelle 10: Seiten-Schlüssel-Flags [21]

Wert	Bezeichnung	Beschreibung
0x01	v	Unbekannt: Werte beinhalten möglicherweise Variable Size Datentypen
0x02	d	Gelöscht: Wert wird nicht mehr verwendet
0x04	c	Common Key: Werte enthalten eine gemeinsame Schlüsselgröße

Tabelle 11: Aufbau des Datendefinitions-Headers [21]

Offset	Länge	Beschreibung
0x00	1	Letzter Fixed Size Datentyp
0x01	1	Letzter Variable Size Datentyp
0x02	2	Offset zu den Variable Size Datentypen, Offset ist relativ zum Start des Datendefinitions-Header

Tabelle 12: Datendefinition der Tabelle MSysObjects [21]

Offset	Länge	Beschreibung
Fixed Size Datentyp-Definitionen		
0x00	4	FDP Objekt-ID
0x04	2	Katalog-Typ
0x06	4	ID
wenn Datendefintions-Typ = 0x02 (Spalte)		
0x0A	4	Spalten-Typ
andere Datendefinitions-Typen		
0x0A	4	FDP-Nummer
wenn Datendefinitions-Typ = 0x01 (Tabelle)		
0x0E	4	Platzbedarf: Anzahl der von der Tabelle verwendeten Seiten
0x12	4	Flags
0x16	4	Initiale Anzahl von Seiten
wenn Datendefinitions-Typ = 0x02 (Spalte)		
0x0E	4	Platzbedarf: Anzahl Bytes die von Spalte belegt werden
0x12	4	Flags
0x16	4	Codeseite
wenn Datendefinitions-Typ = 0x03 (Index)		
0x0E	4	Platzbedarf: Anzahl Seiten die von Index verwendet werden
0x12	4	Flags
0x16	4	Locale Identifier (LCID)
wenn Datendefinitions-Typ = 0x04 (LV)		
0x0E	4	Platzbedarf: Anzahl Seiten die von LV verwendet werden
0x12	4	Flags
0x16	4	Initiale Anzahl von Seiten
alle Datendefinitionen		
0x1A	1	Root Flag
0x1B	2	Record Offset
0x1D	4	LC Map Flags
0x21	2	KeyMost
0x23	4	LVChunkMax

Tabelle 13: Spaltentypen [21]

Wert	Beschreibung
0	Ungültiger Spaltentyp
1	Boolean Spaltentyp, true (nicht 0) oder false (0)
2	8 Bit Integer ohne Vorzeichen
3	16 Bit Integer mit Vorzeichen
4	32 Bit Integer mit Vorzeichen
5	Währung: 8 Byte Integer mit Vorzeichen
6	Gleitkommazahl: Float (32 Bit)
7	Gleitkommazahl: Double (64 Bit)
8	Datum und Uhrzeit (64 Bit), werden als FILETIME gespeichert
9	Binäre Daten, bis zu 255 Bytes groß
10	Text (Extended ASCII oder Unicode), bis 255 ASCII-Zeichen oder 127 Unicode Zeichen
11	Große Binärdaten, bis zu 2.147.483.647 Bytes groß
12	Langer Text (Extended ASCII oder Unicode), bis zu 2.147.483.647 ASCII-Zeichen oder 1.073.741.823 Unicode-Zeichen
in Windows XP eingeführte Werte	
13	Super LV: Spaltentyp ist überholt/veraltet
in Windows Vista eingeführte Werte	
14	32 Bit Integer ohne Vorzeichen
15	64 Bit Integer mit Vorzeichen
16	GUID (128 Bit)
17	16 Bit Integer ohne Vorzeichen

Tabelle 14: Aufbau eines Eintrags im Variable Size Datentyp Array [21]

Offset	Länge	Beschreibung
0x00	2	Länge des Variable Size Datentyps; MSB zeigt an, ob Datentyp leer ist; Größe der vorherigen Variable Size Datentypen muss von der aktuellen Größe subtrahiert werden

Tabelle 15: Aufbau eines Eintrags im Tagged Datentyp Array [21]

Offset	Länge	Beschreibung
0x00	2	Tagged Datentyp ID
0x02	2	Offset der Tagged Daten, ist relativ zum Start des Tagged Datentyp Arrays

Tabelle 16: Pfadangaben zu den Testdaten

Dateiname	Pfad
HCdata.edb	%SYSTEMDRIVE%\Windows\pchealth\helpctr\Database
ntds.dit	%SYSTEMDRIVE%\Windows\NTDS
Windows.edb	%SYSTEMDRIVE%\ProgramData\Microsoft\Search\Data\Applications\Windows
WebCacheV01.dat	%LOCALAPPDATA%\Microsoft\Windows\WebCache
meta.edb	%LOCALAPPDATA%\Microsoft\Windows\SettingSync\remotemetastore\v1
IndexedDB.edb	%LOCALAPPDATA%\Packages\Microsoft.MicrosoftEdge_-8wekyb3d8bbwe\AppData\User\Default\IndexedDB
store.vol	%LOCALAPPDATA%\Comms\UnistoreDB

Literaturverzeichnis

[1] Bundesministerium für Wirtschaft und Klimaschutz, *Auf einen Blick*, 2020.

[2] H. Chivers, „Navigating the Windows Mail database“, *Digital Investigation*, Jg. 26, S. 92–99, 2018, ISSN: 1742-2876. DOI: 10.1016/j.diin.2018.02.001. Adresse: https://www.sciencedirect.com/science/article/pii/S1742287617303547.

[3] S. Bunting, *EnCase Computer Forensics – the Official EnCE: EnCase Certified Examiner Study Guide*, 3. Aufl. Indianapolis, Indiana: John Wiley & Sons, Incorporated, 2012, ISBN: 9781118219409. Adresse: http://ebookcentral.proquest.com/lib/hs-mittweida/detail.action?docID=821657.

[4] X-Ways Software Technology AG, *X-Ways Forensics: Integrierte Software für Computerforensik*, 10.09.2023. Adresse: https://www.x-ways.net/forensics/index-d.html.

[5] B. Shavers und E. Zimmerman, *X-Ways Forensics Practitioner's Guide*. Syngress, 2014, ISBN: 9780124116221.

[6] X-Ways Software Technology AG, *X-Ways Investigator*. Adresse: https://www.x-ways.net/investigator/index-d.html.

[7] S. Fleischmann und X-Ways Software Technology AG, „Benutzerhandbuch X-Ways Forensics & WinHex“, 2023. Adresse: https://www.x-ways.net/winhex/manual-d.pdf.

[8] X-Ways Software Technology AG, *X-Ways Forensics X-Tensions API Documentation*, 10.09.2023. Adresse: https://www.x-ways.net/forensics/x-tensions/api.html.

[9] A. Böttcher und F. Kneissl, *Informatik für Ingenieure: Grundlagen und Programmierung in C*. München: Oldenbourg Wissenschaftsverlag, 2012, ISBN: 9783486717419. DOI: 10.1524/9783486717419.

[10] X-Ways Software Technology AG, *X-Ways Forensics X-Tensions API Documentation: XWF_* functions that you may call*, 10.09.2023. Adresse: https://www.x-ways.net/forensics/x-tensions/XWF_functions.html.

[11] Microsoft, *__stdcall*, 2021. Adresse: https://learn.microsoft.com/en-us/cpp/cpp/stdcall?view=msvc-170.

[12] Microsoft, *Was ist eine DLL*, 2023. Adresse: https://learn.microsoft.com/de-de/troubleshoot/windows-client/deployment/dynamic-link-library.

[13] Á. Rocha, C. Ferrás, P. C. López-López und T. Guarda, *Information technology and systems: ICITS 2021* (Advances in Intelligent Systems and Computing). Cham, Switzerland: Springer, 2021, Bd. volume 1330-1331, ISBN: 978-3-030-68284-2. DOI: 10.1007/978-3-030-68285-9.

[14] P. Mandl, *Grundkurs Betriebssysteme: Architekturen, Betriebsmittelverwaltung, Synchronisation, Prozesskommunikation, Virtualisierung*, 5. Aufl. Wiesbaden: Springer Fachmedien Wiesbaden, 2020, ISBN: 978-3-658-30546-8. DOI: 10.1007/978-3-658-30547-5.

[15] Microsoft, *Advantages of Dynamic Linking*, 2021. Adresse: https://learn.microsoft.com/en-us/windows/win32/dlls/advantages-of-dynamic-linking.

[16] Microsoft, *Extensible-Storage-Engine*, 2021. Adresse: https://github.com/microsoft/Extensible-Storage-Engine.

[17] Microsoft, *Extensible Storage Engine*, 2021. Adresse: https://learn.microsoft.com/en-us/windows/win32/extensible-storage-engine/extensible-storage-engine.

[18] J. Kim, A. Park und S. Lee, „Recovery method of deleted records and tables from ESE database“, *Digital Investigation*, Jg. 18, S118–S124, 2016, ISSN: 1742-2876. DOI: 10.1016/j.diin.2016.04.003. Adresse: https://www.sciencedirect.com/science/article/pii/S1742287616300342.

[19] IBM, „C-ISAM Programmer's Manual“, 2001. Adresse: https://publib.boulder.ibm.com/epubs/pdf/7897b.pdf.

[20] I. Ahmed, G. Smith und E. Pirozzi, *PostgreSQL 10 High Performance: Expert techniques for query optimization, high availability, and efficient database maintenance*, 1. Aufl. Birmingham: Packt Publishing, 2018, ISBN: 9781788472456. Adresse: https://www.wiso-net.de/document/PKEB__9781788472456508.

[21] J. Metz, *Extensible Storage Engine (ESE) Database File (EDB) format specification*, 2023. Adresse: https://github.com/libyal/libesedb/tree/main/documentation.

[22] A. Shaaban und K. Sapronov, *Practical Windows forensics: Leverage the power of digital forensics for Windows systems*, 1st ed. Birmingham: Packt Publishing, 2016, ISBN: 978-1-78355-409-6.

[23] H. Chivers, „Private browsing: A window of forensic opportunity“, *Digital Investigation*, Jg. 11, Nr. 1, S. 20–29, 2014, ISSN: 1742-2876. DOI: 10.1016/j.diin.2013.11.002. Adresse: https://www.sciencedirect.com/science/article/pii/S1742287613001229.

[24] H. Chivers und C. Hargreaves, „Forensic data recovery from the Windows Search Database“, *Digital Investigation*, Jg. 7, Nr. 3, S. 114–126, 2011, ISSN: 1742-2876. DOI: 10.1016/j.diin.2011.01.001. Adresse: https://www.sciencedirect.com/science/article/pii/S1742287611000028.

[25] Microsoft, *Übersicht über Active Directory Domain Services*, 2023. Adresse: https://learn.microsoft.com/de-de/windows-server/identity/ad-ds/get-started/virtual-dc/active-directory-domain-services-overview.

[26] D. Francis, *Mastering Active Directory: Design, deploy, and protect Active Directory Domain Services for Windows Server 2022*, 3. Aufl. Birmingham: Packt Publishing Limited, 2021, ISBN: 9781801073752. Adresse: https://www.wiso-net.de/document/PKEB__9781801073752778.

[27] M. de Rooij und J. Wesselius, *Pro Exchange 2019 and 2016 Administration: For Exchange On-Premises and Office 365*, 2. Aufl. Berkeley, CA: Apress, 2022, ISBN: 978-1-4842-7330-2. DOI: 10.1007/978-1-4842-7331-9.

[28] Microsoft, *Database Overview*, 2021. Adresse: https://learn.microsoft.com/en-us/windows/win32/extensible-storage-engine/database-overview.

[29] P. Fischer und P. Hofer, *Lexikon der Informatik*. Heidelberg: Springer-Verlag Berlin, 2011, ISBN: 9783642151262. DOI: 10.1007/978-3-642-15126-2.

[30] Microsoft, *ESE Deep Dive: Part 1: The Anatomy of an ESE database*, 2019. Adresse: https://techcommunity.microsoft.com/t5/ask-the-directory-services-team/ese-deep-dive-part-1-the-anatomy-of-an-ese-database/ba-p/400496.

[31] H. Knebl, *Algorithmen und Datenstrukturen: Grundlagen und probabilistische Methoden für den Entwurf und die Analyse*, 2., aktualisierte Auflage. Wiesbaden: Springer Vieweg, 2021, ISBN: 978-3-658-32713-2.

[32] H. Ernst, J. Schmidt und G. Beneken, „Datenstrukturen, Bäume und Graphen“, in *Grundkurs Informatik*, H. Ernst, J. Schmidt und G. Beneken, Hrsg., Wiesbaden: Springer Fachmedien Wiesbaden, 2020, S. 609–690, ISBN: 978-3-658-30331-0. DOI: 10.1007/978-3-658-30331-0_14.

[33] R. Bayer und E. M. McCreight, „Organization and maintenance of large ordered indexes“, *Acta Informatica*, Jg. 1, Nr. 3, S. 173–189, 1972, ISSN: 0001-5903. DOI: 10.1007/BF00288683.

[34] G. Saake, K.-U. Sattler und A. Heuer, *Datenbanken: Implementierungstechniken*, 3. Aufl. Heidelberg u. a.: mitp, 2011, ISBN: 9783826691560.

[35] D. Comer, „Ubiquitous B-Tree“, *ACM Computing Surveys*, Jg. 11, Nr. 2, S. 121–137, 1979, ISSN: 0360-0300. DOI: 10.1145/356770.356776.

[36] Microsoft, *Extensible Storage Engine Architecture*, 2014. Adresse: https://learn.microsoft.com/en-us/previous-versions/tn-archive/aa998171(v=exchg.65).

[37] B. Malmström und P. Teveldal, „Forensic analysis of the ESE database in Internet Explorer 10“, Diss., 2013. Adresse: https://www.diva-portal.org/smash/record.jsf?pid=diva2:635743.

[38] W. van Dongen, W. Toorop und J. Blokhuis, „Forensic examination of Windows Live Messenger 2009 Extensible Storage Engine“, 2009. Adresse: https://www.nlnetlabs.nl/~willem/wlm2009_ese_fin.pdf.

[39] Microsoft, *Esentutl*, 2016. Adresse: https://learn.microsoft.com/en-us/previous-versions/windows/it-pro/windows-server-2012-r2-and-2012/hh875546(v=ws.11).

[40] Microsoft, *Walkthrough: Create and use your own Dynamic Link Library (C++)*, 2021. Adresse: https://learn.microsoft.com/de-de/cpp/build/walkthrough-creating-and-using-a-dynamic-link-library-cpp?view=msvc-170.

[41] Microsoft, *Precompiled Header Files*, 2022. Adresse: https://learn.microsoft.com/en-us/cpp/build/creating-precompiled-header-files?view=msvc-170.

[42] Microsoft, *DllMain entry point*, 2022. Adresse: https://learn.microsoft.com/en-us/windows/win32/dlls/dllmain.

[43] Microsoft, *Linker options*, 2023. Adresse: https://learn.microsoft.com/en-us/cpp/build/reference/linker-options?view=msvc-170.

[44] Microsoft, *Moduldefinitionsdateien (.Def)*, 2023. Adresse: https://learn.microsoft.com/de-de/cpp/build/reference/module-definition-dot-def-files?view=msvc-170.

[45] Microsoft, *Set debug and release configurations in Visual Studio*, 2023. Adresse: https://learn.microsoft.com/en-us/visualstudio/debugger/how-to-set-debug-and-release-configurations?view=vs-2022.

[46] X-Ways Software Technology AG, *X-Ways Forensics X-Tensions API Documentation: XT_* functions that you may export*, 10.09.2023. Adresse: https://www.x-ways.net/forensics/x-tensions/XT_functions.html.

[47] Microsoft, *strcmp, wcscmp, _mbscmp, _mbscmp_l*, 2022. Adresse: https://learn.microsoft.com/en-us/cpp/c-runtime-library/reference/strcmp-wcscmp-mbscmp?view=msvc-170.

[48] M. Kittan, „Realisierung eines Plug-ins für die Forensic Software X-Ways Forensics für Apple Konfigurationsdateien“, 2018.

[49] Microsoft, *memcpy, wmemcpy*, 2022. Adresse: https://learn.microsoft.com/en-gb/cpp/c-runtime-library/reference/memcpy-wmemcpy?view=msvc-170.

[50] R. Ishida und W3C, *Das BOM (byte-order mark) in HTML*, 10.09.2023. Adresse: https://www.w3.org/International/questions/qa-byte-order-mark#bomhow.

[51] Microsoft, *FILETIME*, 2022. Adresse: https://learn.microsoft.com/en-gb/office/client-developer/outlook/mapi/filetime.

[52] *Epoch Converter*, 10.09.2023. Adresse: https://www.epochconverter.com/.

[53] Microsoft, *strftime, wcsftime, _strftime_l, _wcsftime_l*, 2022. Adresse: https://learn.microsoft.com/en-us/cpp/c-runtime-library/reference/strftime-wcsftime-strftime-l-wcsftime-l?view=msvc-170.

[54] P. Leach, M. Mealling und R. Salz, *A Universally Unique IDentifier (UUID) URN Namespace*, 2005. Adresse: https://datatracker.ietf.org/doc/html/rfc4122.

Eidesstattliche Erklärung

Hiermit versichere ich – Linda Becker – an Eides statt, dass ich die vorliegende Arbeit selbstständig und nur unter Verwendung der angegebenen Quellen und Hilfsmittel angefertigt habe.

Sämtliche Stellen der Arbeit, die im Wortlaut oder dem Sinn nach Publikationen oder Vorträgen anderer Autoren entnommen sind, habe ich als solche kenntlich gemacht.

Diese Arbeit wurde in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegt oder anderweitig veröffentlicht.

Mittweida, 11. September 2023	
Ort, Datum	Linda Becker