HOCHSCHULE
MITTWEIDA
UNIVERSITY OF
APPLIED SCIENCES

Bachelorarbeit

Herr
Michael Kittan

Realisierung eines Plug-ins für die Forensic Software X-Ways Forensics für Apple Konfigurationsdateien

Mittweida, 2018

Fakultät Angewandte Computer- und Biowissenschaften

Bachelorarbeit

Realisierung eines Plug-ins für die Forensic Software X-Ways Forensics für Apple Konfigurationsdateien

Autor:
Herr Michael Kittan

Studiengang:
Allgemeine und Digitale Forensik

Seminargruppe:
FO15w2-B

Erstprüfer:
Prof. Dr. rer. nat. Christian Hummert

Zweitprüfer:
Prof. Dr. rer. pol. Dirk Pawlaszczyk

Einreichung:
Mittweida, 28.09.2018

Verteidigung/Bewertung:
Mittweida, 2018

Faculty of Applied Computer Sciences &
Biosciences

BACHELOR THESIS

Realisation of a X-Ways Forensics Plug-in to interpret Apple Configuration files

Author:
Michael Kittan

Course of studies:
General and Digital Forensic Science

Seminar group:
FO15W2-B

First examiner:
Prof. Dr. rer. nat. Christian Hummert

Second examiner:
Prof. Dr. rer. pol. Dirk Pawlaszczyk

Submission:
Mittweida, 28.09.2018

Defence/ evaluation:
Mittweida, 2018

Bibliografische Beschreibung:

Kittan, Michael:
Realisierung eines Plug-ins für die Forensic Software X-Ways Forensics für Apple Konfigurationsdateien - 2018. 84 Seiten, 32 Abbildungen, 2 Anlagen
Hochschule Mittweida University of Applied Sciences, Fakultät Angewandte Computer- und Biowissenschaften

Bachelorarbeit, 2018

Referat:

Die vorliegende Arbeit befasst sich mit der Umsetzung eines Parsers für Apple Konfigurationsdateien für die Forensic Software X-Ways Forensics. Dabei werden speziell der Aufbau und die Funktionsweise der Binary Property List Dateien erklärt und ein Programm entwickelt, das in X-Ways Forensics als X-Tension funktionsfähig eingebunden wird. Des Weiteren werden der Aufbau der API und die Schritte erläutert, die notwendig sind, um eine solche X-Tension zu erstellen.

This bachelor-thesis deals with the realisation of a parser for Apple configuration files for the forensic software X-Ways Forensic. In particular the structure and the operating principle of the Binary Property List Data as well as the development of a functioning, to X-Ways forensic integrated program are explained. Furthermore, the structure of the API and the steps to create such an x-tension are illustrated.

Inhalt

Abbildungsverzeichnis ..... **III**

Abkürzungsverzeichnis ..... **V**

1 Einleitung **1**

1.1 Motivation 1

1.2 Zielstellung 2

2 Grundlagen **3**

2.1 X-Ways Forensics 3

2.2 DLL 4

2.3 X-Ways Forensics X-Tensions API 5

2.3.1 XT_* Funktionen 6

2.3.1.1 Viewer X-Tension 7

2.3.2 XWF_* Funktionen 8

2.4 Apple Property List Files 9

2.4.1 Binary Property Lists 10

2.4.1.1 Header (8 Byte) 11

2.4.1.2 Trailer (32 Byte) 11

2.4.1.3 Offset Tabelle (Variable Größe) 12

2.4.1.4 Objekt Tabelle (Variable Größe) 13

2.4.1.5 Beispiel BPList auslesen 17

3 Methoden **20**

3.1 Erstellung einer X-Ways X-Tension 20

3.2 Programmumsetzung 24

3.2.1 Aufruf der XT* Funktionen 24

3.2.2 Auslesen von Header, Trailer und Offset Tabelle 25

3.2.3 Formatieren der Ausgabe 27

3.2.4 Verarbeitung der Objekttypen in der Objekt Tabelle 30

3.2.4.1 Interpretieren der Objekttypen 30

3.2.4.2 Objekttyp Integer 31

3.2.4.3 Objekttyp Real 32

3.2.4.4 Objekttyp Data 33

3.2.4.5 Objekttypen ASCII String, Unicode String, UID und Date 33

3.2.4.6 Containerformate 34

4 Ergebnis 37

5 Diskussion 40

5.1 Vergleich mit anderen Parsern 40

5.2 Fazit 43

5.3 Ausblick 44

5.3.1 Verbesserungen der X-Tension 44

5.3.2 Andere Problembetrachtungen 46

Literatur 47

Anlagen 52

Anlage I - Vergleich mit Kommerzieller Software I

Anlage II - Quellcode der X-Tension 5

Selbstständigkeitserklärung .....

Abbildungsverzeichnis

Abbildung 1: Einbindung X-Tensions, (eigene Darstellung) 6

Abbildung 2: PList File im XML Format (eigene Darstellung) 9

Abbildung 3: PList File im Binary Format (eigene Darstellung) 9

Abbildung 4: Aufbau BPList File, Auszug [APPL14_1] 10

Abbildung 5: BPList Header (eigene Darstellung) 11

Abbildung 6: BPList Trailer (eigene Darstellung) 11

Abbildung 7: BPList Offset Tabelle (eigene Darstellung) 12

Abbildung 8: BPList Objekt Tabelle (eigene Darstellung) 13

Abbildung 9: Dateitypen im BPList File [APPL14_1] 14

Abbildung 10: Integer Zählvariable (eigene Darstellung) 16

Abbildung 11: Trailer auslesen Beispiel (eigene Darstellung) 17

Abbildung 12: Offset Tabelle auslesen Beispiel (eigene Darstellung) 17

Abbildung 13: Auslesen Dictionary Beispiel (eigene Darstellung) 18

Abbildung 14: Ausgabe Beispiel mit PList Viewer Xcode (eigene Darstellung) 19

Abbildung 15: Erstellen eines neuen Projekts (eigene Darstellung) 21

Abbildung 16: Verwendung des Vorkompilierten Headers deaktivieren (eigene Darstellung) 21

Abbildung 17: Moduldefinitionsdatei definieren (eigene Darstellung) 22

Abbildung 18: Projekt mit allen Dateien im Projektmappen-Explorer (eigene Darstellung) 23

Abbildung 19: BPList File ohne Inhalt (eigene Darstellung) 26

Abbildung 20: Beispiel Ausgabe Tabellen Dictionary und Array (eigene Darstellung) 30

Abbildung 21: Datentypen: Date, Integer, Real, Array, Dictionary, String (eigene Darstellung) 37

Abbildung 22: Ausschnitt des generierten HTML Codes im Rohformat (eigene Darstellung).....37

Abbildung 23: Datentyp Data und Verschachtelung (eigene Darstellung) 38

Abbildung 24: Ergebnis: Abfangen der Dateigröße (eigene Darstellung) 38

Abbildung 25: Ergebnis: Datentyp String, Real, Dictionary (eigene Darstellung) 41

Abbildung 26: Ergebnis: Datentyp Date, String, Integer, Array (eigene Darstellung).....42

Abbildung 27: Ergebnis: Datentyp Bool, UID, String, Dictionary, Array (eigene Darstellung).....42

Abbildung 28: Darstellung des NTFS Journal Parsers (eigene Abbildung).....43

Abbildung 29: Anlage 1: Objekttypenvergleich (eigene Darstellung)..... I

Abbildung 30: Anlage 1: Objekttypenvergleich (eigene Darstellung)..... II

Abbildung 31: Anlage 1: Objekttypenvergleich (eigene Darstellung).....III

Abbildung 32: Anlage 1: Objekttypenvergleich (eigene Darstellung)..... IV

Abkürzungsverzeichnis

ASCII	American Standard Code for Information Interchange
API	Application programming interface
BOM	Byte Order Mark
BPList	Binary Property List File
DLL	Dynamic Link Libary
Dylib	Dynamally linked Library
HTML	Hypertext Markup Language
IEEE	Institute of Electrical and Electronics Engineers
Kb	Kilobyte
PList	Property List
XML	Extensible Markup Language
XWFo	X-Ways Forensics

1 Einleitung

Durch die Digitalisierung hat sich in den letzten 20 Jahren das Leben in den industrialisierten Ländern verändert. Das Smartphone ist ein ständiger Begleiter und übernimmt immer mehr wesentliche Aufgaben des Alltags, wie z.B. die des Navigationssystems, des Kalenders oder der Geldbörse. Aber auch die herkömmliche Computertechnik entwickelt sich weiter. Diese bisherigen und zukünftigen Innovationen, die unser Leben geprägt haben oder noch prägen werden, bieten auch der Kriminalität neue Möglichkeiten. Neben den physischen Spuren wird es heute immer wichtiger, auch Spuren in den digitalen Medien zu sichern, auszuwerten und gerichtsfest verwertbar zu machen. Untermauert wird dies von den Zahlen des BKAs, laut denen eine Steigerung von über 80% der Cybercrime Straftaten von 2015 auf 2016 gemessen wurde. Aber auch die Gründung von neuen Stellen wie z.B. der ZiTiS (Zentralen Stelle für Informationstechnik im Sicherheitsbereich) in München zeigen, dass es für die Behörden immer wichtiger wird, digitale Straftaten aufklären zu können [BKA16; BMI17].

Zu Hilfe kommen den Behörden dabei Tools, die auf die Auswertung digitaler Daten spezialisiert sind und stetig weiterentwickelt werden. Trotz dieses großen und hart umkämpften Marktes kann keines dieser Tools alle Daten automatisiert auswerten. Sie sind Hilfsmittel, die Ihre Stärken und Schwächen haben. Nicht nur aus diesem Grund gehört es für den IT-Fahnder zum täglichen Geschäft, eigene Lösungen zu entwickeln, verschiedene Tools zu kombinieren oder Daten komplett von Hand auszuwerten.

1.1 Motivation

Ein weit verbreitetes Forensisches Tool ist die Software „X-Ways Forensics“ von der Firma X-Ways Software Technology AG mit Sitz in Köln. Die Stärke dieser Forensik-Suite liegt darin, dass es hinter einer recht einfachen Benutzeroberfläche mehrere Möglichkeiten gibt, die Funktionen und Methoden aufzurufen. Genau daher stammt auch der Name „X-Ways“: Umgangssprachlich für „viele Wege“ [BRET14, S. 19]. Trotz des großen und stetig wachsenden Funktionsumfangs der Software ist sie in der aktuellen Version nicht in der Lage, Apple Konfigurationsdateien in der binären Speicherform auszuwerten. Obwohl einige Apple Dateien innerhalb der Software ausgelesen und analysiert werden können, erscheint besonders beim Interpretieren von binären Formaten eine unlesbare Ausgabe. Für IT-Fahnder bedeutet dies, die Daten per Hand analysieren zu müssen, um an verwertbare Informationen zu gelangen. Dabei liegt der Gedanke nahe, dass digitale Informationen, egal wie sie gespeichert werden, auch automatisch auswertbar sein sollten. Das System muss die gespeicherten Informationen lesen, ändern oder selbst verwerten können, was nur mithilfe eines Quellcodes umsetzbar ist, der weiß, wie die Daten zu interpretieren sind. Besonders Apple ist dafür bekannt, dass für eine Interpretation Ihrer Software außerhalb ihrer eigenen Lösungen Do-

kumentationen und Ansätze fehlen. Das bedeutet aber nicht, dass es nicht möglich ist, einen solchen Ansatz zu entwickeln.

1.2 Zielstellung

Ziel dieser Arbeit ist die Programmierung einer Erweiterung für die forensische Analysesoftware „X-Ways Forensics“, einer sogenannten X-Tension, deren Funktion es ist, die relevanten Daten aus der Apple Binary Property List (BPList) zu lesen und zu verarbeiten. Dazu ist es zum einen notwendig, eine X-Tension in X-Ways lauffähig einzubinden und zum anderen einen Parser zu konzipieren, welcher die Inhalte der Dateien aus einem Binärformat in einen für den Menschen lesbaren und verständlichen Inhalt optisch ansprechend übersetzt. Weiterhin sollen die Grundlagen der X-Ways API und die notwendigen Schritte, damit die Erweiterung der Software angenommen und potentiellen Nutzern die Entwicklung erleichtert wird, erläutert werden.

2 Grundlagen

In diesem Kapitel sollen die Grundlagen vermittelt werden, die notwendig sind, um das Ziel dieser Arbeit zu erreichen. Dazu zählen unter anderem der Umgang mit DLL-Dateien, die detaillierte Auseinandersetzung mit der X-Ways API inklusive der dazugehörigen Funktionen und vor allem die genaue Analyse des Apple Property List Formates.

2.1 X-Ways Forensics

Im Zuge der letzten Jahre haben sich mit der wachsenden Digitalisierung komplexe forensische Softwarelösungen hervorgetan, die mit der Zeit gehen und sich rasch weiterentwickeln. Neben vielen kleineren Programmen, die sich mit einigen speziellen Gebieten befassen und optimierte Lösungen anbieten, gibt es auch einige große Forensik-Suiten. Diese haben sich auf dem Markt etabliert und werden von Behörden und privaten Firmen zur Analyse von EDV-Daten genutzt. Die Vorteile dieser großen Softwarelösungen sind, dass sie viele Funktionen in einem Programm bündeln, die gewonnen Informationen direkt weiterverarbeitet werden können, sie mehrere Dateisysteme unterstützen, Schnittstellen bereitstellen, Images einlesen, verwalten oder selber erstellen und eine umfangreiche Suche in den Daten ermöglichen. Neben Encase[1] und dem Forensik-Toolkit[2] ist das von Stefan Fleischmann 2004 entwickelte X-Ways Forensics[3] (im Weiteren mit XWFo bezeichnet) eine dieser großen Suiten.

Seitdem die Software 2004 aus dem Hex-Editor[4] WinHex entstanden ist, wird sie permanent von Fleischmann und seiner Firma weiterentwickelt. Dabei reagiert er schnell auf Anregungen seiner Kunden, was dazu beigetragen hat, dass es mit über 35.000 Nutzern zu einem der meist benutzten Forensik-Tools gehört. Weitere Stärken der Software sind unter anderen die gute Performance aufgrund der geringen Systemanforderung, durchsichtige Lösungswege, die Ausführung von externen Programmen sowie die Vielzahl von Filtermöglichkeiten [BRET14, S. 19; XWAY18_1]. Besonders in Deutschland hat sich die Software zu einem etablierten Werkzeug entwickelt, um Beweise aus digitalen Medien zu ermitteln und gerichtsfest auszuwerten [XWAY18_1]. Der größte Kunden-

[1] Encase: Link zur Herstellerseite: https://www.guidancesoftware.com/encase-forensic; letzter Zugriff:26.09.2018 12:12

[2] Forensik Toolkit: Link zur Herstellerseite: https://accessdata.com/products-services/forensic-toolkit-ftk/; letzter Zugriff:26.09.2018 12:12

[3] X-Ways Forensics: Link zur Herstellerseite: http://www.x-ways.net/forensics/index-d.html; letzter Zugriff:26.09.2018 12:12

[4] Hex-Editor: ist ein Programm zum betrachten und Bearbeiten von Daten auf Byte-Ebene wobei auf die Interpretation der Daten verzichtet wird

stamm kommt dabei aus dem behördlichen Umfeld, aber auch in privaten Unternehmen ist XWFo eine weit verbreitete Umgebung [FOR08]. Der X-Ways Investigator ist eine abgeschwächte Version der Software, so dass es auch für Nutzer ohne tiefgreifende IT-Kenntnisse möglich ist, Beweise zu durchsuchen. Zuvor müssen die Asservate durch die IT-Forensiker mit der Hauptsoftware aufgearbeitet werden. Danach werden jedoch die Ermittlungen sowie die Arbeitsteilung erheblich erleichtert [XWAY18_2].

Zum Anzeigen der vielen unterschiedlichen Dateitypen greift XWFo auf interne Viewer-Komponenten[5] zurück, die dafür sorgen, dass eine Vielzahl von Dateiformaten betrachtet werden können. Dabei nutzt die Software neben den internen auch externe Viewer-Komponenten, die separat eingebunden werden müssen und auf der Herstellerseite heruntergeladen werden können. Falls ein Format nicht unterstützt wird, bietet XWFo die Möglichkeit externe Programme festzulegen, die dann aus der Software heraus manuell aufgerufen werden können. Bei Videodateien bietet sich diese Lösung an, aber auch bei fremden Dateiformaten, die von der Software noch nicht unterstützt werden. Dabei wird das externe Programm separat gestartet, was für Geschwindigkeit und die Handhabung eher nachteilig ist. Zuletzt ist es noch möglich, mithilfe der XWFo-API eine eigene Erweiterung zu schreiben, die das Anzeigen innerhalb der Software mit den gleichen Performancevorteilen ermöglicht wie die Hauptsoftware. Genau dieser Weg soll für das Lesen der BPList Files gegangen werden.

Da der Funktionsumfang und der forensische Nutzen von XWFo bzw. den anderen Forensik-Suiten sehr groß ist, werden hier nur einige wenige Funktionen erwähnt. Eine umfangreichere Funktionsbeschreibung bieten die Handbücher der einzelnen Suiten.

2.2 DLL

Das Einbinden der X-Tensions wird von der X-Ways AG mit Hilfe des Programmbibliotheksformats Dynamic Link Library (DLL) umgesetzt. Das DLL-Format ist Teil des Shared Memory Konzepts. Es dient dazu, gemeinsam genutzte Speicherbereiche nur einmal in den Hauptspeicher zu laden, so dass sie von mehreren Prozessen genutzt werden können ohne dass jeder Prozess den Code selbst in den Speicher laden muss. Dieses Konzept spart Platz auf der Festplatte und vor allem im Arbeitsspeicher. Ein weiterer Vorteil besteht darin, dass sich ein Programm in separate Module aufteilen lässt, die sich zur Laufzeit des Programms nachladen lassen. Das Laden des benötigten Moduls erfolgt nur dann, wenn es vom Programm benötigt wird, was ein schnelleres Starten des Programms und auch des jeweiligen Moduls ermöglicht. In der Praxis greifen mehrere Programme auf gleiche DLLs zurück. Wird in einer dieser Bibliotheken eine Sicherheitslücke festgestellt oder sind andere Anpassungen notwendig, die eine Aktualisierung erforderlich machen, ist es nicht notwendig, dass alle Entwickler der zugreifenden Programme eine Aktualisierung einspielen.

[5] Interne Viewer sind Bibliotheken innerhalb von XWFo die es ermöglichen bestimmte Dateiformate darzustellen [XWF18_4, S. 103-104]

Es reicht aus, dass eine Aktualisierung für die jeweilige DLL eingespielt wird, alle anderen Programme profitieren automatisch von diesem Update [MIC18_1; MAN14, S. 244].

Ein Nachteil dieser geteilten Bibliotheken besteht darin, dass die Programme nicht mehr eigenständig sind. Wird eines dieser Objekte vom System gelöscht oder verschoben, besteht die Möglichkeit, dass das komplette Programm nicht mehr lauffähig ist. Auch ein Überschreiben der DLL durch eine ältere Version, beispielsweise durch die Installation eines älteren Programms, kann zu Fehlern bei der Ausführung führen. Im Umkehrschluss bedeutet dies, dass DLLs ein großes Sicherheitsrisiko für Windows-Geräte darstellen, weswegen mit dem Windows-Dateischutz verhindert wird, dass nicht autorisierte Anwendungen Systembibliotheken bei einer Installation überschreiben [MIC18_1].

Das Dynamic Link Libary Format ist in der Windows-Welt ein gängiges Format und fester Bestandteil des Betriebssystems. Auf anderen Betriebssystemen existieren Formate, die ähnliche Funktionen erfüllen wie z.B. „.so“ (shared object) Objekte bei Unix-artigen Systemen wie Linux oder dem „dylib“-Format (Dynamally linked Library) für Apple Systeme [APPL12; MAN14, S. 244].

2.3 X-Ways Forensics X-Tensions API

Zum Erweitern des Funktionsumfangs bietet der Hersteller mit der „X-Ways Forensics X-Tensions API[6]“ die Möglichkeit, eigene Erweiterungen zu implementieren. Die Dokumentation der Schnittstelle findet sich mit allen Funktionen ausschließlich auf der Herstellerseite, worauf sich in diesem Kapitel, wenn nicht anders kenntlich gemacht, bezogen wird. Komplexere Anwendungsbeispiele sind in der Dokumentation nicht vorhanden, weshalb es aus Gründen des allgemeinen Verständnisses notwendig ist, hier auf einige Punkte speziell einzugehen. [XWAY18_3].

Das Erstellen einer X-Tension mit Hilfe der API bietet viele neue Möglichkeiten für die individuelle Anpassung oder das Automatisieren von forensischen Aufgaben. Der Programmcode der API wird im Adressraum der Anwendung selbst ausgeführt, dadurch kann die gleiche Performance erwartet werden wie bei den internen Funktionen des Hauptprogramms. Umgesetzt werden kann alles, was sich in die einzelne DLL-Datei implementieren lässt. Dank der Windows-API beinhaltet das nahezu alle Windows-Funktionen. Nach der Erstellung der DLL muss diese im XWFo Programmordner gespeichert und im Programm selbst aufgerufen werden.

Die notwendigen Dateien, „X-Tension.h“ und „X-Tension.cpp“ der API sind auf der Herstellerseite unter

http://www.x-ways.net/forensics/x-tensions/api.html

[6] API: Application Programming Interface, Anwendungsprogrammierschnittstelle, die definiert, welche Betriebssystem- oder Programmaufrufe einem Anwendungsprogramm zur Verfügung stehen [BÖT99, S. 59]

zum Herunterladen verfügbar. Dabei stehen die Programmiersprachen C, C++, Delphi und Python zur Verfügung. In dieser Arbeit wird im weiteren Verlauf mit C++ Dateien gearbeitet, aber die Programmiersprache C verwendet. Im Anschluss müssen diese in der jeweiligen Programmierumgebung eingebunden werden, worauf im Abschnitt 3.1 genauer eingegangen wird [XWAY18_3].

Die X-Ways API an sich unterteilt sich in zwei Arten von Funktionen. Sie lassen sich durch die Namenskonvertierung unterscheiden:

- XT_* Funktionen - werden von X-Ways aufgerufen
- XWF_* Funktionen - werden vom Entwickler aufgerufen

2.3.1 XT_* Funktionen

Wann eine XT_* Funktion vom Programm adressiert wird, hängt davon ab, an welcher Stelle die Erweiterung im Programm aufgerufen wird. Dafür gibt es vier verschiedene Möglichkeiten:

1. Beim Öffnen einer Partition als „Disk I/O X-Tension“ zum Verfeinern des Partitionsüberblicks (Abbildung 1- Rechts)
2. Im Suchfenster der parallelen Suche, damit Suchtreffer direkt verarbeitet werden
3. Über die Werkzeugleiste oder das Kontextmenü der Dateianzeige, was die X-Tension auf eine bzw. mehrere Dateien anwendet
4. Als Viewer X-Tension, die nach dem Einstellen in den Viewer-Programmen bei jeder Datei automatisch aufgerufen und im Vorschaufenster dargestellt wird (Abbildung 1 - links)

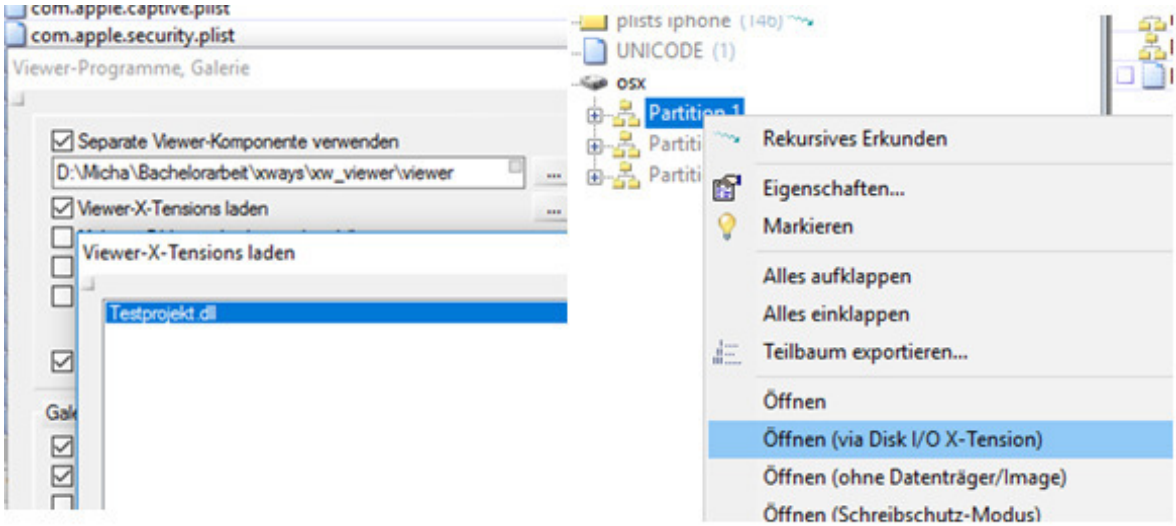


Abbildung 1: Einbindung X-Tensions, Viewer X-Tension (links), Disk I/O X-Tension (rechts) (eigene Darstellung)

Die verschiedenen Funktionen lassen sich zum Bereitstellen von mehr Funktionsvielfalt oder der einfacheren Handhabung in einer einzelnen X-Tension kombinieren. Wie schon erwähnt wird nur

die XT-Funktion ausgeführt, die aus dem jeweiligen Menü aufgerufen wird. Außerhalb dieser Einteilung agieren nur die Funktionen XT_Init und XT_Done. Während letztere vor dem Entladen der DLL aufgerufen wird, um z.B. mögliche Daten dauerhaft zu speichern, wird die XT_Init für die Initialisierung verwendet. Damit ist sie die einzige Funktion, die Pflicht innerhalb der DLL ist. Ohne die XT_Init ist es nicht möglich, eine funktionierende X-Ways Erweiterung in das Programm einzubinden. Des Weiteren muss in ihr die Funktion XT_RetrieveFunctionPointers() aufgerufen werden. Sie beinhaltet die XWF Funktionen und ist laut einem Herstellerkommentar in der X-Tension.cpp Pflicht.

```
21  #define EXPORT extern "C" __declspec(dllexport)
63  EXPORT BOOL __stdcall XT_ReleaseMem(PVOID lpBuffer)
```

Codeblock 1: Export und stdcall Aufruf

Gemäß der Dokumentation der X-Ways API müssen alle XT* Funktionen exportiert und mit der `stdcall` Aufrufkonvention adressiert werden. In Codezeile 63 ist ein Beispiel des Exports für die Programmierumgebung Visual Studio und in Programzeile 21 ein Beispiel für den Aufruf von `stdcall` innerhalb der XT_ReleaseMem Funktion aufgezeigt.

„`__stdcall`“ teilt dem Compiler die Regeln für die Einrichtung des Stacks, das Übergeben von Argumenten und das Abrufen des Rückgabewertes mit [MIC16_2].

„`#define EXPORT extern "C" __declspec(dllexport)` “ sorgt dafür, dass die Klassen und Methoden der Exportdirektive des Compilers hinzugefügt werden. Alternativ kann diese Anweisung auch vor jede Funktion separat geschrieben werden. Zur besseren Lesbarkeit ist die Variante in Codeblock 1 aber zu bevorzugen [MIC16_3].

2.3.1.1 Viewer X-Tension

Ein typisches Anwendungsgebiet für eine Viewer X-Tension ist das Interpretieren eines Byte-Codes in eine lesbare Form, ähnlich wie es beim Lesen von Apple Konfigurationsdateien notwendig ist. Beim Aufruf durch das Programm werden nach der XT_Init folgende zwei Funktionen nacheinander abgearbeitet:

1. `PVOID XT_View(HANDLE hItem, LONG nItemID, HANDLE hVolume, HANDLE hEvidence, PVOID lpReserved, PINT64 nResSize)`

Die einzelnen HANDLE-Variablen und die `nItemID` werden von XWFo intern vergeben und je nach zu implementierender Funktionalität der XWF-Funktionen benötigt. Der `lpReserved` Parameter sollte laut aktueller API Dokumentation ignoriert werden. Der einzige Parameter, der vom Entwickler während der Laufzeit auf einen der vier folgenden Zustände gesetzt und verändert werden muss, ist `nResSize`.

Wert	Bedeutung
-2	Fehler
-1	Viewer X-Tension ist nicht für diese Datei Verantwortlich
0	Keine Daten werden Dargestellt
Positiver Wert	Größe der Daten die zurückgegeben werden

Tabelle 1: Parameter nResSize der XT_View (eigene Darstellung)

Als Rückgabewert für die Funktion dient die Adresse eines Buffers[7], der in der X-Tension selbst definiert werden muss. Dieser Buffer wird im Vorschaufenster angezeigt und interpretiert. Der Inhalt kann normaler Text oder Code sein, sofern er von einer Viewer-Komponente interpretiert werden kann.

```
2. BOOL XT_ReleaseMem(PVOID lpBuffer)
```

Die Funktion XT_ReleaseMem wird nach der XT_View aufgerufen, falls Speicher über die Variable `nResSize` in der XT_View zurückgeben wurde. Sie dient entweder dazu, den Speicher freizugeben oder global benötigte Variablen auf die Ursprungswerte zurückzusetzen. Die Interpretation des Rückgabewertes ist aktuell in der API noch nicht integriert.

Sobald eine Viewer X-Tension eingebunden wurde, wird im Hauptverzeichnis von X-Ways die Datei „Viewer X-Tensions.txt“ erstellt. In dieser Datei kann festgelegt werden, welche X-Tensions in welcher Reihenfolge ausgeführt werden.

2.3.2 XWF_* Funktionen

XWF_* Funktionen ruft der Entwickler an einer beliebigen Stelle des Programms auf. Sie werden aus den XT_* Funktionen aufgerufen und beinhalten zumeist Abfragen oder das Setzen von neuen Informationen, die von XWFo weitergereicht oder an das Programm übergeben werden. Je nach Funktionalität der Erweiterung können unterschiedliche Anweisungen von Bedeutung sein. In dieser Erläuterung wurde als Beispiel die Funktion zum Auslesen des Dateiinhaltes gewählt. Die anderen aktuell 64 Funktionen sind ausreichend auf der Herstellerseite beschrieben und unterscheiden sich in Aufruf und Verarbeitung nur geringfügig von dem gewählten Beispiel.

```
DWORD XWF_Read (HANDLE hVolumeOrItem, INT64 nOffset, BYTE* lpBuffer, DWORD nNumber-
OfBytesToRead);
```

Parameter	Bedeutung
`hVolumeOrItem`	Adresse der Datei auf die Zugegriffen wird
`nOffset`	Offset von dem an gelesen werden soll
`lpBuffer`	Zieladresse der gelesenen Werte
`nNumberOfBytesToRead`	Anzahl der zu lesenden Bytes

Tabelle 2: Bedeutung Parameter XWF_Read (eigene Darstellung)

[7] Buffer (zu Deutsch Puffer) dient zur Zwischenspeicherung von Daten

Der Rückgabewert gibt die Anzahl der tatsächlich gelesenen Bytes zurück, was es ermöglicht, auftretende Fehler abzufangen.

2.4 Apple Property List Files

Beinahe jede Anwendung hat Parameter, die abgefragt, übergeben oder auch gespeichert werden. Dies können Informationen sein wie das Tastaturlayout, die hinterlegte Sprache, der Zeitpunkt der letzten Sicherung oder auch der Name des Nutzers. Durch die Menge an Informationen, die dabei gespeichert werden, aber auch durch das Abfragen von anderen Programmen, sollten diese Informationen transportierbar, speicherbar und effizient hinterlegt sein. Die Betriebssysteme von Apple speichern diese Informationen in Eigenschaftslisten, den sogenannten „Property List“ (PList), die durch den festgelegten Aufbau zum Teil auch eine gewisse Komprimierung der Informationen erlauben [APPL10].

Die Speicherung innerhalb der Betriebssysteme erfolgt dabei in zwei Formaten, dem binären und dem XML[8] Format. Während die XML Speicherung den Vorteil hat, dass es mit jedem Text Editor les- und änderbar ist (Abbildung 2), benötigt die binäre Variante (Abbildung 3) weniger Speicherplatz und ist schneller in den Zugriffszeiten. Eine Konvertierung zwischen den beiden Formaten kann mithilfe des Apple Tools „plutil“ erfolgen. Außerdem ist das Tool in der Lage, die Struktur eines PList Files auf Korrektheit zu überprüfen [APPL10; MAR10, S.39].

com.apple.keyboardtype.plist

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/PropertyList-1.0.dtd">
3  <plist version="1.0">
4  <dict>
5      <key>keyboardtype</key>
6      <dict>
7          <key>45200-1130-0</key>
8          <integer>41</integer>
9          <key>35-1130-0</key>
10         <integer>41</integer>
11     </dict>
12 </dict>
13 </plist>
```

Abbildung 2: PList File im XML Format (eigene Darstellung)

```
Offset(h)  00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F

00000000   62 70 6C 69 73 74 30 30 D1 01 02 5C 6B 65 79 62   bplist00Ñ..\keyb
00000010   6F 61 72 64 74 79 70 65 D2 03 04 05 05 5C 34 35   oardtypeÒ....\45
00000020   32 30 30 2D 31 31 33 30 2D 30 59 33 35 2D 31 31   200-1130-0Y35-11
00000030   33 30 2D 30 10 29 08 0B 18 1D 2A 34 00 00 00 00   30-0.)....*4....
00000040   00 00 01 01 00 00 00 00 00 00 00 06 00 00 00 00   ................
00000050   00 00 00 00 00 00 00 00 00 00 00 36               ...........6
```

Abbildung 3: PList File im Binary Format (eigene Darstellung)

[8] XML (Extensible Markup Language) universelles Datenformat, dass von Menschen und Maschinen gleichermaßen gelesen werden kann, es unterliegt dabei einer strengen Syntax [FIS11, S. 1022]

Aufgrund der einfachen Lesbarkeit von XML Files und des ähnlichen Aufbaus wird an dieser Stelle auf eine genauere Erklärung verzichtet und nur auf die BPList eingegangen.

2.4.1 Binary Property Lists

Die Dokumentation des Herstellers Apple zum Aufbau von BPList Files beschränkt sich auf ein paar Kommentare in der CoreFoundation[9] Datei CFBinaryPList.c (Abbildung 4). Mehrere Anfragen an Apple für eine genauere Spezifikation blieben unbeantwortet, weswegen diese Arbeit größtenteils auf Internetrecherchen oder eigenen Beobachtungen basiert. Auch die Kommunikation mit anderen Entwicklern[10] bestätigte, dass aktuell die CoreFoundation die einzige zitierbare Quelle darstellt.

```
/*
HEADER
    magic number ("bplist")
    file format version (currently "0?")

OBJECT TABLE
    variable-sized objects

OFFSET TABLE
    list of ints, byte size of which is given in trailer
    -- these are the byte offsets into the file
    -- number of these is in the trailer

TRAILER
    byte size of offset ints in offset table
    byte size of object refs in arrays and dicts
    number of offsets in offset table (also is number of objects)
    element # in offset table which is top level object
    offset table offset
*/
```

Abbildung 4: Aufbau BPList File, Auszug [APPL14_1]

In Abbildung 4 ist ersichtlich, dass ein BPList File aus vier Teilen besteht. Dem Header, der die Magic Number[11] und die Version beinhaltet, der Objekt Tabelle, mit den eigentlichen Dateiinformationen, der Offset Tabelle, mit der Positionsangabe der einzelnen Werte und dem Trailer, der einige Kerninformationen über das BPList File enthält. Zuerst ausgewertet werden aufgrund der

[9] CoreFoundation: eine API für Apple Betriebssysteme, die Softwaredienste und Zugriffe auf Datentypen bereitstellt [APPL18_6]

[10] 3breadt ist der Autor der Java Bibliothek „com.dd.plist - A Java library for working with property lists“[BRE18]

[11] Magic Number: Am Anfang einer Datei untergebrachte Ganzzahl, mit der gewisse Dateitypen spezifiziert werden können [FIS11, S. 541]

festen Größen der Header und der Trailer, erst danach folgen die Offset Tabelle und die Objekt Tabelle mit dem eigentlichen Auslesen der Informationen [APPL14_1; CAI10].

Im Folgenden wird auf die einzelnen Bereiche genauer eingegangen. Zum Teil ist es dabei notwendig, auf einen anderen Teil der Property List zu verweisen. Am Ende dieses Kapitels wird an einem Beispiel erläutert, wie genau das Lesen eines BPList Files erfolgt.

2.4.1.1 Header (8 Byte)

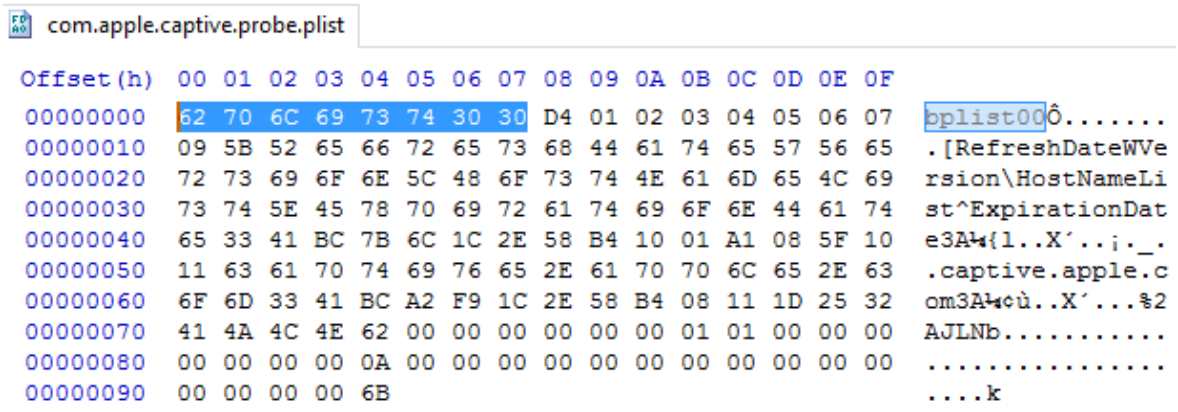


Abbildung 5: BPList Header (eigene Darstellung)

Der Header ist 8 Byte Groß und besteht aus der Magic Number und der Versionsnummer. Die Magic Number besteht aus den ersten 6 Byte und dient zur Identifikation als Binary Property List File. Die letzten 2 Byte sind für die Version des Files bestimmt. In dieser Arbeit werden ausschließlich Files mit der Version 00 untersucht. Gemäß des Apple CoreFoundation Files werden auch andere Versionsnummern verwendet. Auf diesen Punkt wird in Abschnitt 5.2 noch genauer eingegangen. [APPL14_1; KAR17].

2.4.1.2 Trailer (32 Byte)

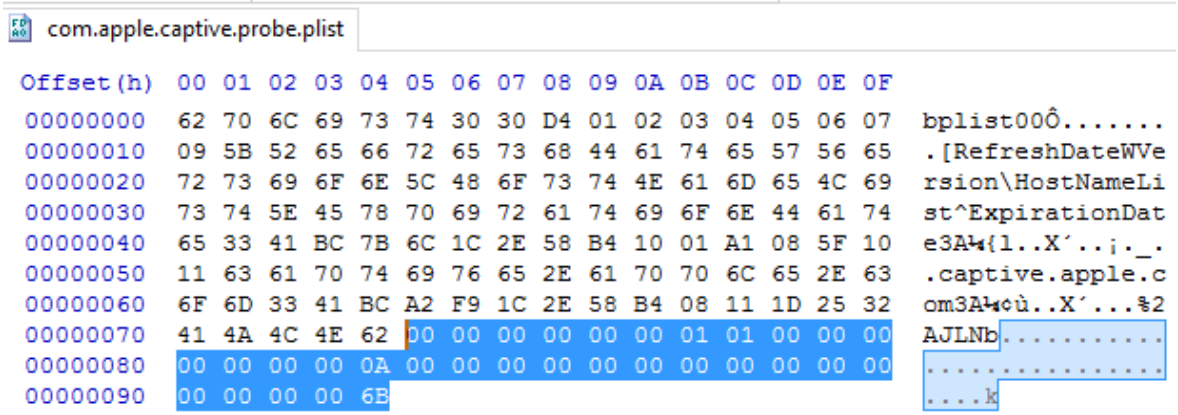


Abbildung 6: BPList Trailer (eigene Darstellung)

Der Trailer ist 32 Byte lang und der letzte der vier Teile. Hier sind an definierten Offsets wichtige Informationen zur Interpretation des Files gespeichert. Der Tabelle 3 ist zu entnehmen, an welchem Offset welche Informationen zu finden sind [KAR17].

Offset	Länge	Beschreibung
0x00	5	Unbenutzt (immer 0)
0x06	1	Offset Tabellen Offset Größe (offsize) Byte Anzahl für die Speicherung eines Offsets in der Offset Tabelle.
0x07	1	Objekt Referenzierungs Byte länge (refsize) Ähnlich wie die Offset Tabellen Offset Größe gibt der Wert an Offset 7 an, wieviel Bytes zum Adressieren in Containerformaten z.B. Array oder Dict benötigt werden.
0x08	8	Anzahl der Objekte in der Objekt Tabelle (numobjekt) Der Wert gibt an, wie viele Objekte in der Datei gespeichert und somit in der Offset Tabelle kodiert sind.
0x10	8	Offset des Wurzel Objekts der Offset Tabelle (markerOffsettable) Der Wert gibt das erste Element der Offset Tabelle an, von dem die Objekte gezählt werden. Normalerweise 0.
0x18	8	Offset der Offset Tabelle (offsetot) Absolute Angabe zur Offset Tabelle innerhalb der Datei

Tabelle 3: Bedeutung Werte im Trailer (eigene Darstellung) [KAR17]

2.4.1.3 Offset Tabelle (Variable Größe)

com.apple.captive.probe.plist

```
Offset(h)  00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
00000000   62 70 6C 69 73 74 30 30 D4 01 02 03 04 05 06 07  bplist00Ô.......
00000010   09 5B 52 65 66 72 65 73 68 44 61 74 65 57 56 65  .[RefreshDateWVe
00000020   72 73 69 6F 6E 5C 48 6F 73 74 4E 61 6D 65 4C 69  rsion\HostNameLi
00000030   73 74 5E 45 78 70 69 72 61 74 69 6F 6E 44 61 74  st^ExpirationDat
00000040   65 33 41 BC 7B 6C 1C 2E 58 B4 10 01 A1 08 5F 10  e3A¼{l..X´..¡._.
00000050   11 63 61 70 74 69 76 65 2E 61 70 70 6C 65 2E 63  .captive.apple.c
00000060   6F 6D 33 41 BC A2 F9 1C 2E 58 B4 08 11 1D 25 32  om3A¼¢ù..X´...%2
00000070   41 4A 4C 4E 62 00 00 00 00 00 00 01 01 00 00 00  AJLNb...........
00000080   00 00 00 00 0A 00 00 00 00 00 00 00 00 00 00 00  ................
00000090   00 00 00 00 6B                                   ....k
```

Abbildung 7: BPList Offset Tabelle (eigene Darstellung)

Der dritte Abschnitt enthält die einzelnen Offsets der Objekte in der Objekt-Tabelle. Ein Offset ist dabei solang, wie im Trailer unter der offsize festgelegt. In Abbildung 7 ist in Offset 0x7B herauszulesen, dass 1 Byte ausreicht, weswegen jedes Byte in der Offset-Tabelle 1 Objekt referenziert. Die Objekte werden dabei intern inklusive der 0 durchnummeriert. Die Anzahl entspricht dabei dem Wert, der im Trailer an Offset 0x08 festgelegt ist [KAR17].

2.4.1.4 Objekt Tabelle (Variable Größe)

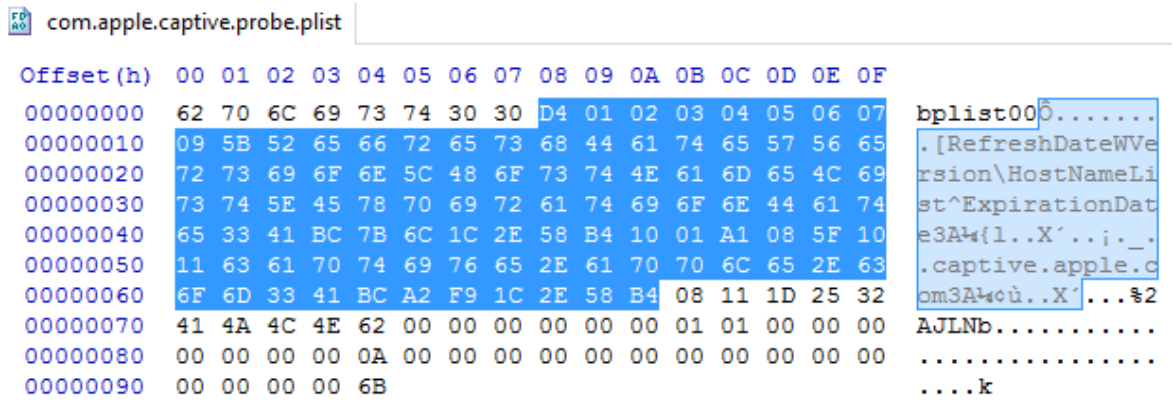


Abbildung 8: BPList Objekt Tabelle (eigene Darstellung)

In Abbildung 8 ist ersichtlich, dass dieser Bereich in den allermeisten Fällen den größten der Datei darstellt. Hier sind die eigentlichen gespeicherten Informationen in verschiedenen Dateitypen hinterlegt. Einige dieser Dateitypen kodieren die Informationen in nur einem Byte, wie z.B. die beiden Bool Typen, andere geben die Länge der darauffolgenden Bytes an, wieder andere sind Containerformate, die andere Objekte beinhalten. Das erste Objekt der Objekt-Tabelle ist dabei fast immer ein Dictionary, Arrays und Sets sind selten und andere Dateitypen können nur theoretisch als erstes Objekt auftreten. Dies bedeutet automatisch, dass alle Objekte innerhalb der Tabelle in Containerformaten angeordnet sind, sodass eine hierarchisch geordnete Unterteilung erfolgen kann [KAR17].

```
Object Formats (marker byte followed by additional info in some cases)
null    0000 0000               // null object [v"1?"+ only]
bool    0000 1000               // false
bool    0000 1001               // true
url     0000 1100   string      // URL with no base URL, recursive encoding of URL string [v"1?"+ only]
url     0000 1101   base string // URL with base URL, recursive encoding of base URL, then recursive
                                encoding of URL string [v"1?"+ only]
uuid    0000 1110               // 16-byte UUID [v"1?"+ only]
fill    0000 1111               // fill byte
int     0001 0nnn   ...         // # of bytes is 2^nnn, big-endian bytes
real    0010 0nnn   ...         // # of bytes is 2^nnn, big-endian bytes
date    0011 0011   ...         // 8 byte float follows, big-endian bytes
data    0100 nnnn   [int]   ... // nnnn is number of bytes unless 1111 then int count follows,
                                // followed by bytes
string  0101 nnnn   [int]   ... // ASCII string, nnnn is # of chars, else 1111 then int count, then bytes
string  0110 nnnn   [int]   ... // Unicode string, nnnn is # of chars, else 1111 then int count, then
                                // big-endian 2-byte uint16_t
string  0111 nnnn   [int]   ... // UTF8 string, nnnn is # of chars, else 1111 then int count, then
                                // bytes [v"1?"+ only]
uid     1000 nnnn   ...         // nnnn+1 is # of bytes
        1001 xxxx               // unused
array   1010 nnnn   [int]   objref* // nnnn is count, unless '1111', then int count follows
ordset  1011 nnnn   [int]   objref* // nnnn is count, unless '1111', then int count follows [v"1?"+ only]
set     1100 nnnn   [int]   objref* // nnnn is count, unless '1111', then int count follows [v"1?"+ only]
dict    1101 nnnn   [int]   keyref* objref* // nnnn is count, unless '1111', then int count follows
        1110 xxxx               // unused
        1111 xxxx               // unused
```

Abbildung 9: Dateitypen im BPList File und gleichzeitig einzige Spezifikation vom Hersteller zur Speicherung innerhalb der Datei [APPL14_1]

In der folgenden Tabelle wird auf die meisten Objekttypen aus Abbildung 9 eingegangen. Auf die fehlenden Objekttypen url, uuid, UTF8 String und orderset kann nicht genauer eingegangen werden, da diese erst ab Version 1 eingebunden werden und aktuell in der CoreFoundation noch nicht implementiert sind. Eine andere Spezifikation der Umsetzung war im Bearbeitungszeitraum nirgends zu finden, weswegen eine Implementierung dieser Typen nicht möglich war.

Dateityp	Bedeutung
Bool	Wert der entweder Wahr oder Falsch ist
Fill	Füll-Byte
Int	Ganzzahl Variable vom Typ Integer in Big Endian[12] Auch für die die Längenangabe der anderen Dateitypen verwendet Zahlen mit 1,2 und 4 Byte Länge sind niemals negativ und werden vorzeichenlos gespeichert. Nur 8 Byte Werte können negativ sein und werden vorzeichenbehaftet gespeichert[APPL14_1].
Real	Gleitkommazahl in 4 (Float) oder 8 Byte (Double) gemäß des IEEE 754[13] Standards
Date	Zeitpunkt, unabhängig von einem bestimmten Kalender oder Zeitzone [APPL18_1] Definition nach der Klasse NSDate der Apple CoreFoundation und einem absoluten Referenzdatum (00:00:00 UTC am 1. Januar 2001). Hierbei kann der Double Wert der Variable auf das Referenzdatum addiert werden, um den abgespeicherten Zeitpunkt zu erhalten [APPL14_2].

[12] Big Endian: Beschreibt die Anordnung einzelner Bytes. Bei Big Endian wird das höchstwertige Byte als erstes gespeichert, bei Little Endian das niederwertigste [FIS11, S. 292]

[13] IEEE 754: (Institute of Electrical and Electronics Engineers); Weltweites Institut zur Normgebung in den Bereichen Elektrotechnik und Informationstechnik. 754 ist die Norm zur Repräsentation von Fliesskommazahlen im Prozessor [FIS11, S. 416-417]

Data	Statischer Byte Buffer gemäß der NSData Klasse der Apple Foundation [APPL18_2] Daten Bytes, die nicht interpretiert werden
String ASCII	Ausgabe von Zeichen gemäß der ASCII Tabelle[14] (1 Byte lang)
String Unicode	Ausgabe von Zeichen gemäß der Unicode Tabelle[15] (2 Byte lang)
UID	Gemäß CFBinaryPList.c ein Integer Wert mit 1, 2, 4, oder 8 Byte Länge UID wird zur Speicherung in PLists genutzt. Apple selbst lässt in seinem Xcode[16] PList Viewer gespeicherte UID aber Werte nicht anzeigen
Array	Geordnete Sammlung mit wahlfreiem Zugriff [APPL18_3] Containerdatentyp, der auf andere Objekte innerhalb der Datei über die Offset Tabelle verweist.
Set	Ähnlich dem Array, stellt aber eine ungeordnete Sammlung von Daten dar [APPL18_4]
Dict	Eine Dictionary Auflistung, deren Elemente Schlüssel/Wert-Paare sind [APPL18_5] Ist ein Containerdatentyp, der andere Datentypen enthält und zum geordneten Aufbau dient. Dabei werden zuerst alle Schlüssel und danach alle Werte referenziert

Tabelle 4: Dateitypen der Objekt Tabelle (eigene Darstellung)

Zur Identifikation der einzelnen Objekttypen dient das erste Byte, das sogenannte Markierungsbyte, des in der Offset-Tabelle genannten Offsets. Dieses Byte unterteilt sich das linke Nibble[17], das den Objekttyp an sich definiert, und das rechte Nibble, das außer bei den Dateitypen mit dem Wert 0 immer die Länge der folgenden Bytes benennt. Falls die Länge mit einem 0xF kodiert ist, gibt erst das nächste Byte mit einem Integer-Zähler die Länge der folgenden Zeichen an. Der Integer-Zähler funktioniert dabei auf die gleiche Weise, als wenn ein Integer-Wert innerhalb der Objekt-Tabelle hinterlegt wird.

Integer Zähler Beispiel

Zur Verdeutlichung wird der ASCII-String „Mittweida“ betrachtet, der mit 0x59 kodiert wird. Die „5“ stellt das linke Nibble dar und kodiert den Objekttypen als ASCII-String gemäß Abbildung 9. Die zweite Ziffer, das rechte Nibble, gibt die Anzahl der folgenden Zeichen an (roter Rahmen in Abbildung 10). Bei dem String „Schönes Mittweida“ übersteigt die Zeichenlänge von 17 Zeichen

[14] ASCII (American Standard Code for Information Interchange); Weltweite 7 Bit Norm zur Repräsentation von 128 Zeichen [FIS11, S. 56]

[15] Unicode System-, Programm- und sprachunabhängiges Projekt, das sämtliche Zeichen in einem einzigen Zeichensatz abbildet, maximal 65536 mögliche Zeichen [FIS11, S. 939]

[16] Xcode: Apples integrierte API zum Entwickeln von Anwendungen, die auf Apple Betriebssystemen laufen [APPL16]

[17] Nibble: 4 Bit eines Bytes [FIS11, S. 612]

die Möglichkeit der Kodierung des rechten Nibble. Innerhalb der Objekt-Tabelle würde dieser String mit 0x5F 10 11 kodiert abgespeichert werden, dahinter folgen 17 Zeichen des ASCII-Strings. Im zweiten Byte steht das linke Nibble für den Objekttyp Integer, der immer eins ist. Das zweite Nibble gibt mit einem vielfachem von 2 die Anzahl der Bytes an, die zum Speichern des Integer-Wertes benötigt werden. In diesem Beispiel $2^0 = 1$. Ab dem dritten Byte ist der eigentliche Wert 17 enthalten und das System liest die folgenden 17 Byte, um den String auszugeben. Der Integer-Wert wird dabei wie alle numerisch gespeicherten Werte in Big Endian ausgelesen (Grüner Rahmen in Abbildung 10) [KAR17].

```
00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
59 4D 69 74 74 77 65 69 64 61 00 00 00 00 00 00  YMittweida......
5F 10 11 53 63 68 F6 6E 65 73 20 4D 69 74 74 77  _..Schönes Mittw
65 69 64 61                                      eida
```

Abbildung 10: Integer Zählvariable (eigene Darstellung)

Neben den Objekttypen, denen Byte-Sequenzen mit den Werten folgen, werden noch Container-Datentypen wie z.B. Array oder Dictionary zur Speicherung verwendet. In diesen Formaten folgen auf das erste Markierungsbyte Referenzen, die auf die Objektnummer in der Offset-Tabelle zeigen. Folgt auf das erste Byte und den möglichen Integer-Zähler die Zahl 3, so wird im nächsten Schritt das dritte Objekt aus der Offset Tabelle aufgerufen. Wieviel Bytes eine Containerangabe benötigt, wird durch die refsize im Trailer angegeben.

Beispiele Containertypen

0xD4 – Ein Dictionary mit 4 Schlüssel/Wert-Paaren. Hierbei folgen 8 Byte (wenn refsize = 1). Die ersten vier Byte geben dabei immer die einzelnen Schlüssel an und die nachfolgenden vier Byte den passenden Wert, sodass Byte 1 und Byte 5 nach dem Markierungsbyte jeweils ein Schlüssel/Wert-Paar ergeben. Falls der Wert wieder ein Containerformat ist, erfolgt das gleiche Verfahren verschachtelt.

0xA8 – Ein Array mit 8 Werten. Der Unterschied zum Dictionary besteht darin, dass es keine Schlüssel gibt, sondern nur 8 Werte, die nacheinander aus der Offset-Tabelle aufgerufen werden [KAR17].

Auch bei diesen Dateitypen kann durch eine 0xF Kodierung immer ein Integer-Zähler folgen. Dieses Verfahren ist anfangs etwas kompliziert, wird aber im folgenden Beispiel deutlich.

2.4.1.5 Beispiel BPList auslesen

Die Erklärung und das Verstehen eines Binary Property List Files ist bereits sehr verständlich im Blog des IOS Entwicklers Christos Karaiskos erläutert. Beim Aufzeigen dieses Beispiels wurde sich stark an seine Ausführungen angelehnt [KAR17].

Als Beispiel wird weiterhin das Apple File „com.apple.captive.probe.plist“ verwendet. Bereits in Kapitel 2.4.1.1 wurde erläutert, dass es sich um ein BPList File der Version 00 handelt. Als nächstes wird der Trailer analysiert, woraus folgende Basisinformationen herauszulesen sind:

```
Offset(h)  00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F

00000000   62 70 6C 69 73 74 30 30 D4 01 02 03 04 05 06 07   bplist00Ô.......
00000010   09 5B 52 65 66 72 65 73 68 44 61 74 65 57 56 65   .[RefreshDateWVe
00000020   72 73 69 6F 6E 5C 48 6F 73 74 4E 61 6D 65 4C 69   rsion\HostNameLi
00000030   73 74 5E 45 78 70 69 72 61 74 69 6F 6E 44 61 74   st^ExpirationDat
00000040   65 33 41 BC 7B 6C 1C 2E 58 B4 10 01 A1 08 5F 10   e3A¼{l..X´..¡._.
00000050   11 63 61 70 74 69 76 65 2E 61 70 70 6C 65 2E 63   .captive.apple.c
00000060   6F 6D 33 41 BC A2 F9 1C 2E 58 B4 08 11 1D 25 32   om3A¼¢ù..X´...%2
00000070   41 4A 4C 4E 62 00 00 00 00 00 00 01 01 00 00 00   AJLNb...........
00000080   00 00 00 00 0A 00 00 00 00 00 00 00 00 00 00 00   ................
00000090   00 00 00 00 6B                                    ....k
```

Abbildung 11: Trailer auslesen Beispiel (eigene Darstellung)

Offset-Tabellen Offset Größe:	0x01 Byte
Objekt-Referenzierungsbyte Länge:	0x01 Byte
Anzahl der gespeicherten Objekte:	0x0A
Offset-Wurzel Objekt Offset-Tabelle:	0x00
Offset der Offset-Tabelle:	0x6B

Nach der Analyse der Basisinformationen aus dem Trailer wird an Offset 0x6B die Offset-Tabelle 10 Byte lang mit den Angaben der Offsets von den jeweiligen Objekten ausgelesen. In Abbildung 12 sind dabei gleiche Farben mit den Markern der einzelnen Objekte in der Objekt-Tabelle markiert.

```
Offset(h)  00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F

00000000   62 70 6C 69 73 74 30 30 D4 01 02 03 04 05 06 07   bplist00Ô.......
00000010   09 5B 52 65 66 72 65 73 68 44 61 74 65 57 56 65   .[RefreshDateWVe
00000020   72 73 69 6F 6E 5C 48 6F 73 74 4E 61 6D 65 4C 69   rsion\HostNameLi
00000030   73 74 5E 45 78 70 69 72 61 74 69 6F 6E 44 61 74   st^ExpirationDat
00000040   65 33 41 BC 7B 6C 1C 2E 58 B4 10 01 A1 08 5F 10   e3A¼{l..X´..¡._.
00000050   11 63 61 70 74 69 76 65 2E 61 70 70 6C 65 2E 63   .captive.apple.c
00000060   6F 6D 33 41 BC A2 F9 1C 2E 58 B4 08 11 1D 25 32   om3A¼¢ù..X´...%2
00000070   41 4A 4C 4E 62 00 00 00 00 00 00 01 01 00 00 00   AJLNb...........
00000080   00 00 00 00 0A 00 00 00 00 00 00 00 00 00 00 00   ................
00000090   00 00 00 00 6B                                    ....k
```

Abbildung 12: Offset Tabelle auslesen Beispiel (eigene Darstellung)

Objekt 0:	0xD4	Dictionary mit 4 Schlüssel/Wert-Paaren	
Objekt 1:	0x5B	ASCII-String mit 11 Zeichen	Inhalt: „*RefreshDate*“
Objekt 2:	0x57	ASCII-String mit 7 Zeichen	Inhalt: „ *Version*“
Objekt 3:	0x5C	ASCII-String mit 12 Zeichen	Inhalt: „ *HostNameList*“
Objekt 4:	0x5E	ASCII-String mit 14 Zeichen	Inhalt: „ *ExpirationDate*“
Objekt 5:	0x33	Datum mit 8 folgenden Bytes	Umrechnung:„*22.02.2016 17:17:32*“
Objekt 6:	0x10	Integer mit 1 folgenden Byte	Wert: „*1*“
Objekt 7:	0xA1	Array mit einem Element	
Objekt 8:	0x5F	ASCII String mit 17 Zeichen	Inhalt: „*captive.apple.com*“
Objekt 9:	0x33	Datum mit 8 folgenden Bytes	Umrechnung:„*23.03.2016 17:17:32*“

Die obere Darstellung zeigt alle gespeicherten Objekte der Datei. Dabei werden die Objekte der Reihe nach aufgezählt, was noch keine geordnete Unterteilung zulässt. Es ist nicht eindeutig zuzuordnen, welcher Wert zu welchem Schlüssel gehört. Dieses Problem wird mit den Containerformaten gelöst und so für eine klare Strukturierung der Datei gesorgt. Zur Verdeutlichung dieses Verfahrens wird in Abbildung 13 beispielhaft die Verarbeitung des Dictionary Objektes aufgezeigt. Dabei wird das erste Schlüssel/Wert-Paar ermittelt. Es kann hier nicht, wie im Beispiel sonst grundsätzlich, davon ausgegangen werden, dass die Objekte der Reihe nach gespeichert sind.

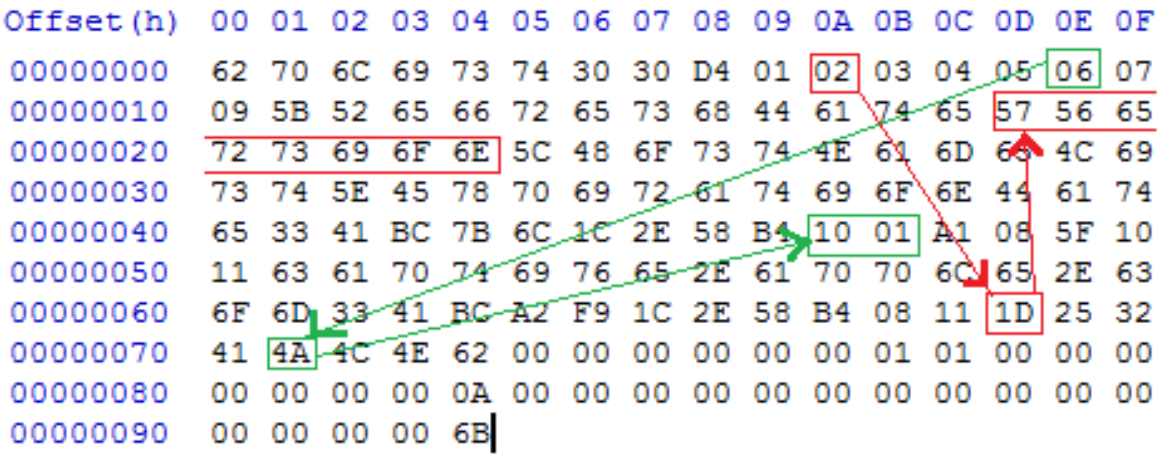


Abbildung 13: Auslesen Dictionary Beispiel (eigene Darstellung)

Wendet man dieses Verfahren auf die komplette Datei an, bekommt man folgende strukturierte Informationen als Ergebnis. In Abbildung 14 ist die Ausgabe mit Apples eigenem BPList Viewer Xcode zu sehen. Apple nimmt bei der Interpretation der Datei noch eine Sortierung der Dateitypen vor.

Dictionary mit 4 Objekten:		
Refresh Date:	22.02.2016 17:17:32	
Version:	1	
HostNameList:	Array Objekt Nr. 1:	captive.apple.com
ExpirationDate:	23.03.2016 17:17:32	

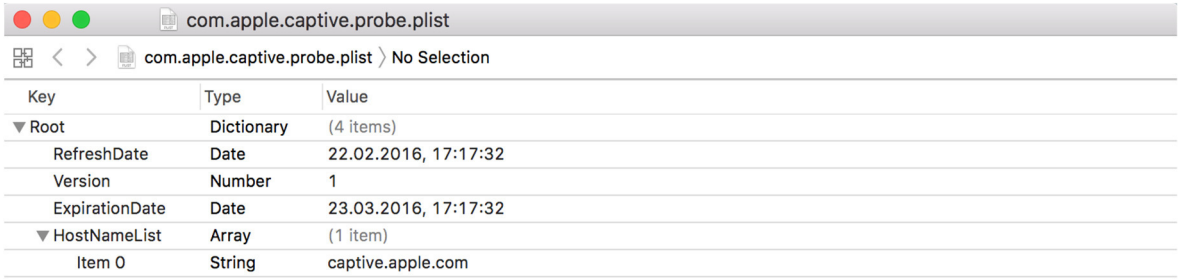


Abbildung 14: Ausgabe Beispiel mit PList Viewer Xcode (eigene Darstellung)

Beim Betrachten dieses Beispiels ist gut zu erkennen, dass nur acht Informationen in der Datei gespeichert sind. Die fehlenden zwei Informationen sind die Containertypen, die der hierarchischen Strukturierung dienen. Eine Komprimierung bzw. das mehrmalige Aufrufen eines Objekts sind in diesem Beispiel nicht enthalten.

3 Methoden

Nach den ersten Versuchen mit der X-Ways API wurde klar, dass für die Erfüllung des Ziels zwei Probleme gelöst werden müssen. Zum einen muss eine funktionierende DLL in X-Ways implementiert und zum anderen ein Programmkonzept entwickelt werden, das erweiterbar für neue Dateitypen ist, aber auch den rekursiven Aufbau der PList Files unterstützt. Auf diese beiden Punkte wird in diesem Kapitel eingegangen. Während das Programmkonzept in der Regel recht allgemein und auf verschiedene Programmiersprachen anwendbar ist, hat sich während der Ausarbeitung dieses Projekts und der Kommunikation mit anderen Entwicklern herausgestellt, dass trotz der Online-Dokumentation viel Zeit benötigt wird, eine X-Tension in X-Ways funktionsfähig zu programmieren. Aus diesem Grund soll im Folgenden erklärt werden, welche Maßnahmen ergriffen werden müssen, damit die X-Tension von XWFo akzeptiert wird.

Zur Erstellung einer DLL existieren plattformübergreifend unterschiedliche Programmierumgebungen. Da diese Arbeit sich in der Windows-Welt bewegt, wurde für die Umsetzung auf Visual Studio 2017 zurückgegriffen. Die API selbst ist in C geschrieben und auch die meisten anderen frei verfügbaren Erweiterungen sind in C umgesetzt. Aus diesem Grund wurde für die hier entwickelte DLL ebenfalls C verwendet.

Des Weiteren wurde mit folgenden Systemkonfigurationen gearbeitet:

Betriebssystem:	Windows 10 Enterprise 64 Bit
Programmierumgebung:	Microsoft Visual Studio 2017 Version 15.7.4
Microsoft.NET Framework:	Version 4.6.01586
X-Ways Forensics:	Version 19.1 SR-3 x86

3.1 Erstellung einer X-Ways X-Tension

In diesem Kapitel wird anhand einer Schritt-für-Schritt-Anleitung aufgezeigt, welche Schritte notwendig sind, eine DLL zu erstellen, zu befüllen und in X-Ways einzubinden.

1. Starten von Microsoft Visual Studio und Erstellen eines neuen DLL Projekts. (Abbildung 15)

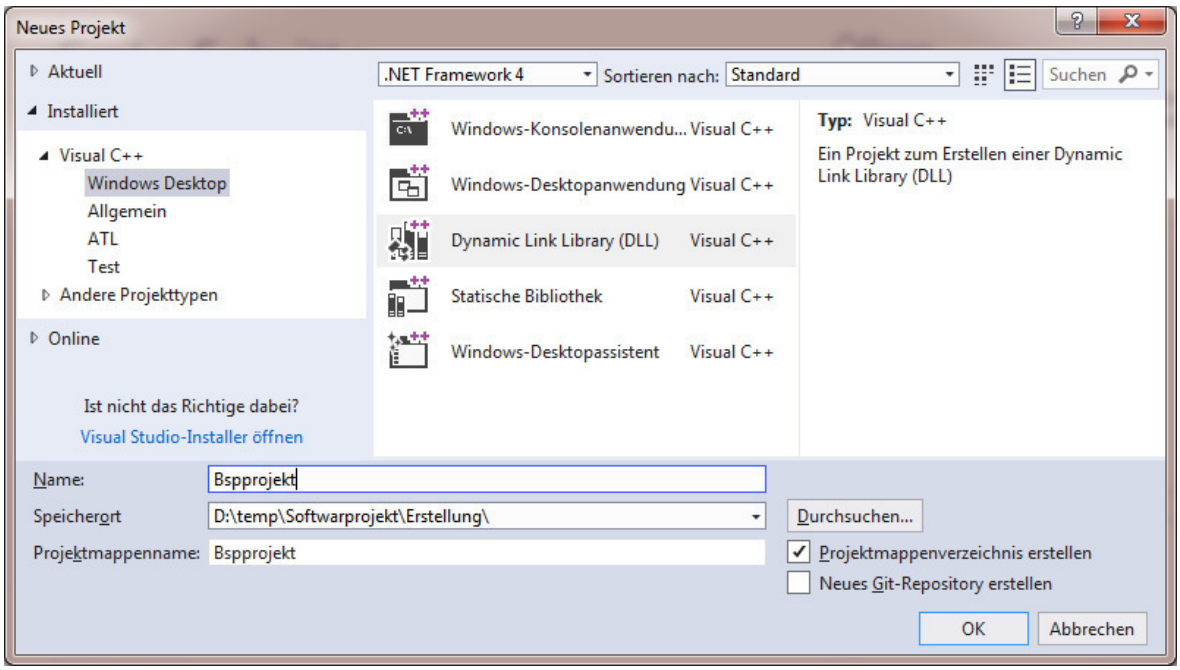


Abbildung 15: Erstellen eines neuen Projekts (eigene Darstellung)

2. In den Einstellungen des Projekts unter dem Punkt C/C++ die Verwendung von vorkompilierten Headern deaktivieren. (Abbildung 16)

Der vorkompilierte Header dient dazu, den statischen und unveränderbaren Quellcode, wie z.B. Systembibliotheken, nicht bei jedem Kompiliervorgang abarbeiten zu müssen. Die Dateien stdafx.cpp und stdafx.h legen dabei fest, was als statischer Code betrachtet wird und wohin er gespeichert werden soll [MIC18_2]. Da dieses Projekt aber kaum Abhängigkeiten benötigt, kann darauf verzichtet werden.

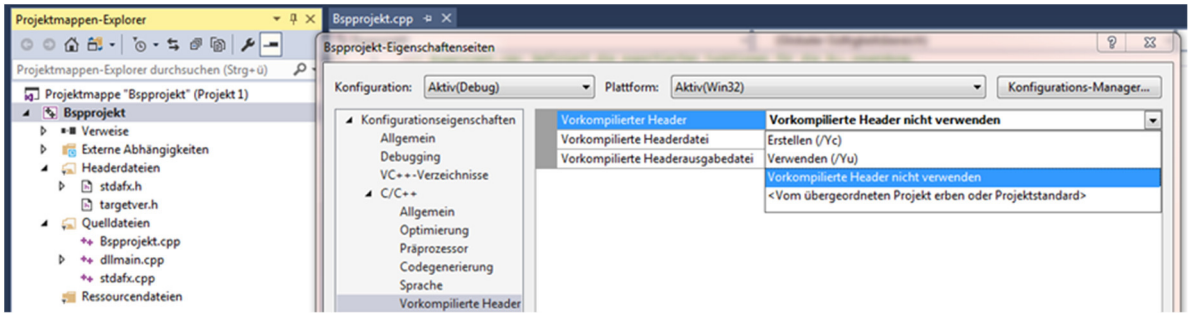


Abbildung 16: Verwendung des Vorkompilierten Headers deaktivieren (eigene Darstellung)

3. Im Projektexplorer alle Quell- und Headerdateien bis auf die Datei Bspprojekt.cpp löschen. (Abbildung 17)

Die Dateien targetver.h und dllmain.cpp sind für Versionsangaben und Einstiegspunkte vorgesehen und werden für eine X-Ways X-Tension nicht benötigt. Bspprojekt.cpp enthält den dynamischen Code und ist damit die Hauptdatei der X-Tension.

4. Eine neue Moduldefinitionsdatei unter „Quelldateien → Hinzufügen → Neues Element → Hilfsprogramm → Textdatei“ erstellen und die Endung in „.def“ ändern. (Abbildung 17)

Die Moduldefinitionsdatei versorgt den Linker[18] mit Informationen. Alle XT* Funktionen, die aus der X-Ways API im Quellcode aufgerufen werden, müssen darin unter Exports definiert werden. (Abbildung 18)

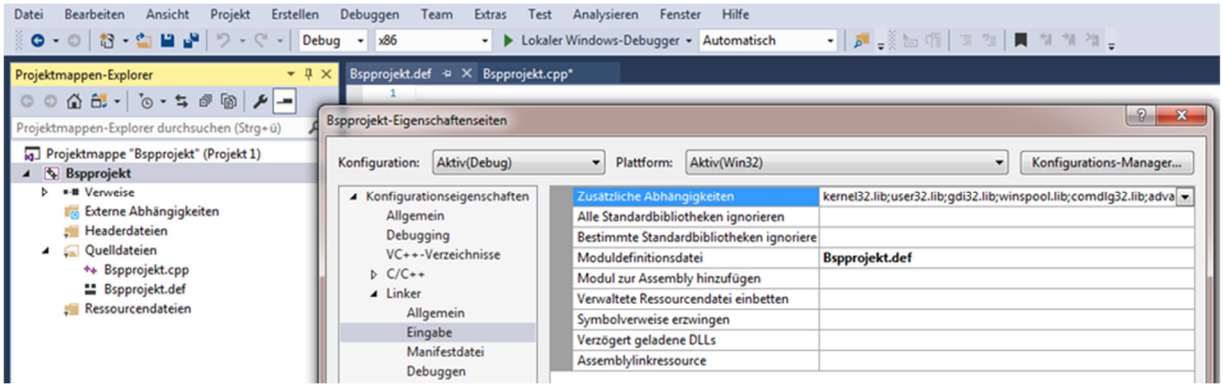


Abbildung 17: Moduldefinitionsdatei definieren (eigene Darstellung)

5. Die Moduldefinitionsdatei in den Einstellungen definieren.
   Rechtsklick Projekt → Eigenschaften → Linker → Eingabe → Moduldefinitionsdatei → Name der Definitionsdatei eingeben. (Abbildung 17)

6. Im Explorer die X-Tension.h und die X-Tension.cpp in den Projektordner kopieren. Optional kann auch die stdafx.h, targetver.h und dllmain.cpp gelöscht werden.

7. Hinzufügen der X-Tension.h und X-Tension.cpp.
   Projekt → Vorhandenes Element hinzufügen (Abbildung 18)

[18] Linker: der Linker dient zum Verbinden von Bibliotheken und Objektdateien, um z.B. eine DLL zu erstellen [MIC16_1]

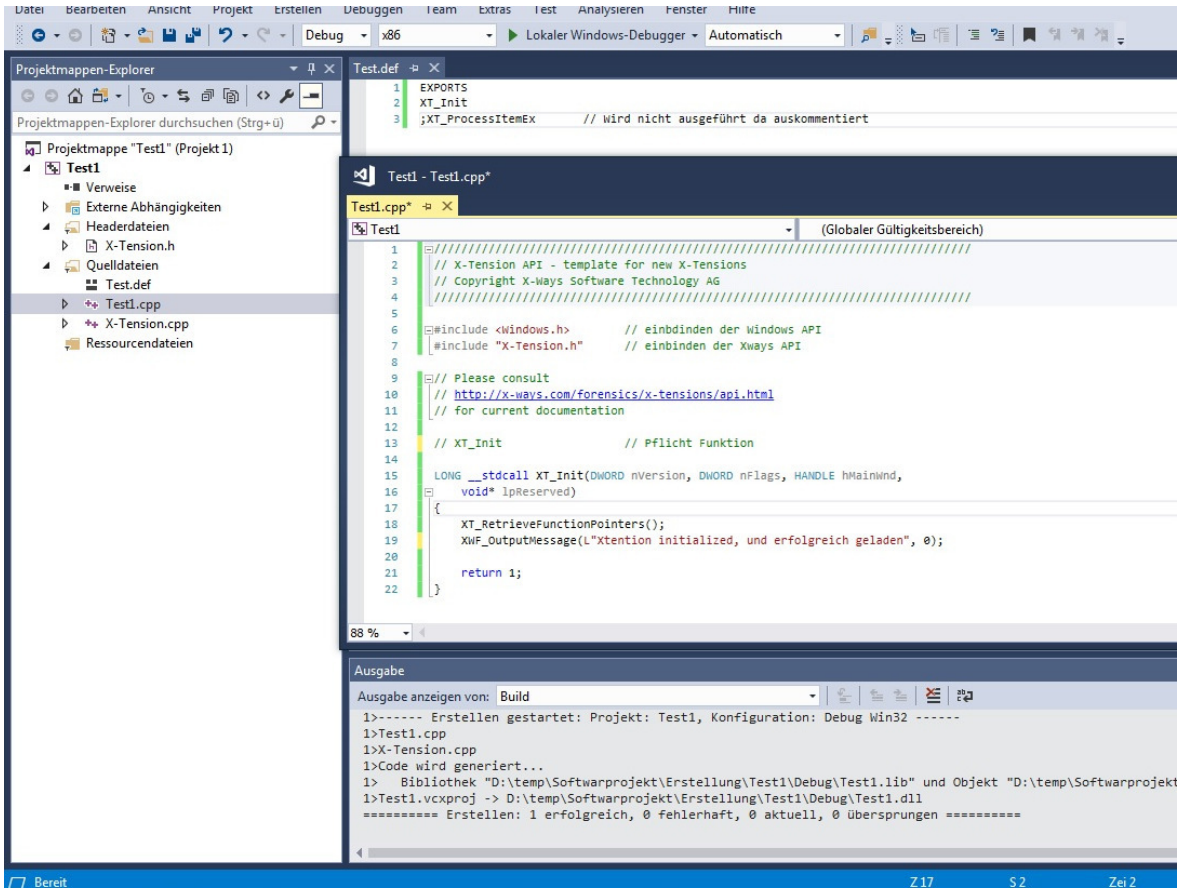


Abbildung 18: Projekt mit allen benötigten Dateien im Projektmappen-Explorer (eigene Darstellung)

8. Die beiden Projektdateien gemäß der X-Ways Dokumentation bzw. Abschnitt 2.3 befüllen. Die #include Anweisung sorgt dafür, dass die X-Ways API in das Programm eingebunden wird. (Abbildung 18)
9. Die DLL über „Erstellen → Projekt Erstellen“ erstellen.
10. DLL in den X-Ways Programmordner kopieren und als X-Tension, wie in Abschnitt 2.3.1 beschrieben, laden.

Nachdem die Programmierung abgeschlossen ist, sollte beim Erstellen der DLL die Projektmappenkonfiguration von „Debug“ auf „Release“ umgestellt werden. Dadurch werden einige Abhängigkeiten zusätzlich in die DLL integriert und die X-Tension auf anderen Systemen nutzbar. Außerdem sollte auf den Hinweis der Herstellerseite geachtet werden, X-Tension-DLLs mit dem Kürzel „XT_“ zu beginnen.

Damit die erstellten Erweiterungen auch auf Windows Systemen funktionieren auf denen sich kein Visual Studio befindet, ist es gemäß der „!readme.txt“ im Viewer-Ordner von XWFo erforderlich die „Microsoft Visual C++ Redistributable Packages“[19] auf dem Zielsystem zu installieren. Auf

[19] Redistributable Packages installieren Laufzeitkomponenten von Bibliotheken die zum Ausführen der Anwendung auf dem Computer benötigt werden [MIC18_3]

den meisten aktuellen Systemen ist das Package bereits enthalten. Sollte dies nicht der Fall sein besteht alternativ die Möglichkeit die fehlenden DLLs in den Programmordner von XWFo zu integrieren. Welche DLLs das betrifft, hängt von dem Quellcode und den benutzten Bibliotheken innerhalb der X-Tension ab. Die benötigten Bibliotheken sind im Vorschaufenster von XWFo unter der Import Table gelistet. Diese können beispielsweise aus dem Redistributable Packages extrahiert und anschließend ins Hauptverzeichnis kopiert werden.

3.2 Programmumsetzung

Nachdem das Grundgerüst der X-Tension funktioniert, musste die eigentliche Umsetzung des Programms realisiert werden. Die Bereiche Header, Trailer und Offset-Tabelle können noch recht statisch und linear abgearbeitet werden. Bei der Objekt Tabelle ist es durch die nahezu unbegrenzt tiefe Verschachtelung der einzelnen Containerobjekte komplizierter, dies programmtechnisch und visuell umzusetzen.

Durch das Springen innerhalb der Datei musste ein Konzept entwickelt werden, welches es ermöglicht, das Parsen jedes Dateityps jederzeit aufrufen zu können. Außerdem muss das Programm und vorzugsweise auch der Nutzer zu jeder Zeit wissen, in welcher hierarchischen Ebene der Containerdatei es sich befindet.

3.2.1 Aufruf der XT* Funktionen

Wie in Abschnitt 2.3.1 erläutert muss die DLL-Datei als Viewer X-Tension in XWFo unter „Optionen → Viewer-Programme“ eingebunden werden. Im Anschluss ruft X-Ways die `XT_Init` zur Initialisierung der DLL auf und führt den darin enthaltenen Code aus. In Codezeile 66 ist ersichtlich, dass nur die Information der erfolgreichen Initialisierung an den Nutzer übergeben werden. Auf weitere interne Informationen wurde in dieser ersten Version verzichtet. Durch die `#define`-Anweisung in Zeile 23 reicht der Begriff `EXPORT` für den Aufruf der XT*-Funktionen aus, damit sie vom Programm gefunden und korrekt gesteuert werden können.

```
23  #define EXPORT extern "C" __declspec(dllexport)

61  EXPORT LONG __stdcall XT_Init(DWORD nVersion, DWORD nFlags, HANDLE hMainWnd,
62  void* lpReserved)
63  {
65    XT_RetrieveFunctionPointers();
66    XWF_OutputMessage(L"Viewer X-Tension BPList Parser von Kittan Michael v0.1 gela-
     den",
      0);
67    return 1;
68  }
69  EXPORT PVOID __stdcall XT_View(HANDLE hItem, LONG nItemID, HANDLE hVolume, HANDLE
    hEvidence, PVOID lpReserved, PINT64 nResSize)

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

Codeblock 2: Aufruf der Funktionen

In der `XT_View` beginnt der eigentliche Code zur Interpretation der PList Files. Alles, was in dieser Funktion enthalten ist, wird im Vorschaufenster von XWFo verarbeitet, nachdem eine Datei angeklickt wurde. Mit dem Auswählen einer neuen Datei wird in Zeile 692 die letzte XT-Funktion aufgerufen und der benutzte Speicher wieder freigegeben oder die Variablen auf die Ursprungswerte zurückgesetzt.

3.2.2 Auslesen von Header, Trailer und Offset Tabelle

Ein Binary Property List File besitzt innerhalb der ersten 8 Byte einen Header. Dieser kann auch als Magic Number beschrieben und zur Identifizierung genutzt werden. Dies ist besonders wichtig um X-Ways zu signalisieren, ob die ausgewählte Datei von der X-Tension interpretiert oder verworfen werden muss.

Der Wert in Zeile 80 `nResSize` signalisiert der `XT_View` die Zuständigkeit. Mit dem Setzen des Wertes auf -1 können mehreren Überprüfungen durchgeführt werden. Sobald eine `return` Anweisung ausgeführt wird und der Wert vorher gesetzt wurde, weiß XWFo, dass die X-Tension nicht für diese Datei zuständig ist. Die eigentliche Prüfung erfolgt in den Zeilen 82-88 mithilfe der Funktionen `XWF_Read` und `strcmp`.

```
79  ////////////////////// Prüfen X-Tension Zuständig ////////////////////
80  *nResSize = -1;
81
82  char sig_buffer[9] = {};
83  XWF_Read(hItem, 0, (BYTE*)sig_buffer, 8);
84
85  if (strcmp("bplist00", sig_buffer) != 0)
86  {
87     return NULL;
88  }
90  wchar_t *buffer = (wchar_t*)calloc(SpeicherZugeordnet,
91  sizeof(wchar_t));
103 if (file_size <= 40)
105    *nResSize = 150;
106    wsprintf(buffer, L"ERROR: Dieses BPLIST File scheint Fehlerhaft zu sein(Kleiner
       als 40 Byte)");
107    return buffer;
```

Codeblock 3: Prüfen der Zuständigkeit

Aufgrund dessen, dass in diesem Projekt nur BPLists der Version 00 ausgelesen werden, wird die Übereinstimmung der ersten 8 Bytes überprüft. Wegen der besonderen Struktur der PList Files ist es nicht möglich, dass die zu interpretierende Datei kleiner als 40 Byte ist, weswegen eine Überprüfung in Zeile 103 durchgeführt wird. Ein vollständiges BPList File ist ohne Informationen 42 Bytes groß. In diesem ist nur ein Objekt gespeichert, ein Containerformat, welches keine eigenen Objekte beinhaltet. In Abbildung 19 ist an Offset 0x08 ein solches Dictionary zu erkennen, danach folgt direkt die Offset-Tabelle. Diese Überprüfung auf unvorhersehbare Fehler erfolgt auch an vielen anderen Stellen des Quellcodes, um Programmabstürze zu vermeiden. Der in Codezeile 90 definierte dynamische `buffer` wird, wie für alle anderen Informationen, als Rückgabewert an

XWFo verwendet. Es ist möglich, ihn mehrmals zu erweitern oder zu verkleinern, damit auch größere Dateien verarbeitet werden können ohne den Speicher fest vorhalten zu müssen. Taucht ein Fehler auf, wird dieser innerhalb des Vorschaufensters angezeigt. Dabei ist es sowohl möglich, dass nur ein bestimmter Dateityp übersprungen wird, als auch, dass wie in diesem Beispiel nur der Fehler ausgegeben wird.

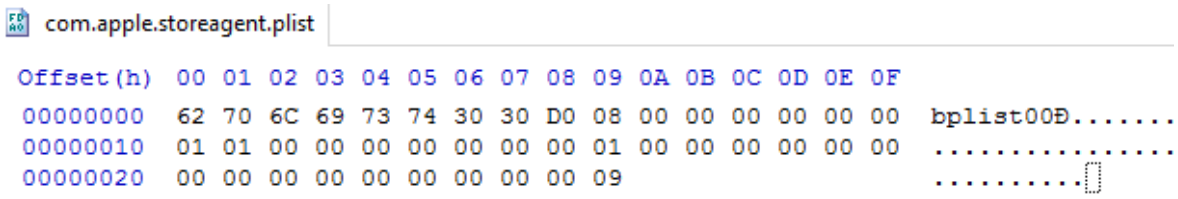


Abbildung 19: BPList File ohne Inhalt (eigene Darstellung)

Nachdem der Header vollständig ausgewertet wurde, folgt die Interpretation des Trailers. Unabhängig vom Trailer lässt sich feststellen, dass grundsätzlich in den einzelnen Programmteilen Speicher durch eine Heap[20]-Funktion reserviert und danach durch die XWF-Funktion Read mit dem benötigten Inhalt der Datei byteweise befüllt wird.

```
121 HANDLE heap = GetProcessHeap();
125 BYTE *trailer = (BYTE*)HeapAlloc(heap, 0, file_size);
126
127 XWF_Read(hItem, file_size - 32, trailer, file_size);
129 // Null Bytes von index 0 -5
130 offsize = trailer[6];
131 refsize = trailer[7];
133 // numobjekt                  Anzahl der Kodierten Objekte
134 ptr = &trailer[8];            // Zeiger zum auslesen des 64 Bit Wertes
135 numobjekt = _byteswap_uint64(*((uint64_t *)ptr));

153 HeapFree(heap, 0, trailer);          // Speicher vom Trailer wieder Freigeben
```

Codeblock 4: Auslesen Trailer, Auszug

Obwohl die ersten 5 Bytes des Trailers nicht befüllt sind, werden diese der Vollständigkeit halber in Zeile 127 eingelesen. Die Variablen `offsize`, `refsize`, `numobjekt` und `markerOfsettable` sind global definiert, da diese auch in ausgelagerten Funktionen zur Berechnung noch benötigt werden. Das Offset der Offset-Tabelle wird ausschließlich im nächsten Schritt benötigt, weswegen ein lokales Deklarieren ausreicht. Die Funktion `_byteswap_uint64` in Codezeile 135 tauscht den 8 Byte langen Werte von Big zu Little Endian. Abschließend wird der Speicher wieder freigegeben. Die restlichen Elemente des Trailers werden auf die gleiche Weise ausgelesen, weswegen im Codeblock 4 auf die restlichen Zeilen verzichtet wurde.

[20] Heap: Speicherbereich für Daten, der bei Bedarf zur Laufzeit des Programms angelegt wird [BÖT99, S. 220]

```
165 uint64_t markeroffset[4500];
173 for (int i = 0; i < numobjekt; i++)
174 {
179   else if (offsize == 2)                          // 16 Bit lesen
180   {
181         markeroffset[i] = (uint64_t)(((uint16_t *)file_buf)[i]);
182         markeroffset[i] = _byteswap_ushort((uint16_t)markeroffset[i]);
183   }
```

Codeblock 5: Auslesen Offset Tabelle, Auszug

Die Offset-Tabelle enthält alle Startoffsets der gespeicherten Objekte innerhalb der Datei. Um diese auszulesen, werden die gespeicherten Informationen des Trailers benötigt. Nachdem der benötigte Bereich mit `XWF_Read` in einen temporären Buffer gespeichert wurde, wird die Offset-Tabelle mit einer Schleife Objekt für Objekt ausgelesen und in einem Array gespeichert. In Zeile 179 ist beispielhaft ersichtlich, dass die unterschiedlichen Werte der `offsize` mitbetrachtet werden. In dieser Version wird die maximale Anzahl der Objekte aufgrund der Performance auf 4.500 reduziert. Das Array `markeroffset` ist zum weiteren Auslesen der Datei essentiell und beherbergt danach alle Offsets der Objekte innerhalb der Datei.

3.2.3 Formatieren der Ausgabe

Nachdem die ersten drei Teile der Datei ausgewertet und gespeichert wurden, muss sich vor dem Verarbeiten der Objekt-Tabelle Gedanken über eine strukturierte Ausgabe gemacht werden. Dafür wird der im Codeblock 3 deklarierte `buffer` genutzt. Um diesen jederzeit mit Informationen befüllen und erweitern zu können, wurde die Funktion „Ausgabe“ integriert. Darin wird bei jedem Aufrufen überprüft, ob die Bytes, die hinzugefügt wurden, den bisherigen reservierten Speicher überschreiten. Ist dies der Fall, wird der Speicher um den Faktor 2 vergrößert. Im Anschluss wird in Zeile 1177 mit `wmemcpy` der neue Inhalt an den `buffer` angehangen und zum Schluss die aktuelle `Speichernutzung` aktualisiert. Dadurch ist es möglich, unbegrenzt Daten zum `buffer` hinzuzufügen, was die benötigte Flexibilität sicherstellt. In den folgenden Codezeilen ist der Quellcode der Ausgabemethode kommentarbereinigt dargestellt.

```
1157 wchar_t *Ausgabe(wchar_t *buffer, const wchar_t* Zeile, wchar_t* Ausgabebuf, int
     zeichencount){
1160   size_t Zeilenlänge = wcslen(Zeile)+zeichencount;
1163   while (Speichernutzung + Zeilenlänge >= SpeicherZugeordnet)
1164   {
1166         SpeicherZugeordnet *= 2;
1169         buffer = (wchar_t *)realloc(buffer, SpeicherZugeordnet* sizeof
           (wchar_t));
1170         if ((buffer == NULL))
1171         {
1173               return NULL;
1174         }
1175   }
1177 wmemcpy(buffer + Speichernutzung          // Zielbuffer + aktueller Inhalt
1178         , Ausgabebuf                      // Quellbuffer
1179         , Zeilenlänge);                   // Anzahl der zu kopierenden Zeichen
1182 Speichernutzung += Zeilenlänge;
1184 return buffer;
```

Codeblock 6: Funktion Ausgabe, Auszug

Die notwendigen Variablen `Speichernutzung` und `SpeicherZugeordnet` werden am Anfang des Programms global deklariert. Als Startwert des zu reservierenden Speichers wurden 512 Byte festgelegt. Die Aufrufparameter aus Zeile 1157 sind folgendermaßen definiert:

`buffer`: Speicherort der Daten; dient gleichzeitig auch als Rückgabewert
`Zeile`: wchar_t Array zum Zählen der übergebenen Zeichen ohne Werte
`Ausgabebuf`: wchar_t Array mit fester Größe zum Übergeben des kompletten Inhalts
`zeichencount`: Integer-Wert, indem z.B. die Zeichenzahl einer Dezimalzahl übergeben wird

Durch die mögliche Verschachtelung innerhalb eines Binary Property List Files ist es unerlässlich, die Ausgabe so zu gestalten, dass sie für den Nutzer intuitiv und leicht lesbar ist. Ein gesondertes Fenster, das sich beim Betrachten einer Datei öffnet, wäre beim Durchsehen mehrerer Dateien hinderlich, sodass die Seitenbeschreibungssprache HTML für die Umsetzung eingesetzt wurde. Dazu ist es notwendig, XWFo durch ein BOM[21] mitzuteilen, dass der folgende Code mit der zugehörigen internen Viewer-Komponente interpretiert wird. In den ersten zwei Byte muss dem Buffer dieses BOM übergeben werden. Dies ist das einzige Fall innerhalb der X-Tension, bei der der Buffer nicht über die definierte Ausgabemethode befüllt wird. Für UTF-16[22] lautet das BOM in Big Endian 0xFEFF [RIC16].

Nach dem BOM muss die HTML Datei gemäß der W3C[23]-Syntax befüllt werden. XWFo aktualisiert seine externen Viewer-Komponenten in sinnvollen Abständen und stellt diese auf der Website zur Verfügung. Aufgrund der Verschachtelungen von PList Files ist es notwendig, dem Nutzer anzuzeigen, in welcher Ebene die einzelnen Objekte angezeigt werden. Die Dateitypen werden dabei generell in einzelnen Tabellen angezeigt und, falls ein Containerobjekt endet oder beginnt, außerhalb der Tabelle ausgegeben. Für diese Ausgabe und Zählung der einzelnen Ebenen müssen einige Variablen global definiert werden.

```
48  wchar_t Stringbuffer[3000];  // Buffer für String-Ausgabe
53  int ebene = 0;               // Zähler für die Ebene
54  int ebenenlaenge[40];        // Speicherung der Länge für die Ebene
55  int stringlength = 0;        // Länge der einzelnen Strings
56  wchar_t tiefenanzeige[3000]; // Speicherung der kompletten Ebenenanzeige
57  size_t nutzungebene = 10;    // Anfangslänge der Ebenausgabe
58  bool table = 0;              // Tabellenausgabe ja oder nein
```

Codeblock 7: Variablen Definition für Hierarchische Ausgabe

[21] BOM (Byte Order Mark): Eine Bytefolge die am Anfang einer Website verwendet wird, um die Unicode Zeichenkodierung zu definieren [RIC16]

[22] UTF-16 (Universal Multiple-Octet Coded Character Set (UCS) Transformation Format for 16 Planes of Group 00): Stellt eine Computercodierung von Unicode dar [FIS11, S. 950]

[23] W3C (World Wide Web Consortium): Vergeben Normen und Standardisierungen im Internet, unter anderem HTML und XML [FIS11, S. 1012]

`Stringbuffer`:	Enthält den letzten gespeicherten String für eine aussagefähige Bezeichnung
`ebene`:	Zählt die Tiefe der Verschachtelung der einzelnen Containerobjekte
`ebenenlaenge`:	Speichert im Integer Array die Länge der benötigten Zeichen für jede aufgerufene Verschachtelung, um beim Verlassen des Containerobjekts die notwendigen Zeichen inklusive der Trennzeichen zurückrechnen zu können
`stringlength`:	Speichert die Länge des letzten übergebenen Strings
`tiefenanzeige`:	In diesem Array ist der komplette Inhalt der ausgegebenen Ebene gespeichert
`nutzungebene`:	definiert die Anfangslänge, diese bleibt auf 10 für den Ausgangsstring “ <b>Ebene:“
`table`:	Falls dieser Wert auf 1 gesetzt wird, wird dem Programm signalisiert, dass eine neue Tabelle aufgebaut werden muss, bei 0 wird dieser Schritt übersprungen

Tabelle 5: Bedeutung Variablen der Funktion Ausgabe (eigene Darstellung)

Nachdem die Variablen erklärt worden sind, zeigt der folgende Codeblock, wie die Ausgabe von Informationen in den Containerformaten organisiert wird. In den Codezeilen 776 bis 782 wird die Ausgabe der aktuellen Ebene dargestellt.

```
776  wsprintf(tiefenanzeige + nutzungebene, L"--> %s", Stringbuffer);
777  nutzungebene = nutzungebene + stringlength + 4;
778  buffer = Ausgabe(buffer, L"", tiefenanzeige, nutzungebene);
779  ebenenlaenge[ebene] = stringlength + 4;
780
781  wsprintf(Ausgabebuf, L"</b><br><br>");
782  buffer = Ausgabe(buffer, L"</b><br><br>", Ausgabebuf, 0);

801  if (table == 1)
802  {
803   buffer = Ausgabe(buffer, L"<table border=\"1\"><tr><th width=70> Item </th><th
      width=250>Wert</th> <th width=100>Typ</th><th width=550>Inhalt</th></tr>", Aus-
      gabebuf, 0);
805   table = 0;
806  }
```

Codeblock 8: Strukturierte Ausgabe (Auszug)

In Zeile 776 ist zu erkennen, dass beim Hereingehen in eine tiefere Ebene zur Abgrenzung folgender String verwendet wird „`-->`“. Im Anschluss folgt die Bezeichnung der neuen Ebene und die Daten werden an die Ausgabefunktion übergeben. Die neue Länge der Gesamtausgabe wird in Zeile 779 im Integer Array hinterlegt und durch den Befehl „`<br>`“ erfolgt eine visuelle Abgrenzung. Ab Codezeile 801 wird beispielhaft für ein Dictionary-Objekt der Aufbau einer neuen Tabelle anhand der HTML Befehle dargestellt. Für die Ausgabe der Ebenen wird eine Tabelle geschlossen und anschließend wiedereröffnet. Die Tabellenparameter sind dabei fest gewählt und für die einzelnen Containerformate unterschiedlich. Ein Dictionary wird in vier Spalten dargestellt, weil es

theoretisch möglich ist, dass der erste Wert keinen String darstellt, während für die Datentypen Array und Set drei Spalten für die Zählung der Objekte ausreichen. In Abbildung 20 ist neben den Tabellentypen auch das Anzeigen der Ebene mit der dazugehörigen Bezeichnung zu erkennen.

Dictonary mit 5 Objekten

ITEM	Wert	Typ	Inhalt
String:	stocks	Array	mit 6 Objekten

Ebene:--> stocks

ITEM Nr.	Typ	Inhalt
Array Item Nr: 1	Dictonary	mit 8 Objekten

Abbildung 20: Beispiel Ausgabe Tabellen Dictionary und Array (eigene Darstellung)

Die Befüllung der Tabelle erfolgt dabei zum Teil innerhalb der einzelnen Objekttypen und zum Teil innerhalb der Verarbeitungsfunktionen der Containerfiles. Die zugehörigen HTML Befehle, lauten `<td>`,`</td>` zum Öffnen und Schließen einer Zelle und `<tr>`,`</tr>` zum Eröffnen und Beenden einer Zeile.

3.2.4 Verarbeitung der Objekttypen in der Objekt Tabelle

Nachdem die strukturierte Ausgabe und die Informationen aus Header, Trailer und Offset-Tabelle zur Verarbeitung des BPList Files gespeichert wurden, wird in der Objekt-Tabelle begonnen, die einzelnen Objekttypen auszulesen. Dazu wird in Codeblock 9 die Funktion `parseobjekt` aufgerufen.

```
237  buffer = parseobjekt(file_buf,      // gespeicherter Dateiinhalt
238          markeroffset,               // gespeicherte Offsets der Offset Tabelle
239          0,                          // Objektnummer
240          buffer);                    // Buffer, in den geschrieben wird
```

Codeblock 9: Aufruf der Funktion parseobjekt

Diese Funktion wird bei jedem neuen Objekt erneut aufgerufen und interpretiert jeden Dateityp nacheinander bzw. ruft weitere Unterfunktionen auf. Der Aufruf der Funktion erfolgt beim ersten Mal statisch mit dem Objekt 0 (Codezeile 239) aus der Funktion XT_View, danach wird sie bei Bedarf aus den Containerdateien rekursiv aufgerufen. Dadurch ist auch der Fall abgedeckt, dass das erste Objekt kein Containerformat ist und die Funktion nur einmal durchläuft.

3.2.4.1 Interpretieren der Objekttypen

Ähnlich wie in Abschnitt 2.4.1.5 vorgeführt, wird in der Methode `parseobjekt` zuerst das Markierungsbyte des jeweiligen Objektes interpretiert. Dazu wird das Byte in Codeblock 10 in die einzelnen Nibble getrennt und in den Variablen `objtyp` und `objinfo` gespeichert.

```
279  objtyp = (file_buf[inhalt] & 0xF0);       // bitweise UND Verknüpfung
280  objtyp = objtyp >> 4;                     // 4 Bits nach rechts verschieben
281  objinfo = (file_buf[inhalt] & 0x0F);      // zweites Nibble speichern
282
283  switch (objtyp)
284  {
285  case 0x0:
286   switch (objinfo)
287   {
298   case 0x9:
300          wsprintf(Ausgabebuf, L"Bool</td><td>True");
301          return Ausgabe(buffer, L"Bool</td><td>True", Ausgabebuf,0);
```

Codeblock 10: Objekte Bestimmen (Auszug)

Nach der Trennung des Bytes in die einzelnen Nibble wird durch eine switch-Anweisung die Interpretation des Dateityps durchgeführt. Beispielhaft ist dies für die einfache Information „True“, auszugsweise ab der Codezeile 283 dargestellt. Gut ersichtlich ist die doppelte Anwendung der switch-Anweisung, erst für den Objekttypen und im Anschluss für die Objektinfo. Dieses doppelte Verfahren lässt sich nur für die Objekttypen Null, Bool und das Fill Byte umsetzen, da diese verschiedenen Typen unter der Objekt Kennzeichnung 0 vereint sind und keine Längenangabe benötigen (vgl. Abbildung 9).

Nachdem die Überprüfung der beiden Werte abgeschlossen ist, folgt die Verarbeitung der enthaltenen Daten. In Zeile 300 ist zu sehen, dass es in diesem Beispiel ausreicht, die Information an den `buffer` zu übergeben. Ebenfalls zu sehen ist, dass die Ausgabe HTML formatiert in zwei Spalten erfolgt, worauf im vorherigen Abschnitt schon genauer eingegangen wurde.

3.2.4.2 Objekttyp Integer

Der Datentyp Integer wird nicht nur für die Ausgabe benötigt, sondern auch zum Auslesen von Informationen mit Hilfe der Integer-Kodierung (siehe Abschnitt 2.4.1.4). Deswegen muss die Berechnung des Wertes in einer separaten Funktion erfolgen. Die Funktion `parseint` wird mit folgender Konvention aufgerufen.

```
315  ergebnis = parseint(objinfo, // Länge der Folgenden Bytes
316             inhalt,           // Position innerhalb der Datei
317             file_buf);        // Buffer mit der Datei

332          intlaengeu =ergebnis;
333          zeichencount = 1;
334          while ((intlaengeu = intlaengeu / 10) != 0)     zeichencount++;
```

Codeblock 11: Aufruf Funktion parseint

Als Rückgabewert wird die Integer-Zahl zurückgegeben, die je nach Aufruf an die Ausgabe übergeben oder zum Auslesen der folgenden Bytes in den anderen Datentypen verwendet wird. Für die Ausgabe ist es bei allen Zahlendatentypen notwendig, die Zeichen zu zählen, die für die Ausgabe auf dem Bildschirm benötigt werden. Dafür wird die Integer-Variable Zeichencount in allen notwendigen Datentypen verwendet. Die Umsetzung erfolgt durch eine while-Schleife, die exempla-

risch ab Zeile 332 für die anderen Datentypen aufgezeigt wird. Die Variable `intlaengeu` enthält temporär den Integer-Wert, der vorher innerhalb der Funktion ermittelt wurde.

Innerhalb der Methode `parseint` wird ein Zeiger definiert, der auf die aktuelle Position des Dateibuffers `file_buf` zeigt. Ein Integer-Wert kann nur in den Potenzen von „2“ physisch gespeichert werden, wobei ein 8-Byte-Integer die größte Zahl darstellt. Diese Berechnung kann am besten mit einer Switch-Anweisung und dem bitweisen Operator „<<“ durchgeführt werden. Dieser verschiebt die einzelnen Bits der Variable `objinfo` um ein Bit nach links, was den gleichen Effekt hat, wie das Berechnen der Zweierpotenz. Im Codeblock 12 ist dies beispielhaft für einen 8–Byte-Integer dargestellt.

```
711  switch (0x01 << objinfo) {
725  case 8:              // 8-Byte-Integer werden nicht unsigned definiert
726  ergebnis = (int64_t)_byteswap_uint64(*((int64_t *)ptr));
727  negativint = (int64_t)_byteswap_uint64(*((int64_t *)ptr));
728  break;
```

Codeblock 12: Funktion parseint (Auszug)

Besonderheit an diesem 8-Byte-Beispiel ist, dass nicht der Integer-Typ uint64_t verwendet werden darf. In den Byte-Längen 1,2 und 4 dürfen gemäß Apple keine negativen Zahlen gespeichert werden. Der Rückgabewert der Funktion muss wegen der vielfachen Nutzung immer ein Integer vom Typ uint64_t sein. Dieser Sonderfall wird in diesem Projekt mit einer Globalen Variable `negativint` abgefangen, die vor der Ausgabe des Wertes innerhalb der Funktion `parseobjekt` überprüft wird.

3.2.4.3 Objekttyp Real

Der Datentyp Real kann in Apples Binary Property List File als Float (4 Byte lang) auftreten oder als Double (8 Byte lang). Das zweite Nibble im Markierungsbyte gibt die Länge an. Um Rechenleistung durch die komplizierte Berechnung der Kommawerte zu sparen, wird der jeweilige Wert als Integer ausgelesen und mithilfe von memcpy in den reservierten Speicherbereich der Float-Variable übertragen und anschließend als solcher interpretiert.

```
343  if (length == 4)
344  {
345   ergebnis = file_buf[inhalt]<<24;
346   ergebnis = ergebnis +(file_buf[inhalt + 1] << 16);
347   ergebnis = ergebnis + (file_buf[inhalt + 2] << 8);
348   ergebnis = ergebnis + (file_buf[inhalt + 3]);
349
350   memcpy(&ergfloat, &ergebnis, sizeof(float));
```

Codeblock 13: Berechnung des Real Wertes (Auszug)

In den Zeilen 345 bis 348 werden die Bytes nacheinander ausgelesen und mithilfe des Bitweisen Operators „<<“ an die richtige Stelle gesetzt, wodurch ein Tausch der Bytes von Little in Big Endian nicht mehr notwendig ist. Nachfolgend und hier nicht abgebildet folgt die Zählung der benö-

tigten Zeichen und die formatierte Ausgabe auf die gleiche Weise wie bei den anderen Datentypen. Die 8-Byte-Methode erfolgt auf die gleiche Art und Weise.

3.2.4.4 Objekttyp Data

Ähnlich wie bei den Containerdatentypen ist es bei String, Unicode und Data notwendig, mit und ohne einen Integer-Wert die Länge zu bestimmen. Die Umsetzung erfolgt mithilfe einer if-Anweisung, in der entweder die Objektinfo direkt als Länge angenommen oder sie berechnet wird. Im folgenden Codeblock ist dies für den Datentyp Data Beispielhaft abgebildet.

```
423  if (objinfo != 0xf)
424  {
425   ergebnis = objinfo;
426   position = inhalt + 1;
427  }
428  else if (objinfo == 0xf)
429  {
430   objinfo = (file_buf[inhalt + 1] & 0x0F);
431   length = (int)pow(2, objinfo);
432   ergebnis = parseint(objinfo, inhalt + 1, file_buf);
433   position = inhalt + 2 + length;
434  }
435
445  XWF_Read(hitem1, position, (BYTE *)file_tmp, (DWORD)(position + ergebnis));
447  ergebnis = ergebnis * 2;
```

Codeblock 14: Befüllen der Temporären Buffer (Auszug)

Ersichtlich ist in Zeile 426 und 433, dass der Startpunkt zum Lesen der Daten aus dem Buffer `file_buf` unterschiedlich weit nach hinten verschoben werden muss, je nachdem, ob 14 Zeichen für die Anzeige ausreichend sind oder ein Integer-Zähler folgen muss. Nachdem der Buffer `file_tmp` mithilfe der Funktion `XWF_Read` befüllt wurde, muss die Dateilänge für die folgende Schleife verdoppelt werden. Anders ist es nicht möglich, das linke und das rechte Nibble nacheinander an die Ausgabe zu übergeben. Dies passiert innerhalb einer Schleife, in der ein temporäres Array nacheinander befüllt wird.

Zur Verbesserung der Lesbarkeit wird in der Ausgabe alle 16 Byte ein Leerzeichen eingefügt. Außerdem können Datenströme eine nahezu unbegrenzte Länge annehmen, weswegen es für eine fließende Ausgabe erforderlich ist, die Länge zu prüfen und zu begrenzen. In dieser Arbeit wird die Anzahl der Zeichen ausgegeben und die Ausgabe auf 400 Zeichen beschränkt. Dadurch besteht die Möglichkeit, den ausgegebenen String aus der Ausgabe heraus zu kopieren und innerhalb der Datei zu suchen, um im Anschluss den kompletten Datenstrom zu betrachten.

3.2.4.5 Objekttypen ASCII String, Unicode String, UID und Date

Die Datentypen ASCII und Unicode werden mit den bereits vorgestellten Methoden interpretiert und benötigen keiner weiteren Erläuterung. Lediglich einige kleinere Anpassungen mussten vorgenommen werden, wie z.B. das Auslesen von jeweils zwei Byte bei dem Datentyp Unicode.

Der Datentyp UID ist in der Apple CoreFoundation Datei zwar zu finden und auch implementiert [APPL14_1], dennoch werden UIDs im Apple eigenen PList-Viewer Xcode nicht angezeigt. Der Datentyp kommt aber häufig in den getesteten BPList-Files vor, weshalb es für diese Arbeit notwendig ist, ihn zu implementieren. Die Umsetzung erfolgt dabei ähnlich wie bei einem Integer-Wert, der einzige Unterschied besteht darin, dass das zweite Nibble vom Markierungsbyte um eins erhöht werden muss.

Der Datentyp Date hat eine feste Länge von 8 Bytes. Dieser wird als Integer ausgelesen und ähnlich wie beim Datentyp Real mit dem Befehl `memcpy` in den Speicher der Datumsvariablen kopiert.

```
407  memcpy(&datum, &ergebnis, 8);// Kopiert von Ergebnis in Datum ohne Konvertierung
408  // 978307200 = Datum Zeit Format geparst vom 01.01.2001 GMT
409  datum += 978307200;          // + EPOCH Time berechnen
410  ergebnis = (uint64_t)datum;
411
412  localtime_s(&ts, (const time_t *)&ergebnis);
413
414  wcsftime(Ausgabebuf, 150, L"Datum:</td><td>%a %Y-%m-%d %H:%M:%S", &ts);
415  return Ausgabe(buffer, L"Datum:</td><td>                        ", Ausgabebuf,0);
```

Codeblock 15: Datentyp Datum Verarbeiten (Auszug)

Im Anschluss wird in Zeile 409 der feste Wert für die Apple-Referenz-Time (00:00:00 UTC am 1. Januar 2001) auf den double-Wert addiert [APPL14_2]. Zuletzt wird der Wert mit der internen Typenumwandlung wieder in die Integer-Variable Ergebnis überführt und mit der Funktion localtime_s in ein Format gebracht, mit dem der Zeitpunkt in beliebiger Formatierung ausgeben werden kann. Die Ausgabe erfolgt in der letzten Zeile, darin wird der Zeitwert in den Buffer und die HTML-Tabelle übergeben. Die Ausgabe im Programm erfolgt dann gemäß der Zeile 414 im Format „Wochentag Jahr-Monat-Tag Stunde:Minute:Sekunde“.

3.2.4.6 Containerformate

Die drei Containerformate Dictionary, Array und Set haben einen vergleichbaren Funktionsaufbau. Ähnlich wie beim manuellen Lesen dieser Formate unterscheidet sich die programmiertechnische Umsetzung nur marginal. Besonders die Datentypen Set und Array unterscheiden sich gemäß Dokumentation nur in der Sortierung [APPL18_3; APPL18_4]. Auch wenn in der CoreFoundation von Apple schon seit der Version 00 genannt, war es nicht möglich, den Datentyp Set in einem der zu Verfügung stehenden PList Objekte zu finden [APPL14_1]. Aus diesem Grund wurde dieser Datentyp zwar implementiert, konnte aber nicht auf Funktionsfähigkeit getestet werden.

Alle selbst ausgelagerten Funktionen müssen am Anfang des Quellcodes einmal definiert werden. Dies gilt nicht nur für die drei Containerformate, sondern auch für die bereits genannten Funktionen `parseobjekt`, `parseint` und `Ausgabe`.

Im Codeblock 16 ist ersichtlich, dass alle drei Funktionen den gleichen Rückgabewert und die gleichen Aufrufparameter benutzen. Im Einzelnen sind die Parameter folgendermaßen zu interpretieren:

```
30  wchar_t* parseArray(int objinfo, uint64_t markernummer, BYTE *file_buf, uint64_t
    markeroffset[4500], wchar_t *buffer);
31  wchar_t* parseSet(int objinfo, uint64_t markernummer, BYTE *file_buf, uint64_t
    markeroffset[4500], wchar_t *buffer);
32  wchar_t* parseDict(int objinfo, uint64_t markernummer, BYTE *file_buf, uint64_t
    markeroffset[4500], wchar_t *buffer);
```

Codeblock 16: Definieren der Containerfunktionen

`objinfo:`	Rechtes Nibble für die Längenangabe des folgenden Typs
`markernummer:`	Nummer des Objektes in der Offset Tabelle, das analysiert wird
`*file_buf:`	Zeiger auf den Buffer, in dem der Inhalt der Datei gespeichert ist
`markeroffset[4500]:`	Die Offsets aller Objekte der Datei
`*buffer:`	Zeiger auf den Ausgabe Buffer, in dem die Ergebnisse gespeichert werden dieser dient gleichzeitig auch als Rückgabewert der Funktionen

Zum Verarbeiten der Objekttypen innerhalb der Containerdateien wird, nach dem Bestimmen von Anzahl und Start-Offset der Objekte, eine Schleife verwendet. Darin wird unter Berücksichtigung der im Codeblock 4 bestimmten `refsize` Größe die jeweilige Objektnummer aus dem Containerformat ausgelesen und an die Funktion `parseobjekt` übergeben. Beispielhaft wird im folgenden Codeblock das Verfahren für den Objekttyp Dictionary verkürzt aufgezeigt.

```
798  for (int i = 1; i <= objektpaare; i++)
799  {
807   position = inhalt    // Aktuelle Position in der Datei
808          + i           // Nummer des auszulesenden Objekts
809          + length      // Länge des Integer Counters
810          + j;          // Byte Zähler für refsize 2
811
825   else if (refsize == 2)
826   {
827    objektnummer = (file_buf[position]) << 8;                    // 1 Byte
828    objektnummer = objektnummer + file_buf[position + 1];        // 2 Byte
829    objektnummer2 = (file_buf[position+ objektpaare * 2]) << 8;  // 1 Byte
830    objektnummer2 = objektnummer2 + file_buf[position+1+objektpaare * 2];// 2 Byte
831     j++;                       // Zähler für das Doppelbyte
832   }
846
847   wsprintf(Ausgabebuf, L"<td>");
848   buffer = Ausgabe(buffer, L"<td>", Ausgabebuf, 0);
849   buffer = parseobjekt(file_buf, markeroffset, objektnummer, buffer);
850   wsprintf(Ausgabebuf, L"</td><td>");
851   buffer = Ausgabe(buffer, L"</td><td>", Ausgabebuf, 0);
852   buffer = parseobjekt(file_buf, markeroffset, objektnummer2, buffer);
853   wsprintf(Ausgabebuf, L"</td></tr>");
854   buffer = Ausgabe(buffer, L"</td></tr>", Ausgabebuf, 0);
856  }
```

Codeblock 17: Verarbeitung Containerdateien (Auszug)

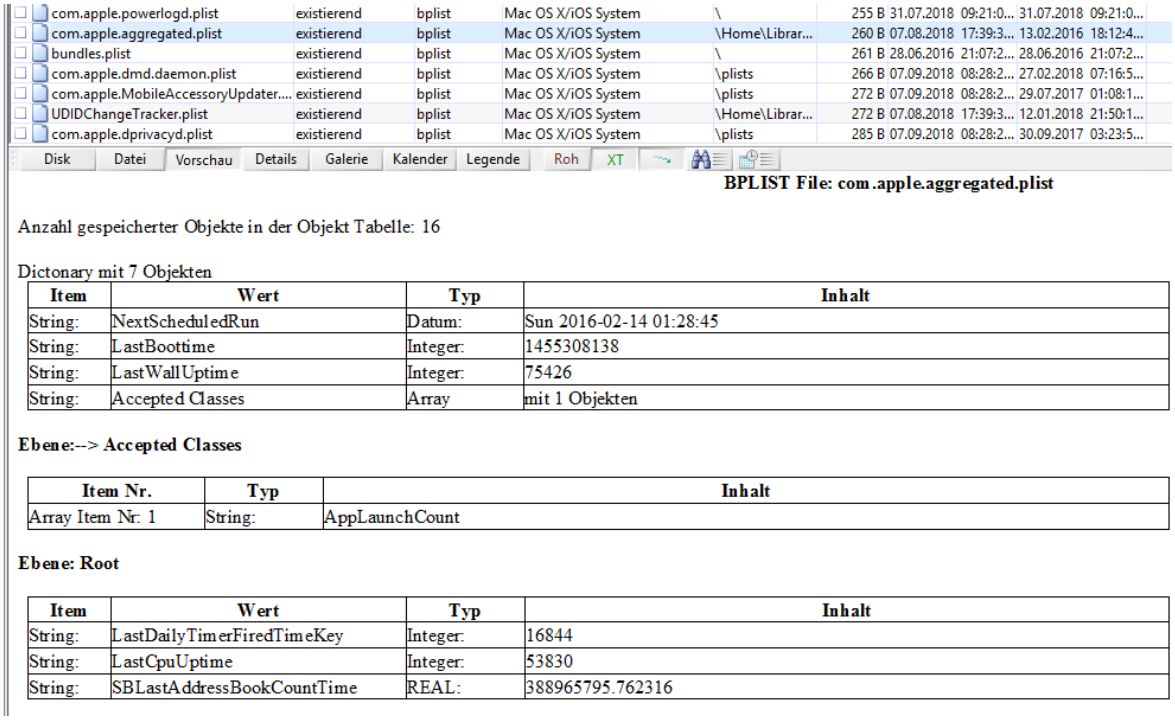
Nachdem die Schleife begonnen hat jedes Objektpaar separat abzuarbeiten, werden in der Hilfsvariablen `position` die ersten Informationen zusammengefasst. Sie dienen im Anschluss zum Auslesen des gesuchten Objektes innerhalb der gesamten Datei. Ähnlich wie bei dem Dateityp Real werden die Werte in Codezeile 827 und 829 mit dem Operator „<<“ um ein Byte nach links versch-

ben, um die passenden Objektnummern zu speichern. Nach der Übergabe der notwendigen Tabelleninformationen in Zeile 848 wird die Funktion `parseobjekt` für den Schlüssel und den Wert nacheinander aufgerufen. So wird die Tabelle mit vier Spalten zeilenweise durch die Schleife befüllt. Bei den Dateitypen Array und Set wird nur der Wert übergeben und die erste Spalte mit einem festen String beschriftet, weswegen die Funktion `parseobjekt` nur einmal rekursiv aufgerufen werden muss.

Nachdem alle Objekte interpretiert wurden, springt das Programm zurück in die die Hauptfunktion XT_View. Darin wird zuletzt mit der stets aktuell gehaltenen Variable Speichernutzung die Größe des zu übergebenen Buffers an die Variable `nResSize` und abschließend der `buffer` mit allen befüllten Inhalten an X-Ways Forensics übergeben. Auf Grund der Tatsache, dass in dieser ersten Version die maximale Anzahl der Objekte auf 4.500, die maximale einzulesende Dateigröße auf 100 KB begrenzt wird und keine Testdateien der nötigen Größe vorlagen, wurde auf eine Umsetzung der `refsize` 4 und 8 verzichtet. Der Komplette Quellcode der X-Tension ist in Anlage 2 einzusehen.

4 Ergebnis

Im Folgenden werden einige Screenshots der X-Tension für die Software X-Ways Forensics aufgezeigt, die das Ergebnis dieses konzipierten BPList Parsers darstellen.



com.apple.powerlogd.plist existierend bplist Mac OS X/iOS System \ 255 B 31.07.2018 09:21:0... 31.07.2018 09:21:0...
com.apple.aggregated.plist existierend bplist Mac OS X/iOS System \Home\Librar... 260 B 07.08.2018 17:39:3... 13.02.2016 18:12:4...
bundles.plist existierend bplist Mac OS X/iOS System \ 261 B 28.06.2016 21:07:2... 28.06.2016 21:07:2...
com.apple.dmd.daemon.plist existierend bplist Mac OS X/iOS System \plists 266 B 07.09.2018 08:28:2... 27.02.2018 07:16:5...
com.apple.MobileAccessoryUpdater.... existierend bplist Mac OS X/iOS System \plists 272 B 07.09.2018 08:28:2... 29.07.2017 01:08:1...
UDIDChangeTracker.plist existierend bplist Mac OS X/iOS System \Home\Librar... 272 B 07.08.2018 17:39:3... 12.01.2018 21:50:1...
com.apple.dprivacyd.plist existierend bplist Mac OS X/iOS System \plists 285 B 07.09.2018 08:28:2... 30.09.2017 03:23:5...

Disk Datei Vorschau Details Galerie Kalender Legende Roh XT

BPLIST File: com.apple.aggregated.plist

Anzahl gespeicherter Objekte in der Objekt Tabelle: 16

Dictonary mit 7 Objekten

Item	Wert	Typ	Inhalt
String:	NextScheduledRun	Datum:	Sun 2016-02-14 01:28:45
String:	LastBoottime	Integer:	1455308138
String:	LastWallUptime	Integer:	75426
String:	Accepted Classes	Array	mit 1 Objekten

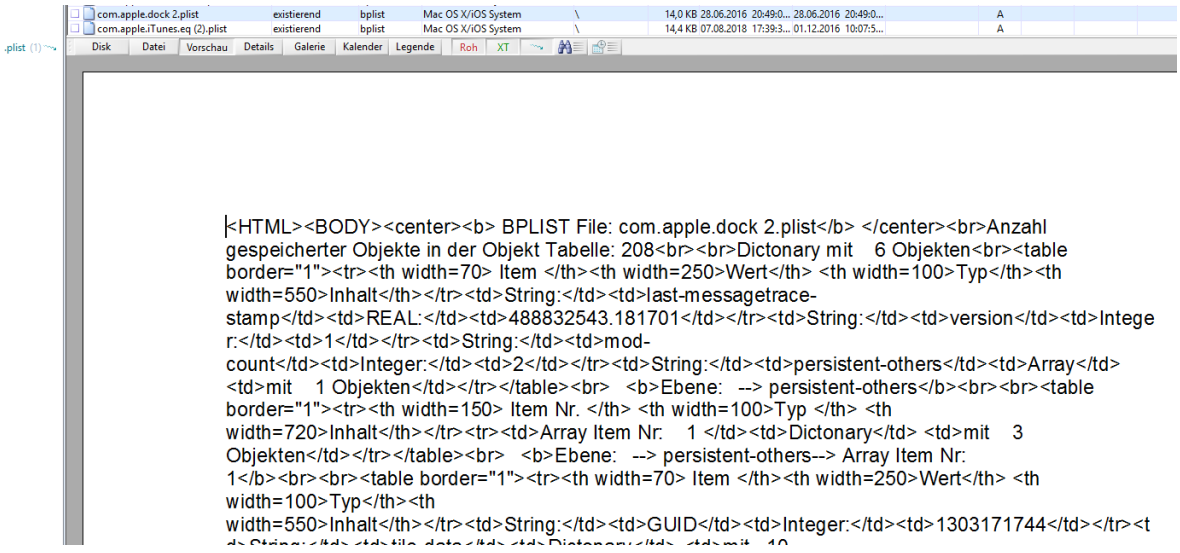
Ebene:--> Accepted Classes

Item Nr.	Typ	Inhalt
Array Item Nr: 1	String:	AppLaunchCount

Ebene: Root

Item	Wert	Typ	Inhalt
String:	LastDailyTimerFiredTimeKey	Integer:	16844
String:	LastCpuUptime	Integer:	53830
String:	SBLastAddressBookCountTime	REAL:	388965795.762316

Abbildung 21: Datentypen Date, Integer, Real, Array, Dictionary und String (eigene Darstellung)



com.apple.dock 2.plist existierend bplist Mac OS X/iOS System \ 14,0 KB 28.06.2016 20:49:0... 28.06.2016 20:49:0... A
com.apple.iTunes.eq (2).plist existierend bplist Mac OS X/iOS System \ 14,4 KB 07.08.2018 17:39:3... 01.12.2016 10:07:5... A

Disk Datei Vorschau Details Galerie Kalender Legende Roh XT

<HTML><BODY> BPLIST File: com.apple.dock 2.plist
Anzahl gespeicherter Objekte in der Objekt Tabelle: 208

Dictonary mit 6 Objekten
<table border="1"><tr><th width=70> Item </th><th width=250>Wert</th> <th width=100>Typ</th><th width=550>Inhalt</th></tr><td>String:</td><td>last-messagetrace-stamp</td><td>REAL:</td><td>488832543.181701</td></tr><td>String:</td><td>version</td><td>Integer:</td><td>1</td></tr><td>String:</td><td>mod-count</td><td>Integer:</td><td>2</td></tr><td>String:</td><td>persistent-others</td><td>Array</td><td>mit 1 Objekten</td></tr></table>
 Ebene: --> persistent-others

<table border="1"><tr><th width=150> Item Nr. </th> <th width=100>Typ </th> <th width=720>Inhalt</th></tr><tr><td>Array Item Nr: 1 </td><td>Dictonary</td> <td>mit 3 Objekten</td></tr></table>
 Ebene: --> persistent-others--> Array Item Nr: 1

<table border="1"><tr><th width=70> Item </th><th width=250>Wert</th> <th width=100>Typ</th><th width=550>Inhalt</th></tr><td>String:</td><td>GUID</td><td>Integer:</td><td>1303171744</td></tr><t

Abbildung 22: Ausschnitt des generierten HTML Codes im Rohformat (eigene Darstellung)

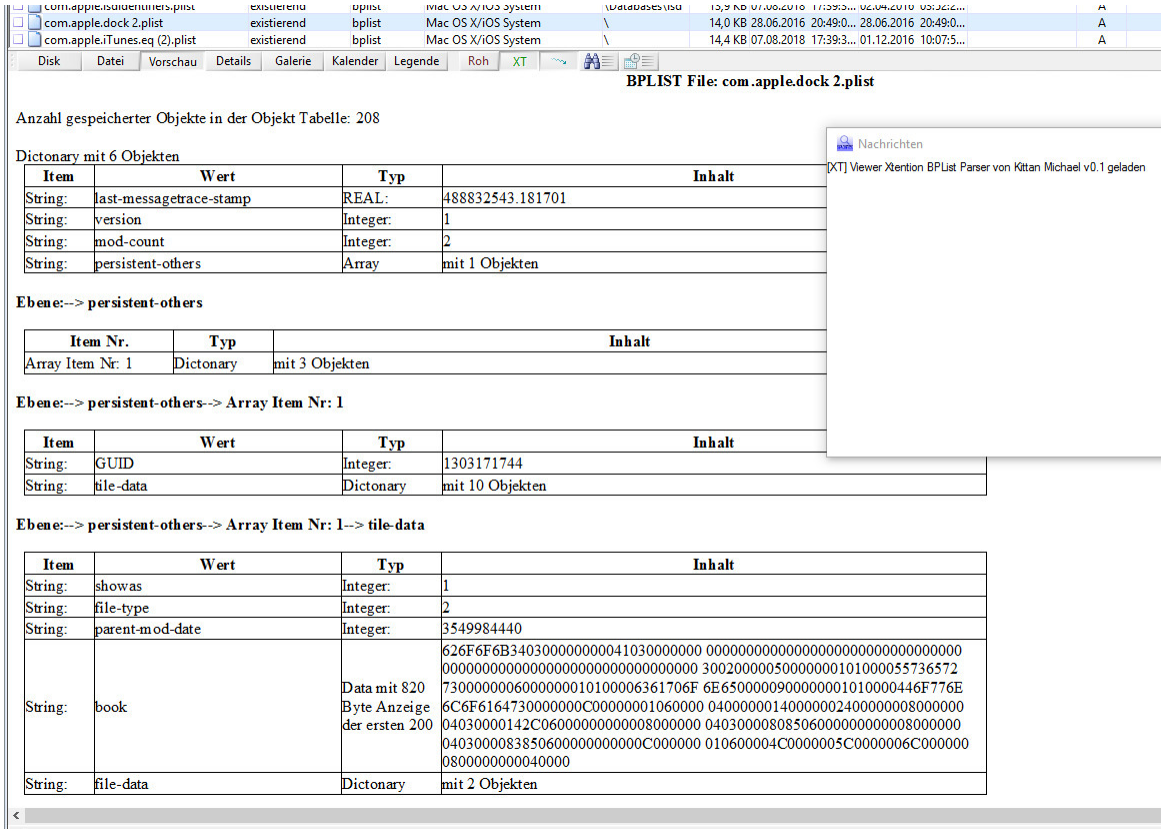


Abbildung 23: Datentyp Data und Verschachtelung (eigene Darstellung)

Der Testdatensatz umfasste 562 Binary Property List Files aus vier unterschiedlichen Apple-Geräten, denen stichprobenartig Dateien entnommen wurden:

1. Iphone 5s mit IOS 11.4.1
2. MacBook Pro mit macOS High Sierra 10.13.6
3. MacBook Pro mit macOS High Sierra 10.13.6
4. Schulungsimage mit OS X El Capitan 10.11.3

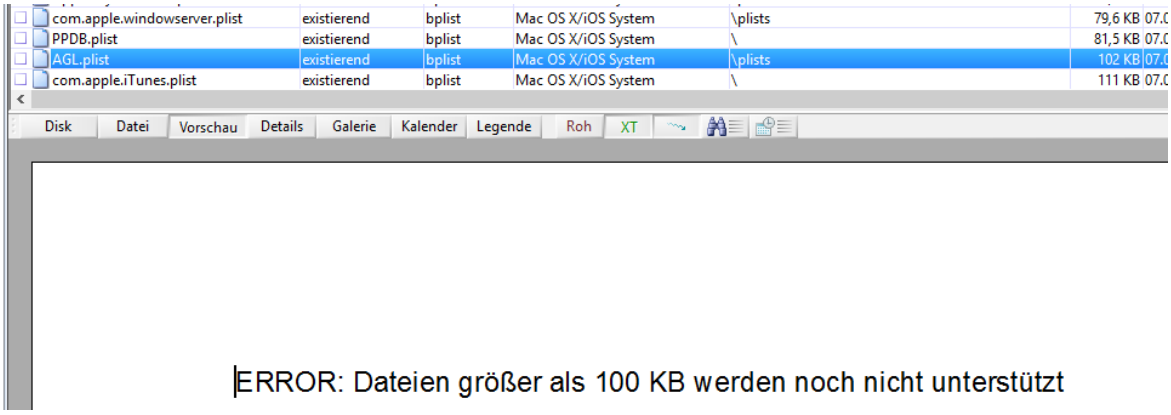


Abbildung 24: Ergebnis: Abfangen der Dateigröße (eigene Darstellung)

Aktuell existiert noch ein Fehler bei einigen größeren Dateien, der die Software zum Abstürzen bringt. Um dies zu verhindern, wurden einige Routinen eingefügt, die den Fehler unterbinden. Mit der Begrenzung auf 100 KB und max. 4.500 Objekten in der Datei werden 561 der 562 Dateien Absturzsicher durch die X-Tension angezeigt. Bei 14 Dateien greifen diese Begrenzungen (Abbildung 24), wovon vier dieser Dateien das Programm alternativ zum Absturz bringen würden.

5 Diskussion

Mit dieser Arbeit wurden die Grundlagen gelegt, die notwendig sind, um eine X-Ways X-Tension zu erstellen und in das Programm auf verschiedenen Wegen einzubinden. Dazu war es notwendig, sich ausgiebig mit der X-Ways API auseinanderzusetzen, was aufgrund der kurz gehaltenen Dokumentation und den nicht vorhandenen Beispielen viel Zeit beanspruchte. Eine Fehlersuche innerhalb der Programmierung war zu jederzeit schwierig, da es durch die API-Funktionen nicht möglich ist, einzelne Programmteile im Compiler zu testen bzw. Fehler zu lokalisieren. Um Debugging-Informationen zu erhalten, mussten eigenhändig Ausgabeinformationen zum Quelltext hinzugefügt, die DLL erstellt und in X-Ways eingebunden werden.

Im Weiteren liefert diese Arbeit die Grundlage zum Verstehen des Apple-File-Formates Property List. Um dies zu gewährleisten, wurde der Aufbau des Formates anhand von Beispielen, dem Apple File „CFBinaryPList.c“ und einigen Internetquellen erarbeitet und niedergeschrieben. Auf ausgiebige Literaturquellen, eine Dokumentation vom Hersteller oder eine Kooperation mit diesem konnte trotz mehrmaliger Anfragen nicht zurückgegriffen werden. Nachdem diese Grundlagen geschaffen waren, konnte mit der Konzipierung der programmtechnischen Umsetzung begonnen werden. Dafür wurden die einzelnen Teile des BPList Files ausgelesen und verarbeitet, um zuletzt die einzelnen Objekttypen zueinander passend im Vorschaufenster von XWFo auszugeben. Die Umsetzung des Parsers erfolgte in C und beanspruchte nicht nur wegen der genannten Schwierigkeiten die meiste Zeit in diesem Projekt.

Als Ergebnis kann ein funktionsfähiger Parser vorgestellt werden, der in dieser Version noch nicht ganz absturzsicher läuft, aber sich in der Funktionalität mit der hauseigenen Apple-Software messen kann.

5.1 Vergleich mit anderen Parsern

Zum Einordnen der Funktionalität und der Nutzerfreundlichkeit der X-Tension muss ein Vergleich zu anderen Programmen gezogen werden, die ähnliche Funktionen bieten. Obwohl die visuellen Möglichkeiten in einer X-Ways X-Tension durch die Hauptsoftware begrenzt werden und der forensische Nutzen beachtet werden muss, bleibt der Hintergrund, das BPList File menschenlesbar darzustellen, unverändert.

Für die Windows-Umgebung konnte online neben einigen freien Bibliotheken, wie z.B. in Java von 3bread[24] oder in Javascript von joeferner[25], kein freier BPList-Parser mit Oberfläche gefunden wer-

[24] 3bread: https://github.com/3breadt/dd-plist; letzter Zugriff:26.09.2018 12:12

[25] Joeferner: https://github.com/joeferner/node-bplist-parser; letzter Zugriff:26.09.2018 12:12

den. Einige kostenpflichtige Programme wie z.B. OSForensics[26] von PassMark oder der plistEditor Pro[27] von VOWSoft stehen jedoch als Probeversion zur Verfügung. Leider kann aufgrund des geschlossenen Quellcodes dieser Programme nicht sichergestellt werden, wie genau diese umgesetzt wurden, weswegen für einen Vergleich maßgeblich der von Apple unter macOS bereitgestellte PList Viewer Xcode bleibt.

Dennoch soll kurz ein Vergleich mit den beiden kommerziellen Tools gezogen werden. In Anlage I sind die Abbildungen 29-32 enthalten, worauf jeweils die gleiche Datei mit den drei Tools dargestellt ist. Die Unterschiede der Programme sind dabei gering. Die kommerziellen Tools greifen beide auf das optische Format von Apple zurück, in dem die Containerformate beliebig geöffnet und geschlossen werden können. Es gibt einzelne Unterschiede in der Formatierung wie z.B. beim Objekttyp Date und der Darstellung von Data. Genau wie diese Arbeit zeigt OSForensics bei Containerformaten die Anzahl der beinhaltenden Objekte an, worauf der plistEditor verzichtet (Abbildung 32). Ein großer Nachteil bei dem Tool von PassMark besteht darin, dass negative Integer-Werte nicht dargestellt werden können, stattdessen taucht der maximale Integer-Wert für einen vorzeichenlosen 64-Bit-Integer auf. Die Ursache dieses Fehlers wurde in Tabelle 4 erläutert. Diese X-Ways X-Tension und das Tool von VOWSoft stellen den Wert fehlerfrei dar (Abbildung 31). Ähnlich wie Apple verzichtet OSForensics auf die Anzeige der UID Werte. Diese Arbeit und der plistEditor Pro zeigen die Werte korrekt an, das kommerzielle Tool fügt aber vor jeden UID-Wert noch ein leeres Dictionary an, was die Lesbarkeit zusätzlich erschwert.

Im Folgenden werden einige Screenshots der Software X-Ways Forensics aufgezeigt, die das Ergebnis dieses BPList-Parsers (oben) im Vergleich zu dem Apple eigenen BPList-Viewer Xcode (unten) aufzeigen. Die untersuchten Dateien sind jeweils identisch.

BPLIST File: com.apple.powerlogd.plist

Anzahl gespeicherter Objekte in der ObjektTabelle: 11

Dictonary mit 5 Objekten

ITEM	Wert	Typ	Inhalt
String:	TimeOffset-kernel	REAL:	-1532001869.982873
String:	BootSessionUUID	String:	67E422D6-A983-47B9-AE6D-0C0BCEB7A462
String:	TimeOffset-monotonic	REAL:	1533022447.714088
String:	TimeOffset-system	REAL:	-782.214084
String:	PLUUID	String:	3FDFF463-FF5B-4660-A905-23E414448CE9

com.apple.powerlogd.plist

com.apple.powerlogd.plist 〉 No Selection

Key	Type	Value
▼ Root	Dictionary	(5 items)
TimeOffset-kernel	Number	-1.532.002.893,98287
BootSessionUUID	String	67E422D6-A983-47B9-AE6D-0C0BCEB7A462
TimeOffset-monotonic	Number	1.533.022.447,71409
TimeOffset-system	Number	-782,214084386826
PLUUID	String	3FDFF463-FF5B-4660-A905-23E414448CE9

Abbildung 25: Ergebnis: Datentyp String, Real, Dictionary (eigene Darstellung)

[26] OSForensics: Herstellerseite https://www.osforensics.com/; letzter Zugriff:26.09.2018 12:12

[27] plistEditor: Herstellerseite: http://www.icopybot.com/plist-editor.htm; letzter Zugriff:26.09.2018 12:12

BPLIST File: com.apple.captive.probe.plist

Anzahl gespeicherter Objekte in der ObjektTabelle: 10

Dictonary mit 4 Objekten

ITEM	Wert	Typ	Inhalt
String:	RefreshDate	Datum:	Mon 2016-02-22 17:17:32
String:	Version	Integer:	1
String:	HostNameList	Array	mit 1 Objekten

Ebene:--> HostNameList

ITEM Nr.	Typ	Inhalt
Array Item Nr: 1	String:	captive.apple.com

Ebene: Root

ITEM	Wert	Typ	Inhalt
String:	ExpirationDate	Datum:	Wed 2016-03-23 17:17:32

com.apple.captive.probe.plist

com.apple.captive.probe.plist 〉 No Selection

Key	Type	Value
▼ Root	Dictionary	(4 items)
RefreshDate	Date	22.02.2016, 17:17:32
Version	Number	1
ExpirationDate	Date	23.03.2016, 17:17:32
▼ HostNameList	Array	(1 item)
Item 0	String	captive.apple.com

Abbildung 26: Ergebnis: Datentyp Date, String, Integer, Array (eigene Darstellung)

Ebene:--> $objects

ITEM Nr.	Typ	Inhalt
Array Item Nr: 59	String:	[hybrid]
Array Item Nr: 60	String:	HSMW VPN All
Array Item Nr: 61	Dictonary	mit 12 Objekten

Ebene:--> $objects--> Array Item Nr: 61

ITEM	Wert	Typ	Inhalt
String:	profileSource	Integer:	0
String:	ApplicationCreated	Bool	False
String:	PayloadType	Uid:	0
String:	$class	Uid:	65
String:	ProfileUUID	Uid:	64
String:	ProfileIdentifier	Uid:	63
String:	isSetAside	Bool	False
String:	ProfileOrganization	Uid:	0
String:	systemVersion	Uid:	0
String:	PayloadOrganization	Uid:	62
String:	profileIngestionDate	Uid:	0
String:	PayloadUUID	Uid:	61

Ebene:--> $objects

com.apple.networkextension.plist

com.apple.networkextension.plist 〉 No Selection

Key	Type	Value
▶ Item 57	Dictionary	(1 item)
Item 58	String	[hybrid]
Item 59	String	HSMW VPN All
▼ Item 60	Dictionary	(3 items)
profileSource	Number	0
ApplicationCreated	Boolean	NO
isSetAside	Boolean	NO

Abbildung 27: Ergebnis: Datentyp Bool, UID, String, Dictionary, Array (eigene Darstellung)

Auf den Abbildungen 25-27 ist ersichtlich, dass alle Datentypen korrekt dargestellt werden. Dabei sind nur geringe Abweichungen zu erkennen, wie z.B. in Abbildung 21 das Abschneiden der letz-

ten Kommastelle in der XWFo X-Tension, in Abbildung 22 die unterschiedliche Sortierreihenfolge der beiden Parser oder in Abbildung 23 das zusätzliche Anzeigen der UID-Werte, was für eine forensische Software einen zusätzlichen Nutzen generieren kann.

Größere Unterschiede sind in der visuellen Darstellung zu erkennen. Während Apples hauseigenes Tool ein separates Fenster öffnet, in dem die Datei in einer großen Tabelle mit sich öffnenden Containerdateien dargestellt wird, werden in dieser Arbeit mehrere Tabellen erstellt und auch wieder geschlossen. Dies kann bei stark verschachtelten PLists zu Unübersichtlichkeit führen. Ein Vorteil gegenüber der Apple-Methode besteht darin, dass aufgrund des fehlenden zu öffnenden Fensters viele Dateien durchsehbar sind, ohne die Dateien separat öffnen zu müssen. Ein flüssiges Durchnavigieren durch die einzelnen Dateien, z.B. mit den Pfeiltasten, ist dadurch gewährleistet.

Zuletzt ist diese Anzeigemethode angelegt an die restlichen Viewer-Komponenten von XWFo, wie z.B. beim Betrachten von DLL-Dateien, ausführbaren Dateien oder dem NTFS-Journal-Parser[28] (Abbildung 28), was für den Nutzer den Umgang zusätzlich erleichtert.

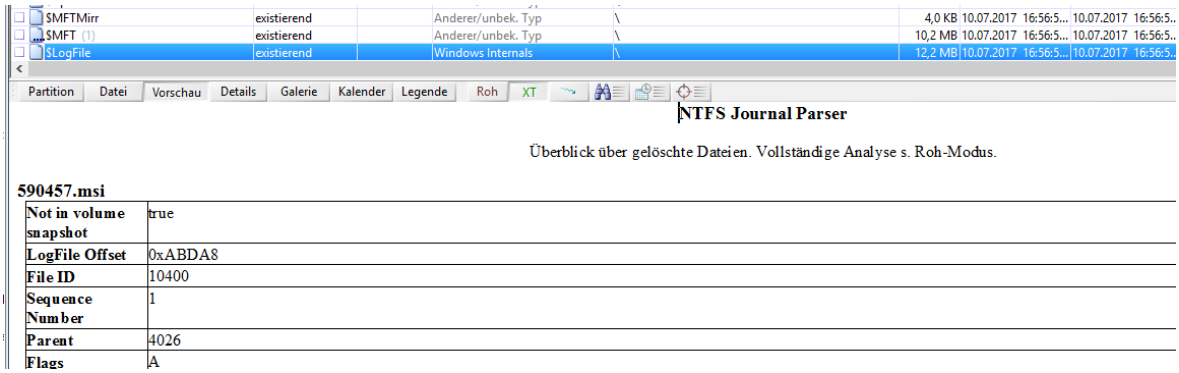


Abbildung 28: Darstellung des NTFS Journal Parsers (eigene Abbildung)

5.2 Fazit

Zusammenfassend kann gesagt werden, dass das Ziel, eine X-Ways X-Tension zu erstellen, die das Apple Format Binary Property Lists interpretiert und menschenlesbar innerhalb der Software ausgibt, erfolgreich umgesetzt wurde. Die Erweiterung für die Hauptsoftware funktioniert weitestgehend fehlerfrei und kann alle Dateitypen, die in der BPList Version 00 enthalten sind, korrekt interpretieren. Andere Versionen der Binary Files sind durch Apple in der „CFBinaryPList.c“ von Zeile 765 -767 erwähnt[APPL14_1]. Darin wird darauf verwiesen, dass ab der Mac OS X Version Snow Leopard eine Version größer 0 verwendet wird. Diese Mac OS Version ist seit dem 28. August 2009 auf jedem neuen Mac vorinstalliert [APPL11]. Beide MacBooks, die als Quelle des Testdatensatz dienten, sind mit einer aktuellen Version von MAC OS ausgestattet. Dennoch waren

[28] NTFS Journal Parser: Anzeige beim Öffnen der Datei $LogFile in einem NTFS Image

ausnahmslos nur Dateien der Version 00 auf den Geräten zu finden. Im Umkehrschluss lässt sich aufgrund der Zeitdifferenz und dem Testdatensatz von über 500 Dateien darauf schließen, dass aktuelle Apple-Systeme ihre Informationen noch im Format der Version 00 hinterlegen. Des Weiteren sind die neuen Dateitypen url, uuid, UTF8 String und OrderSet in der „CFBinaryPList.c“ zwar erwähnt, aber noch nicht implementiert [APPL14_1]. Eine spezielle Versionsprüfung innerhalb des Headers wurde für diese Arbeit anfangs zwar angestrebt, aber aus den genannten Gründen und mangels Testobjekten wieder verworfen.

Auch das zweite Ziel der Arbeit, eine X-Tension funktionstüchtig in XWFo einzubinden, konnte erfolgreich umgesetzt werden. Mit der Schritt–für-Schritt-Anleitung in Abschnitt 3.1 ist es mit Visual Studio innerhalb von kurzer Zeit möglich, eine eigene Erweiterung für die Forensic-Suite zu entwickeln. Nachdem die nötigen Dateien erstellt wurden und das Grundgerüst der Erweiterung steht, ist es möglich jedes andere forensische Problem aufzugreifen und in XWFo zu implementieren. Die Vorgehensweise in anderen Programmiersprachen oder Programmierumgebungen ist dabei ähnlich der hier vorgestellten.

Zum Abschluss dieses Fazits muss auf einen Fehler der Erweiterung eingegangen werden, der die Software aktuell zum Absturz bringt. Der Fehler trat im Testdatensatz ausschließlich bei einigen Dateien mit einer Größe von über 60KB auf und wurde mit der Größenbeschränkung von 100KB und max. 4500 anzuzeigenden Objekten eingegrenzt. Die Fehlerausgabe innerhalb der Datei „error.log“ gibt folgende Meldung aus:

„*Error significance: high or unknown Eine Ausnahmesituation vom Typ 216 (page protection fault) ist bei Offset 77EB7C42 aufgetreten als ich [bitte ergänzen]…*“

Der Fehler tritt nicht auf, wenn die DLL ohne die Verarbeitung des Unicode-Blocks erstellt wird. Ein weiteres Eingrenzen des Fehlers blieb aufgrund der ungenauen Fehlerausgabe erfolglos.

5.3 Ausblick

Im letzten Kapitel soll auf Themen eingegangen werden, die in der Erweiterung noch verbessert und erweitert werden können. Außerdem soll auf andere Lösungswege verwiesen werden, die alternativ zu der vorgestellten Variante ebenfalls zum Ziel führen könnten.

5.3.1 Verbesserungen der X-Tension

Das Testen der Erweiterung konnte aufgrund der Hochschulversion von XWFo nur innerhalb der Version 19.1 SR3 erfolgen. Aktuell wird auf der Herstellerseite die Version 19.7 vertrieben. Ein umfangreicherer Test der Erweiterung in Kombination mit anderen XWFo-Versionen ist aber notwendig, um mögliche weitere Fehler aufzuspüren und zu beheben und damit eine störungsfreie Nutzung der Erweiterung auch für die Zukunft gewährleistet zu können. Für weitere Tests sollte ebenfalls der Testdatensatz erhöht werden und weitere Apple-Geräte wie z.B. den Ipod, die Apple-Watch oder Apple TV betrachtet werden. Besonders bei neueren Apple-Geräten erhöht sich

dadurch auch die Möglichkeit, BPList Files mit anderen Versionen zu finden und analysieren zu können und somit der Erweiterung einen höheren Funktionsumfang zu bieten.

Im anderen Speicherformat der PList Files, dem XML Format, sind die Informationen für jeden Menschen lesbar und werden von XWFo durch die internen Viewer bereits korrekt dargestellt. Für einen schnellen Programmfluss und die einheitliche Lesbarkeit innerhalb des Programms sollte die XML Version von diesem Parser ebenfalls aufgegriffen und in gleicher Form von der Software dargestellt werden. Alternativ besteht die Möglichkeit, sich das Apple Tool „plutil“ zu Hilfe zu nehmen bzw. in die X-Tension zu integrieren, um ein XML File vorher zu einem BPList-File zu konvertieren und es anschließend mit der Erweiterung zu interpretieren. Apple setzt dies bei ihren macOS Betriebssystemen bereits in dieser Form um.

Um sich weiter an die Apple Lösung anzulehnen, sollte die Sortierung innerhalb von Xcode genauer betrachtet werden (Abbildung 26). Eine Schwäche des Parsers besteht darin, dass bei stark verschachtelten Dateien die Übersicht durch die vielen Tabellen verloren gehen kann. Eine Sortierung, nach der z.B. die Containerformate in jeder Ebene erst zum Schluss analysiert werden, könnte die Lesbarkeit des Parsers erhöhen.

Insbesondere von den beiden Windows Tools „OSForensics“ und „plistEditor Pro“ aufgegriffen ist die Idee, den Objekttyp Data vollständig anzeigen zu lassen. Diese beiden Tools besitzen integriert einen HexEditor, der bei Bedarf aufgerufen wird und den Datenstrom betrachten lässt. Dies erleichtert erheblich die Suche nach versteckten Daten innerhalb dieses uminterpretierten Stroms. Insbesondere größere Dateien beinhalten teils sehr große Datenströme innerhalb des BPList-Files, wobei viele Informationen verloren gehen können. Um noch einen Schritt weiter zu gehen, sollte eine Routine entwickelt werden, die nach Informationen wie z.B. Strings oder anderen Dateiformaten in den Strömen sucht, diese überprüft und bei Bedarf ausgibt. Denkbar wäre auch die Suche nach der Magic Number des kompletten BPList Files oder nach speziellen Abständen im Trailer. Dies könnte durch die Abstände der einzelnen Werte untereinander geschehen unter der Annahme, dass die ersten beschriebenen Felder des Trailers nur 0x1, 0x2, 0x3 oder 0x4 beinhalten können [CAI10].

In einem Dokument von ccl-forensics aus dem Jahr 2010 wird angenommen, dass einige dieser großen Datenströme zum Archivieren von mehreren Objekten im Speicher dienen. Zum Auslesen wird ein Dictionary mit vier Schlüssel/Wert-Paaren benötigt, wobei der Datentyp UID eine spezielle Bedeutung beim Auslesen dieser archivierten Daten einnimmt [CAI10]. In weiterer Betrachtung dieses Projekts muss für einen zusätzlichen forensischen Nutzen auch diese Möglichkeit genauer betrachtet werden.

Zur weiteren Verbesserung der Erweiterung muss eine X-Tension erstellt werden, die vorzugsweise in der 64Bit und der 32 Bit Version der Hauptsoftware funktioniert. Ebenfalls ist es denkbar, dem Nutzer die Wahl zu lassen, die Anzeige innerhalb des Vorschaufensters zu realisieren oder, wie die anderen in Punkt 5.1 vorgestellten Parser, mithilfe eines separaten Fensters. Nicht zuletzt muss der in Punkt 5.2 beschrieben Fehler behoben werden, um die eingebauten Beschränkungen auf 4.500 Objekte und die 100KB aufzuheben. Abschließend müssen noch die die refsize Größen 4 und 8 implementiert werden.

5.3.2 Andere Problembetrachtungen

Auch wenn es vor dem Beginn und während dieser Arbeit nicht gelungen ist, komplett andere Betrachtungsweisen zu finden, um einen BPList-Parser in XWFo zu integrieren, so gibt es zumindest theoretisch einige weiterführende Lösungsansätze, die denkbar sind.

Zum einen ist es möglich, in einem externen Parser oder einer Bibliothek die Dateien in das XML Format umzuwandeln, zu speichern und anschließend mit XWFo zu betrachten. Diese Methode funktioniert komplett extern und ohne das Programmieren einer X-Tension. Es ist aber auch möglich, diesen Konverter in eine X-Tension zu integrieren und den konvertierten Inhalt in eine temporäre Datei zu schreiben, die anschließend von XWFo ausgewertet wird. Bei Office-Dokumenten oder Email-Datenbanken kann diese Funktionalität des Hauptprogramms bereits betrachtet werden. Aber auch die API stellt diese Funktionalität z.B. mit der Funktion XWF_CreateItem bereit [XWAY18_3]. Alternativ stehen in jeder Programmiersprache Funktionen zur Verfügung, die temporäre Dateien erstellen und nach dem Verarbeiten wieder löschen können. Diese Lösungswege beinhalten automatisch, dass nicht mehr die Originaldatei interpretiert wird.

Eine andere Herangehensweise an diese Arbeit besteht darin, eine bereits fertige Bibliothek zu benutzen, die anschließend in die Erweiterung eingebunden wird. Beispiele für fertige Bibliotheken in unterschiedlichen Programmiersprachen sind unter Punkt 5.1 aufgezählt. Aber auch die „CFBinaryPList.c“ aus der Apple CoreFoundation kann für diesen Zweck angepasst und eingebunden werden.

Für die allgemeine Vereinfachung der Entwicklung von Erweiterungen für XWFo oder die allgemeine programmiertechnische Lösung von forensischen Problemen sollte in Betracht gezogen werden, zuerst den Parser ohne die API zu entwickeln und ähnliche Funktionen zu nutzen, die in der API enthalten sind. Somit besteht die Möglichkeit, den Quellcode zu testen und mögliche Fehler im Debugger des Compilers nachzuvollziehen und zu analysieren. Erst nach dem Sicherstellen der Funktionsweise des eigenständigen Programms kann die Anpassung an die API und die Integration in die Hauptsoftware erfolgen. Dadurch ist es möglich, Fehler im Quellcode, in der Übergabe des Programms oder in einzelnen Funktionen, die von der API bereitgestellt werden, zu lokalisieren und gezielt zu beheben.

Literatur

[APPL10] Apple Inc: *Introduction to Property Lists*- 2010 - https://developer.apple.com/library/archive/documentation/Cocoa/Conceptual/PropertyLists/Introduction/Introduction.html, verfügbar am 25.09.18, 07:28

[APPL11] Apple Inc: *Technische Daten*- 2011 - https://web.archive.org/web/20110510193957/http://www.apple.com:80/de/macosx/specs.html, verfügbar am 25.09.18, 09:30

[APPL12] Apple Inc: *Using Dynamic Libraries*- 2012 - https://developer.apple.com/library/archive/documentation/DeveloperTools/Conceptual/DynamicLibraries/100-Articles/UsingDynamicLibraries.html, verfügbar am 25.09.18, 07:28

[APPL14_1] Apple Inc: *Introduction to Property Lists*- 2014 - https://opensource.apple.com/source/CF/CF-855.17/CFBinaryPList.c.auto.html, verfügbar am 25.09.18, 07:28

[APPL14_2] Apple Inc CFDate.c- 2014 - https://opensource.apple.com/source/CF/CF-855.17/CFDate.c.auto.html, verfügbar am 26.09.18, 08:00

[APPL16] Apple Inc: *Überblick*- 2016 - https://developer.apple.com/jp/documentation/ToolsLanguages/Conceptual/Xcode_Overview/, verfügbar am 26.09.18, 07:28

[APPL18_1] Apple Inc: *Date* - 2018 - https://developer.apple.com/documentation/foundation/date, verfügbar am 25.09.18, 07:28

[APPL18_2] Apple Inc: *NSData*- 2018 - https://developer.apple.com/documentation/foundation/nsdata, verfügbar am 25.09.18, 07:28

[APPL18_3] Apple Inc: *Array*- 2018 - https://developer.apple.com/documentation/swift/array, verfügbar am 25.09.18, 07:28

[APPL18_4] Apple Inc: *Set*- 2018 - https://developer.apple.com/documentation/swift/Set, verfügbar am 25.09.18, 07:28

[APPL18_5] Apple Inc: *Dictionary*- 2018 - https://developer.apple.com/documentation/swift/dictionary, verfügbar am 25.09.18, 07:28

[APPL18_6] Apple Inc: *Core Foundation* - 2018 - https://developer.apple.com/documentation/corefoundation, verfügbar am 26.09.18, 08:28

[BKA16] BKA: *Bundeslagebild Cybercrime 2016*, 2016 - https://www.bka.de/DE/AktuelleInformationen/StatistikenLagebilder/Lagebilder/Cybercrime/cybercrime_node.html, verfügbar am 25.09.18, 07:25

[BMI17] BMI: *Startschuss für ZITiS* , 2017 - https://www.bmi.bund.de/SharedDocs/pressemitteilungen/DE/2017/01/zitis-vorstellung.html, verfügbar am 25.09.18, 07:28

[BRE18] Dreibrodt, Daniel: *com.dd.plist - A Java library for working with property lists* – 2018 - https://github.com/3breadt/dd-plist, verfügbar am 25.09.18, 07:28

[BRET14] Shavers, Brett; Zimmerman, Eric: *X-Ways Forensics Practitioner's Guide* - 2013 – Syngress Verlag; Auflage: 2014 – ISBN 978-0-124-11605-4

[BÖT99] Böttcher, Alex; Kneißel, Franz: *Informatik für Ingenieure – Grundlagen und Programmierung in C* - 1999 – Oldenburg Verlag; Auflage: 1999 – ISBN 978-3-486-24800-6

[CAI10] Caithness, Alex: *Property Lits in Digital Forensics* - 2010 - http://citeseerx.ist.psu.edu/viewdoc/dowlload?doi=10.1.1.190.762&rep=rep1&type=pdf, verfügbar am 26.09.18, 08:30

[FOR08] Forensic Focus: *Interview with Stefan Fleischmann, CEO X-Ways Software Technology AG*, 2008 - https://www.forensicfocus.com/stefan-fleischmann-interview-270208, verfügbar am 25.09.18, 07:28

[FIS11] Fischer, Peter; Hofer, Peter: *Lexikon der Informatik* - 2011 – Springer-Verlag; Auflage: 15 – ISBN 978-3-642-15125-5

[KAR17] Karaiskos, Christos: *Understanding Apple's binary property list format* – 2017 - https://medium.com/@karaiskc/understanding-apples-binary-property-list-format-281e6da00dbd, verfügbar am 25.09.18, 07:28

[MAD14] Mandl, Peter: *Grundkurs Betriebssysteme* - 2014 - Springer Vieweg; Auflage: 2014 – ISBN 978-3-658-06218-7

[MAR10] Marczak, Edward; Neagle, Greg: *Enterprise Mac Managed Preferences*- 2010 - Apress; Auflage: 2010 – ISBN 978-1-4302-2937-7

[MIC16_1] Microsoft: *Linkeroptionen* - 2016 - https://msdn.microsoft.com/de-de/library/y0zzbyt4.aspx, verfügbar am 25.09.18, 07:28

[MIC16_2] Microsoft: *__stdcall* - 2016 - https://msdn.microsoft.com/de-

de/library/zxk0tw93.aspx, verfügbar am 25.09.18, 07:28

[MIC16_3] Microsoft: *Exportieren aus einer DLL mithilfe von "__declspec(dllexport)"*- 2016 - https://msdn.microsoft.com/de-de/library/a90k134d.aspx, verfügbar am 25.09.18, 07:28

[MIC18_1] Microsoft: *Was ist eine DLL?*- 2018 - https://support.microsoft.com/de-de/help/815065/what-is-a-dll, verfügbar am 25.09.18, 07:28

[MIC18_2] Microsoft: *Vorkompilierte Headerdateien*- 2018 - https://msdn.microsoft.com/de-de/library/h552b3ca.aspx, verfügbar am 25.09.18, 07:28

[MIC18_3] Microsoft: *Informationen zu Microsoft Visual C++ Redistributable Packages* - https://support.microsoft.com/de-de/help/2977003/the-latest-supported-visual-c-downloads, verfügbar am 25.09.18, 07:28

[RIC16] Ishida, Richard, W3C: Das BOM (byte-order mark) in HTML – 2016 - https://www.w3.org/International/questions/qa-byte-order-mark.de, verfügbar am 25.09.18, 07:28

[XWAY18_1] X-Ways AG: *X-Ways Forensics: Integrierte Software für Computerforensik*- 2018 - http://www.x-ways.net/forensics/index-d.html, verfügbar am 25.09.18, 07:28

[XWAY18_2] X-Ways AG: *X-Ways Investigator*- 2018 - http://www.x-ways.net/investigator/index-d.html, verfügbar am 25.09.18, 07:28

[XWAY18_3] X-Ways AG: *X-Ways Forensics X-Tensions API Documentation*- 2018 - http://www.x-ways.net/forensics/x-tensions/api.html, verfügbar am 25.09.18, 07:28

[XWAY18_4] X-Ways AG: *X-Ways Forensics/ WinHex*- 2018 - http://www.x-ways.net/winhex/manual-d.pdf, verfügbar am 25.09.18, 07:28

Anlagen

Vergleich mit Kommerzieller Software …….. A-I

Quellcode der X-Tension ……...……. A-III

Anlage I - Vergleich mit Kommerzieller Software

Im Folgenden ist der Vergleich der verschiedenen BPList Parser zu sehen, oben die Version von „OSForensics“, in der Mitte „plist Editor Pro 2.5“ und unten das Ergebnis dieser Arbeit.

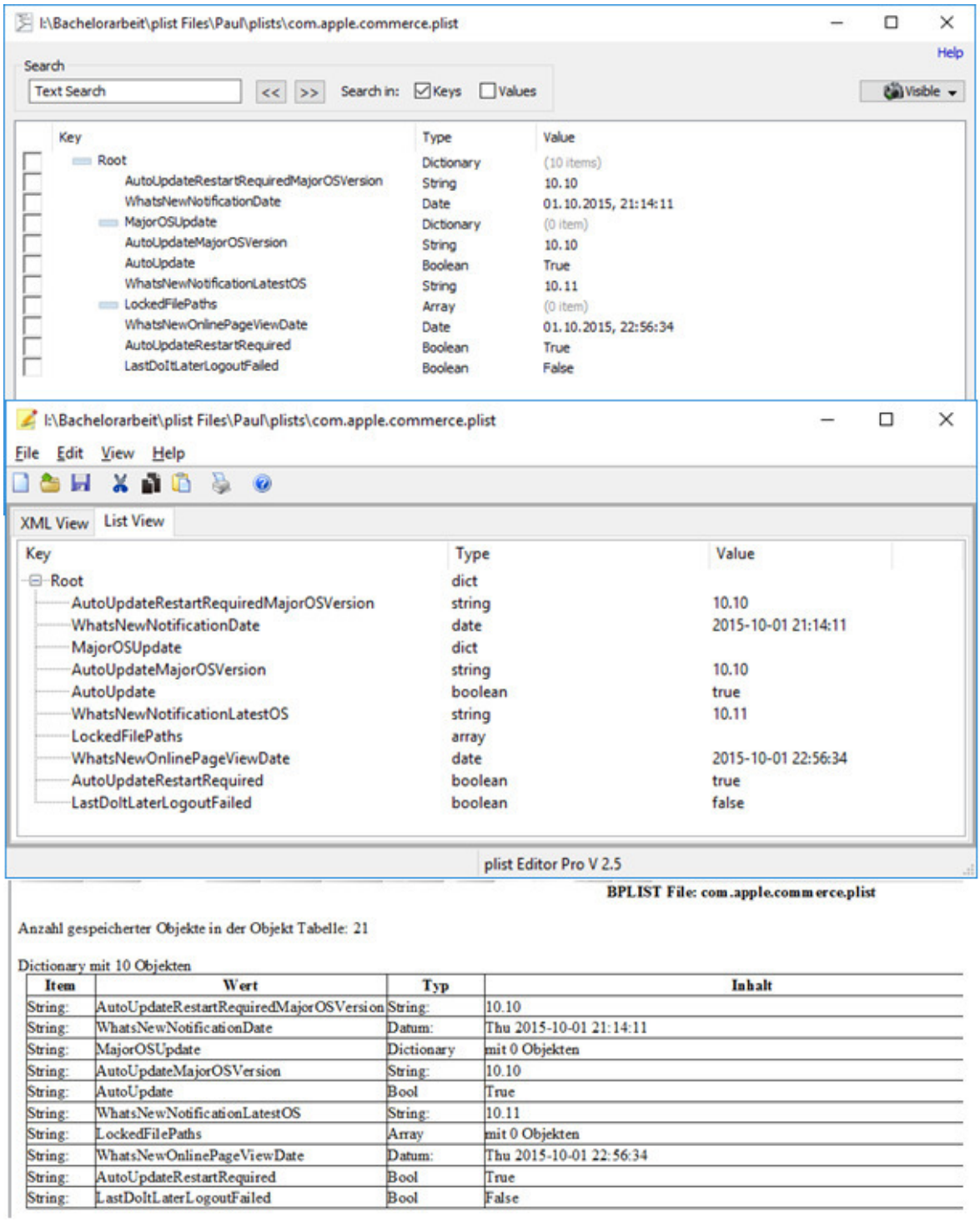
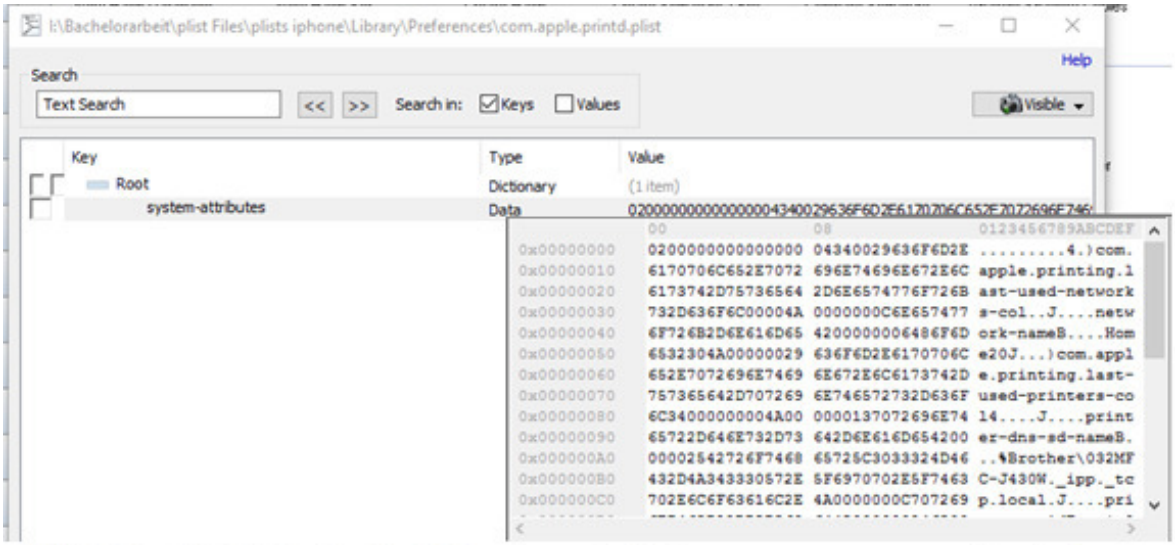
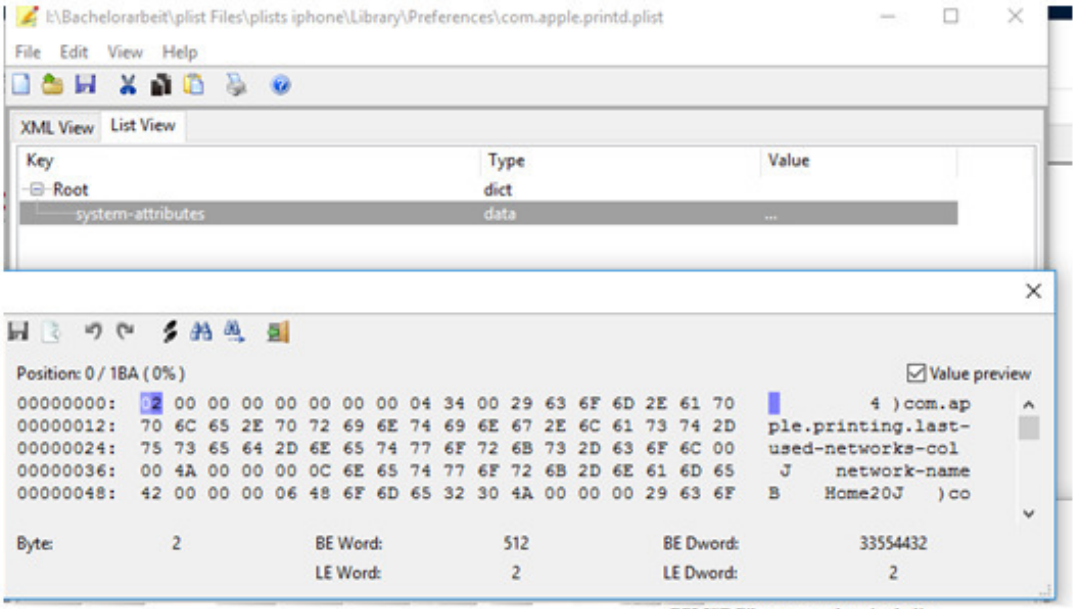


Abbildung 29: Anlage 1: Objekttypenvergleich (eigene Darstellung)





BPLIST File: com.apple.printd.plist

Anzahl gespeicherter Objekte in der Objekt Tabelle: 3

Dictionary mit 1 Objekten

Item	Wert	Typ	Inhalt
String:	system-attributes	Data mit 443 Byte Anzeige der ersten 200	020000000000000004340029636F6D2E 6170706C652E7072696E74696E672E6C 6173742D757365642D6E6574776F726B 732D636F6C00004A0000000C6E657477 6F726B2D6E616D6542000000006486F6D 6532304A00000029636F6D2E6170706C 652E7072696E74696E672E6C6173742D 757365642D7072696E746572732D636F 6C34000000004A00000013707269 6E7465722D646E732D73642D6E616D654200 00002542726F746865725C3033324D46 432D4A343330572E5F6970702E5F7463 702E6C6F63616C2E

Abbildung 30: Anlage 1: Objekttypenvergleich (eigene Darstellung)

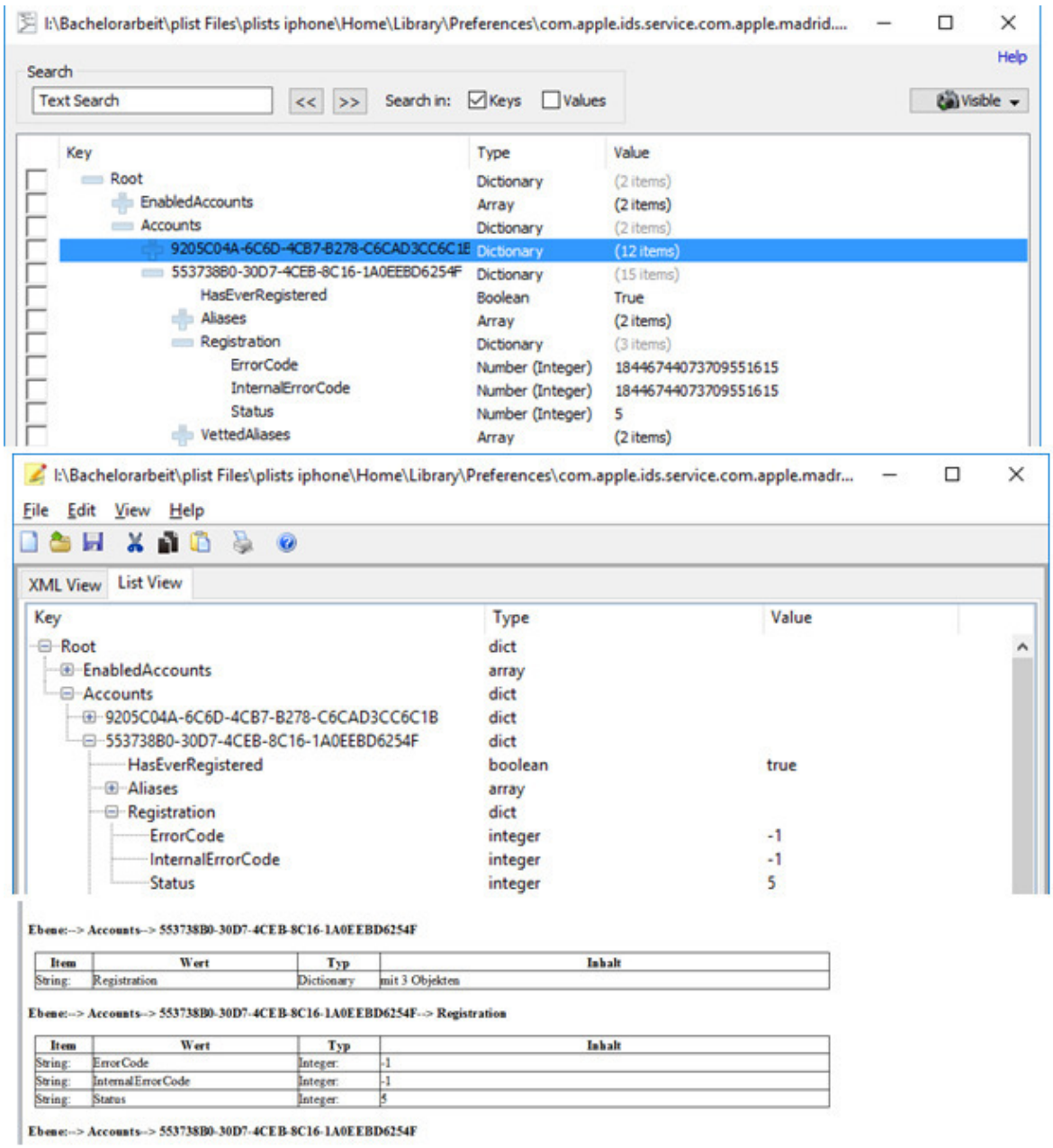


Abbildung 31: Anlage 1: Objekttypenvergleich (eigene Darstellung)

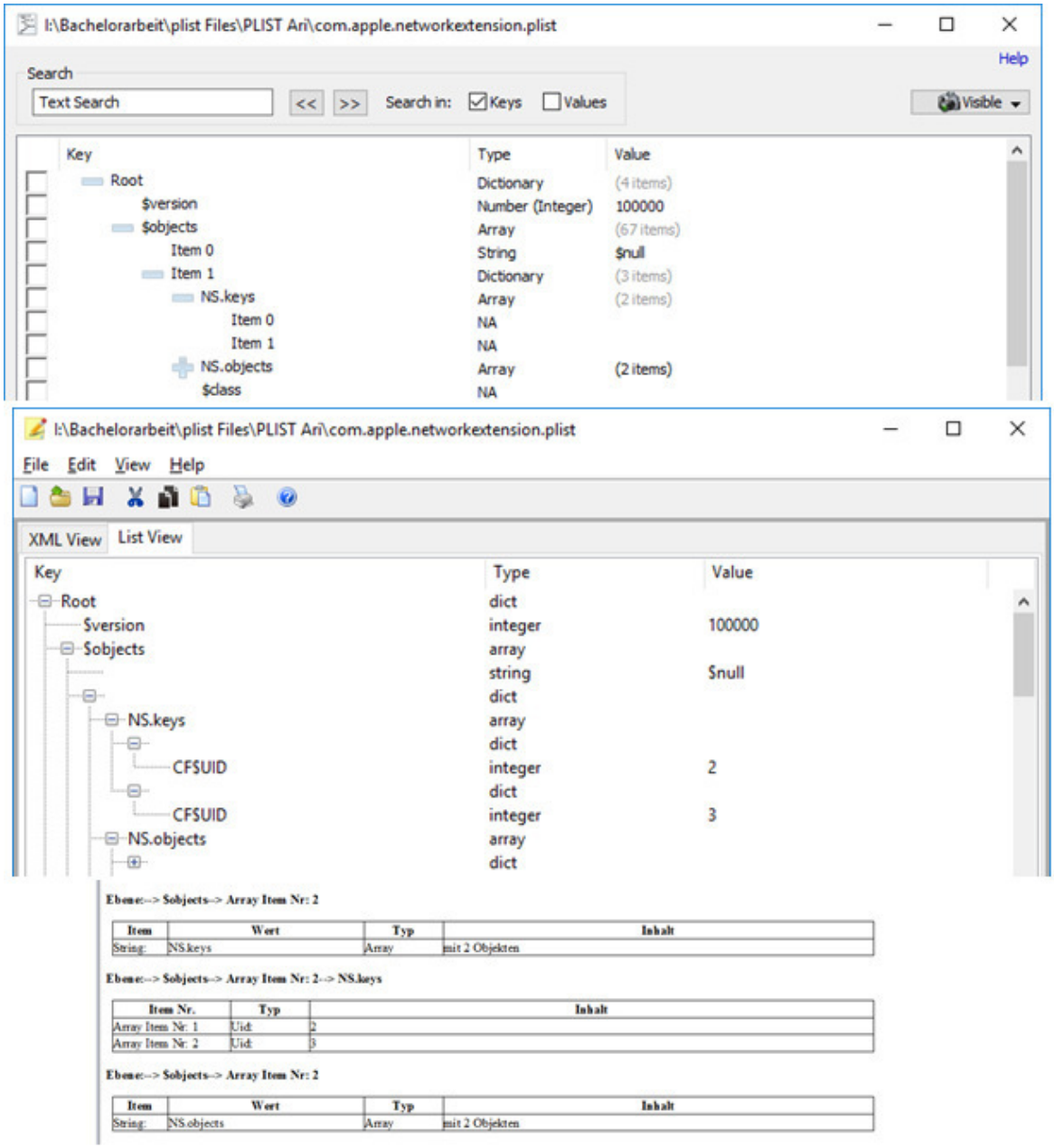


Abbildung 32: Anlage 1: Objekttypenvergleich (eigene Darstellung)

Anlage II - Quellcode der X-Tension

```
1  ////////////////////////////////////////////////////////////////////////////
2  // X-Tension API - template for new X-Tensions
3  // Copyright X-Ways Software Technology AG
4  // Please consult
5  // http://x-ways.com/forensics/x-tensions/api.html
6  // for current documentation
7
8  ////////////////////////////// BPListParser v. 0.1 ////////////////////////////
9  //   Dieses Projekt entstand im Rahmen einer Bachelorarbeit im Studiengang      //
10 //   Allgemeine und Digitale Forensik an der Hochschule Mittweida im Jahr       //
11 //   2018. Betreut wurde diese Arbeit durch Prof. Christian Hummert und         //
12 //   Prof. Dirk Pawlaszczyk. Der Quellcode darf unter Beachtung des Copyright   //
13 //   der X-Ways Software Technology AG kostenlos verwendet, verändert oder      //
14 //   weitergegeben werden. Es gibt keine Gewährleistung auf Funktionalität      //
15 //   oder Aktualisierung.   gez. Kittan Michael     michaelkittan@gmx.de        //
16 ////////////////////////////////////////////////////////////////////////////////
17
18 #include "x-tension.h"
19 #include <math.h>
20 #include <stdint.h>
21 #include <time.h>
22 #include <wchar.h>
23
24 #define EXPORT extern "C" __declspec(dllexport)
25
26 ////////////////////Funktionen definieren/////////////////
27 // Parsen INT Counters ( rechtes Nibble, Offset, Buffer der Datei, Rückgabewert Int
   Zahl )
28 uint64_t parseint(int objinfo, uint64_t inhalt, BYTE *file_buf);
29
30 // Containerdateitypen verarbeiten ( rechtes Nibble, Objekt in Offset Tabelle,
   Datei, Offsets der Offsettabelle, Ausgabebuffer, Rückgabewert: Ausgabebuffer)
31 wchar_t* parseArray(int objinfo, uint64_t markernummer, BYTE *file_buf, uint64_t
   markeroffset[4500], wchar_t *buffer);
32 wchar_t* parseSet(int objinfo, uint64_t markernummer, BYTE *file_buf, uint64_t
   markeroffset[4500], wchar_t *buffer);
33 wchar_t* parseDict(int objinfo, uint64_t markernummer, BYTE *file_buf, uint64_t
   markeroffset[4500], wchar_t *buffer);
34
35 // Inhalte zum Ausgabebuffer hinzufügen ( Zielbufferpointer, Zeichenanzahl zum
   zählen, Inhalt, wieviele Zeichen zusätzlich, Rückgabewert: Ausgabebuffer )
36 wchar_t *Ausgabe(wchar_t *buffer, const wchar_t *Zeile, wchar_t *Ausgabebuf, int
   zeichencount);
37
38 // Aufruf der Objektauswahl ( Datei, Offsets der Offset Tabelle, Objekt der Offset
   Tabelle, Ausgabebuffer
39 wchar_t *parseobjekt(BYTE *file_buf, uint64_t markeroffset[4500], int markernummer,
   wchar_t *buffer);
40
41 ////////////////////// Globale Variablendefinition //////////////////////
42 size_t SpeicherZugeordnet = 512;     // Reservierter Speicher
43 size_t Speichernutzung = 0;          // Aktuelle Speichernutzung
44 HANDLE hitem1 = 0;                   // Variable für Hitem Global
45 size_t file_size = 0;                // Dateigröße
46 int static offsize = 0, refsize = 0;// Offsize und efsize Global
47 INT64 numobjekt = 0;                 // Anzahl gespeicherter Objekte in der Datei
48 INT64 markerOfsettable = 0;          // Marker für Verschiebung der Offsettable
49 wchar_t Stringbuffer[3000];          // Buffer für String Ausgabe
50 int64_t negativint = 0;              // Variable für negativ Angabe Integer
51
52
53                                      // Variablen für die Zählung und Anzeige der
                                        Array,Set oder Dict
54 int ebene = 0;                       // Zähler für die Ebene
55 int ebenenlaenge[40];                // Speicherung der Länge für die Ebene
56 int stringlength = 0;                // Länge der einzelnen Strings
57 wchar_t tiefenanzeige[3000];         // Speicherung der kompletten Ebenenanzeige
58 size_t nutzungebene = 10;            // Anfangslänge der Ebenausgabe
59 bool table = 0;                      // Tabellenausgabe ja oder nein
60
61
62 EXPORT LONG __stdcall XT_Init(DWORD nVersion, DWORD nFlags, HANDLE hMainWnd,
63     void* lpReserved)
```

```
64  {
65      // Initialisieren der X-Ways API Funktionen
66      XT_RetrieveFunctionPointers();
67      XWF_OutputMessage(L"Viewer X-Tension BPList Parser von Kittan Michael v0.1
        geladen", 0);
68      return 1;
69  }
70  EXPORT PVOID __stdcall XT_View(HANDLE hItem, LONG nItemID, HANDLE hVolume, HANDLE
    hEvidence, PVOID lpReserved, PINT64 nResSize)
71  {
72      using namespace std;
73
74      // hitem1 und file_size global setzen
75      hitem1 = hItem;
76      file_size = (size_t)XWF_GetSize(hItem, (LPVOID)1);
77
78      wsprintf(tiefenanzeige, L"<b>Ebene:");
79
80      ////////////////////// Prüfen X-Tension Zuständig ////////////////////
81      *nResSize = -1;     // Rückgabewert das die Datei nicht mit dieser Xtention
        betrachtet werden soll
82
83      char sig_buffer[9] = {};
84      XWF_Read(hItem, 0, (BYTE*)sig_buffer, 8);
85
86      if (strcmp("bplist00", sig_buffer) != 0)
87      {
88          return NULL;
89      }
90
91      wchar_t *buffer = (wchar_t*)calloc(SpeicherZugeordnet,  // Anzahl zu
        reservierenden Elemente
92          sizeof(wchar_t));                                   // Größe des jeweeilgen
            Elements ( Hier 2 Byte für wchar_t)
93
94                                                              // Buffer mit Nullen
                                                                füllen
95      wmemset(buffer, 0, SpeicherZugeordnet);
96
97      // Prüfen auf mögliche Fehler beim Einlesen der Datei
98      if (buffer == NULL)
99      {
100         XWF_OutputMessage(L"BPListParser ERROR: Fehler beim Zuordnen des Speichers",
            0);
101         return NULL;
102     }
103
104     if (file_size <= 40)
105     {
106         *nResSize = 150;
107         wsprintf(buffer, L"ERROR: Dieses BPLIST File scheint Fehlerhaft zu
            sein(Kleiner als 40 Byte)");
108         return buffer;
109     }
110
111     if (file_size >= 100000)
112     {
113         *nResSize = 124;
114         wsprintf(buffer, L"ERROR: Dateien größer als 100 KB werden noch nicht
            unterstützt");
115         return buffer;
116     }
117
118     *nResSize = -2;     // -2 gibt Fehler beim Abarbeiten der Xtention an
119
120                         ////////////////////// Trailer auslesen ////////////////////
121
122     HANDLE heap = GetProcessHeap();
123     void *ptr;                      // Pointer
124     INT64  offsetot = 0;            // das Offset der Offsettable
125
126     BYTE *trailer = (BYTE*)HeapAlloc(heap, 0, file_size);
127
```

```
128         XWF_Read(hItem, file_size - 32, trailer, file_size);
129
130         // Null Bytes von index 0 -5
131         offsize = trailer[6];
132         refsize = trailer[7];
133
134         // numobjekt            Anzahl der Kodierten Objekte
135         ptr = &trailer[8];      // Zeiger zum Auslesen des 64 Bit Wertes
136         numobjekt = _byteswap_uint64(*((uint64_t *)ptr));
137
138         // topobjekt
139         ptr = &trailer[16];
140         markerOfsettable = _byteswap_uint64(*((uint64_t *)ptr));
141
142         // Offset Tabellen Offset
143         ptr = &trailer[24];
144         offsetot = _byteswap_uint64(*((uint64_t *)ptr));
145
146         if (offsetot > file_size || offsetot <9)
147         {
148             *nResSize = 90;
149             wsprintf(buffer, L"ERROR: Falsche Angabe des Offsets im Trailer");
150             HeapFree(heap, 0, trailer);
151             return buffer;
152         }
153
154         HeapFree(heap, 0, trailer);         // Speicher vom Trailer wieder Freigeben
155
156                                             // ---------------------- Offsets Tabelle
                                                auslesen -----------------------
157
158         BYTE *file_buf = (BYTE*)HeapAlloc(heap, 0, file_size);
159
160         XWF_Read(hItem,             // Handle auf die Datei
161             offsetot,               // von wo an gelesen werden soll
162             file_buf,               // wo rein geschrieben werden soll
163             file_size);             // bis wohin gelesen werden soll
164
165                                     // in dem OffsetArray stehen danach alle Offsets in
                                        der Datei wo ein Objekt beginnt, das i gibt dabei
                                        die Nummer des Objekts an ( Marker )
166         uint64_t markeroffset[4500];
167         if (numobjekt > 4500)
168         {
169             *nResSize = 90;
170             wsprintf(buffer, L"ERROR: Die BPLIST hat mehr als 4500 Objekte.");
171             HeapFree(heap, 0, file_buf);
172             return buffer;
173         }
174         for (int i = 0; i < numobjekt; i++)
175         {
176             if (offsize == 1)
177             {
178                 markeroffset[i] = file_buf[i];
                    // 8 Bit lesen
179             }
180             else if (offsize == 2)
                // 16 Bit lesen
181             {
182                 markeroffset[i] = (uint64_t)(((uint16_t *)file_buf)[i]);
183                 markeroffset[i] = _byteswap_ushort((uint16_t)markeroffset[i]);
184             }
185             else if (offsize == 4)
186             {
187                 markeroffset[i] = (uint64_t)(((uint32_t *)file_buf)[i]);
                    // 32 Bit lesen
188                 markeroffset[i] = _byteswap_ulong((uint32_t)markeroffset[i]);
189             }
190             else if (offsize == 8)
191             {
192                 markeroffset[i] = (uint64_t)(((uint64_t *)file_buf)[i]);
                    // 64 Bit lesen
193                 markeroffset[i] = _byteswap_uint64((uint64_t)markeroffset[i]);
```

```
194             }
195             else
196             {
197                 *nResSize = 110;
198                 wsprintf(buffer, L"ERROR: Fehler in der Datei, nicht unterstützte
                    Offsize");
199                 HeapFree(heap, 0, file_buf);
200                 return buffer;
201             }
202
203             // Prüfen Werte der Offset Tabelle
204             if (markeroffset[i] > file_size)
205             {
206                 *nResSize = 94;
207                 wsprintf(buffer, L"ERROR: Fehler beim Auslesen der Offset Tabelle");
208                 HeapFree(heap, 0, file_buf);
209                 return buffer;
210             }
211         }
212         HeapFree(heap, 0, file_buf);            // den File Buffer wieder freigeben
213
214                                                 // ----------------------- Auswerten
                                                    derObjectTable
                                                    ----------------------------
215
216         file_buf = (BYTE*)HeapAlloc(heap, 0, file_size);
217         XWF_Read(hItem, 0, file_buf, file_size);            // Komplette Datei wird in
            file_buf geschrieben
218
219                                                             /// HTML Einbindung
220         wchar_t Ausgabebuf[500];    // Temporärer Buffer für Ausgabe
221         buffer[0] = 0xFeFF;         // BOW für HTML Code
222         Speichernutzung = 1;        // Zähler wieviel Byte im Buffer aktuell gespeichert
            sind
223         int zeichencount = 1;       // Zähler für Länge des Int Wertes
224         uint64_t intlaengeu;        // Variable zum Berechnen der Zeichenlänge eines
            Integers
225
226         wsprintf(Ausgabebuf, L"<HTML><BODY><center><b> BPLIST File: %s</b>
            </center><br>", XWF_GetItemName(nItemID));
227         buffer = Ausgabe(buffer, L"<HTML><BODY><center><b> BPLIST File: </b>
            </center><br>", Ausgabebuf, wcslen(XWF_GetItemName(nItemID)));
228
229         intlaengeu = numobjekt;
230         while ((intlaengeu = intlaengeu / 10) != 0) zeichencount++;
231
232         wsprintf(Ausgabebuf, L"Anzahl gespeicherter Objekte in der Objekt Tabelle:
            %d<br><br>", numobjekt);
233         buffer = Ausgabe(buffer, L"Anzahl gespeicherter Objekte in der Objekt Tabelle:
            <br><br>", Ausgabebuf, zeichencount);
234
235         // ---------------------------
236
237         // Aufruf des 0 Objekts
238         buffer = parseobjekt(file_buf,      // Gespeicherter Dateiinhalt
239             markeroffset,                   // Gespeicherte Offsets der Offset Tabelle
240             0,                              // Objektnummer
241             buffer);                        // Buffer in den geschrieben wird
242
243                                             // Html schließen
244         wsprintf(Ausgabebuf, L"</BODY></HTML>");
245         buffer = Ausgabe(buffer, L"</BODY></HTML>", Ausgabebuf, 0);
246
247         *nResSize = Speichernutzung * 2;    // mal 2 weil wchar
248
249         return buffer;
250     }
251
252     wchar_t *parseobjekt(BYTE *file_buf, uint64_t markeroffset[4500], int markernummer,
        wchar_t *buffer) {
253
254         // --------------------- Variablen Definieren ---------------------
255         uint64_t inhalt = markeroffset[markernummer];   // Offset des Objekts ( Alle
```

```
        Werte der Offsetable )
256     uint64_t ergebnis = 0;                  // Ergebnisvariable
257     int objtyp = 0;                         // Objekt Typen Variable festlegen z.B: int,
        fil, bool
258     int objinfo = 0;                        // Info zu dem Typ festlegen z.B. Länge
259     struct tm ts;                           // Hilfsvariable Datumsausgabe
260     double datum;                           // Hilfsvariable für Datumsausgabe
261     int length = 0;                         // Byte länge wenn Int Counter
262     wchar_t *StringZeiger = Stringbuffer;   // Zeiger für ASCII Ausgabe
263     float ergfloat = 0;                     // Real 4 Byte Variable
264     double ergdouble = 0;                   // Real 8 Byte Variable
265     void *ptr = 0;                          // Pointer für UID und Byteswap
266     uint64_t position = 0;                  // Position für Auslesen der Werte
267     wchar_t Ausgabebuf[2000];               // Temporärer Buffer für Ausgabe
268     int64_t intlaenge = 0;                  // Hilfsvariable bei Zeichenberechnung von
        Int 8 Byte Werten
269     uint64_t intlaengeu = 0;                // Hilfsvariable bei Zeichenberechnung von
        Int 8 Byte Werten unsignd
270     HANDLE heap1;                           // Zusätzlichen Speicher initialisieren
271     char *file_tmp;                         // Variable für Temp Buffer
272     char* hex;                              // Pointer für Data Ausgabe
273     int j = 0;                              // Hilfszählvariable
274     int zeichencount = 0;                   // Zählvariable für Länge der Int Zahlen
275     wchar_t *file_tmp16;                    // Filetemp 16 Bit für UNICODE
276     HANDLE heapunicode;                     // Speicher für Unicode initalisieren
277
278                                             // -------------------- Auswahl des
                                                Objekttyps ---------------------
279
280     objtyp = (file_buf[inhalt] & 0xF0);         // Bitweise UND Verknüpfung
281     objtyp = objtyp >> 4;                       // 4 Bits nach rechts verschieben
282     objinfo = (file_buf[inhalt] & 0x0F);        // zweites Nibble Speichern
283
284     switch (objtyp)
285     {
286     case 0x0:
287         switch (objinfo)
288         {
289         case 0x0:
290             //  XWF_OutputMessage(L" Datentyp: NULL ", 0);
291             wsprintf(Ausgabebuf, L"NA");
292             return Ausgabe(buffer, L"NA", Ausgabebuf, 0);
293
294         case 0x8:
295             //  XWF_OutputMessage(L" Datentyp: BOOL = FALSE", 0);
296             wsprintf(Ausgabebuf, L"Bool</td><td>False");
297             return Ausgabe(buffer, L"Bool</td><td>False", Ausgabebuf, 0);
298
299         case 0x9:
300             //XWF_OutputMessage(L" Datentyp: BOOL = TRUE", 0);
301             wsprintf(Ausgabebuf, L"Bool</td><td>True");
302             return Ausgabe(buffer, L"Bool</td><td>True", Ausgabebuf, 0);
303
304         case 0xF:
305             //XWF_OutputMessage(L" Datentyp: FILL BYTE", 0);
306             wsprintf(Ausgabebuf, L"Fill Byte");
307             return Ausgabe(buffer, L"Fill Byte", Ausgabebuf, 0);
308
309         default:
310             XWF_OutputMessage(L"SIMPLE PARSE - Nichts\n", 0);
311             wsprintf(Ausgabebuf, L"NA");
312             return Ausgabe(buffer, L"NA", Ausgabebuf, 0);
313         }
314
315     case 0x1:                                                   // Datentyp INT
316         ergebnis = parseint(objinfo,    // Länge der Folgenden Bytes
317             inhalt,                     // Position innerhalb der Datei
318             file_buf);                  // Buffer mit der Datei
319
320         if (negativint < 0)
321         {
322             intlaenge = negativint;
323
```

```
324                 zeichencount = 2;
325                 while ((intlaenge = intlaenge / 10) != 0)   zeichencount++;
326
327                 wsprintf(Ausgabebuf, L"Integer:</td><td>%d", negativint);
328                 negativint = 0;
329                 return Ausgabe(buffer, L"Integer:</td><td>", Ausgabebuf, zeichencount);
330             }
331             else
332             {
333                 intlaengeu = ergebnis;
334                 zeichencount = 1;
335                 while ((intlaengeu = intlaengeu / 10) != 0) zeichencount++;
336
337                 wsprintf(Ausgabebuf, L"Integer:</td><td>%I64u", ergebnis);
338                 return Ausgabe(buffer, L"Integer:</td><td>", Ausgabebuf, zeichencount);
339             }
340
341         case 0x2:                                                   // Datentyp
            Real
342             length = 0x1 << objinfo;
343             inhalt++;
344             if (length == 4)
345             {
346                 ergebnis = file_buf[inhalt] << 24;
347                 ergebnis = ergebnis + (file_buf[inhalt + 1] << 16);
348                 ergebnis = ergebnis + (file_buf[inhalt + 2] << 8);
349                 ergebnis = ergebnis + (file_buf[inhalt + 3]);
350
351                 memcpy(&ergfloat, &ergebnis, sizeof(float));
352
353                 intlaengeu = (long int)ergfloat;
354                 swprintf(Ausgabebuf, 200, L"REAL:</td><td>%f", ergfloat);
355
356             }
357             else if (length == 8) {
358
359                 ergebnis = file_buf[inhalt] << 24;
360                 ergebnis = ergebnis + (file_buf[inhalt + 1] << 16);
361                 ergebnis = ergebnis + (file_buf[inhalt + 2] << 8);
362                 ergebnis = ergebnis + ((file_buf[inhalt + 3]));
363                 ergebnis = ergebnis << 32;
364                 ergebnis = ergebnis + (file_buf[inhalt + 4] << 24);
365                 ergebnis = ergebnis + (file_buf[inhalt + 5] << 16);
366                 ergebnis = ergebnis + (file_buf[inhalt + 6] << 8);
367                 ergebnis = ergebnis + (file_buf[inhalt + 7]);
368
369                 memcpy(&ergdouble, &ergebnis, sizeof(double));
370
371                 intlaengeu = (long int)ergdouble;
372
373                 swprintf(Ausgabebuf, 200, L"REAL:</td><td>%f", ergdouble);
374             }
375             else
376             {
377                 wsprintf(Ausgabebuf, L"REAL</td><td> Falsche Byte Länge:%3d Byte lang",
                    length);
378                 return Ausgabe(buffer, L"REAL</td><td> Falsche Byte Länge:%3d Byte
                    lang", Ausgabebuf, 0);
379             }
380
381
382             zeichencount = 1;
383             if (intlaengeu == 0) zeichencount++;
384
385             // Zeichen zählen mit 6 Nachkommastellen
386             else if (ergdouble < 0 || ergfloat < 0)     // Vorzeichenstelle
387             {
388                 zeichencount++;
389                 intlaengeu = intlaengeu * (-1);
390             }
391
392             while ((intlaengeu = intlaengeu / 10) != 0)
393                 zeichencount++;
```

```
394
395             zeichencount = zeichencount + 7;        // 6 Kommastellen + 1 Punkt
396
397             return Ausgabe(buffer, L"REAL:</td><td>", Ausgabebuf, zeichencount);
398
399         case 0x3:                                           // Date Format
400             if (objinfo != 3)
401             {
402                 wsprintf(Ausgabebuf, L"Datum</td><td> Ungültiges Datum");
403                 return Ausgabe(buffer, L"Datum</td><td> Ungültiges Datum", Ausgabebuf, 0);
404             }
405             ptr = &file_buf[inhalt + 1];                // Übergibt Zeiger vom Beginn
                des Datums
406
407             ergebnis = _byteswap_uint64(*((uint64_t *)ptr));
408             memcpy(&datum, &ergebnis, 8);               // Kopiert von Ergebnis in Datum
                ohne Konvertierung
409                                                         // 978307200 = Datum Zeit Format
                                                            geparst vom 01.01.2001 GMT
410             datum += 978307200;                         // + EPOCH Time berechnen
411             ergebnis = (uint64_t)datum;
412
413             localtime_s(&ts, (const time_t *)&ergebnis);
414
415             wcsftime(Ausgabebuf, 150, L"Datum:</td><td>%a %Y-%m-%d %H:%M:%S", &ts);
416             return Ausgabe(buffer, L"Datum:</td><td>                        ",
                Ausgabebuf, 0);
417
418         case 0x4:                                                   // Datenty DATA
419             heap1 = GetProcessHeap();   // Neuen TeBuffer definieren
420             file_tmp = (char*)HeapAlloc(heap1, 0, file_size);
421             hex = file_tmp;
422             if (objinfo != 0xf)
423             {
424                 ergebnis = objinfo;
425                 position = inhalt + 1;
426             }
427             else if (objinfo == 0xf)
428             {
429                 objinfo = (file_buf[inhalt + 1] & 0x0F);
430                 length = (int)pow(2, objinfo);
431                 ergebnis = parseint(objinfo, inhalt + 1, file_buf);
432                 position = inhalt + 2 + length;
433             }
434
435             if (position + ergebnis > file_size)
436             {
437                 wsprintf(Ausgabebuf, L"ERROR: Falsche Byte Länge beim Typ Data");
438                 buffer = Ausgabe(buffer, L"ERROR: Falsche Byte Länge beim Typ Data",
                    Ausgabebuf, 0);
439                 HeapFree(heap1, 0, file_tmp);
440                 return buffer;
441             }
442
443             // File_temp befüllen
444             XWF_Read(hitem1, position, (BYTE *)file_tmp, (DWORD)(position + ergebnis));
445
446             ergebnis = ergebnis * 2;
447
448             if (ergebnis> 400)
449             {
450                 wsprintf(Ausgabebuf, L"Data mit %8d Byte Anzeige der ersten
                    200</td><td>", ergebnis / 2);
451                 buffer = Ausgabe(buffer, L"Data mit          Byte Anzeige der ersten
                    200</td><td>", Ausgabebuf, 0);
452                 ergebnis = 400;
453             }
454             else
455             {
456                 wsprintf(Ausgabebuf, L"Data mit %8d Byte</td><td>", ergebnis / 2);
457                 buffer = Ausgabe(buffer, L"Data mit          Byte</td><td>", Ausgabebuf,
                    0);
458             }
```

```
459
460             j = 0;
461             for (int i = 0; i < ergebnis; i++)  // Zeichen einzeln in den Buffer
                schreiben ( *2 wegen whitechar)
462             {
463                 wsprintf(Ausgabebuf + i * 2 + j, L"%02X", (*(hex + i) & 0xFF));
464                 // Leerzeichen alle 16 Bytes
465                 if (((i + 1) % 16 == 0) && i != 0)
466                 {
467                     j = j + 2;
468                     wsprintf(Ausgabebuf + i * 2 + j, L" ");
469                 }
470             }
471
472             // Erweiterung des Speichers wenn Speicherbedarf zu Groß
473             while (Speichernutzung + ergebnis + j / 2 >= SpeicherZugeordnet)
474             {
475                 // Exponentielle Vergößerung
476                 SpeicherZugeordnet *= 2;
477
478                 // Speicherzuordnung erfolgreich
479                 buffer = (wchar_t *)realloc(buffer, SpeicherZugeordnet * sizeof(wchar_t));
480                 if ((buffer == NULL))
481                 {
482                     XWF_OutputMessage(L"BPListParser ERROR: Fehler beim Zuordnen des
                    größeren Speichers", 0);
483                     HeapFree(heap1, 0, file_tmp);
484                     return NULL;
485                 }
486             }
487
488             // Neuen Inhalt an Speicher anhängen
489             wmemcpy(buffer + Speichernutzung        // Zielbuffer
490                 , Ausgabebuf                        // Quellbuffer
491                 , (size_t)ergebnis + j / 2 - 1);        // Anzahl der zu kopierenden
                    Zeichen
492
493
494
495                                                     // Aktuelle Speichernutzung
                                                        festhalten
496             Speichernutzung += (size_t)ergebnis + j / 2 - 1;
497             HeapFree(heap1, 0, file_tmp);
498             return buffer;
499
500         case
            0x5:
                    // ASCII
501
502             heap1 = GetProcessHeap();               // Neuen Temporären Buffer definieren
503             file_tmp = (char*)HeapAlloc(heap1, 0, file_size);
504             if (objinfo != 0xf)
505             {
506                 ergebnis = objinfo;
507                 position = inhalt + 1;
508             }
509             else if (objinfo == 0xf)
510             {
511                 objinfo = (file_buf[inhalt + 1] & 0x0F);                // objektInfo
                    neu einlesen da ein Byte weiter
512
513                 length = (int)pow(2, objinfo);
514                 ergebnis = parseint(objinfo, inhalt + 1, file_buf);
515                 position = inhalt + 2 + length;
516             }
517
518             if (position + ergebnis > file_size)
519             {
520                 wsprintf(Ausgabebuf, L"ERROR: Falsche Byte Länge beim Typ String");
521                 buffer = Ausgabe(buffer, L"ERROR: Falsche Byte Länge beim Typ String",
                    Ausgabebuf, 0);
522                 HeapFree(heap1, 0, file_tmp);
523                 return buffer;
```

```
524         }
525         // File_temp befüllen von Anfang bis Ende des Strings
526         XWF_Read(hitem1, position, (BYTE *)file_tmp, (DWORD)(position + ergebnis));
527 
528         wsprintf(Ausgabebuf, L"String:</td><td>");
529         buffer = Ausgabe(buffer, L"String:</td><td>", Ausgabebuf, 0); ;
530 
531         StringZeiger = Stringbuffer;
532 
533         // Prüfen ob String länger als Stringbuffer sonst abschneiden
534         if (ergebnis > 1495)
535             ergebnis = 1495;
536 
537         for (int i = 0; i < ergebnis; i++)
538             *(StringZeiger++) = file_tmp[i];
539 
540         // Variable für EbenenAngabe
541         stringlength = (size_t)ergebnis;
542 
543         // Erweiterung des Speichers wenn Speicherbedarf zu Groß
544         while (Speichernutzung + ergebnis >= SpeicherZugeordnet)
545         {
546             // Exponentielle Vergößerung
547             SpeicherZugeordnet *= 2;
548 
549             // Speicherzurdnung erfolgreich
550             buffer = (wchar_t *)realloc(buffer, SpeicherZugeordnet * sizeof(wchar_t));
551             if ((buffer == NULL))
552             {
553                 XWF_OutputMessage(L"Fehler beim Zuordnen des größeren Speichers", 0);
554                 HeapFree(heap1, 0, file_tmp);
555                 return NULL;
556             }
557         }
558         // Neuen Inhalt an Speicher anhängen
559         wmemcpy(buffer + Speichernutzung        // Zielbuffer
560             , Stringbuffer                      // Quellbuffer
561             , (size_t)ergebnis);                // Anzahl der zu kopierenden Zeichen
562 
563                                                 // Aktuelle Speichernutzung festhalten
564         Speichernutzung += (size_t)ergebnis;
565         //      wsprintf(Ausgabebuf, L"</td>");
566         //      buffer = Ausgabe(buffer, L"</td>", Ausgabebuf,0);
567         HeapFree(heap1, 0, file_tmp);
568         return buffer;
569 
570     case 0x6:                                   // UNICODE
571 
572                                                 // Buffer defninieren
573         heapunicode = GetProcessHeap();
574         file_tmp16 = (wchar_t*)HeapAlloc(heapunicode, 0, (uint16_t)file_size);
575         if (objinfo != 0xf)
576         {
577             ergebnis = (objinfo);
578             position = inhalt + 1;
579         }
580         else if (objinfo == 0xf)
581         {
582             objinfo = (file_buf[inhalt + 1] & 0x0F);                // objektInfo
                neu einlesen da ein Byte weiter
583             length = (int)pow(2, objinfo);
584             ergebnis = (parseint(objinfo, inhalt + 1, file_buf));
585             position = inhalt + 2 + length;
586         }
587 
588         if (position + ergebnis * 2 > file_size)
589         {
590             wsprintf(Ausgabebuf, L"ERROR: Falsche Byte Länge beim Typ Unicode");
591             buffer = Ausgabe(buffer, L"ERROR: Falsche Byte Länge beim Typ Unicode",
                Ausgabebuf, 0);
592             HeapFree(heapunicode, 0, file_tmp16);
593             return buffer;
594         }
```

```
595
596             // Einlesen in file_tmp vom Ende des Intcount bis zum Ende des Strings
597             XWF_Read(hitem1, position, (BYTE *)file_tmp16, (DWORD)(position + ergebnis *
                2));
598
599             wsprintf(Ausgabebuf, L"Unicode:</td><td>");
600             buffer = Ausgabe(buffer, L"Unicode:</td><td>", Ausgabebuf, 0);
601
602             StringZeiger = Stringbuffer;
603
604             if (ergebnis > 1495)
605                 ergebnis = 1495;
606
607             for (int i = 0; i < ergebnis; i++)
608                 *(StringZeiger++) = _byteswap_ushort((uint16_t)file_tmp16[i]);
609
610             // Variable für EbenenAngabe
611             stringlength = (size_t)ergebnis;
612
613             while (Speichernutzung + ergebnis >= SpeicherZugeordnet)
614             {
615                 // Exponentielle Vergößerung
616                 SpeicherZugeordnet *= 2;
617
618                 // Speicherzuordnung erfolgreich
619                 buffer = (wchar_t *)realloc(buffer, SpeicherZugeordnet * sizeof(wchar_t));
620                 if ((buffer == NULL))
621                 {
622                     XWF_OutputMessage(L"Fehler beim Zuordnen des größeren Speichers", 0);
623                     HeapFree(heapunicode, 0, file_tmp16);
624                     return NULL;
625                 }
626             }
627
628             // Neuen Inhalt an Speicher anhängen
629             wmemcpy(buffer + Speichernutzung        // Zielbuffer
630                 , Stringbuffer                      // Quellbuffer
631                 , (size_t)ergebnis);                // Anzahl der zu kopierenden Zeichen
632
633                                                     // Aktuelle Speichernutzung festhalten
634             Speichernutzung += (size_t)ergebnis;
635             HeapFree(heapunicode, 0, file_tmp16);
636
637             return buffer;
638
639         case 0x8:                                                   // Datentyp
            UID
640             ergebnis = objinfo + 1;
641
642             ptr = &file_buf[inhalt + 1];            // liest den Inhalt von dieser
                Position aus
643             switch (ergebnis) {
644             case 1:
645                 ergebnis = *((uint8_t *)ptr);               // hier kein Byteswap da nur
                    8 Bit
646                 break;
647             case 2:
648                 ergebnis = (uint64_t)_byteswap_ushort(*((uint16_t *)ptr));      //
                    Byteswap wechselt Big Endian zu Little Endian mit 16 Bit
649                 break;
650             case 4:
651                 ergebnis = (uint64_t)_byteswap_ulong(*((uint32_t *)ptr));       //
                    Byteswap wechselt Big Endian zu Little Endian mit 32 Bit
652                 break;
653             case 8:
654                 ergebnis = (uint64_t)_byteswap_uint64(*((uint64_t *)ptr));      //
                    Byteswap wechselt Big Endian zu Little Endian mit 64 Bit
655                 break;
656
657             default:
658
659                 wsprintf(Ausgabebuf, L"Uid:</td><td>Fehler beim Auslesen des UID Wertes");
660                 return Ausgabe(buffer, L"Uid</td><td>Fehler beim Auslesen des UID
```

```
                Wertes", Ausgabebuf, 0);
661         }
662
663         wsprintf(Ausgabebuf, L"Uid:</td><td>%4d", ergebnis);
664         return Ausgabe(buffer, L"Uid:</td><td>    ", Ausgabebuf, 0);
665
666     case 0xA:                                                           //
        Datentyp Array
667         buffer = parseArray(objinfo, markernummer, file_buf, markeroffset, buffer);
668         return buffer;
669
670     case 0xB:                                                           //
        Datentyp SET
671         buffer = parseSet(objinfo, markernummer, file_buf, markeroffset, buffer);
672         return buffer;
673
674     case 0xD:                                                           //
        Datentyp Dict
675         buffer = parseDict(objinfo, markernummer, file_buf, markeroffset, buffer);
676         return buffer;
677
678     default:
679         wsprintf(Ausgabebuf, L"Unbekannter Datentyp");
680         return Ausgabe(buffer, L"Unbekannter Datentyp", Ausgabebuf, 0);
681
682     }
683     wsprintf(Ausgabebuf, L"Unbekannter Datentyp");
684     return Ausgabe(buffer, L"Unbekannter Datentyp", Ausgabebuf, 0);
685
686 }
687
688 // Speicher freigeben Variablen zurücksetzen
689 EXPORT BOOL __stdcall XT_ReleaseMem(PVOID lpBuffer)
690 {
691     if (NULL != lpBuffer)
692     {
693         free(lpBuffer);
694         SpeicherZugeordnet = 512;
695         Speichernutzung = 0;
696         nutzungebene = 10;
697         ebene = 0;
698         wsprintf(tiefenanzeige, L"<b>Ebene:");
699         return true;
700     }
701     return false;
702 }
703
704 // Int Counter auswerten Rückgabewert in Int
705 // Übergabe: Länge ( vor ^2), Stelle im Buffer, Buffer
706 uint64_t parseint(int objinfo, uint64_t inhalt, BYTE *file_buf) {
707
708     uint64_t ergebnis = 0;
709
710     void *ptr = &file_buf[inhalt + 1];          // liest den Inhalt von dieser
        Position aus
711     switch (0x01 << objinfo) {                  // verschiebt den Wert bitweise nach
        Links für die Anzahl von objinfo
712                                                 // ergibt dann automatisch die
                                                    Potenzen von 2
713     case 1:
714         ergebnis = *((uint8_t *)ptr);               // hier kein Byteswap da nur 8 Bit
715         break;
716
717     case 2:
718         ergebnis = (uint64_t)_byteswap_ushort(*((uint16_t *)ptr));      // Byteswap
            wechselt Big Endian zu Little Endian mit 16 Bit
719         break;
720
721     case 4:
722         ergebnis = (uint64_t)_byteswap_ulong(*((uint32_t *)ptr));       // Byteswap
            wechselt Big Endian zu Little Endian mit 32 Bit
723         break;
724
```

```
725         case 8:     // 8 Byte Integer werden nicht unsigned definiert
726             ergebnis = (int64_t)_byteswap_uint64(*((int64_t *)ptr));        // Byteswap
                wechselt Big Endian zu Little Endian mit 64 Bit hier kein uint
727             negativint = (int64_t)_byteswap_uint64(*((int64_t *)ptr));      // Variable
                für Minus Werte
728             break;
729
730         default:
731             XWF_OutputMessage(L"BPListParser ERROR: Fehler bei der Integer Berechnung",
                0);
732             break;
733         }
734         return ergebnis;
735     }
736
737     wchar_t* parseDict(int objinfo, uint64_t markernummer, BYTE *file_buf, uint64_t
        markeroffset[4500], wchar_t *buffer) {
738
739         uint64_t inhalt = markeroffset[markernummer], objektpaare = 0, position = 0;
740         wchar_t Ausgabebuf[500];            // Temporärer Buffer für Ausgabe
741         int objektnummer, objektnummer2, j = 0;
742
743         int length = 0;
744         if (objinfo != 0xf)
745         {
746             objektpaare = objinfo;      // Anzahl der Objektpaare
747         }
748         else if (objinfo == 0xf)
749         {
750             inhalt++;
751             objinfo = (file_buf[inhalt] & 0x0F);                // objektInfo neu
                einlesen da ein Byte weiter
752             length = (int)pow(2, objinfo);                      // length rechnet für
                die eigene Funktion
753             objektpaare = parseint(objinfo, inhalt, file_buf);  // in INT wird länge des
                Int selber berechnet
754         }
755
756         if (objektpaare != 0)
757         {
758             ebene++;
759             if (markernummer == 0)
760             {
761                 wsprintf(Ausgabebuf, L"Dictionary mit %4d Objekten<br>", objektpaare);
762                 buffer = Ausgabe(buffer, L"Dictionary mit      Objekten<br>",
                    Ausgabebuf, 0);
763             }
764             else
765             {
766                 wsprintf(Ausgabebuf, L"Dictionary</td> <td>mit %4d
                    Objekten</td></tr></table><br>", objektpaare);
767                 buffer = Ausgabe(buffer, L"Dictionary</td> <td> mit
                    Objekten</td></tr></table><br>", Ausgabebuf, 0);
768
769                 // nutzungsebene = Aktueller Pointer auf Array tiefenanzeige
770                 // Stringbuffer = letzer gespeicherter String
771                 // stringlength = Länge des letzten Objekts
772
773                 // Ausgabe Ebene und Prüfung auf Max Länge der Buffer
774                 if (((nutzungebene + stringlength + 4) < 1500) && (ebene <40))
775                 {
776                     wsprintf(tiefenanzeige + nutzungebene, L"--> %s", Stringbuffer);
777                     nutzungebene = nutzungebene + stringlength + 4;
778                     buffer = Ausgabe(buffer, L"", tiefenanzeige, nutzungebene);
779                     ebenenlaenge[ebene] = stringlength + 4;
780
781                     wsprintf(Ausgabebuf, L"</b><br><br>");
782                     buffer = Ausgabe(buffer, L"</b><br><br>", Ausgabebuf, 0);
783                 }
784                 else
785                 {
786                     wsprintf(Ausgabebuf, L"<b>Zu viele Ebenen für Anzeige</b><br><br>");
787                     buffer = Ausgabe(buffer, L"<b>Zu viele Ebenen für
```

```
                Anzeige</b><br><br>", Ausgabebuf, 0);
788             }
789         }
790
791         wsprintf(Ausgabebuf, L"<table border=\"1\"><tr><th width=70> Item </th><th
            width=250>Wert</th> <th width=100>Typ</th><th width=550>Inhalt</th></tr>");
792         buffer = Ausgabe(buffer, L"<table border=\"1\"><tr><th width=70> Item
            </th><th width=250>Wert</th> <th width=100>Typ</th><th
            width=550>Inhalt</th></tr>", Ausgabebuf, 0);
793         table = 0;
794
795         // objektnummer = Objektnummer im DICT ( z.B. 01,02,03 ) --> neue markernummer
796         // markernummer = Objekt in der Offsettabelle
797         // markeroffset = Offsets der Offsettabelle ( mit allen offsize
798
799         for (int i = 1; i <= objektpaare; i++)
800         {
801             if (table == 1)
802             {
803                 wsprintf(Ausgabebuf, L"<table border=\"1\"><tr><th width=70> Item
                    </th><th width=250>Wert</th> <th width=100>Typ</th><th
                    width=550>Inhalt</th></tr>");
804                 buffer = Ausgabe(buffer, L"<table border=\"1\"><tr><th width=70>
                    Item </th><th width=250>Wert</th> <th width=100>Typ</th><th
                    width=550>Inhalt</th></tr>", Ausgabebuf, 0);
805                 table = 0;
806             }
807
808             position = inhalt   // Aktuelle Position in der Datei
809                 + i         // Nummer des auszulesenden Objekts
810                 + length    // Länge des Int Counters
811                 + j;        // Byte zähler für refsize 2
812
813             if (position + objektpaare * 2> file_size)
814             {
815                 wsprintf(Ausgabebuf, L"</table><br><b> Fehler: Auslesewert außerhalb
                    der Datei</b><br>");
816                 buffer = Ausgabe(buffer, L"</table><br><b> Fehler: Auslesewert
                    außerhalb der Datei</b><br>", Ausgabebuf, 0);
817                 return buffer;
818             }
819
820             if (refsize == 1)
821             {
822                 objektnummer = file_buf[position];
823                 objektnummer2 = file_buf[position + objektpaare];
824
825             }
826             else if (refsize == 2)
827             {
828                 objektnummer = (file_buf[position]) <<
                    8;                                  // 1 Byte
829                 objektnummer = objektnummer + file_buf[position +
                    1];                         // 2 Byte
830                 objektnummer2 = (file_buf[position + objektpaare * 2]) <<
                    8;                      // 1 Byte
831                 objektnummer2 = objektnummer2 + file_buf[position + 1 + objektpaare
                    * 2];   // 2 Byte
832                 j++;                // Zähler für das doppelbyte
833             }
834             else
835             {
836                 XWF_OutputMessage(L"BPListParser ERROR: Unsupportet Dict refsize", 0);
837                 return buffer;
838             }
839
840             // Prüfen Objektnummer
841             if (objektnummer > numobjekt && objektnummer2 > numobjekt)
842             {
843                 wsprintf(Ausgabebuf, L"</table><br><b> Fehler: nicht vorhandene
                    Objektnummer</b><br>");
844                 buffer = Ausgabe(buffer, L"</table><br><b> Fehler: nicht vorhandene
                    Objektnummer</b><br>", Ausgabebuf, 0);
```

```
845                     return buffer;
846                 }
847
848                 wsprintf(Ausgabebuf, L"<td>");
849                 buffer = Ausgabe(buffer, L"<td>", Ausgabebuf, 0);
850                 buffer = parseobjekt(file_buf, markeroffset, objektnummer, buffer);
851                 wsprintf(Ausgabebuf, L"</td><td>");
852                 buffer = Ausgabe(buffer, L"</td><td>", Ausgabebuf, 0);
853                 buffer = parseobjekt(file_buf, markeroffset, objektnummer2, buffer);
854                 wsprintf(Ausgabebuf, L"</td></tr>");
855                 buffer = Ausgabe(buffer, L"</td></tr>", Ausgabebuf, 0);
856
857             }
858             if (ebene<40)
859                 nutzungebene = nutzungebene - ebenenlaenge[ebene];
860
861             wsprintf(Ausgabebuf, L"</table><br>");
862             buffer = Ausgabe(buffer, L"</table><br>", Ausgabebuf, 0);
863             ebene--;
864             table = 1;
865
866             if (ebene > 1)
867             {
868                 buffer = Ausgabe(buffer, L"", tiefenanzeige, nutzungebene);
869                 wsprintf(Ausgabebuf, L"<br><br></b>");
870                 buffer = Ausgabe(buffer, L"<br><br></b>", Ausgabebuf, 0);
871             }
872             else if (ebene > 0)
873             {
874                 wsprintf(Ausgabebuf, L"<b>Ebene: Root</b><br><br>");
875                 buffer = Ausgabe(buffer, L"<b>Ebene: Root</b><br><br>", Ausgabebuf, 0);
876             }
877         }
878         else
879         {
880             wsprintf(Ausgabebuf, L"Dictionary</td><td> mit 0 Objekten</td></tr>");
881             buffer = Ausgabe(buffer, L"Dictionary</td><td> mit 0 Objekten</td></tr>",
                Ausgabebuf, 0);
882         }
883         return buffer;
884     }
885
886
887     wchar_t* parseArray(int objinfo, uint64_t markernummer, BYTE *file_buf, uint64_t
    markeroffset[4500], wchar_t *buffer)
888     {
889         uint64_t inhalt = markeroffset[markernummer], objekte = 0, position = 0;;
890         int length = 0, objektnummer = 0;
891         wchar_t Ausgabebuf[500];            // Temporärer Buffer für Ausgabe
892         int j = 0;
893         if (objinfo != 0xf)
894         {
895             objekte = objinfo;      // Anzahl der Objektpaare
896         }
897         else if (objinfo == 0xf)
898         {
899             inhalt++;
900             objinfo = (file_buf[inhalt] & 0x0F);                // objektInfo neu
                einlesen da ein Byte weiter
901             length = (int)pow(2, objinfo);
902             objekte = parseint(objinfo, inhalt, file_buf);
903         }
904
905         if (objekte != 0)
906         {
907             ebene++;
908             if (markernummer == 0)
909             {
910                 wsprintf(Ausgabebuf, L"Array mit %4d Objekten<br>", objekte);
911                 buffer = Ausgabe(buffer, L"Array mit      Objekten<br>", Ausgabebuf, 0);
912             }
913             else
914             {
```

```
915                 wsprintf(Ausgabebuf, L"Array</td> <td>mit %4d
                    Objekten</td></tr></table><br>", objekte);
916                 buffer = Ausgabe(buffer, L"Array</td> <td> mit
                    Objekten</td></tr></table><br>", Ausgabebuf, 0);
917
918                 // Prüfen auf Überschreitung der Buffer in Ebenenlaenge und Tiefenanzeige
919                 if (((nutzungebene + stringlength + 4) < 1500) && (ebene <40))
920                 {
921                     wsprintf(tiefenanzeige + nutzungebene, L"--> %s", Stringbuffer);
922                     nutzungebene = nutzungebene + stringlength + 4;
923                     buffer = Ausgabe(buffer, L"", tiefenanzeige, nutzungebene);
924                     ebenenlaenge[ebene] = stringlength + 4;
925
926                     wsprintf(Ausgabebuf, L"</b><br><br>");
927                     buffer = Ausgabe(buffer, L"</b><br><br>", Ausgabebuf, 0);
928                 }
929                 else
930                 {
931                     wsprintf(Ausgabebuf, L"<b>Zu viele Ebenen für Anzeige</b><br><br>");
932                     buffer = Ausgabe(buffer, L"<b>Zu viele Ebenen für
                        Anzeige</b><br><br>", Ausgabebuf, 0);
933                 }
934             }
935
936             // HTML Einbindung
937             wsprintf(Ausgabebuf, L"<table border=\"1\"><tr><th width=150> Item Nr. </th>
                <th width=100>Typ </th> <th width=720>Inhalt</th></tr>");
938             buffer = Ausgabe(buffer, L"<table border=\"1\"><tr><th width=150> Item Nr.
                </th> <th width=100>Typ </th> <th width=720>Inhalt</th></tr>", Ausgabebuf, 0);
939             table = 0;
940
941             for (int i = 1; i <= objekte; i++)
942             {
943                 if (table == 1)
944                 {
945                     wsprintf(Ausgabebuf, L"<table border=\"1\"><tr><th width=150> Item
                        Nr. </th> <th width=100>Typ </th> <th width=720>Inhalt</th></tr>");
946                     buffer = Ausgabe(buffer, L"<table border=\"1\"><tr><th width=150>
                        Item Nr. </th> <th width=100>Typ </th> <th
                        width=720>Inhalt</th></tr>", Ausgabebuf, 0);
947                     table = 0;
948                 }
949
950                 wsprintf(Ausgabebuf, L"<tr><td>Array Item Nr: %4d </td><td>", i);
951                 buffer = Ausgabe(buffer, L"<tr><td>Array Item Nr:      </td><td>",
                    Ausgabebuf, 0);
952
953                 wsprintf(Stringbuffer, L"Array Item Nr: %4d", i);
954                 stringlength = wcslen(L"Array Item Nr:     ");
955
956                 position = inhalt   // Aktuelle Position in der Datei
957                     + i             // Nummer des auszulesenden Objekts
958                     + length        // Länge des Int Counters
959                     + j;            // Byte zähler für refsize 2
960
961                 if (position + 1> file_size)
962                 {
963                     wsprintf(Ausgabebuf, L"</table><br><b> Fehler: Auslesewert außerhalb
                        der Datei</b><br>");
964                     buffer = Ausgabe(buffer, L"</table><br><b> Fehler: Auslesewert
                        außerhalb der Datei</b><br>", Ausgabebuf, 0);
965                     return buffer;
966                 }
967
968                 if (refsize == 1)
969                     objektnummer = file_buf[position];
970                 else if (refsize == 2)
971                 {
972                     objektnummer = (file_buf[position]) << 8;               // 1 Byte
973                     objektnummer = objektnummer + file_buf[position + 1];   // 2 Byte
974                     j++;
975                 }
976                 else
```

```
977                 {
978                     XWF_OutputMessage(L"BPListParser ERROR: Nicht unterstützte Array
                    refsize", 0);
979                     return buffer;
980                 }
981 
982                 // Prüfen Objektnummer in Datei vorhanden
983                 if (objektnummer > numobjekt)
984                 {
985                     wsprintf(Ausgabebuf, L"</table><br><b> Fehler: nicht vorhandene
                    Objektnummer</b><br>");
986                     buffer = Ausgabe(buffer, L"</table><br><b> Fehler: nicht vorhandene
                    Objektnummer</b><br>", Ausgabebuf, 0);
987                     return buffer;
988                 }
989 
990                 buffer = parseobjekt(file_buf, markeroffset, objektnummer, buffer);
991                 wsprintf(Ausgabebuf, L"</td></tr>");
992                 buffer = Ausgabe(buffer, L"</td></tr>", Ausgabebuf, 0);
993             }
994 
995             if (ebene<40)
996                 nutzungebene = nutzungebene - ebenenlaenge[ebene];
997 
998             wsprintf(Ausgabebuf, L"</table><br>");
999             buffer = Ausgabe(buffer, L"</table><br>", Ausgabebuf, 0);
1000            ebene--;
1001            table = 1;
1002 
1003            if (ebene > 1)
1004            {
1005                buffer = Ausgabe(buffer, L"", tiefenanzeige, nutzungebene);
1006                wsprintf(Ausgabebuf, L"<br><br></b>");
1007                buffer = Ausgabe(buffer, L"<br><br></b>", Ausgabebuf, 0);
1008            }
1009            else if (ebene > 0)
1010            {
1011                wsprintf(Ausgabebuf, L"<b>Ebene: Root<br><br></b>");
1012                buffer = Ausgabe(buffer, L"<b>Ebene: Root<br><br></b>", Ausgabebuf, 0);
1013            }
1014        }
1015        else {
1016            wsprintf(Ausgabebuf, L"Array</td><td> mit 0 Objekten</td></tr>");
1017            buffer = Ausgabe(buffer, L"Array</td><td> mit 0 Objekten</td></tr>",
                Ausgabebuf, 0);
1018        }
1019        return buffer;
1020    }
1021 
1022    wchar_t* parseSet(int objinfo, uint64_t markernummer, BYTE *file_buf, uint64_t
        markeroffset[4500], wchar_t *buffer)
1023    {
1024        uint64_t inhalt = markeroffset[markernummer], objekte = 0, position = 0;;
1025        int length = 0, objektnummer = 0;
1026        wchar_t Ausgabebuf[500];            // Temporärer Buffer für Ausgabe
1027        int j = 0;
1028        if (objinfo != 0xf)
1029        {
1030            objekte = objinfo;      // Anzahl der Objektpaare
1031        }
1032        else if (objinfo == 0xf)
1033        {
1034            inhalt++;
1035            objinfo = (file_buf[inhalt] & 0x0F);                // objektInfo neu
                einlesen da ein Byte weiter
1036            length = (int)pow(2, objinfo);
1037            objekte = parseint(objinfo, inhalt, file_buf);
1038        }
1039 
1040        if (objekte != 0)
1041        {
1042            ebene++;
1043            if (markernummer == 0)
```

```
1044            {
1045                wsprintf(Ausgabebuf, L"Set mit %4d Objekten<br>", objekte);
1046                buffer = Ausgabe(buffer, L"Set mit      Objekten<br>", Ausgabebuf, 0);
1047            }
1048            else
1049            {
1050                wsprintf(Ausgabebuf, L"Set</td> <td>mit %4d
                    Objekten</td></tr></table><br>", objekte);
1051                buffer = Ausgabe(buffer, L"Set</td> <td> mit
                    Objekten</td></tr></table><br>", Ausgabebuf, 0);
1052
1053                // Prüfen auf Überschreitung der Buffer in Ebenenlaenge und Tiefenanzeige
1054                if (((nutzungebene + stringlength + 4) < 1500) && (ebene <40))
1055                {
1056                    wsprintf(tiefenanzeige + nutzungebene, L"--> %s", Stringbuffer);
1057                    nutzungebene = nutzungebene + stringlength + 4;
1058                    buffer = Ausgabe(buffer, L"", tiefenanzeige, nutzungebene);
1059                    ebenenlaenge[ebene] = stringlength + 4;
1060
1061                    wsprintf(Ausgabebuf, L"</b><br><br>");
1062                    buffer = Ausgabe(buffer, L"</b><br><br>", Ausgabebuf, 0);
1063                }
1064                else
1065                {
1066                    wsprintf(Ausgabebuf, L"<b>Zu viele Ebenen für Anzeige</b><br><br>");
1067                    buffer = Ausgabe(buffer, L"<b>Zu viele Ebenen für
                        Anzeige</b><br><br>", Ausgabebuf, 0);
1068                }
1069            }
1070
1071            // HTML Einbindung
1072            wsprintf(Ausgabebuf, L"<table border=\"1\"><tr><th width=150> Set Nr. </th>
                <th width=100>Typ </th> <th width=720>Inhalt</th></tr>");
1073            buffer = Ausgabe(buffer, L"<table border=\"1\"><tr><th width=150> Set Nr.
                </th> <th width=100>Typ </th> <th width=720>Inhalt</th></tr>", Ausgabebuf, 0);
1074            table = 0;
1075
1076            for (int i = 1; i <= objekte; i++)
1077            {
1078                if (table == 1)
1079                {
1080                    wsprintf(Ausgabebuf, L"<table border=\"1\"><tr><th width=150> Set
                        Nr. </th> <th width=100>Typ </th> <th width=720>Inhalt</th></tr>");
1081                    buffer = Ausgabe(buffer, L"<table border=\"1\"><tr><th width=150>
                        Set Nr. </th> <th width=100>Typ </th> <th
                        width=720>Inhalt</th></tr>", Ausgabebuf, 0);
1082                    table = 0;
1083                }
1084
1085                wsprintf(Ausgabebuf, L"<tr><td>Set Item Nr: %4d </td><td>", i);
1086                buffer = Ausgabe(buffer, L"<tr><td>Set Item Nr:      </td><td>",
                    Ausgabebuf, 0);
1087
1088                wsprintf(Stringbuffer, L"Set Item Nr: %4d", i);
1089                stringlength = wcslen(L"Set Item Nr:     ");
1090
1091                position = inhalt   // Aktuelle Position in der Datei
1092                    + i             // Nummer des auszulesenden Objekts
1093                    + length        // Länge des Int Counters
1094                    + j;            // Byte zähler für refsize 2
1095
1096                if (position + 1> file_size)
1097                {
1098                    wsprintf(Ausgabebuf, L"</table><br><b> Fehler: Auslesewert außerhalb
                        der Datei</b><br>");
1099                    buffer = Ausgabe(buffer, L"</table><br><b> Fehler: Auslesewert
                        außerhalb der Datei</b><br>", Ausgabebuf, 0);
1100                    return buffer;
1101                }
1102
1103                if (refsize == 1)
1104                    objektnummer = file_buf[position];
1105                else if (refsize == 2)
```

```
1106                {
1107                    objektnummer = (file_buf[position]) << 8;                // 1 Byte
1108                    objektnummer = objektnummer + file_buf[position + 1];    // 2 Byte
1109                    j++;
1110                }
1111                else
1112                {
1113                    XWF_OutputMessage(L"BPListParser ERROR: Nicht unterstützte Set
                    refsize", 0);
1114                    return buffer;
1115                }
1116
1117                // Prüfen Objektnummer in Datei vorhanden
1118                if (objektnummer > numobjekt)
1119                {
1120                    wsprintf(Ausgabebuf, L"</table><br><b> Fehler: nicht vorhandene
                    Objektnummer</b><br>");
1121                    buffer = Ausgabe(buffer, L"</table><br><b> Fehler: nicht vorhandene
                    Objektnummer</b><br>", Ausgabebuf, 0);
1122                    return buffer;
1123                }
1124
1125                buffer = parseobjekt(file_buf, markeroffset, objektnummer, buffer);
1126                wsprintf(Ausgabebuf, L"</td></tr>");
1127                buffer = Ausgabe(buffer, L"</td></tr>", Ausgabebuf, 0);
1128            }
1129
1130            if (ebene<40)
1131                nutzungebene = nutzungebene - ebenenlaenge[ebene];
1132
1133            wsprintf(Ausgabebuf, L"</table><br>");
1134            buffer = Ausgabe(buffer, L"</table><br>", Ausgabebuf, 0);
1135            ebene--;
1136            table = 1;
1137
1138            if (ebene > 1)
1139            {
1140                buffer = Ausgabe(buffer, L"", tiefenanzeige, nutzungebene);
1141                wsprintf(Ausgabebuf, L"<br><br></b>");
1142                buffer = Ausgabe(buffer, L"<br><br></b>", Ausgabebuf, 0);
1143            }
1144            else if (ebene > 0)
1145            {
1146                wsprintf(Ausgabebuf, L"<b>Ebene: Root<br><br></b>");
1147                buffer = Ausgabe(buffer, L"<b>Ebene: Root<br><br></b>", Ausgabebuf, 0);
1148            }
1149        }
1150        else {
1151            wsprintf(Ausgabebuf, L"Set</td><td> mit 0 Objekten</td></tr>");
1152            buffer = Ausgabe(buffer, L"Set</td><td> mit 0 Objekten</td></tr>",
                Ausgabebuf, 0);
1153        }
1154        return buffer;
1155    }
1156
1157    wchar_t *Ausgabe(wchar_t *buffer, const wchar_t* Zeile, wchar_t* Ausgabebuf, int
    zeichencount) {
1158
1159        // Anzahl zu schreibenden Zeichen zählen
1160        size_t Zeilenlänge = wcslen(Zeile) + zeichencount;
1161
1162        // Erweiterung des Speichers wenn Speicherbedarf zu Groß
1163        while (Speichernutzung + Zeilenlänge >= SpeicherZugeordnet)
1164        {
1165            // Exponentielle Vergößerung
1166            SpeicherZugeordnet *= 2;
1167
1168            // Speicherzuordnung erfolgreich
1169            buffer = (wchar_t *)realloc(buffer, SpeicherZugeordnet * sizeof(wchar_t));
1170            if ((buffer == NULL))
1171            {
1172                XWF_OutputMessage(L"BPListParser ERROR: Fehler beim Zuordnen des
                    größeren Speichers", 0);
```

```
1173                return NULL;
1174            }
1175        }
1176        // Neuen Inhalt an Speicher anhängen
1177        wmemcpy(buffer + Speichernutzung        // Zielbuffer + aktueller Inhalt
1178            , Ausgabebuf                        // Quellbuffer
1179            , Zeilenlänge);                     // Anzahl der zu kopierenden Zeichen
1180
1181                                                // Aktuelle Speichernutzung festhalten
1182        Speichernutzung += Zeilenlänge;
1183
1184        return buffer;
1185    }
```

Selbstständigkeitserklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig und nur unter Verwendung der angegebenen Literatur und Hilfsmittel angefertigt habe.

Stellen, die wörtlich oder sinngemäß aus Quellen entnommen wurden, sind als solche kenntlich gemacht.

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

Mittweida, den 28.09.2018

Michael Kittan