HOCHSCHULE MITTWEIDA
University of Applied Sciences

Fraunhofer SIT

Bundeskriminalamt

Betriebssysteme

macOS - Teil 3

Autor: Prof. Ronny Bodach

macOS Agenda

1. Einführung in macOS
2. macOS Bedienung
3. macOS Lab & Image Einbindung
4. Bootcamp Besonderheiten (& Parallels)
5. Mac FHS und Speicherstrukturen
6. Datenformate SQLite und Plist
7. Zuletzt genutzte Elemente & Nutzeraktivitäten
8. Spotlight und erweiterte Metadaten
9. Gelöschte Dateien
10. Schlüsselbund
11. Logdateien
12. Mac Disk Images
13. Time Machine und lokale Backups
14. Kommunikations-Apps
15. Browser Artefakte
16. Cloud
17. iOS Backups

macOS Agenda

5. Mac FHS und Speicherstrukturen
6. Datenformate SQLite und Plist

BETRIEBSSYSTEM macOS

Mac FHS und Speicherstrukturen

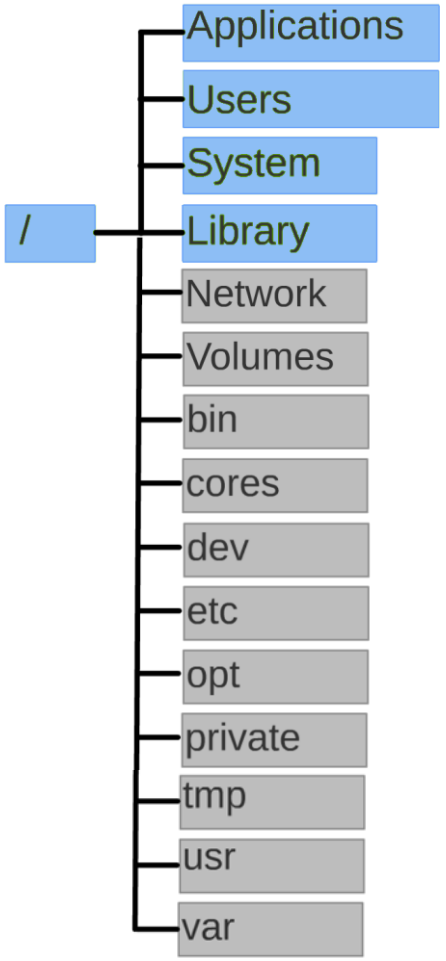
Betriebssystem macOS

Wichtige Verzeichnisstrukturen:

Die Verzeichnisstrukturen eines Mac OS X lässt den UNIX-Ursprung erkennen. Unterschiede gibt es jedoch bei den von Apple entwickelten Bestandteilen.

Apple hält sich bei seiner Verzeichnisstruktur nicht streng an den Filesystem Hierarchy Standard (FHS). Der Standard ist aber noch erkennbar.

Zudem sind viele Systemordner bei OSX versteckt (graue Verzeichnisse).



Betriebssystem macOS

Wichtige Verzeichnisstrukturen:

Das MAC OS X Betriebssystem teilt Daten ebenfalls in die Kategorien Systemdaten, Benutzerdaten und Softwaredaten auf.

Anwendungen befinden sich im Verzeichnis „***Application***“ im Root Verzeichnis.

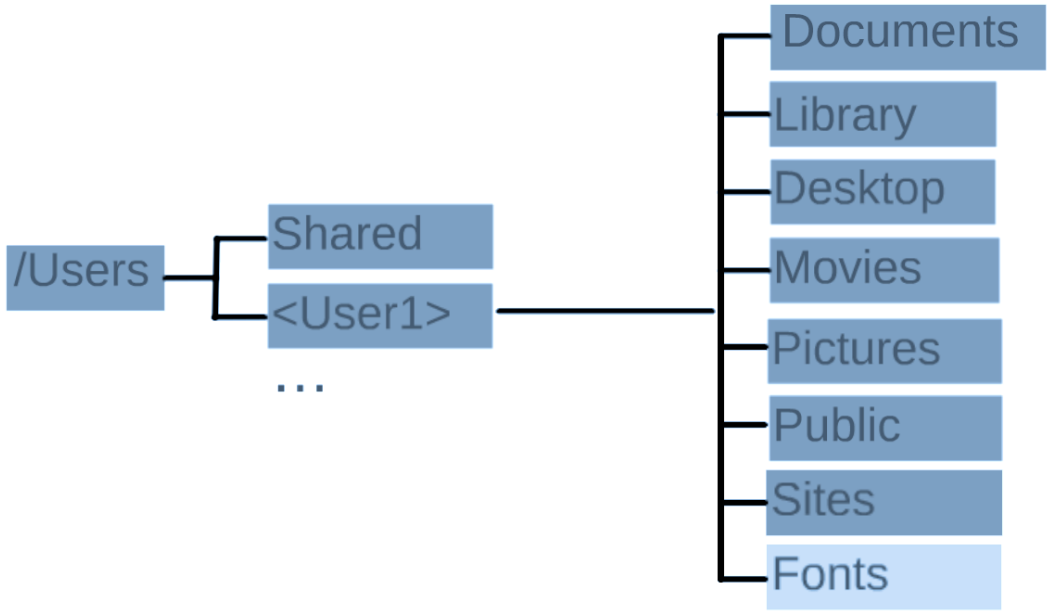
Systemdaten finden sich zum Einen im Verzeichnis ***„…/System/Library/***“ und zum Anderen im ***„…/Library***“ Verzeichnis im Root Verzeichnis selbst.

Die für jeden Benutzer gültigen Systemeinstellungen werden im Benutzerverzeichnis ebenfalls im Unterverzeichnis ***„…/Users/[Name]/ Library“*** abgelegt.

Betriebssystem macOS

Wichtige Verzeichnisstrukturen:

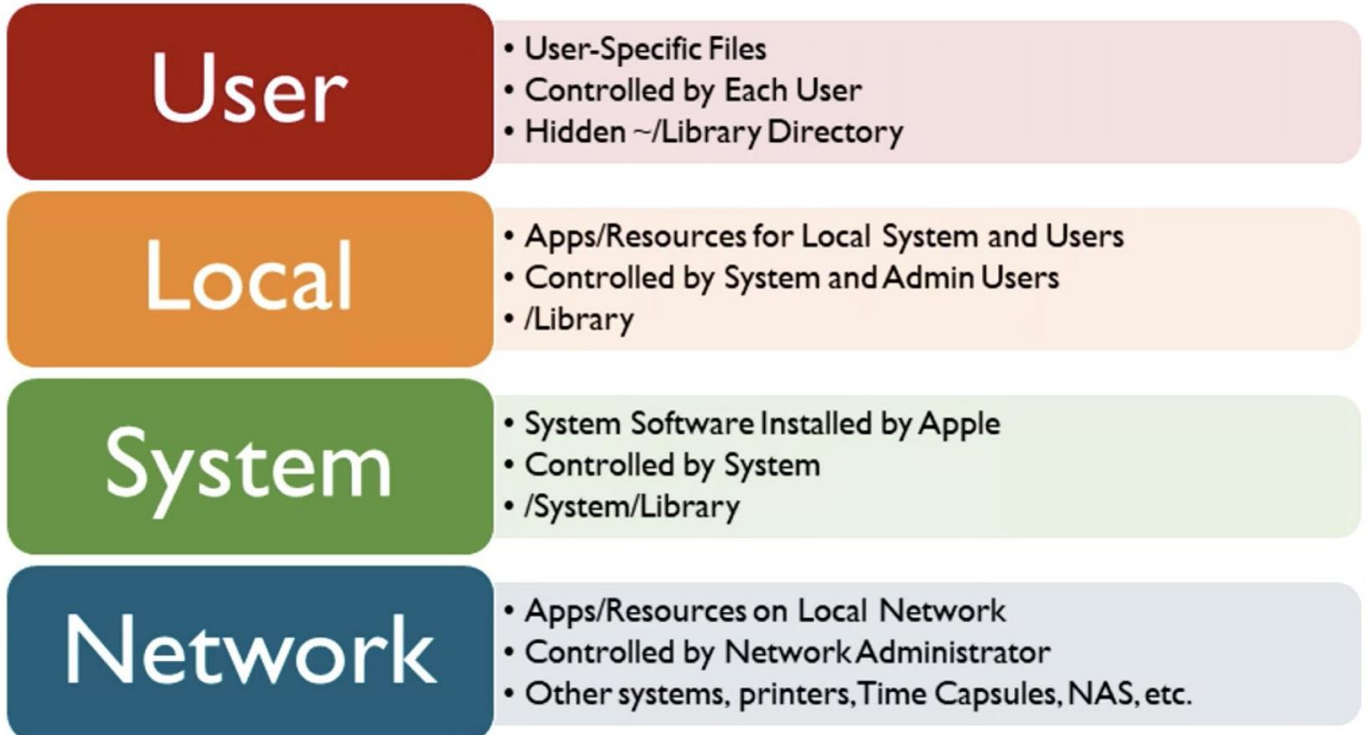
Benutzerdaten befinden sich im jeweiligen Unterverzeichnis des Benutzers im Verzeichnis Users im Root Verzeichnis.



Betriebssystem macOS

Wichtige Verzeichnisstrukturen:

Folgende Übersicht zeigt zudem die von System und Netzwerk benutzten Bestandteile auf:

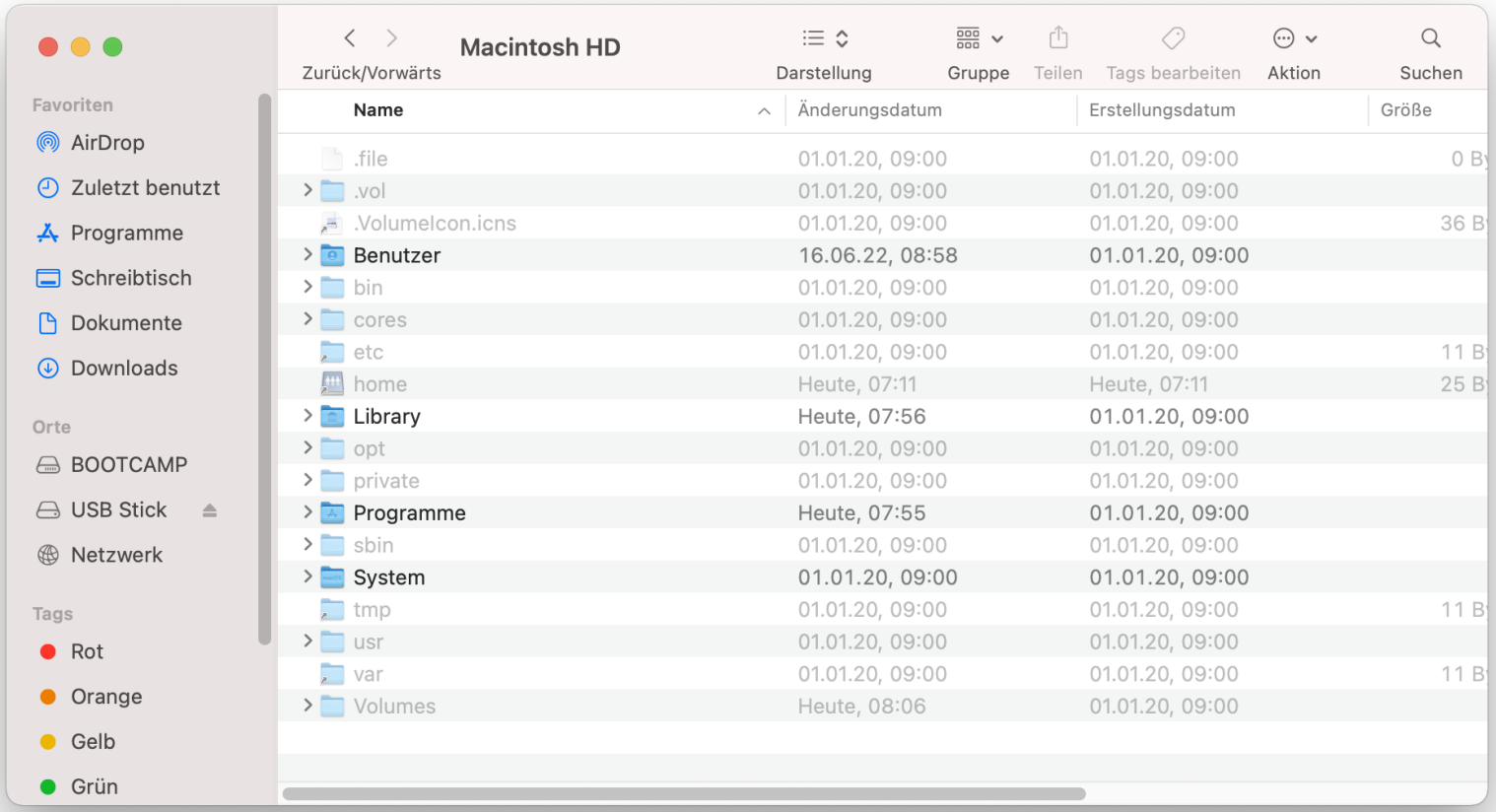


Quelle: Sysadmin and Networking Institut

Betriebssystem macOS

Die Vollansicht im Finder muss explizit aktiviert werden:

(**Shift** + Befehlstaste/**Command** (⌘) + **.**)



Betriebssystem macOS

Zusätzliche Anwendungsdaten:

Im Library Benutzerverzeichnis existiert ein Unterverzeichnis mit dem Namen ***„Application Support“***.

Hier befinden sich Benutzerdaten für die installierten Anwendungen. Dieses Verzeichnis kann mit dem auf Windows bekannten Anwendungsdatenverzeichnis gleichgesetzt werden.

BETRIEBSSYSTEM macOS

Datenformate SQLite und Plist

Betriebssystem macOS

Formate

OS X speichert jede Menge Informationen die für den IT-Forensiker von Interesse sein können. Diese Informationen sind in einer ganzen Reihe verschiedener Formate abgelegt.

Einige sind leicht zu interpretieren, wie Plain Text, XML oder Datenbanken. Andere haben aber proprietäre Binärformate. Einige Dateiformate sind gut dokumentiert, andere leider gar nicht und mussten Reverse Engineered werden.

Für die meisten Binärformate bringt OSX aber eigene Viewer mit. Das ist ein Grund, dass sich Apple Rechner am besten auf einem Apple auswerten lassen.

Betriebssystem macOS

Formate

- **Basic Security Modules (BSM):** Binärdateien, welche die Kernel Logs speichern. Es gibt mehrere BSM-Dateien, die jeweils mehrere Token fester oder variabler Länge enthalten. Ähnlich zu den EVT, EVTX Dateien bei Windows.
- **Binary Apple System Log (ASL):** Standardformat für die Daemon Logs, Binärdaten in doppelt verlinkten Listen.
- **Keychain**: Ein binäres Datenbank-Format in dem Passwörter und Zertifikate von Applikationen, Webseiten, Netzwerken und ähnlichem abgelegt werden.
- **Plist:** OS X und die Applikationen legen ihre Konfiguration in Plist Dateien ab. Es gibt zwei Unterformate: Plists die XML enthalten Binäre Plists (BPlist)
- **SQLite:** Userapplikationen wie Chrome, Skype, Firefox, etc. aber auch einige Caches von OSX selbst werden im SQLite Datenbankformat abgelegt.

Betriebssystem macOS

macOS Library

Systemeinstellungen werden in **.plist* Dateien den Property Listen gespeichert.

Diese Dateien sind vom Aufbau her XML Dateien mit Apple spezifischen Schlüsselpaaren. Es besteht jedoch die Möglichkeit diese Property Listen auch als binärcodierte Dateien abzulegen.

Für die Auswertung ist es dann notwendig diese Binary Property List in eine XML basierte Datei umzuwandeln.

BETRIEBSSYSTEM macOS

Plattformabhängige Speichermethode

Plist

Plattformabhängige Speichermethode

Plist Konvertierung

OSX bietet ein Tool um zwischen beiden Formaten zu konvertieren:

- **plutil -convert xml1 some_file.plist -** konvertiert eine bplist in eine XML-plist
- **plutil -convert binary1 some_other_file.plist -** konvertiert eine XML-plist in eine bplist

Plattformabhängige Speichermethode

Aufbau des Plist Formates

Die BPlist Datei besteht zunächst aus vier Bestandteilen:

- 8 Byte Header
- var. Byte Objekttabelle
- var. Byte Offset Table
- 32 Byte Trailer

Eine Binary Property List (bplist)-Datei besteht aus 4 Teilen. Sie werden im Folgenden nicht in der Reihenfolge beschrieben, in der sie auftreten, sondern in der Reihenfolge, die beim Lesen der Datei am sinnvollsten ist.

Plattformabhängige Speichermethode

Aufbau des Plist Formates - Header

Die bplist-Datei trägt im Header eine Signatur:

- **`0x62706C6973743030 = bplist`**,
- gefolgt von der **`Version`** des bplist Files (**`00,15,16,etc.`**).

Die Untersuchung von BPlist Dateien setzt in der Regel voraus, dass es sich um eine BPlist der Version 00 handelt, da andere Versionen von Apple bisher nicht dokumentiert sind.

Plattformabhängige Speichermethode

Aufbau des Plist Formates - Trailer

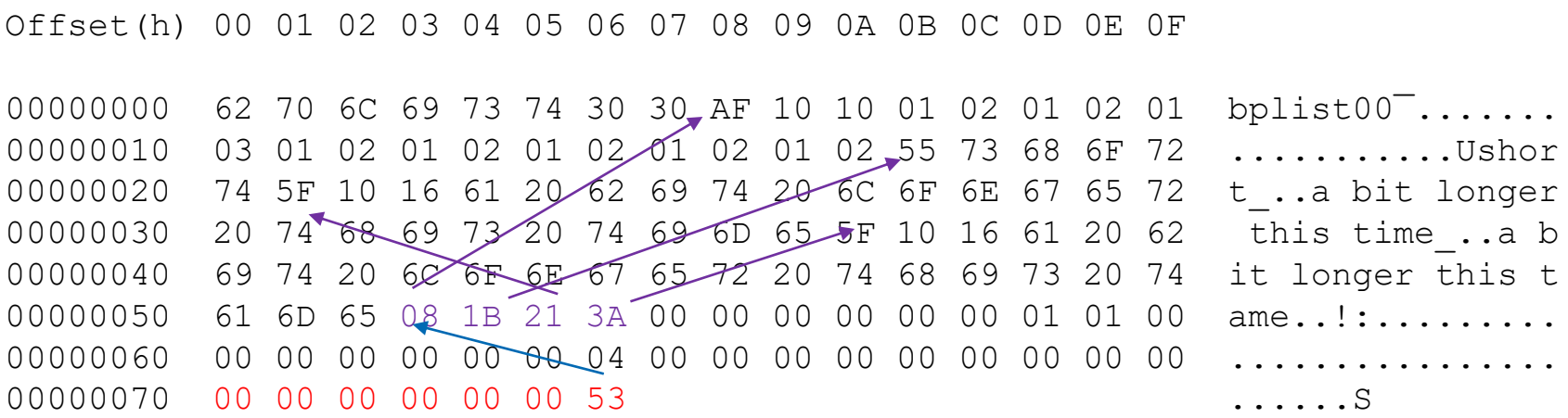
Der Trailer besteht im Wesentlichen nur aus drei, für uns relevanten Teilen. Zum einen die Offsetgröße in Byte, welche uns verrät, wie groß ein Offset in der Offset Tabelle ist. Weiter ist hier auch die Anzahl aller gespeicherten Objekte vermerkt. Abschließend ist der Offset der Offset Tabelle ebenfalls noch enthalten.

Offset	Länge	Datenstruktur	Beschreibung
0	5	unbenutzt	unbenutzt
6	1	8-bit unsigned integer	Offsetgröße in Byte
7	1	8-bit unsigned integer	Referenzgröße in Dictionaries
8	8	64-bit unsigned big endian integer	Anzahl der gespeicherten Objekte
16	8	64-bit unsigned big endian integer	Top Level Object Index
24	8	64-bit unsigned big endian integer	Offset der Offset Tabelle
32			

Plattformabhängige Speichermethode

Aufbau des Plist Formates – Offset-Tabelle

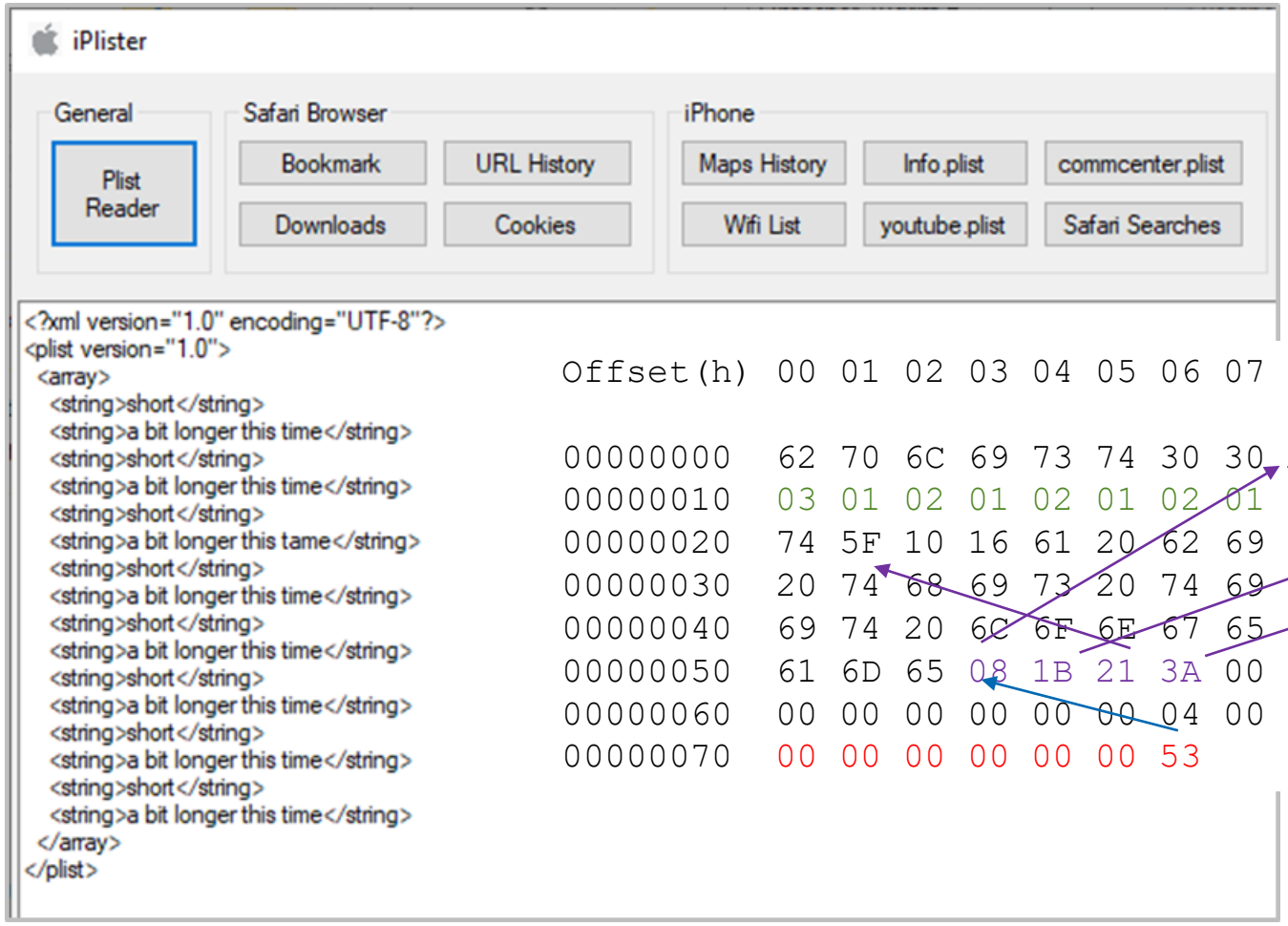
Die Offset-Tabelle ist der dritte Abschnitt in der Datei. Der Startoffset in der Datei wird im Trailer angegeben („Offset der Offset-Tabelle“). Die Länge kann durch Multiplizieren der Trailer-Felder „Anzahl der gespeicherten Objekte“ mit „Offsetgröße in Byte“ ermittelt werden. Die Offsettabelle enthält ein Array von Offsets mit einem Index von Null (dh der erste Eintrag wird als Eintrag 0 bezeichnet, der zweite als Eintrag 1 usw.), die auf Objekte in der Objekttabelle verweisen.



Beispiel Offset-Tabelle und referenzierte Objekte dazu

Plattformabhängige Speichermethode

Aufbau des Plist Formates – Offset-Tabelle



Plattformabhängige Speichermethode

Aufbau des Plist Formates – die Objekttabelle

Die Objekttabelle ist der zweite Abschnitt in der Datei, der direkt nach dem Header bei Offset 8 beginnt und bis zum Beginn der Offset-Tabelle fortgesetzt wird. Es enthält die binären Darstellungen jedes „Objekts“ in der Plist.

Jedes Objekt besteht aus einem Typ-Deskriptor-Byte, optional gefolgt von einer Länge (abhängig vom Datentyp kann die Länge im Datentyp-Byte impliziert sein), gefolgt von den Daten selbst.

Um die Länge der Objekte festzustellen, muss zunächst der Typ erkannt werden. Dieser wird in den meisten Fällen durch die ersten 4 Bit bestimmt. Die Länge ergibt sich dann, durch die nächsten 4 Bit. Ist die Länge kleiner als 15 Byte so reichen diese 4 Bit aus um die Länge anzugeben. Ist das Objekt größer, so werden die 4 Bit alle mit 1 belegt. Das bedeutet, dass die nächsten Bytes für die Größe ausgewertet werden müssen.

Plattformabhängige Speichermethode

Aufbau des Plist Formates – die Objekttabelle

Typ	Binär	Größe	Inhalt
null	0000 0000		
bool	0000 1000		FALSE
bool	0000 1001		TRUE
fill	0000 1111		Füll Byte
int	0001 nnnn	…	2^nnnn Big-Endian Bytes
real	0010 nnnn	…	2^nnnn Big-Endian Bytes
date	0011 0011	…	8 Byte Float (Big-Endian)
data	0100 nnnn	[int] …	nnnn Bytes bis '1111', danach int Typ und int Count gefolgt von Inhalt
string	0101 nnnn	[int] …	ASCII String, nnnn Chars bis '1111', danach int Typ und int Count gefolgt von Inhalt
string	0110 nnnn	[int] …	Unicode String, nnnn Chars bis '1111', danach int Typ und int Count gefolgt von Inhalt

Plattformabhängige Speichermethode

Aufbau des Plist Formates – die Objekttabelle

Typ	Binär	Größe	Inhalt
	0111 xxxx		reserviert
uid	1000 nnnn	…	nnnn+1 Bytes
	1001 xxxx		reserviert
array	1010 nnnn	[int] objref*	nnnn Einträge bis ’1111’, danach int Typ und int Count
	1011 xxxx		reserviert
set	1100 nnnn	[int] objref*	nnnn Einträge bis ’1111’, danach int Typ und int Count
dict	1101 nnnn	[int] keyref	nnnn Einträge bis ’1111’, danach int Typ und int Count
	1110 xxxx		reserviert
	1111 xxxx		reserviert
	0111 xxxx		reserviert

Plattformabhängige Speichermethode

Aufbau des Plist Formates – Beispiel

```
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 AF 10 10 01 02 01 02 01  bplist00¯.......
00000010  03 01 02 01 02 01 02 01 02 01 02 55 73 68 6F 72  ...........Ushor
00000020  74 5F 10 16 61 20 62 69 74 20 6C 6F 6E 67 65 72  t_..a bit longer
00000030  20 74 68 69 73 20 74 69 6D 65 5F 10 16 61 20 62   this time_..a b
00000040  69 74 20 6C 6F 6E 67 65 72 20 74 68 69 73 20 74  it longer this t
00000050  61 6D 65 08 1B 21 3A 00 00 00 00 00 00 01 01 00  ame..!:.........
00000060  00 00 00 00 00 00 04 00 00 00 00 00 00 00 00 00  ................
00000070  00 00 00 00 00 00 53                             ......S
```

Plattformabhängige Speichermethode

Aufbau des Plist Formates – Beispiel

```
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 AF 10 10 01 02 01 02 01  bplist00¯.......
00000010  03 01 02 01 02 01 02 01 02 01 02 55 73 68 6F 72  ...........Ushor
00000020  74 5F 10 16 61 20 62 69 74 20 6C 6F 6E 67 65 72  t_..a bit longer
00000030  20 74 68 69 73 20 74 69 6D 65 5F 10 16 61 20 62   this time_..a b
00000040  69 74 20 6C 6F 6E 67 65 72 20 74 68 69 73 20 74  it longer this t
00000050  61 6D 65 08 1B 21 3A 00 00 00 00 00 00 01 01 00  ame..!:.........
00000060  00 00 00 00 00 00 04 00 00 00 00 00 00 00 00 00  ................
00000070  00 00 00 00 00 00 53                             ......S
```

AF = 1010 1111

Plattformabhängige Speichermethode

Aufbau des Plist Formates – Beispiel

```
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 AF 10 10 01 02 01 02 01  bplist00¯.......
00000010  03 01 02 01 02 01 02 01 02 01 02 55 73 68 6F 72  ...........Ushor
00000020  74 5F 10 16 61 20 62 69 74 20 6C 6F 6E 67 65 72  t_..a bit longer
00000030  20 74 68 69 73 20 74 69 6D 65 5F 10 16 61 20 62   this time_..a b
00000040  69 74 20 6C 6F 6E 67 65 72 20 74 68 69 73 20 74  it longer this t
00000050  61 6D 65 08 1B 21 3A 00 00 00 00 00 00 01 01 00  ame..!:.........
00000060  00 00 00 00 00 00 04 00 00 00 00 00 00 00 00 00  ................
00000070  00 00 00 00 00 00 53                             ......S
```

AF = 1010 1111

array	1010 nnnn	[int] objref*	nnnn Einträge bis ’1111’, danach int Typ und int Count

Plattformabhängige Speichermethode

Aufbau des Plist Formates – Beispiel

```
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 AF 10 10 01 02 01 02 01  bplist00¯.......
00000010   03 01 02 01 02 01 02 01 02 01 02 55 73 68 6F 72  ...........Ushor
00000020   74 5F 10 16 61 20 62 69 74 20 6C 6F 6E 67 65 72  t_..a bit longer
00000030   20 74 68 69 73 20 74 69 6D 65 5F 10 16 61 20 62   this time_..a b
00000040   69 74 20 6C 6F 6E 67 65 72 20 74 68 69 73 20 74  it longer this t
00000050   61 6D 65 08 1B 21 3A 00 00 00 00 00 00 01 01 00  ame..!:.........
00000060   00 00 00 00 00 00 04 00 00 00 00 00 00 00 00 00  ................
00000070   00 00 00 00 00 00 53                             ......S
```

AF = 1010 1111

array	1010 nnnn	[int] objref*	nnnn Einträge, bis ‘1111’, danach int Typ und int Count

Plattformabhängige Speichermethode

Aufbau des Plist Formates – Beispiel

```
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 AF 10 10 01 02 01 02 01  bplist00¯.......
00000010  03 01 02 01 02 01 02 01 02 01 02 55 73 68 6F 72  ...........Ushor
00000020  74 5F 10 16 61 20 62 69 74 20 6C 6F 6E 67 65 72  t_..a bit longer
00000030  20 74 68 69 73 20 74 69 6D 65 5F 10 16 61 20 62   this time_..a b
00000040  69 74 20 6C 6F 6E 67 65 72 20 74 68 69 73 20 74  it longer this t
00000050  61 6D 65 08 1B 21 3A 00 00 00 00 00 00 01 01 00  ame..!:.........
00000060  00 00 00 00 00 00 04 00 00 00 00 00 00 00 00 00  ................
00000070  00 00 00 00 00 00 53                             ......S
```

AF = 1010 1111

nnnn = Anzahl im nachfolgenden Byte

array	1010 nnnn	[int] objref*	nnnn Einträge bis '1111', danach int Typ und int Count

Plattformabhängige Speichermethode

Aufbau des Plist Formates – Beispiel

```
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 AF 10 10 01 02 01 02 01  bplist00¯.......
00000010   03 01 02 01 02 01 02 01 02 01 02 55 73 68 6F 72  ...........Ushor
00000020   74 5F 10 16 61 20 62 69 74 20 6C 6F 6E 67 65 72  t_..a bit longer
00000030   20 74 68 69 73 20 74 69 6D 65 5F 10 16 61 20 62   this time_..a b
00000040   69 74 20 6C 6F 6E 67 65 72 20 74 68 69 73 20 74  it longer this t
00000050   61 6D 65 08 1B 21 3A 00 00 00 00 00 00 01 01 00  ame..!:.........
00000060   00 00 00 00 00 00 04 00 00 00 00 00 00 00 00 00  ................
00000070   00 00 00 00 00 00 53                             ......S
```

AF = 1010 1111

nnnn = Anzahl im nachfolgenden Byte

array	1010 nnnn	[int] objref*	nnnn Einträge bis ’1111’, danach int Typ und int Count

Plattformabhängige Speichermethode

Aufbau des Plist Formates – Beispiel

```
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 AF 10 10 01 02 01 02 01  bplist00¯.......
00000010   03 01 02 01 02 01 02 01 02 01 02 55 73 68 6F 72  ...........Ushor
00000020   74 5F 10 16 61 20 62 69 74 20 6C 6F 6E 67 65 72  t_..a bit longer
00000030   20 74 68 69 73 20 74 69 6D 65 5F 10 16 61 20 62   this time_..a b
00000040   69 74 20 6C 6F 6E 67 65 72 20 74 68 69 73 20 74  it longer this t
00000050   61 6D 65 08 1B 21 3A 00 00 00 00 00 00 01 01 00  ame..!:.........
00000060   00 00 00 00 00 00 04 00 00 00 00 00 00 00 00 00  ................
00000070   00 00 00 00 00 00 53                             ......S
```

10 Type = 0001 0000

2^0 = 1 Byte

int	0001 nnnn	...	2^nnnn Big-Endian Bytes

Plattformabhängige Speichermethode

Aufbau des Plist Formates – Beispiel

```
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 AF 10 10 01 02 01 02 01  bplist00¯.......
00000010   03 01 02 01 02 01 02 01 02 01 02 55 73 68 6F 72  ...........Ushor
00000020   74 5F 10 16 61 20 62 69 74 20 6C 6F 6E 67 65 72  t_..a bit longer
00000030   20 74 68 69 73 20 74 69 6D 65 5F 10 16 61 20 62   this time_..a b
00000040   69 74 20 6C 6F 6E 67 65 72 20 74 68 69 73 20 74  it longer this t
00000050   61 6D 65 08 1B 21 3A 00 00 00 00 00 00 01 01 00  ame..!:.........
00000060   00 00 00 00 00 00 04 00 00 00 00 00 00 00 00 00  ................
00000070   00 00 00 00 00 00 53                             ......S
```

10 Type = 0001 0000

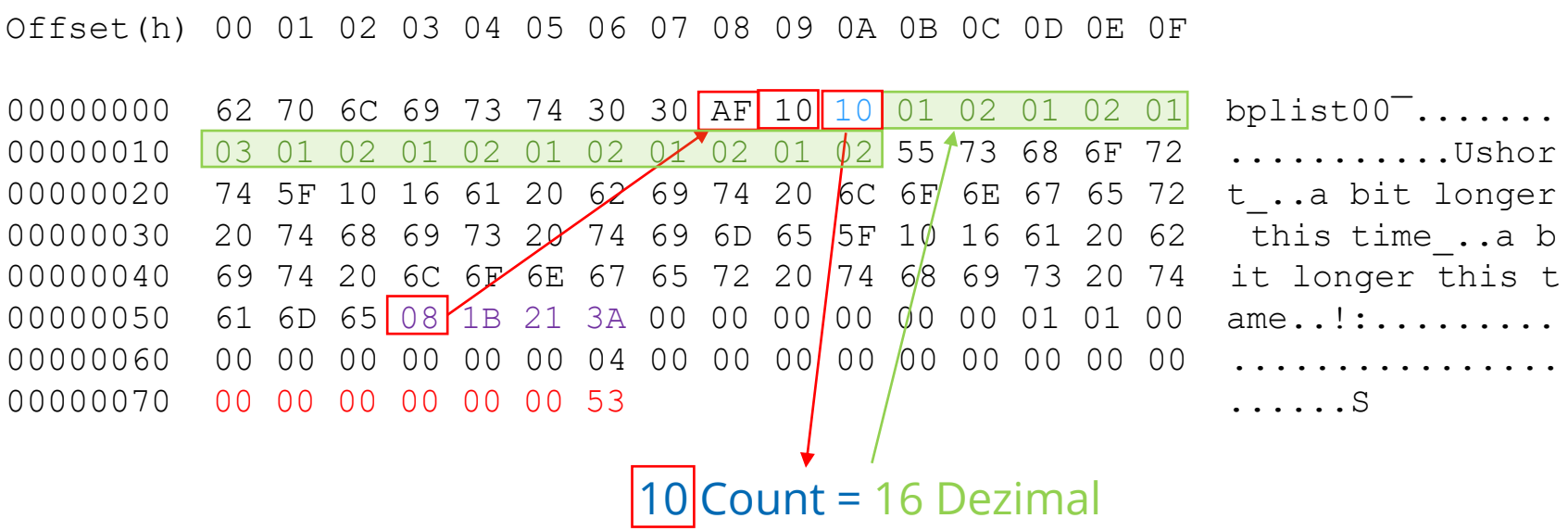
10 Count = 16 Dezimal

$2^{\wedge 0}$ = 1 Byte

int	0001 nnnn	...	$2^{\wedge nnnn}$ Big-Endian Bytes

Plattformabhängige Speichermethode

Aufbau des Plist Formates – Beispiel



Plattformabhängige Speichermethode

Aufbau des Plist Formates – Beispiel

```
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 AF 10 10 01 02 01 02 01  bplist00¯.......
00000010   03 01 02 01 02 01 02 01 02 01 02 55 73 68 6F 72  ...........Ushor
00000020   74 5F 10 16 61 20 62 69 74 20 6C 6F 6E 67 65 72  t_..a bit longer
00000030   20 74 68 69 73 20 74 69 6D 65 5F 10 16 61 20 62   this time_..a b
00000040   69 74 20 6C 6F 6E 67 65 72 20 74 68 69 73 20 74  it longer this t
00000050   61 6D 65 08 1B 21 3A 00 00 00 00 00 00 01 01 00  ame..!:.........
00000060   00 00 00 00 00 00 04 00 00 00 00 00 00 00 00 00  ................
00000070   00 00 00 00 00 00 53                             ......S
```

10 Count = 16 Dezimal

16 Array Member – Nummer bezieht sich auf Position innerhalb der Plist Offsets (beginnend bei 0):

01 02 01 02 01 03 01 02 01 02 01 02 01 02 01 02

Plattformabhängige Speichermethode

Aufbau des Plist Formates – Beispiel

```
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 AF 10 10 01 02 01 02 01   bplist00¯.......
00000010   03 01 02 01 02 01 02 01 02 01 02 55 73 68 6F 72   ...........Ushor
00000020   74 5F 10 16 61 20 62 69 74 20 6C 6F 6E 67 65 72   t_..a bit longer
00000030   20 74 68 69 73 20 74 69 6D 65 5F 10 16 61 20 62    this time_..a b
00000040   69 74 20 6C 6F 6E 67 65 72 20 74 68 69 73 20 74   it longer this t
00000050   61 6D 65 08 1B 21 3A 00 00 00 00 00 00 01 01 00   ame..!:.........
00000060   00 00 00 00 00 00 04 00 00 00 00 00 00 00 00 00   ................
00000070   00 00 00 00 00 00 53                              ......S
```

16 Array Member – Nummer bezieht sich auf Position innerhalb der Plist Offsets (beginnend bei 0):

01 02 01 02 01 03 01 02 01 02 01 02 01 02 01 02

Plattformabhängige Speichermethode

Aufbau des Plist Formates – Beispiel

```
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 AF 10 10 01 02 01 02 01   bplist00¯.......
00000010   03 01 02 01 02 01 02 01 02 01 02 55 73 68 6F 72   ...........Ushor
00000020   74 5F 10 16 61 20 62 69 74 20 6C 6F 6E 67 65 72   t_..a bit longer
00000030   20 74 68 69 73 20 74 69 6D 65 5F 10 16 61 20 62    this time_..a b
00000040   69 74 20 6C 6F 6E 67 65 72 20 74 68 69 73 20 74   it longer this t
00000050   61 6D 65 08 1B 21 3A 00 00 00 00 00 00 01 01 00   ame..!:.........
00000060   00 00 00 00 00 00 04 00 00 00 00 00 00 00 00 00   ................
00000070   00 00 00 00 00 00 53                              ......S
```

```
<plist version="1.0">
<array>
    <string>short</string>
    <string>a bit longer this time</string>
    <string>short</string>
    <string>a bit longer this time</string>
    <string>short</string>
    <string>a bit longer this tame</string>
    <string>short</string>
    <string>a bit longer this time</string>
    <string>short</string>
    <string>a bit longer this time</string>
    <string>short</string>
    <string>a bit longer this time</string>
    <string>short</string>
    <string>a bit longer this time</string>
    <string>short</string>
    <string>a bit longer this time</string>
</array>
</plist>
```

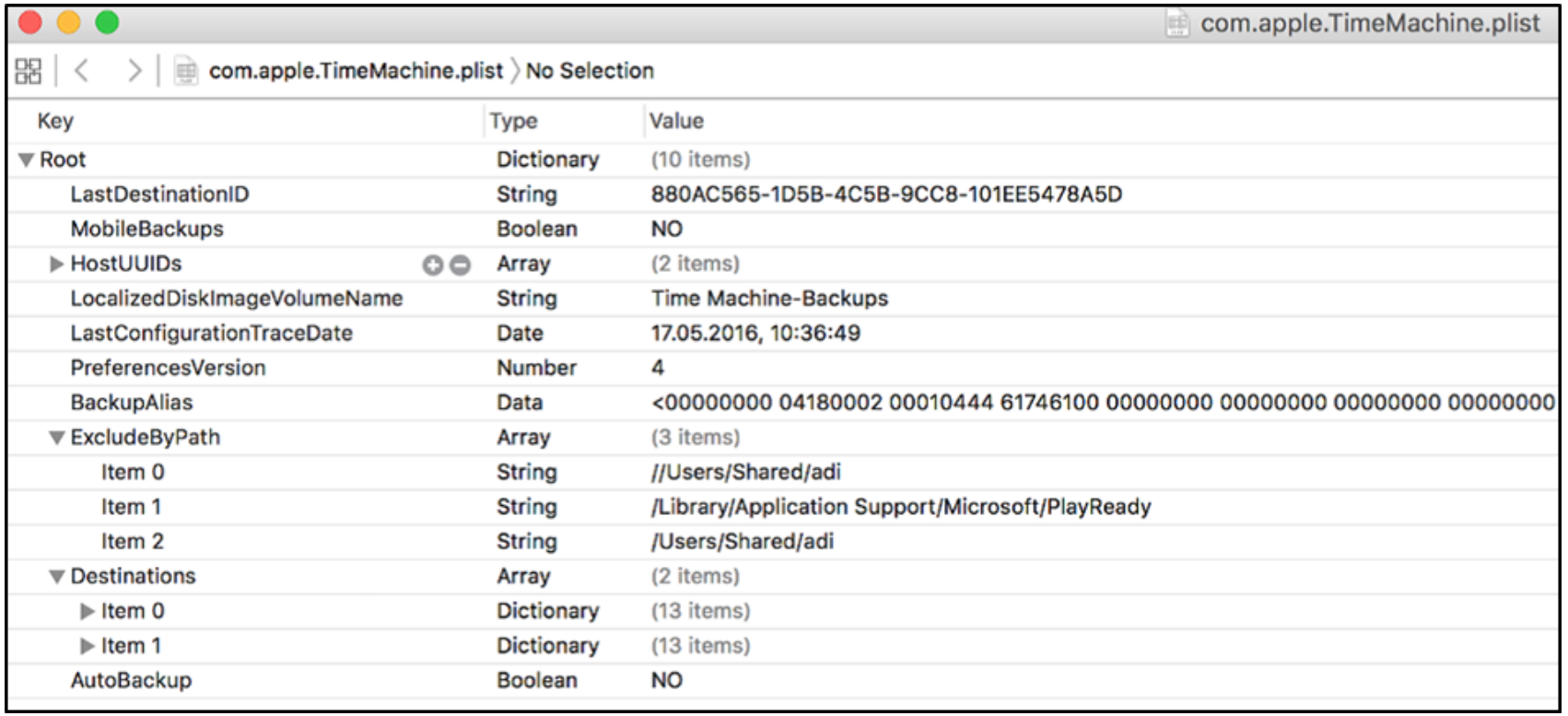
16 Array Member – Nummer bezieht sich auf Position innerhalb der Plist Offsets (beginnend bei 0):

01 02 01 02 01 03 01 02 01 02 01 02 01 02 01 02

Plattformabhängige Speichermethode

Interpretieren und Auswerten von *.plist Dateien

Neben plutil zum konvertieren von Property Lists liefert Apple einen kostenlosen Property List Editor zusammen mit der XCode Entwicklungsumgebung aus.

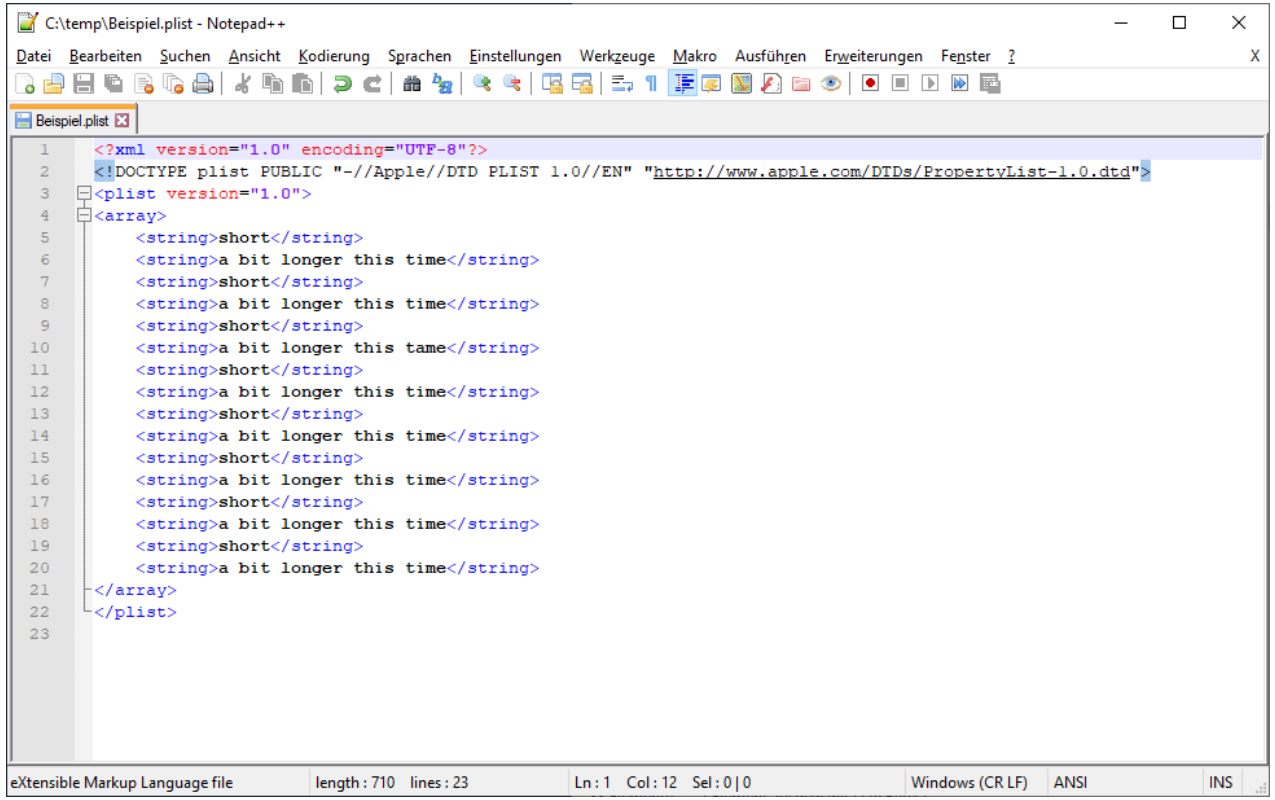


Key	Type	Value
▼ Root	Dictionary	(10 items)
LastDestinationID	String	880AC565-1D5B-4C5B-9CC8-101EE5478A5D
MobileBackups	Boolean	NO
▶ HostUUIDs	Array	(2 items)
LocalizedDiskImageVolumeName	String	Time Machine-Backups
LastConfigurationTraceDate	Date	17.05.2016, 10:36:49
PreferencesVersion	Number	4
BackupAlias	Data	<00000000 04180002 00010444 61746100 00000000 00000000 00000000 00000000
▼ ExcludeByPath	Array	(3 items)
Item 0	String	//Users/Shared/adi
Item 1	String	/Library/Application Support/Microsoft/PlayReady
Item 2	String	/Users/Shared/adi
▼ Destinations	Array	(2 items)
▶ Item 0	Dictionary	(13 items)
▶ Item 1	Dictionary	(13 items)
AutoBackup	Boolean	NO

Plattformabhängige Speichermethode

Interpretieren und Auswerten von *.plist Dateien

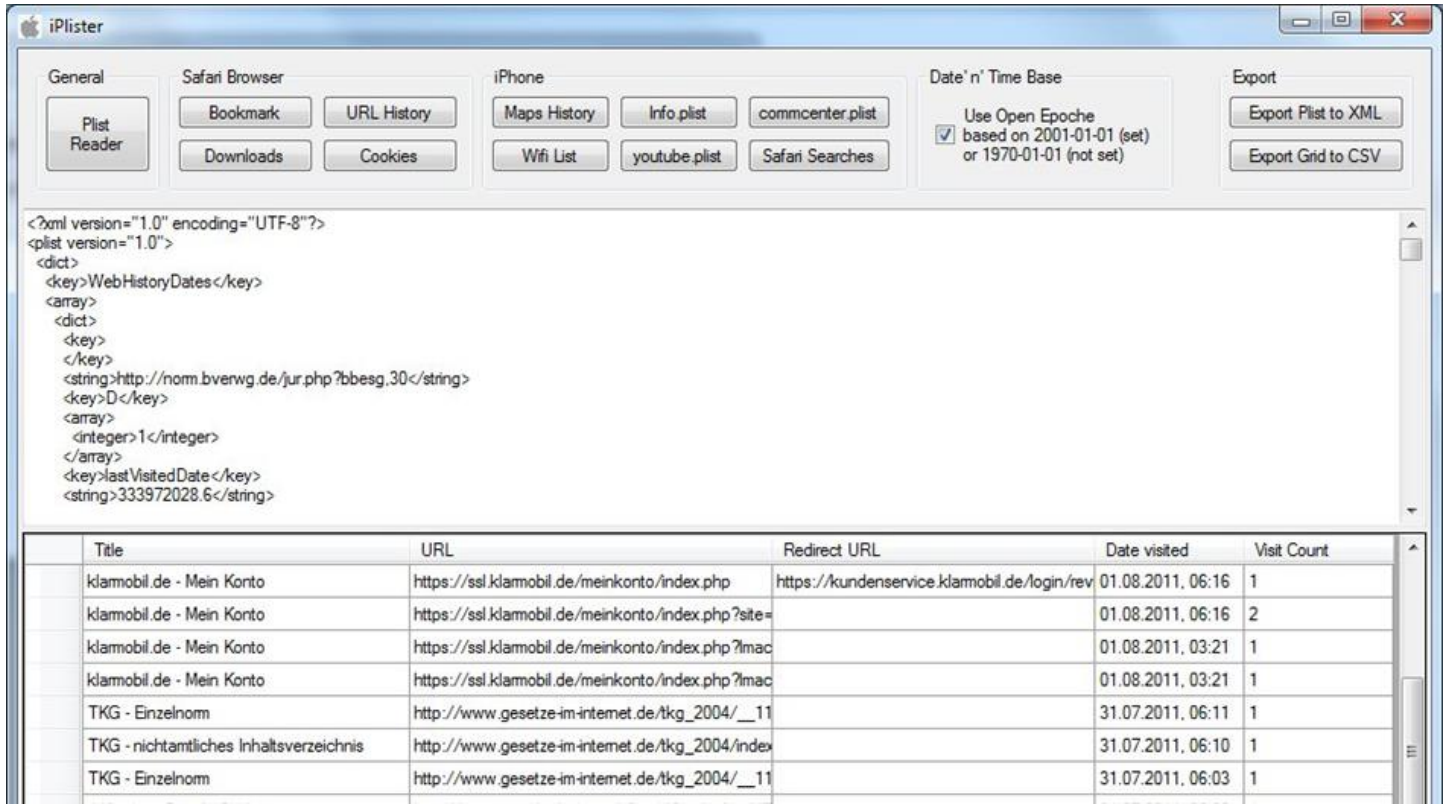
Daneben gibt es unzählige Tools für OSX um Property Lists zu bearbeiten oder anzuzeigen. Auch für Windows sind Programme verfügbar, wie etwa ein Bplist Plugin für Notepad++.



Plattformabhängige Speichermethode

Interpretieren und Auswerten von *.plist Dateien

Oder die freie Software iPlister aus dem Forensik Bereich von Eyewitness Forensic Software (www.eyewitnessforensic.com).



BETRIEBSSYSTEM macOS

Plattformunabhängige Speichermethoden

SQLite

Plattformunabhängige Speichermethoden

SQLite DB (relationale SQL Datenbank) (1)

Bekannteste und am weitesten verbreitete Datenbankformat für kleine und mittlere Desktop sowie mobile Anwendungen

Viele native Apps in Android und auch iOS nutzen SQLite Datenbanken

Bereits früh im Focus forensischer Untersuchungen

Gelöschte Datenbankeintragungen wiederherstellbar

Plattformunabhängige Speichermethoden

SQLite DB (relationale SQL Datenbank) (2)

Relationale Datenbank

Unterschiedliche Datenfelder in einem Datensatz zusammengefassbar und hinzugefügt, geändert oder gelöscht werden.

Abfragen mit SQL, Relationen zwischen Datenfeldern und Datensätzen zulässig

Nicht parallel verarbeitbar

Plattformunabhängige Speichermethoden

SQLite DB (relationale SQL Datenbank) (3)

Sehr viele Betriebssystemportierungen → große Verbreitung

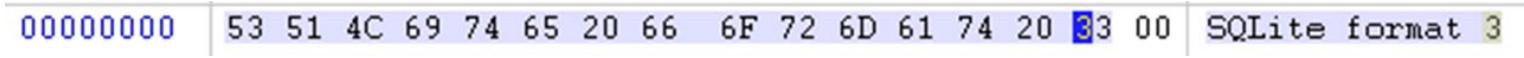
Bekannte Anwendungen die SQLite Datenbanken nutzen: Mozilla Firefox, Google Chrome, ICQ bis 7, Skype und native iOS Apps wie das Adressbuch sowie in Android die Kontakte.

Plattformunabhängige Speichermethoden

SQLite DB (relationale SQL Datenbank) (4)

SQLite Datenbank Aufbau:

- SQLITE 3 Header-Signatur:



- Bestehen aus Pages einer vordefinierten Größe
- Pages mit einzelnen Datensätzen (Records) beschrieben
- B-Tree basiert
- Können mit Journal Funktion versehen werden (Rollback oder Write Ahead Log - WAL)

Plattformunabhängige Speichermethoden

SQLite DB (relationale SQL Datenbank) (5)

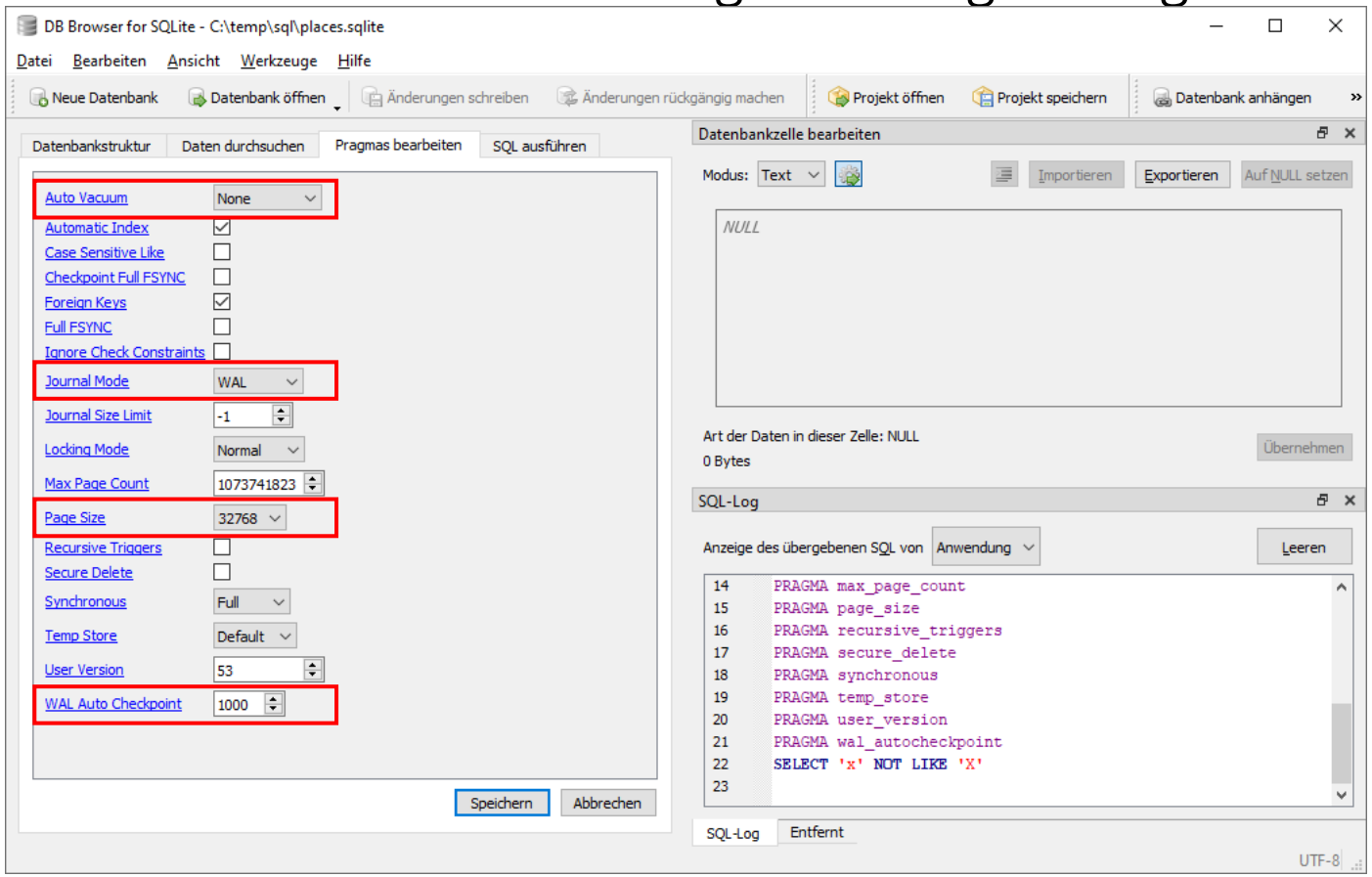
SQLite Datenbank Funktion:

- Änderungen an Datensätzen durch Ändern von Pages, dabei können neue Pages mit geänderten Daten entstehen
- alte Pages entweder freigegeben oder reorganisiert
- Automatische Compactionen möglich (Auto Vacuum), in dem freie Pages ans Ende gestellt und Datenbank automatisch gekürzt
- ohne Auto Vacuum freie Pages in Free Page List verwaltet und wiederverwendet
- Compaction (Vacuum) kann manuell erzwungen werden

Plattformunabhängige Speichermethoden

SQLite DB (relationale SQL Datenbank) (6)

Einstellungen der Datenbank können über Pragma Abfragen festgestellt werden:



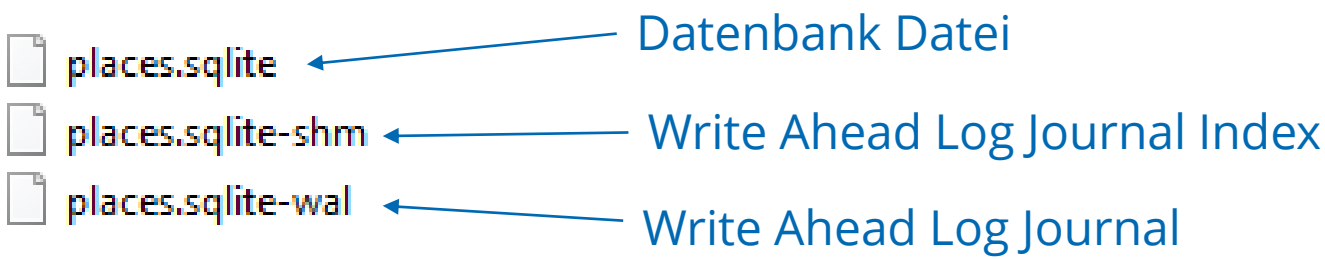
Plattformunabhängige Speichermethoden

SQLite DB (relationale SQL Datenbank) (7)

SQLite Datenbank Dateisystem Aufbau mit WAL Datei:

- Rollback oder WAL speichert Page Änderungen in zusätzlichen Dateien außerhalb der Datenbank und erst nach Transaktionsbestätigung eingearbeitet

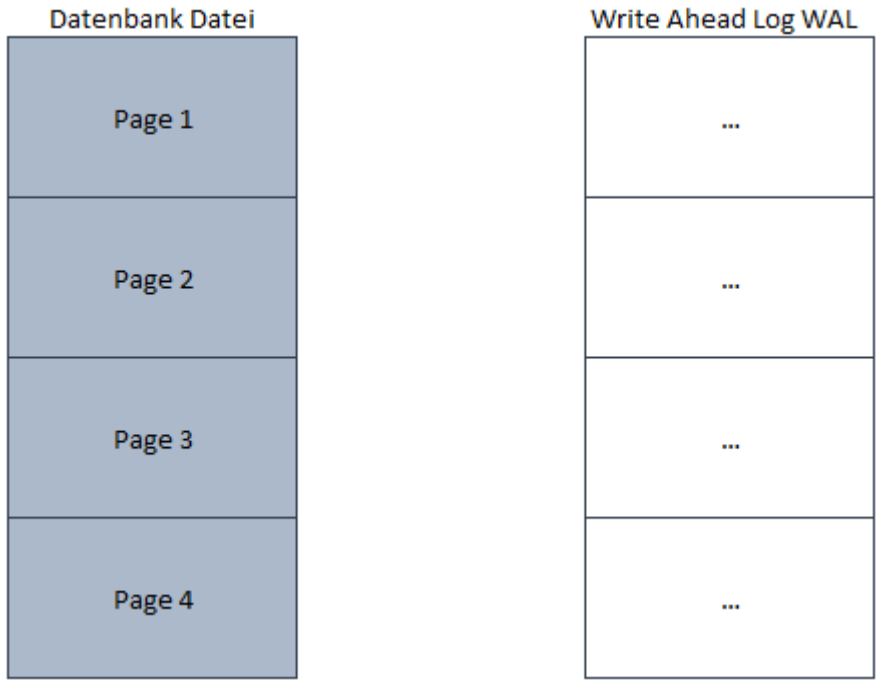
Dateisystem Aufbau mit WAL Datei:



Plattformunabhängige Speichermethoden

SQLite DB (relationale SQL Datenbank) (8)

SQLite Datenbank Funktion mit WAL Datei:

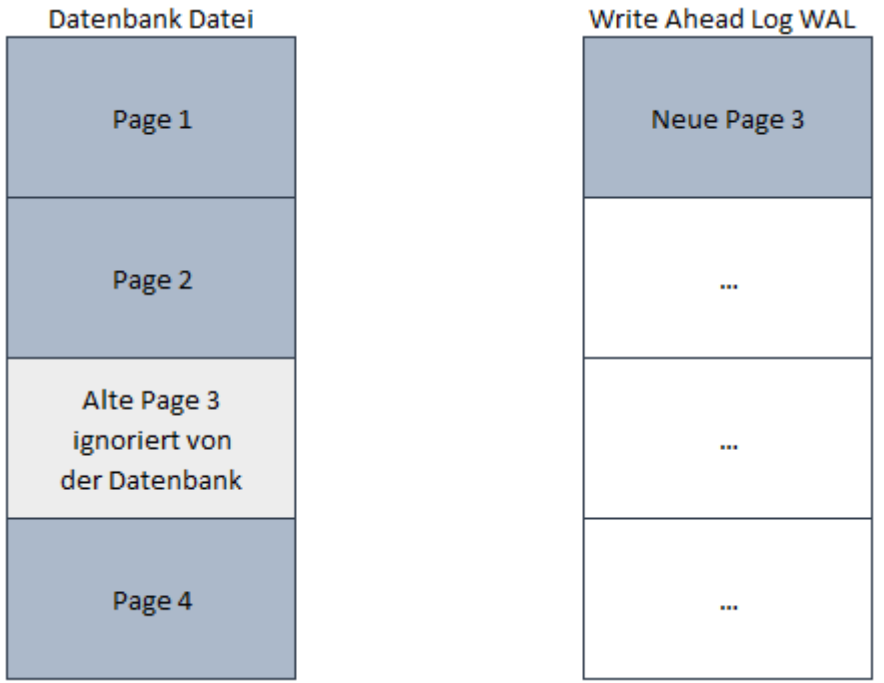


Initialisierungszustand der Datenbank

Plattformunabhängige Speichermethoden

SQLite DB (relationale SQL Datenbank) (9)

SQLite Datenbank Funktion mit WAL Datei:

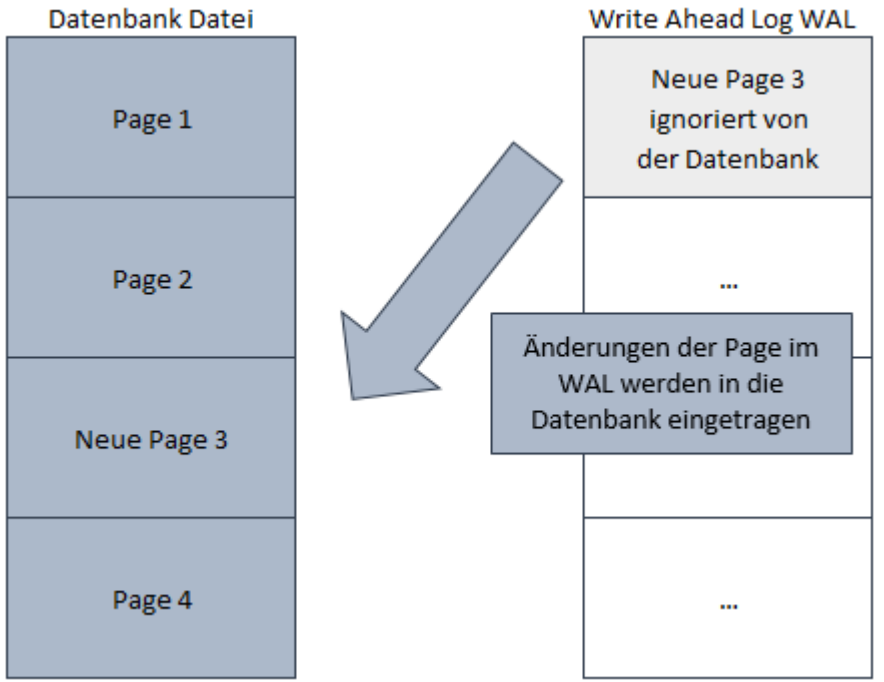


Page 3 wird geändert und in das WAL eingetragen

Plattformunabhängige Speichermethoden

SQLite DB (relationale SQL Datenbank) (10)

SQLite Datenbank Funktion mit WAL Datei:



Page 3 wird geändert vom WAL in die Datenbank geschrieben

Plattformunabhängige Speichermethoden

SQLite DB (relationale SQL Datenbank) (11)

SQLite Datenbank Dateisystem Aufbau mit WAL Datei:

- je nach eingestellter WAL Checkpoint Größe kann eine Eintragung der WAL Daten, wie auch Löschungen erst spät erfolgen
- Eintragungen aus WAL werden bei jedem Start und Ende der SQLite Datenbank Engine durchgeführt
- Bei Crash/Kill, sind diese Änderungen womöglich noch nicht durchgeführt worden
- Bei forensischen Untersuchungen daher auch die WAL Dateien entscheidende Rolle

BETRIEBSSYSTEM macOS

Plattformunabhängige Speichermethoden

SQLite - Wiederherstellung gelöschter Records

Plattformunabhängige Speichermethoden

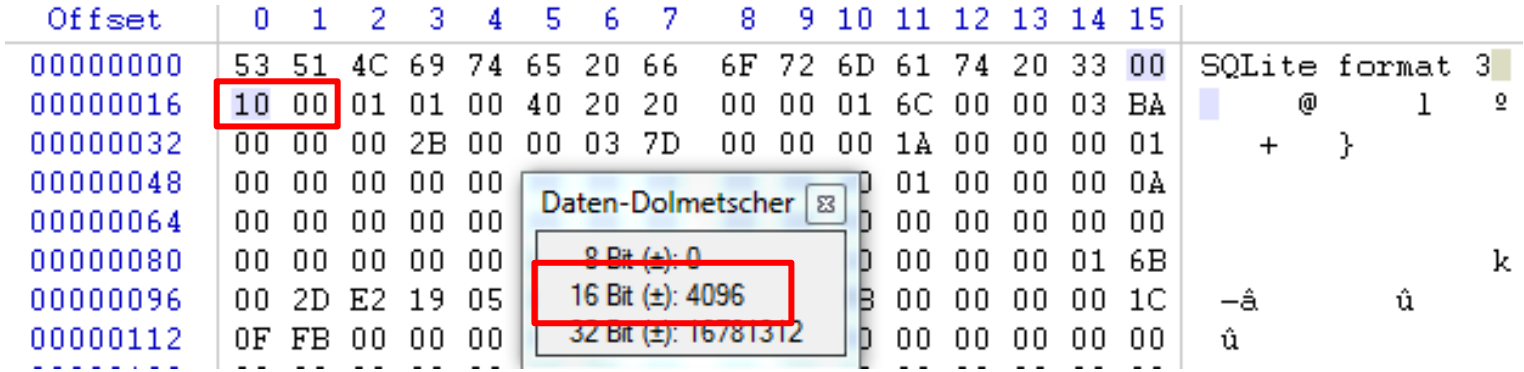
SQLite DB – Wiederherstellung gelöschter Records (1)

SQLITE 3 Datenbanken Header

Besitzen Header mit den wichtigsten Konfigurationsparameter als Big Endian Integers

Speichern Einträge ähnlich einem Dateisystem in Blöcken/Pages, mit variabel eingestellbarer Größe.

Page Größe variiert zwischen 512 und 32768 Bytes und befindet sich an Offset 16 der Datenbankdatei als 2 Byte BE Integer:



Plattformunabhängige Speichermethoden

SQLite DB – Wiederherstellung gelöschter Records (2)

SQLITE 3 Datenbanken Header

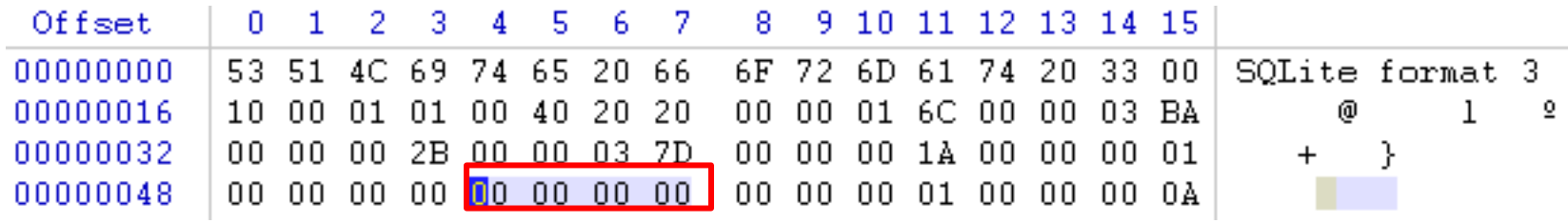
Offset 52: AutoVacuum Datenbank oder wachsende Größe.

AutoVacuum Datenbanken reorganisieren sich beim Löschen von Daten/Records neu.

In AutoVacuum Datenbanken lassen sich gelöschte Records nicht wiederherstellen.

AutoVacuum Datenbanken beinhalten in Offset 52 vier Non-Zero Byte.

Widerherstellbar wenn Offset 52 vier Zero Byte.



Offset	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
00000000	53	51	4C	69	74	65	20	66	6F	72	6D	61	74	20	33	00	SQLite format 3
00000016	10	00	01	01	00	40	20	20	00	00	01	6C	00	00	03	BA	@ l º
00000032	00	00	00	2B	00	00	03	7D	00	00	00	1A	00	00	00	01	+ }
00000048	00	00	00	00	00	00	00	00	00	00	00	01	00	00	00	0A	

Plattformunabhängige Speichermethoden

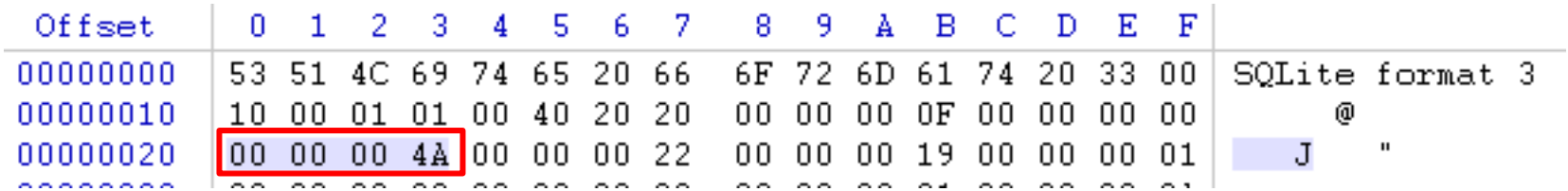
SQLite DB – Wiederherstellung gelöschter Records (3)

Vom Header zur FreepageList

SQLITE 3 Datenbanken speichern gelöschte Records in Free Pages.

Offset 32: erste Sprungadresse der Freelist Page.

Sprungadresse = (4 Byte BE in Offset 32 – 1) * Pagesize



Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
00000000	53	51	4C	69	74	65	20	66	6F	72	6D	61	74	20	33	00	SQLite format 3
00000010	10	00	01	01	00	40	20	20	00	00	00	0F	00	00	00	00	@
00000020	00	00	00	4A	00	00	00	22	00	00	00	19	00	00	00	01	J "

Sprungadresse hex = (0x4A - 1) * 0x1000 = 0x49000
Sprungadresse dec = (74 - 1) * 4096 = 299008

Plattformunabhängige Speichermethoden

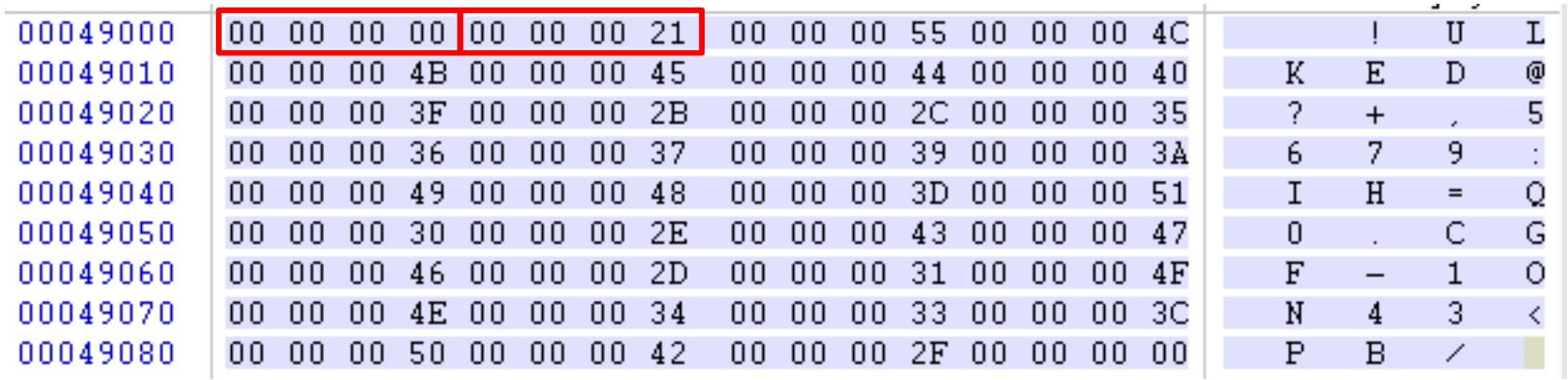
SQLite DB – Wiederherstellung gelöschter Records (4)

Die FreepageList

Speichern freie Pages, für ein Neubeschreiben in denen aber auch gelöschte Daten enthalten sein können.

Ersten 4 Byte BE verweisen auf weitere Free List Page oder andernfalls Zero.

Darauffolgenden 4 Byte stellen die Anzahl der folgenden Listeneinträge dar.



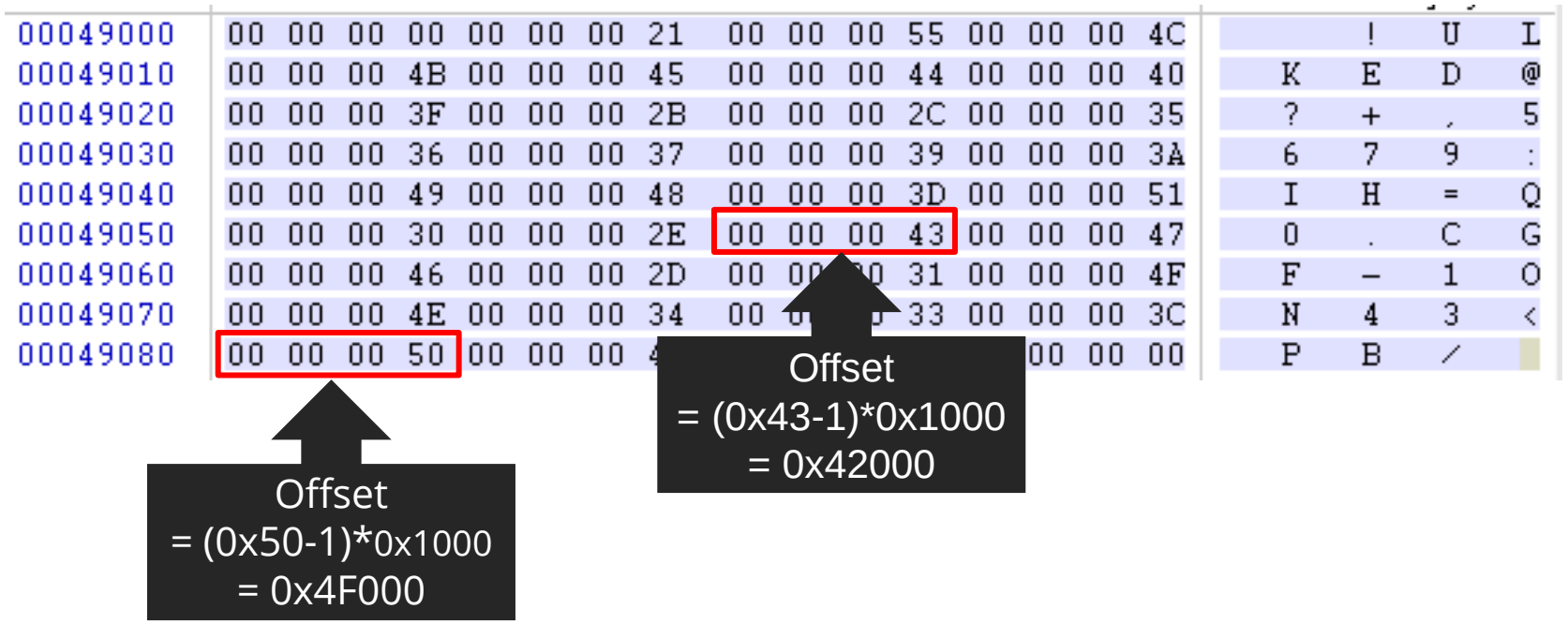
00049000	00	00	00	00	00	00	00	21	00	00	00	55	00	00	00	4C			!	U	L
00049010	00	00	00	4B	00	00	00	45	00	00	00	44	00	00	00	40	K	E	D	@	
00049020	00	00	00	3F	00	00	00	2B	00	00	00	2C	00	00	00	35	?	+	,	5	
00049030	00	00	00	36	00	00	00	37	00	00	00	39	00	00	00	3A	6	7	9	:	
00049040	00	00	00	49	00	00	00	48	00	00	00	3D	00	00	00	51	I	H	=	Q	
00049050	00	00	00	30	00	00	00	2E	00	00	00	43	00	00	00	47	0	.	C	G	
00049060	00	00	00	46	00	00	00	2D	00	00	00	31	00	00	00	4F	F	-	1	O	
00049070	00	00	00	4E	00	00	00	34	00	00	00	33	00	00	00	3C	N	4	3	<	
00049080	00	00	00	50	00	00	00	42	00	00	00	2F	00	00	00	00	P	B	/		

Plattformunabhängige Speichermethoden

SQLite DB – Wiederherstellung gelöschter Records (5)

Die FreepageList

Freepages als 4 Byte Eintragungen, die um 1 subtrahiert mit Pagesize multipliziert den Offset der gelöschten freien Page vom Beginn an ergeben.



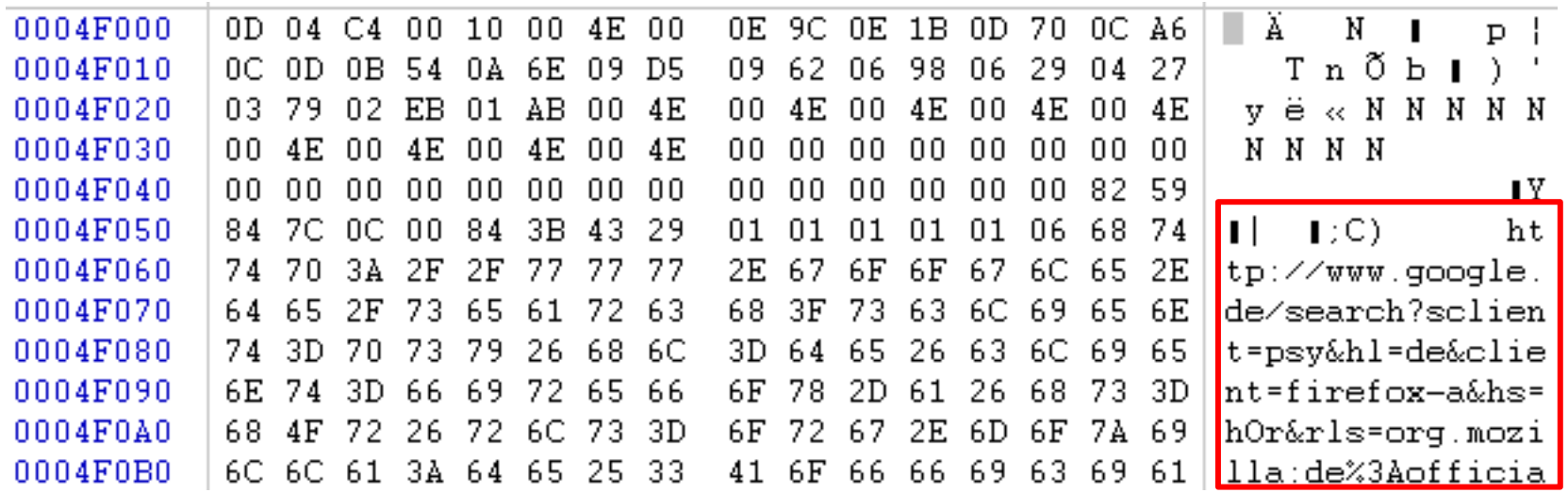
Plattformunabhängige Speichermethoden

SQLite DB – Wiederherstellung gelöschter Records (5)

Freepage einer FFX3 DB

Eine der gelöschten Freepages mit gelöschten Records Einträgen einer Mozilla Firefox moz_places Tabelle:

Offset= (0x50-1)*0x1000= 0x4F000



0004F000	0D	04	C4	00	10	00	4E	00	0E	9C	0E	1B	0D	70	0C	A6
0004F010	0C	0D	0B	54	0A	6E	09	D5	09	62	06	98	06	29	04	27
0004F020	03	79	02	EB	01	AB	00	4E	00	4E	00	4E	00	4E	00	4E
0004F030	00	4E	00	4E	00	4E	00	4E	00	00	00	00	00	00	00	00
0004F040	00	00	00	00	00	00	00	00	00	00	00	00	00	00	82	59
0004F050	84	7C	0C	00	84	3B	43	29	01	01	01	01	01	06	68	74
0004F060	74	70	3A	2F	2F	77	77	77	2E	67	6F	6F	67	6C	65	2E
0004F070	64	65	2F	73	65	61	72	63	68	3F	73	63	6C	69	65	6E
0004F080	74	3D	70	73	79	26	68	6C	3D	64	65	26	63	6C	69	65
0004F090	6E	74	3D	66	69	72	65	66	6F	78	2D	61	26	68	73	3D
0004F0A0	68	4F	72	26	72	6C	73	3D	6F	72	67	2E	6D	6F	7A	69
0004F0B0	6C	6C	61	3A	64	65	25	33	41	6F	66	66	69	63	69	61

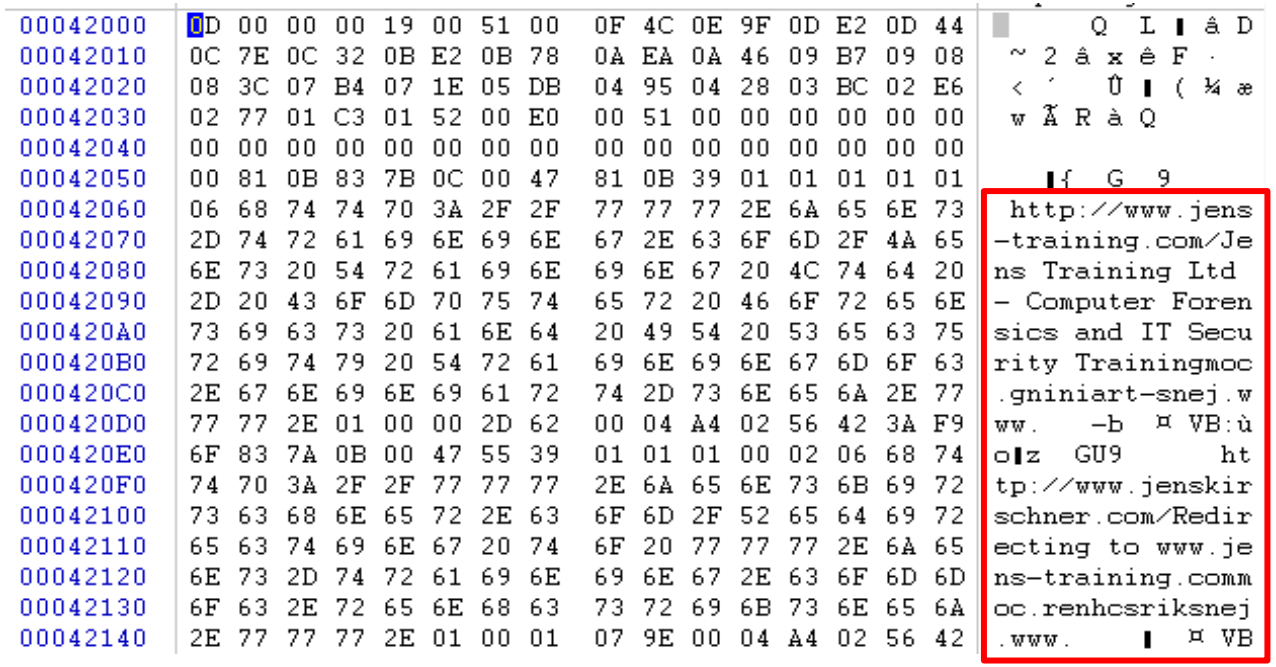
Plattformunabhängige Speichermethoden

SQLite DB – Wiederherstellung gelöschter Records (5)

Freepage einer FFX3 DB

Eine der gelöschten Freepages mit gelöschten Records Einträgen einer Mozilla Firefox moz_places Tabelle:

Offset= (0x43-1)*0x1000= 0x42000



```
00042000  0D 00 00 00 19 00 51 00  0F 4C 0E 9F 0D E2 0D 44
00042010  0C 7E 0C 32 0B E2 0B 78  0A EA 0A 46 09 B7 09 08
00042020  08 3C 07 B4 07 1E 05 DB  04 95 04 28 03 BC 02 E6
00042030  02 77 01 C3 01 52 00 E0  00 51 00 00 00 00 00 00
00042040  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
00042050  00 81 0B 83 7B 0C 00 47  81 0B 39 01 01 01 01 01
00042060  06 68 74 74 70 3A 2F 2F  77 77 77 2E 6A 65 6E 73
00042070  2D 74 72 61 69 6E 69 6E  67 2E 63 6F 6D 2F 4A 65
00042080  6E 73 20 54 72 61 69 6E  69 6E 67 20 4C 74 64 20
00042090  2D 20 43 6F 6D 70 75 74  65 72 20 46 6F 72 65 6E
000420A0  73 69 63 73 20 61 6E 64  20 49 54 20 53 65 63 75
000420B0  72 69 74 79 20 54 72 61  69 6E 69 6E 67 6D 6F 63
000420C0  2E 67 6E 69 6E 69 61 72  74 2D 73 6E 65 6A 2E 77
000420D0  77 77 2E 01 00 00 2D 62  00 04 A4 02 56 42 3A F9
000420E0  6F 83 7A 0B 00 47 55 39  01 01 01 00 02 06 68 74
000420F0  74 70 3A 2F 2F 77 77 77  2E 6A 65 6E 73 6B 69 72
00042100  73 63 68 6E 65 72 2E 63  6F 6D 2F 52 65 64 69 72
00042110  65 63 74 69 6E 67 20 74  6F 20 77 77 77 2E 6A 65
00042120  6E 73 2D 74 72 61 69 6E  69 6E 67 2E 63 6F 6D 6D
00042130  6F 63 2E 72 65 6E 68 63  73 72 69 6B 73 6E 65 6A
00042140  2E 77 77 77 2E 01 00 01  07 9E 00 04 A4 02 56 42
```

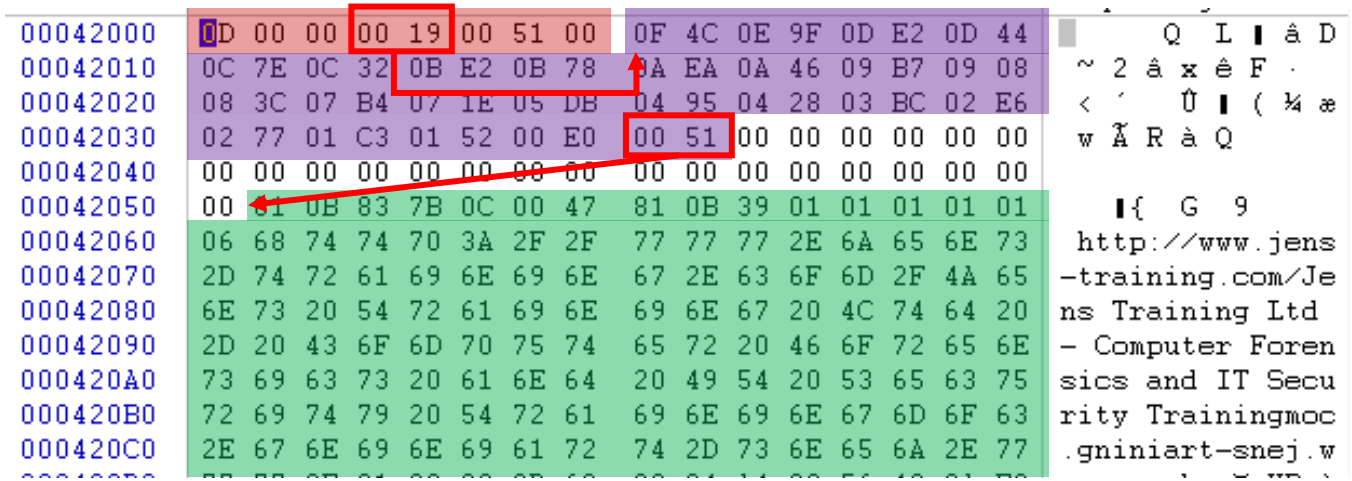
```
  Q  L ▌ â D
 ~ 2 â x ê F ·
 <  ´   Û ▌ ( ¼ æ
 w Ã R à Q

  ▌{  G  9
 http://www.jens
-training.com/Je
ns Training Ltd
- Computer Foren
sics and IT Secu
rity Trainingmoc
.gniniart-snej.w
ww.   -b  ¤ VB:ù
o▌z  GU9     ht
tp://www.jenskir
schner.com/Redir
ecting to www.je
ns-training.comm
oc.renhcsriksnej
.www.    ▌  ¤ VB
```

Plattformunabhängige Speichermethoden

SQLite DB – Wiederherstellung gelöschter Records (5)

Freepage Aufbau

B-Baum und besteht aus Header, Cell Pointer Array und Cell Einträgen:

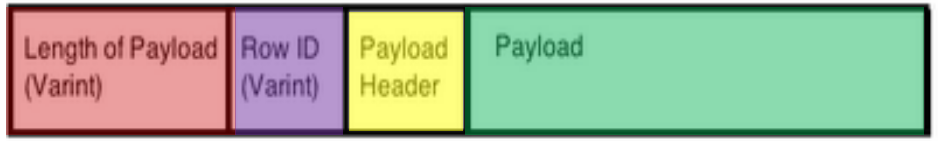


- Offset 0: 0x0A Indexpage und 0x0D Datenpage
- Offset 3+4 i:Anzahl folgender Cellpointer
- ab Offset 8 je 2 Byte Cellpointer, Offset der Cell zum Page Anfang

Plattformunabhängige Speichermethoden

SQLite DB – Wiederherstellung gelöschter Records (6)

Free Page Records



Payload Header Länge (Varint)

```
00042050  00 81 0B 83 7B 0C 00 47  81 0B 39 01 01 01 01 01   ▮{  G  9
00042060  06 68 74 74 70 3A 2F 2F  77 77 77 2E 6A 65 6E 73   http://www.jens
00042070  2D 74 72 61 69 6E 69 6E  67 2E 63 6F 6D 2F 4A 65  -training.com/Je
00042080  6E 73 20 54 72 61 69 6E  69 6E 67 20 4C 74 64 20  ns Training Ltd
00042090  2D 20 43 6F 6D 70 75 74  65 72 20 46 6F 72 65 6E  - Computer Foren
000420A0  73 69 63 73 20 61 6E 64  20 49 54 20 53 65 63 75  sics and IT Secu
000420B0  72 69 74 79 20 54 72 61  69 6E 69 6E 67 6D 6F 63  rity Trainingmoc
000420C0  2E 67 6E 69 6E 69 61 72  74 2D 73 6E 65 6A 2E 77  .gniniart-snej.w
000420D0  77 77 2E 01 00 00 2D 62  00 04 A4 02 56 42 3A F9  ww.    -b  ¤ VB:ù
000420E0  6F 83 7A 0B 00 47 55 39  01 01 01 00 02 06 68 74  o▮z  GU9       ht
000420F0  74 70 3A 2F 2F 77 77 77  2E 6A 65 6E 73 6B 69 72  tp://www.jenskir
00042100  73 63 68 6E 65 72 2E 63  6F 6D 2F 52 65 64 69 72  schner.com/Redir
00042110  65 63 74 69 6E 67 20 74  6F 20 77 77 77 2E 6A 65  ecting to www.je
00042120  6E 73 2D 74 72 61 69 6E  69 6E 67 2E 63 6F 6D 6D  ns-training.comm
00042130  6F 63 2E 72 65 6E 68 63  73 72 69 6B 73 6E 65 6A  oc.renhcsriksnej
00042140  2E 77 77 77 2E 01 00 01  07 9E 00 04 A4 02 56 42  .www.     ▮   ¤ VB
```

Plattformunabhängige Speichermethoden

SQLite DB – Wiederherstellung gelöschter Records (6)

Ein Payload Header ist nach folgendem Schema aufgebaut:

Serial Type	Content Size	Meaning
0	0	NULL
1	1	8-bit twos-complement integer
2	2	Big-endian 16-bit twos-complement integer
3	3	Big-endian 24-bit twos-complement integer
4	4	Big-endian 32-bit twos-complement integer
5	6	Big-endian 48-bit twos-complement integer
6	8	Big-endian 64-bit twos-complement integer
7	8	Big-endian IEEE 754-2008 64-bit floating point number
8	0	Integer constant 0. Only available for schema format 4 and higher.
9	0	Integer constant 1. Only available for schema format 4 and higher.
10,11		*Not used. Reserved for expansion.*
N≥12 and even	(N-12)/2	A BLOB that is (N-12)/2 bytes in length
N≥13 and odd	(N-13)/2	A string in the database encoding and (N-13)/2 bytes in length. The nul terminator is omitted.

```
00 47 81 0B 39 01 01 01 01 01 06
```

0x00 NULL (ROWID – Primary Key)
0x47 String Länge 29 Bytes (URL)
0x810B String Länge 63 Bytes (Titel)
0x39 String Länge 22 Bytes (reverse URL)
0x01 Int8
0x01 Int8
0x01 Int8
0x01 Int8
0x01 Int8
0x01 Int8
0x06 Int64 (Datum Unix Epoche)

Länge gespeicherter Informationen entweder durch Typ vorgegeben oder als VarInt angegeben

Plattformunabhängige Speichermethoden

SQLite DB – Wiederherstellung gelöschter Records (7)

Der damit entschlüsselte Dateninhalt einer FFX3 moz_places Zeile:

```
00 47 81 0B 39 01 01 01 01 01 06
```

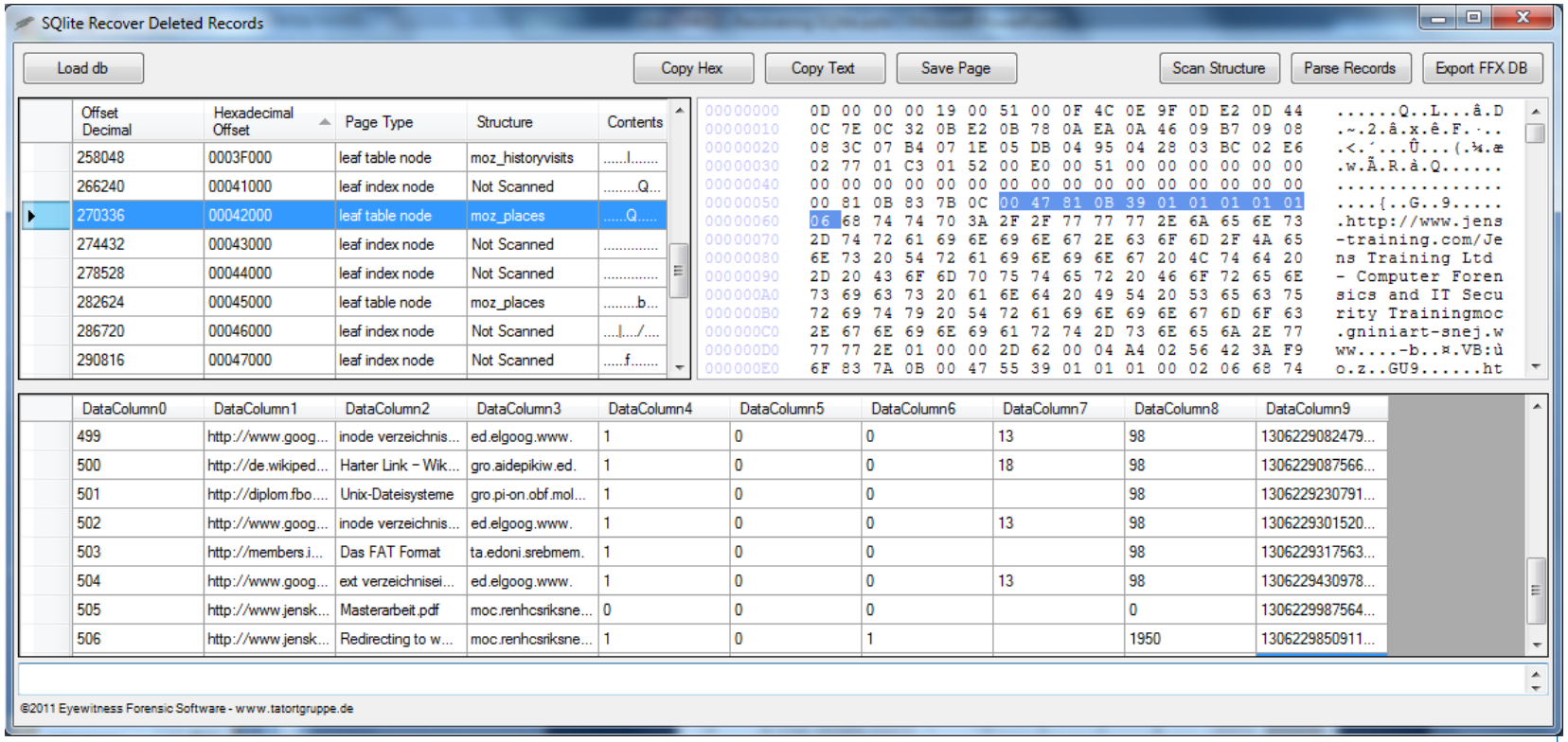
0x00 NULL (ROWID – Primary Key)
0x47 String Länge 29 Bytes (URL)
0x810B String Länge 63 Bytes (Titel)
0x39 String Länge 22 Bytes (reverse URL)
0x01 Int8
0x01 Int8
0x01 Int8
0x01 Int8
0x01 Int8
0x06 Int64 (Datum Unix Epoche)

```
http://www.jens
-training.com/Je
ns Training Ltd
- Computer Foren
sics and IT Secu
rity Trainingmoc
.gniniart-snej.w
ww.    -b  ¤ VB:ù
o▮z  GU9      ht
```

Plattformunabhängige Speichermethoden

SQLite DB – Wiederherstellung gelöschter Records (8)

Es existieren derzeit einige Forensic Tools am Markt, die in der Lage sind gelöschte Datenbank Records wiederherzustellen:

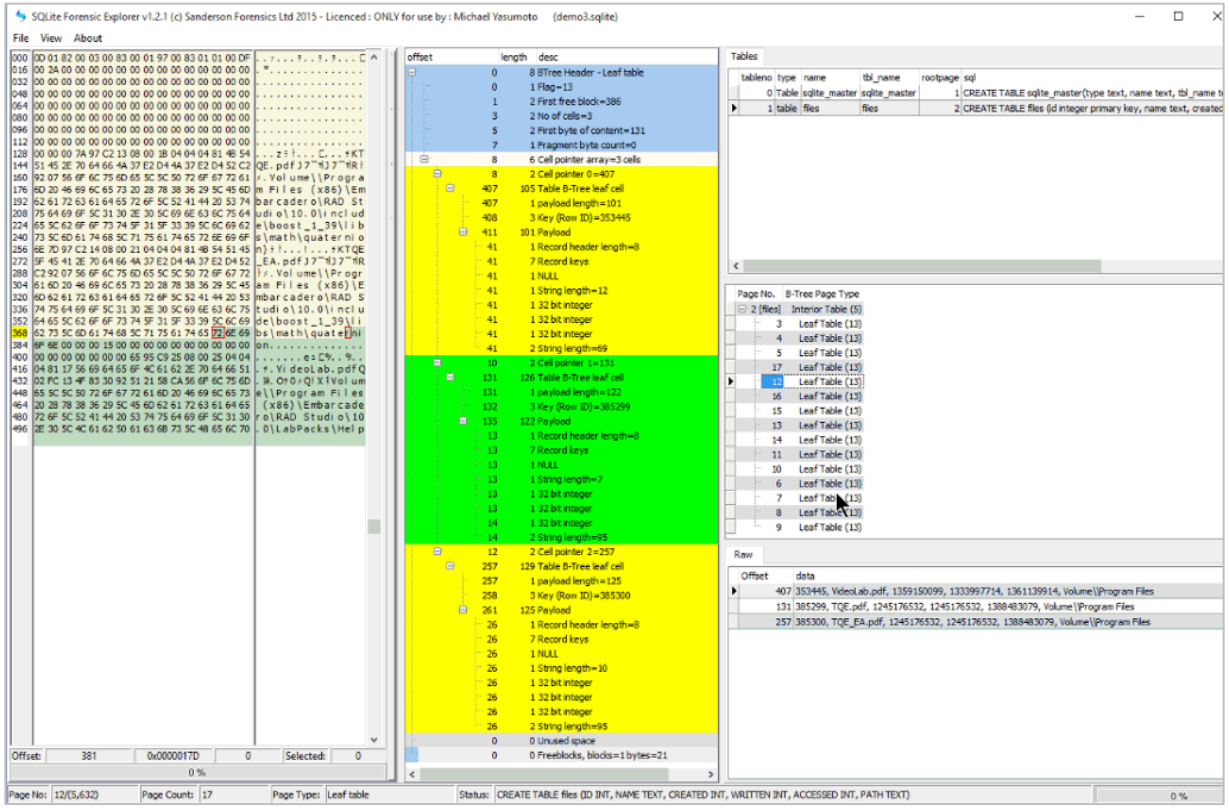


Quelle: Autor

Plattformunabhängige Speichermethoden

SQLite DB – Wiederherstellung gelöschter Records (9)

Eines der umfangreichsten Toolkits ist jedoch des SQLite Forensik Toolkit von Sanderson Forensics:



Quelle: Sqliteforensictoolkit.com

Plattformunabhängige Speichermethoden

SQLite DB – Wiederherstellung gelöschter Records (10)

Auch ein Python Script existiert, welches gelöschte Records parsen kann:

https://github.com/mdegrazia/SQLite-Deleted-Records-Parser

D21: Ghttp://www.reddit.com/r/funny/comments/1pnx9t/my_cousins_costume_hes_a_dentistand_yes_hes_at/My cousin's costume. He's a dentist...and yes, he's at work. : funny.H\

	A	B	C	D
1	Type	Offset	Length	Data
11	Unallocate	393270	66	H0
12	Unallocate	434266	56	\6
13	Unallocate	446544	71	g0
14	Free Block	446700	180	- Ghttp://www.reddit.com/r/funny/comments/1pn79p/i_may_technically_be_an_adult_but_i_laughed_oc/I may technically be an adult, bu
15	Free Block	448557	204	M/http://www.reddit.com/r/funny/comments/1pmnxs/my_five_year_old_dressed_as_michael_jackson_today/My five year old dressed as
16	Unallocate	454740	76	p0
17	Free Block	457046	201	G/http://www.reddit.com/r/funny/comments/1pn24h/this_kid_in_school_was_just_running_into_walls/This kid in school was just running
18	Unallocate	491598	254	LLLLkkk Eahttp://www.reddit.com/r/funny/comments/1pnih3/someone_in_my_school_drew_jfk_and_the_teacher/Someone in my school
19	Free Block	492158	240	}http://www.reddit.com/r/funny/?count=225&after=t3_1pn7ssfunny.He 5ghttp://www.reddit.com/r/funny/comments/1pn7ss/my_pregnar
20	Free Block	492707	169	Awhttp://www.reddit.com/r/funny/comments/1pnx46/i_work_at_a_retirement_home_it_is_halloween/I work at a retirement home. It is
21	Free Block	493833	188	Ghttp://www.reddit.com/r/funny/comments/1pnx9t/my_cousins_costume_hes_a_dentistand_yes_hes_at/My cousin's costume. He's a der
22	Free Block	494974	177	}http://www.reddit.com/r/funny/?count=200&after=t3_1pmifjfunny.H=\w+http://www.reddit.com/r/funny/comments/1pmifj/genius/Gen
23	Unallocate	503894	77	m6
24				
25				

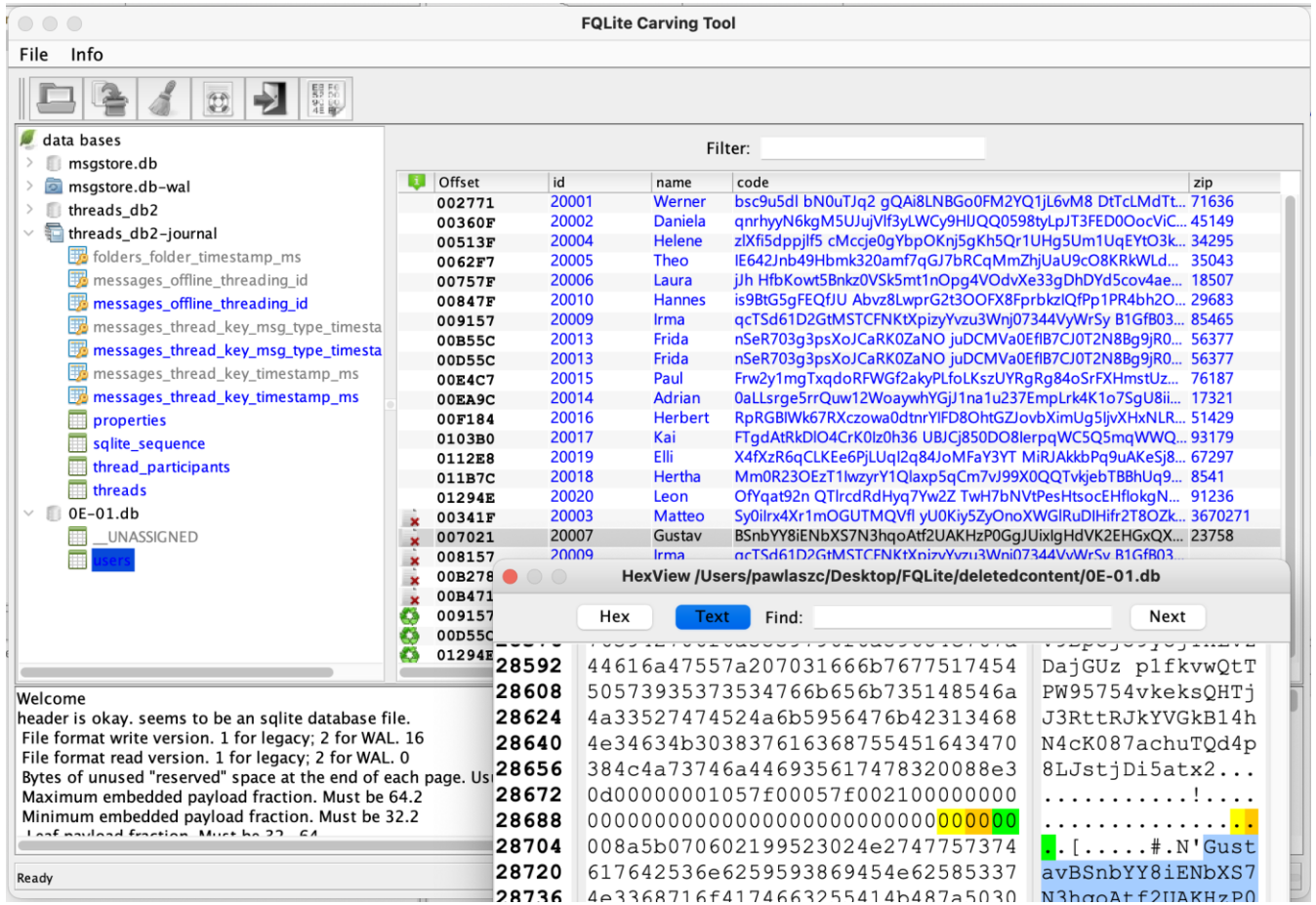
Quelle: http://az4n6.blogspot.com/2013/11/python-parser-to-recover-deleted-sqlite.html

Plattformunabhängige Speichermethoden

SQLite DB – Wiederherstellung gelöschter Records (11)

Ein freies Tool, welches die derzeit bekannten Ansätze zur Wiederherstellung gelöschter Daten beinhaltet ist **FQLite**:

https://github.com/pawlaszczyk/**fqlite**



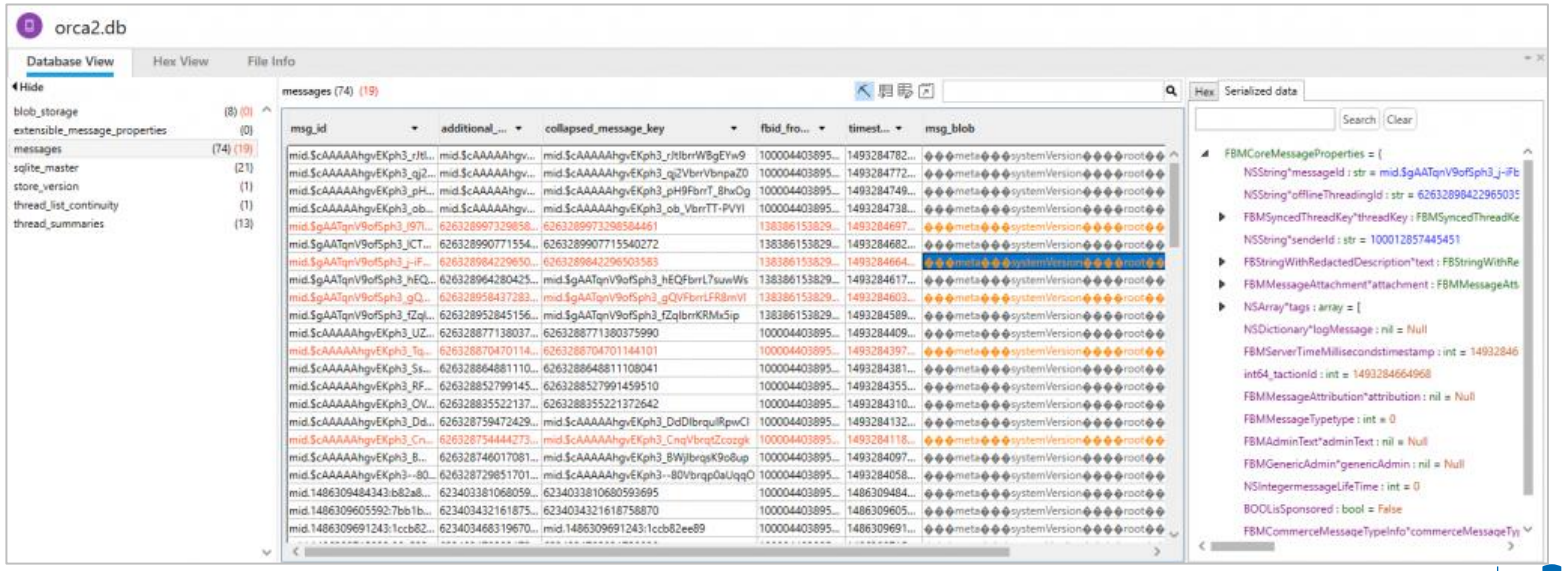
Quelle: https://www.staff.hs-mittweida.de/~pawlaszc/fqlite/

Plattformunabhängige Speichermethoden

SQLite DB – Wiederherstellung gelöschter Records (12)

Cellebrite Software Physical Analyzer ist ebenfalls in der Lage SQLite Datenbanken aufzubereiten und gelöschte Records wiederherzustellen.

Zusätzlich existiert ein Tool, um unbekannte SQLite Datenbanken aufzubereiten und in Tabellenformate zu überführen, um sie berichtsfähig zu machen.



Quelle: Cellebrite

Vielen Dank

Prof. Ronny Bodach

Hochschule Mittweida | University of Applied Sciences
Technikumplatz 17 | 09648 Mittweida
Fakultät Angewandte Computer- und Biowissenschaften

T +49 (0) 3727 58-1011
F +49 (0) 3727 58-21011
bodach@hs-mittweida.de
www.cb.hs-mittweida.de

Haus 8 | Richard-Stücklen Bau | Raum 8-205
Am Schwanenteich 6b | 09648 Mittweida

HOCHSCHULE MITTWEIDA
University of Applied Sciences

hs-mittweida.de