Eigene Formatsignaturen für DROID und Zuarbeit zu PRONOM

Diese Anleitung beschreibt die technischen Grundlagen der in PRONOM und DROID verwendeten Formatsignaturen und erklärt, wie eigene Formatsignaturen entwickelt und in PRONOM integriert werden können.

Grundlagen

Dateiformate werden in PRONOM bzw. DROID hauptsächlich anhand charakteristischer Bitmuster identifiziert. PRONOM hält Beschreibungen solcher Muster vor, und DROID gleicht diese Muster mit zu untersuchenden Dateien ab.

Um diese charakteristischen Bitmuster zu verstehen, muss man sich zunächst bewusst machen, dass alle digitalen Daten, unabhängig von ihrem Dateiformat (sowohl Binär- als auch Textdateien!), aus Folgen von Bits bestehen. Ein Bit ist die kleinste Einheit, in der ein Computer Informationen speichert. Es kann genau zwei Zustände annehmen, die häufig mit `0` und `1` bezeichnet werden. Legt man nun für eine bestimmte Folge von Bits eine definierte Bedeutung fest, z.B. den Buchstaben A oder die Zahl 65 oder "ein grünes Pixel" für die Bitfolge `01000001`, dann können so unterschiedlichste Daten codiert werden: Text-, Bild-, Audio- und Videoformate, aber auch ausführbare Programme.

Dass die Bedeutung solcher Bitfolgen reine Definitionssache ist, lässt sich leicht daran nachvollziehen, dass unterschiedliche Anwendungsprogramme dieselben Bitfolgen auf unterschiedliche Weise interpretieren: Öffnet man eine Bilddatei mit einem passenden Viewer, wird das durch eine lange Bitfolge codierte Bild angezeigt. Öffnet man dieselbe Datei aber mit einem Texteditor, wird man mit einer Reihe kryptischer Zeichen konfrontiert, weil der Texteditor in der Bitfolge keine Muster findet, die er als Buchstaben interpretieren kann.

Will man eine Bitfolge näher untersuchen, dann ist ihre interpretierte Darstellung z.B. als Bild oder Text hinderlich, denn tatsächlich ist in diesem Fall die rohe Bitfolge selbst von Interesse. Um diese unmittelbar zu betrachten, werden sogenannte Hex-Editoren verwendet. Ein Hex-Editor zeigt eine Datei als Folge von Bits an, wobei meist nicht die binäre Darstellung mit den Symbolen `0` und `1` zum Einsatz kommt, sondern die Bitfolgen stattdessen durch kompaktere und dadurch besser lesbare Hexadezimalzahlen auf Basis der Ziffern `0`-`F` dargestellt werden (daher der Name Hex-Editor). Die Eigenschaft der Hexadezimalzahlen als *Zahlen* ist hier übrigens nicht weiter wichtig. Entscheidend ist nur, dass durch jede zweistellige Hexadezimalzahl ein Byte, das ist eine Folge von acht Bits, eindeutig dargestellt wird.

Der Anfang einer XML-Datei beispielsweise sieht in einem Hex-Editor üblicherweise so oder so ähnlich aus:

```
0000000: 3C3F 786D 6C20 7665 7273 696F 6E3D 2231  <?xml version="1
0000010: 2E30 2220 656E 636F 6469 6E67 3D22 5554  .0" encoding="UT
0000020: 462D 3822 3F3E 0A3C 6561 6420 786D 6C6E  F-8"?>.<ead xmln
```

In der linken Spalte (bis zum Doppelpunkt) stehen die Nummern der Bytepositionen, grob vergleichbar einer Zeilennummerierung. Ganz rechts versucht der Hex-Editor, den Inhalt der Datei als Text zu interpretieren, was in diesem Fall auch recht gut gelingt, denn XML ist ein textbasiertes Format. In den mittleren acht Spalten findet sich die in der Datei enthaltene Bitfolge in Hexadezimaldarstellung. Nur diese acht Spalten machen den eigentlichen Dateiinhalt aus! Die linke und die rechte Spalte sowie sämtliche Leerzeichen hingegen sind bloß schmückendes Beiwerk, das der Hex-Editor als Orientierungshilfe für menschliche Leser hinzugefügt hat. Wollte man nur die reine Bitfolge in Hexadezimaldarstellung anzeigen, sähe das so aus:

```
3C3F786D6C2076657273696F6E3D22312E302220656E636F64696E673D225554 (usw.)
```

Der Anfang einer TIFF-Datei stellt sich etwas anders dar:

```
0000000: 4949 2A00 0800 0000 1700 FE00 0400 0100  II*.............
0000010: 0000 0000 0000 0001 0300 0100 0000 9C0F  ................
0000020: 0000 0101 0300 0100 0000 B60A 0000 0201  ................
```

Da sich die Bitfolge, aus der eine TIFF-Datei besteht, nicht sinnvoll als Text interpretieren lässt, werden in der rechten Spalte hier für die meisten Bytes bloß Punkte als Platzhalter eingesetzt. Selbst wenn manche Bitfolgen auch als Buchstaben interpretiert werden können (wie hier im Fall der zwei `I` ganz am Anfang), ist das oft Zufall und hat nicht unbedingt eine Bedeutung. Was den reinen Dateiinhalt auf Ebene der Bits angeht, unterscheiden sich XML und TIFF und viele andere Formate auf den ersten Blick kaum; auch eine TIFF-Datei besteht aus einer langen Folge von Bits:

```
49492A00080000001700FE0004000100000000000000000103000100000009C0F (usw.)
```

Genau wie ein Hex-Editor beschreiben auch die Signaturen in PRONOM Bitmuster durch Hexadezimalzahlen. So ist beispielsweise im Eintrag https://www.nationalarchives.gov.uk/PRONOM/fmt/353 vermerkt, dass TIFF-Dateien stets mit der charakteristischen Bitfolge `49492A00` beginnen. Öffnet man wie im gerade betrachteten Beispiel eine TIFF-Datei in einem Hex-Editor, wird man am Anfang der Datei auf genau diese Bitfolge stoßen. Vergleichbar arbeitet auch DROID: Findet sich die genannte Bitfolge am Anfang einer Datei, wird sie als TIFF identifiziert.

Signatur erstellen

Wie kann nun eine neue Signatur für PRONOM bzw. DROID erstellt werden?

Ein charakteristisches Bitmuster sollte zwei wesentliche Eigenschaften erfüllen:

- Das Muster muss in *allen* Dateien des zu beschreibenden Dateiformats vorkommen, bestenfalls immer an der selben Stelle (z.B. am Anfang der Datei).
- Das Muster sollte möglichst *nur* in Dateien des zu beschreibenden Dateiformats vorkommen, damit nicht ein anderes Dateiformat falsch identifiziert wird. Diese Eigenschaft kann natürlich nie restlos sichergestellt werden, denn das gewählte Muster könnte ja durchaus in einem anderen Dateiformat in einer anderen Funktion auftreten.

Die erste Eigenschaft lässt sich um so leichter erfüllen, je kürzer und allgemeiner das Muster ist. Die zweite Eigenschaft lässt sich hingegen um so leichter erfüllen, je länger und spezieller das Muster ist. Wie so oft liegt daher auch bei der Auswahl eines guten Bitmusters die Wahrheit irgendwo in der Mitte.

Oft ist es aber gar nicht schwer, eine geeignete Bitfolge zu bestimmen. Denn viele Dateiformate folgen einer formalen Spezifikation, in der u.U. charakteristische Bitmuster bzw. Texte festgelegt werden, die in allen Dateien dieses Formats auftreten. Ist eine solche Spezifikation vorhanden, empfiehlt sich daher deren Studium. Andernfalls wird die Mustersuche allerdings aufwändiger, denn dann muss eine möglichst große Sammlung von Beispieldateien zusammengetragen und untersucht werden.

Bitmuster in einer Spezifikation finden

Manche Dateiformatspezifikationen definieren eine feste Bitfolge, die in allen Dateien dieses Formats auftritt und entweder erklärtermaßen als Hilfsmittel zur Identifizierung des Dateiformats dient oder zumindest dafür zweckentfremdet werden kann. Oft werden solche Bitfolgen oder Texte als magic number, format string oder ähnlich bezeichnet und stehen in den Abschnitten der Spezifikation, die sich mit der Dateistruktur beschäftigen. Ein schönes Beispiel dafür findet sich in der Formatspezifikation des Datenbankformats SQLite 3 unter dem Titel "Magic Header String":

> *Every valid SQLite database file begins with the following 16 bytes (in hex): 53 51 4c 69 74 65 20 66 6f 72 6d 61 74 20 33 00. This byte sequence corresponds to the UTF-8 string "SQLite format 3" including the nul terminator character at the end.*

Tatsächlich wird man, wenn man eine beliebige SQLite 3-Datei im Hex-Editor öffnet, diese Bitfolge wiederfinden:

```
0000000: 5351 4c69 7465 2066 6f72 6d61 7420 3300  SQLite format 3.
0000010: 0400 0101 0040 2020 0000 0003 0000 0002  .....@  ........
0000020: 0000 0000 0000 0000 0000 0001 0000 0004  ................
```

Und auch DROID verwendet genau diese Bitfolge zur Identifizierung von SQLite 3-Dateien, wie ein Blick auf den Reiter "Signatures" im PRONOM-Eintrag für das Dateiformat zeigt. Dort ist sowohl die genannte Bitfolge `53514C69746520666F726D6174203300` als auch ihre Position in der Datei vermerkt: "Position type Absolute from BOF, Offset 0", das heißt ausgehend vom Dateianfang (BOF, beginning of file) beginnt die Bitfolge nach 0 Bytes (offset), also direkt am Anfang.

Solche Angaben in einer Dateiformatspezifikation liefern im Grunde schon alle wesentlichen Informationen, die für die Formaterkennung benötigt werden, und können unmittelbar in eine Signatur für PRONOM "übersetzt" werden. Allerdings liest sich nicht jede Formatspezifikation so leicht wie die gerade gezeigte, und nicht jede Spezifikation liefert direkt die für PRONOM benötigte Hexadezimaldarstellung mit. Einige Beispiele dieser Art mit unterschiedlichem Schwierigkeitsgrad finden sich in den Formatspezifikationen von PDF, TIFF und XML.

Die Formatspezifikationen von PDF 1.4, beschreibt in den Abschnitten "File Header" und "File Trailer" genau die Muster `%PDF-1.4` und `%%EOF`, die die Basis für den entsprechenden PRONOM-Eintrag bilden. Diese Muster sind in der Spezifikation allerdings als Text dargestellt, und nicht als Bitmuster in Hexadezimaldarstellung, wie sie von PRONOM benötigt werden. Um daraus eine Signatur abzuleiten, muss deshalb der ASCII-Text bzw. die diesem zugrundeliegenden Bits in Hexadezimaldarstellung überführt werden. Dafür gibt es etliche Hilfsmittel, angefangen von ASCII-Tabellen, die eine manuelle Übertragung ermöglichen, bis hin zu Online-Konvertern wie http://www.asciitohex.com/. Allerdings ist die Hexadezimaldarstellung keine zwingende Voraussetzung, um dem PRONOM-Team Informationen über ein Dateiformat mitzuteilen! Schon der Hinweis auf eine solche Spezifikation ist hilfreich, selbst wenn man nicht selbst in der Lage ist, daraus eine vollständige und technisch einwandfreie Signatur abzuleiten.

Auch in der Spezifikation von XML 1.0 findet sich implizit das Muster, das den entsprechenden PRONOM-Eintrag stützt. Allerdings ist dieses Muster hier deutlich schwieriger zu finden. Es besteht letztendlich aus dem Text `<?xml version=("|')1.0("|')` (die geklammerten Passagen bezeichnen Alternativen, d.h. es können entweder doppelte oder einfache Anführungszeichen auftreten). Dieser Text steht aber nicht so in der Spezifikation, sondern er ergibt sich aus formalen Grammatiken in den Abschnitten "Well-Formed XML Documents" und "Prolog and Document Type Declaration". Diese Grammatiken sind zwar nach einer gewissen Eingewöhnung auch für Nicht-Informatiker verständlich, sehen beim ersten Kontakt aber recht seltsam aus.

Die Spezifikation von TIFF 6.0 schließlich beschreibt im Abschnitt "Image File Header" die Muster, die die Basis für den PRONOM-Eintrag bilden, und das sogar in Hexadezimaldarstellung. Allerdings sind die Angaben in der Spezifikation nur mit etwas mehr Hintergrundwissen über Binärzahlen, insbesondere den Unterschied zwischen "Little Endian" und "Big Endian" zu verstehen.

Dateiformatspezifikationen sind natürlich technische Dokumente und können ohne Hintergrundwissen aus der Informatik oder aus dem fachlichen Entstehungskontext des Dateiformats schwer verständlich sein. Manchmal erlauben sie aber auch eine überraschend flüssige Lektüre, wie im oben gezeigten Beispiel. In jedem Fall lohnt sich, wenn man sich mit der Erkennung eines Dateiformats beschäftigt, immer der Blick in solche Spezifikationen.

Bitmuster in einer Dateisammlung finden

Wenn eine Signatur für ein Dateiformat erstellt werden soll, das keiner formalen Spezifikation folgt, oder wenn diese Spezifikation nicht verfügbar ist, dann muss die Suche nach einem geeigneten Bitmuster anders angegangen werden. An die Stelle des Lesens der Dateiformatspezifikation tritt dann die Analyse einer möglichst großen Menge von Beispieldateien aus möglichst vielen unterschiedlichen Quellen. Dabei wird nach einem Bitmuster gesucht, das zuverlässig in allen Dateien auftritt.

Zur ersten Orientierung empfiehlt sich bei dieser Arbeit immer ein Blick auf einige Dateien im Hex-Editor; möglicherweise zeigt sich dabei bereits ein auffälliges Bitmuster. Interessante Angaben, beispielsweise Formatnamen oder Versionsnummern, stehen oft (aber nicht immer) relativ weit am Anfang einer Datei. Manchmal sind sie sogar als ASCII-Text wie `SQLite format 3` oder `%PDF-1.4` in den Bitstrom von Binärdateien eingebettet und deshalb leicht in der Textdarstellung (in der rechten Spalte) eines Hex-Editors zu erkennen. Häufig sind die entscheidenden Muster aber auch weniger auffällig, haben keine schöne textuelle Entsprechung und machen sich daher erst bemerkbar, wenn man sie in vielen Dateien immer wieder an derselben Stelle gesehen hat.

Für die systematische Suche nach einem noch unbekannten Bitmuster bietet sich deshalb der Einsatz hilfreicher Tools an. Neben Unix-Klassikern wie `strings` und `file` muss hier ganz besonders das Programm TrIDScan erwähnt werden, denn es tut genau das, was mit bloßem Auge im Hex-Editor schwer fällt: gemeinsame Muster in einer Menge von Dateien finden. Die Benutzung des Tools wird auf seiner Website erläutert. In Kurzform: Es setzt Python voraus und wird dann folgendermaßen auf der Kommandozeile aufgerufen, um beispielsweise alle PDF-Dateien in einem Verzeichnis namens `dateien` zu analysieren:

```
$ python tridscan.py dateien/*.pdf
```

Damit wird eine neue Datei namens `newtype.trid.xml` erzeugt, die die Ergebnisse der Analyse enthält. TrIDScan ist eigentlich für die Erstellung von Signaturen für ein anderes Formaterkennungstool, TrID gedacht (dessen Autor um Zusendung interessanter Resultate bittet) und erzeugt deshalb eine etwas gewöhnungsbedürftige Ausgabe im XML-Format, aus der man sich die relevanten Informationen heraussuchen muss. Im Beispiel der PDF-Dateien könnten sich u.a. folgende Zeilen in der XML-Datei finden:

```
<Pattern>
    <Bytes>255044462D312E340D</Bytes>
    <ASCII> % P D F - 1 . 4</ASCII>
    <Pos>0</Pos>
</Pattern>
<Pattern>
    <Bytes>74</Bytes>
    <ASCII> t</ASCII>
    <Pos>68</Pos>
</Pattern>
```

Das bedeutet, das sich in allen untersuchten Dateien an Position 0 (also direkt am Dateianfang) die Bitfolge `255044462D312E340D` findet, außerdem an Position 68 die Bitfolge `74`. Die erste Bitfolge steht im Wesentlichen so auch im PRONOM-Eintrag zu PDF 1.4. Die zweite hingegen dürfte, weil sie so kurz ist, kaum ein Alleinstellungsmerkmal von PDF-Dateien sein, sondern auch in anderen Formaten vorkommen. Bevor sie für die Erstellung einer Signatur herangezogen wird, sollte daher geprüft werden, welche Funktion dieses Bitmuster in PDF-Dateien haben könnte.

Das letzte Beispiel verdeutlicht einen wichtigen Aspekt: Egal, wie man ein Bitmuster gefunden hat, sollte man sich über seine ungefähre Bedeutung im Klaren sein, bevor man es zur Grundlage einer Signatur macht. Denn stammen beispielsweise alle Dateien in der untersuchten Sammlung vom selben Autor oder wurden auf dem selben Computer erzeugt, könnte ein Bitmuster, das in allen Dateien auftritt, womöglich nicht etwa das Dateiformat, sondern bloß den Autor oder den Computernamen bezeichnen! Auch deshalb ist es wichtig, eine Dateisammlung möglichst heterogen aufzubauen und aus unterschiedlichen Quellen zusammenzutragen. Zusätzliche Beispieldateien kann übrigens auch eine Google-Suche nach der Dateiendung, beispielsweise nach `filetype:pdf` zu Tage fördern.

Exkurs: Beschreibung komplexer Bitmuster

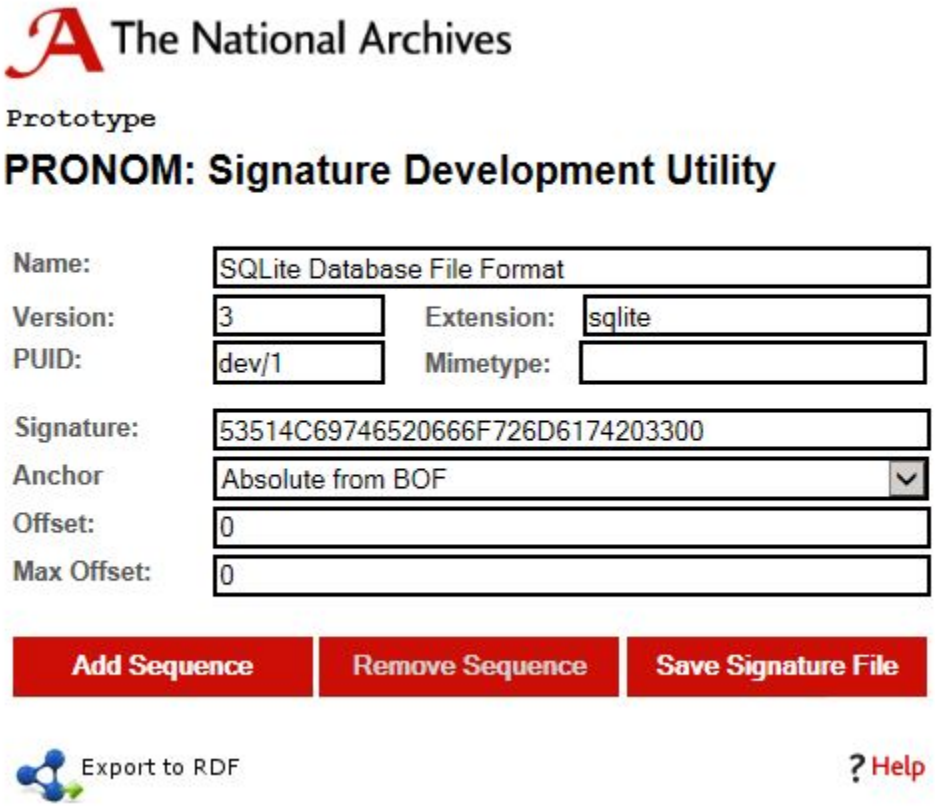
(Dieser Abschnitt ist nicht notwendig für das Verständnis der restlichen Anleitung.)

Manchmal können Bitmuster in unterschiedlichen Varianten auftreten oder eine wechselnde Länge haben. Beispielsweise könnten in einem Muster an derselben Position die Bitfolgen `312E31` und `312E32` beide gleichermaßen akzeptabel sein, oder eine Bitfolge könnte mit `4C4156` anfangen und mit `4E5257` enden, dazwischen aber vier beliebige weitere Bytes umfassen. Um solche Varianten kompakt zu beschreiben, werden in PRONOM spezielle Abkürzungen verwendet, sogenannte Reguläre Ausdrücke. Dabei kommen Sonderzeichen zum Einsatz, die jeweils eine festgelegte Bedeutung haben. Die oben skizzierten Beispiele würden damit als `312E(31|32)` und `4C4156{4}4E5257` beschrieben. Folgende Ausdrücke stehen zur Verfügung:

`??`	Zwei beliebige Hexadezimalziffern, also ein Byte. Das Muster passt u.a. auf `FF`, `A0` oder `31`.
`{n}`	Eine Folge von genau n beliebigen Bytes. Das Muster `{2}` passt u.a. auf `3132` oder `00FF`.
`{m-n}`	Eine Folge von m bis n beliebigen Bytes.
`(a|b)`	Entweder die Bitfolge a oder die Bitfolge b. Das Muster `(31|32)` passt auf `31` und `32`.
`[a:b]`	Eine Bitfolge im Bereich von a bis b. Das Muster `[31:34]` passt auf `31`, `32`, `33` und `34`.
`[!a]`	Eine Bitfolge, die *nicht* a ist. Das Muster `[!FF]` passt u.a. auf `F3` und `4B`, aber nicht auf `FF`.
`[!a:b]`	Eine Bitfolge, die *nicht* im Bereich von a bis b liegt. Das Muster `[!31:33]` passt u.a. auf `30` und `34`, aber nicht auf `31`, `32` und `33`.
`*`	Eine beliebig lange Folge von Bytes (null bis unendlich). Das Muster passt u.a. auf `31` oder `FF314A57`. Achtung: Wird dieses Muster in einer Signatur verwendet, kann das negativen Einfluss auf die Performance von DROID haben, weil dann eine potentiell sehr große Datei im Allgemeinen vollständig durchsucht werden muss!

Signature File erstellen

Eine DROID Signature File, also eine XML-Datei, die Informationen über Formatsignaturen aus PRONOM in einer für DROID verständlichen Form enthält, macht beim ersten Blick hinein einen recht abschreckenden Eindruck. Tatsächlich ist es aber dank eines kleinen Online-Tools namens PRONOM: Signature Development Utility sehr einfach, eine solche Datei selbst zu erstellen. Es nimmt einige Angaben zu der gewünschten Formatsignatur in einem Formular entgegen und erstellt dann eine neue Signature File. Dabei muss kein XML von Hand geschrieben werden.



Angaben wie Name, Version, Extension und Mimetype dürfen sinnvoll ausgefüllt werden, müssen für erste Tests aber noch nicht perfekt sein und können auch leer bleiben, wenn es z.B. für ein Format gar keinen Mimetype gibt. Die PUID ist ein bloßer Platzhalter, bis die Signatur bei PRONOM eingereicht und dort eine offizielle PUID vergeben wird. Interessanter und entscheidend für die Formaterkennung auf Basis der neuen Signature File sind die Felder Signature, Anchor, Offset und Max Offset.

In das Feld Signature wird das charakteristische Bitmuster eingetragen, das zur Identifizierung des Dateiformats dient. Im Beispiel von SQLite 3 wäre das die oben beschriebene Bitfolge `53514C69746520666F726D6174203300`.

Die Felder Anchor, Offset und Max Offset beschreiben gemeinsam die Position des im Signature-Feld eingetragenen Bitmusters in der Datei. Im einfachsten Fall steht das Bitmuster immer direkt am Dateianfang, dann lauten die Einträge Anchor "Absolute from BOF" (beginning of file), Offset 0 und Max Offset 0. Beginnt das Muster nach genau 42 Bytes ausgehend vom Dateianfang, wird der Offset stattdessen zu 42. Beginnt es nach 42, vielleicht aber auch erst nach 43, 44 oder 52 Bytes, muss zusätzlich der Max Offset auf den Wert 52 gesetzt werden. (Achtung, die Zählung der Bytes in einer Datei beginnt üblicherweise bei 0, nicht bei 1!) Alternativ zum Dateianfang kann das Muster mit Anchor "Absolute from EOF" (end of file) ausgehend vom Dateiende oder mit Anchor "Variable" an einer beliebigen Position in der Datei lokalisiert werden. Letzteres sollte allerdings, sofern das möglich ist, vermieden werden, weil es die Geschwindigkeit von DROID negativ beeinflussen kann.

Bei Bedarf können mit dem Button "Add Sequence" weitere Bitmuster hinzugefügt werden.

Mit Klick auf den Button "Save Signature File" wird eine Signature File im XML-Format zum Download angeboten, die anschließend zum Testen der neuen Signatur in DROID geladen werden kann.

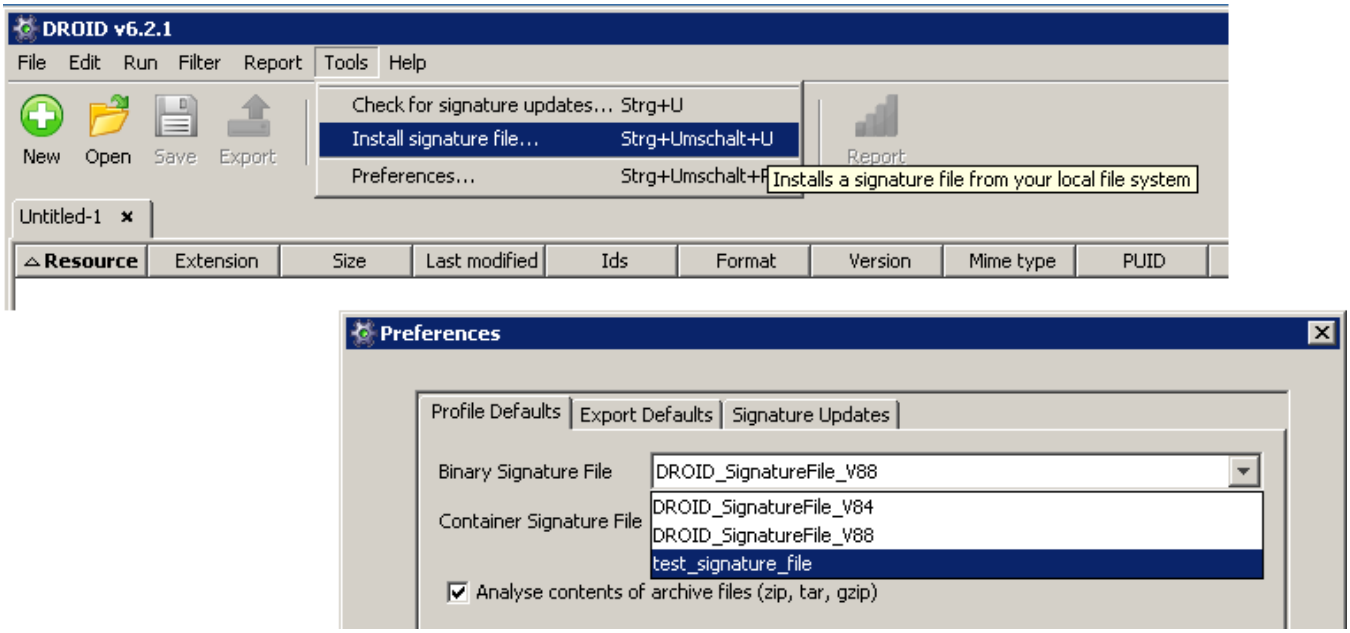
Mit DROID testen

Um zu gewährleisten, dass DROID mit der neuen Signatur korrekt arbeitet, sollte sie ausführlich getestet werden. Angenommen, die neue Signatur beschreibt das SQLite 3-Format. Dann müssen zwei Bedingungen überprüft werden:

- Identifiziert DROID alle SQLite 3-Dateien korrekt als SQLite 3?
- Identifiziert DROID keine Dateien in anderen Formaten fälschlicherweise als SQLite 3?

Beide Fragen sind gleichermaßen wichtig und sollten durch die Analyse einer möglichst großen Zahl unterschiedlicher Dateien, sowohl im Format SQLite 3 als auch in verschiedensten anderen Formaten, beantwortet werden.

DROID muss zunächst mit der neuen Signature File bekannt gemacht werden. Dafür sind zwei Schritte erforderlich: Zuerst wird über den Menüpunkt "Tools > Install signature file ..." die im letzten Abschnitt erstellte XML-Datei in DROID ergänzt; und danach wird diese Datei im Dialog "Preferences" (zu öffnen über den Menüpunkt "Tools > Preferences") als "Binary Signature File" ausgewählt. Anschließend kann die Analyse der Testdateien starten, um die Qualität der neuen Signatur sicherzustellen.



Noch einfacher geht es, wenn DROID auf der Kommandozeile benutzt wird. Hier kann eine Analyse auf Basis der neuen Signature File direkt gestartet werden:

```
$ droid -R -Ns test_signature_file.xml -Nr testfiles
```

Bei PRONOM einreichen

Wenn sich die neu erstellte Signatur als nützlich erwiesen hat, dann sollte sie veröffentlicht werden. Der beste und einfachste Weg dafür ist es, die ermittelten Signaturinformationen dem Betreiber der PRONOM-Datenbank, den britischen National Archives mitzuteilen. Dadurch wird die Signatur nach einer internen Prüfung durch die National Archives erstens online in PRONOM recherchierbar, sie wird zweitens über die dort vergebene PUID eindeutig und permanent referenzierbar und sie wird drittens über die offiziellen DROID-Signature Files automatisch in DROID verwendet, ohne dass eine eigene Signature File manuell eingebunden werden müsste.

Für die Mitteilung von Informationen über Dateiformate stellen die National Archives ein Webformular bereit. Dort werden übrigens auch unvollständige Angaben dankbar entgegengenommen! Die selbständige Erstellung einer Signature File wird nicht vorausgesetzt (ist aber natürlich hilfreich für eigene Tests der Signatur), und auch eine perfekt formulierte Bitfolge ist nicht zwingend notwendig. Jede Information über ein Dateiformat, auch bloße Hinweise z. B. auf Spezifikationen, die man vielleicht selbst nicht restlos versteht, sind hilfreich für die weitere Recherche und können so letzten Endes zu einer neuen Signatur in PRONOMs Datenbestand führen.

Anhang: Empfehlenswerte Hex-Editoren

Windows:

- HxD (GUI, auch als portable Version erhältlich, die keine Installation erfordert)
- Frhed (GUI, auch als portable Version erhältlich, die keine Installation erfordert)

Linux:

- ghex (GUI)
- xxd (Kommandozeile)
- hexer (Kommandozeile)

Anhang: Literatur

- Adrian Brown, The National Archives: "Digital Preservation Technical Paper 1: Automatic Format Identification Using PRONOM and DROID" Tech nische Spezifikation des in PRONOM verwendeten Signaturschemas, der DROID-signature file und Beschreibung der in DROID implementierten Algorithmen zur Formatidentifizierung. Nicht unbedingt nötig für das Erstellen eigener Signaturen und schon gar nicht für die Benutzung von PRONOM/DROID. Anspruchsvoll, aber interessant fürs tiefere Verständnis.
- Adrian Brown, The National Archives: "Digital Preservation Technical Paper 2: The PRONOM Unique Identifier Scheme. A scheme of persistent unique identifiers for representation information" Technische Spezifikation der in PRONOM verwendeten PUIDs. Kurzform: PUIDs für Dateiformate haben das Schema "fmt/<Nummer>" oder (inzwischen historisch) "x-fmt/<Nummer>".

- NN, The National Archives: "How to research and develop signatures for file format identification" High Level-Überblick über den Prozess der Entwicklung von Formatsignaturen. Gut zur Orientierung und um die Zusammenhänge zu verstehen. Liefert allerdings nicht genug Details, um als alleinige Quelle für den Einstieg zu dienen.
- Jay Gattuso, National Library of New Zealand: "How to write a new signature file for DROID" Erste Hälfte Erläuterung der DROID-signature file und Einführung in das Arbeiten mit einem Hex-Editor. Zweite Hälfte Beschreibung der Schritte zur Erstellung einer signature file für DROID. Leicht verständliches, stellenweise etwas ausführliches Tutorial.