How to write a new signature file for DROID

A guide by NLNZ

www.natlib.govt.nz

Te Puna Mātauranga o Aotearoa
NATIONAL LIBRARY
OF NEW ZEALAND

Document Control

Revision history

Revision	Date	Author	Reason for Change
V1	25.1.12	J Gattuso	DRAFT
V1.1	09.02.12	J Gattuso	RELEASE

Table of Contents

1. How to write a new signature file for DROID

This guide describes the method used by NLNZ to create new file format signatures to submit to PRONOM for inclusion in the PRONOM format register.

To be able to follow this guide, you will need to have access to a working version of the TNA tool DROID[1], some files to play with, a hex viewer (explained in detail below), and XML viewer/editor (explained in detail below) and an hour or so to work through the examples.

Notes:

a) In some cases, new signatures are a best guess based on limited access to example files. This means they might not be perfect, and cause either false positives (matches against files that are not of the format being described) or false negatives (files of the format being described not being matched by the signature). This is unfortunately unavoidable when small sets of example files are being assessed, although this paper describes the steps taken to try and mitigate this problem.

b) There is often little or no detail about the ownership, technical notes or other supporting information for old formats. Assumptions are made that older formats are either abandoned, or have no restrictions on the sharing of specific format patterns.

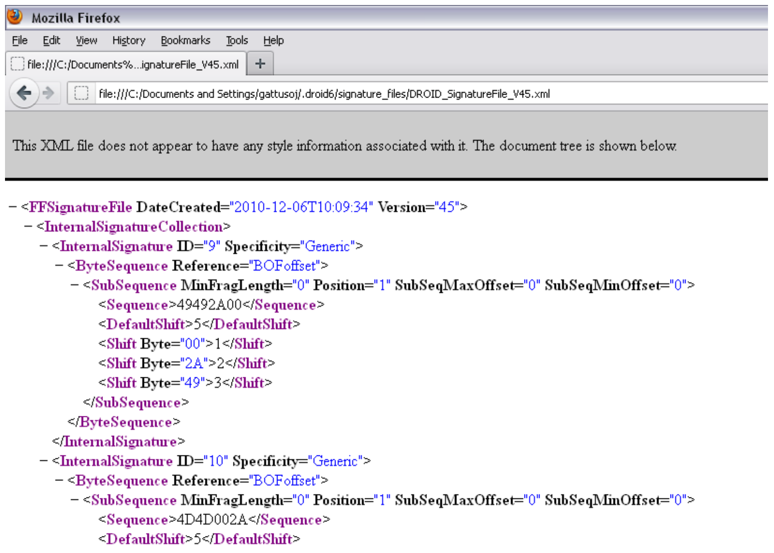
1.1. Understanding the DROID signature file structure

Before we step into the specifics of how to create a signature for a new format, it’s worth exploring the tools in hand and becoming familiar with the types of information we are going to use.
Firstly, take a look at a DROID signature file. I am currently running DROID v6, and find the format signatures in the following location on my Windows XP machine:

```
C:\Documents and Settings\MY_USER_NAME\.droid6\signature_files.
```

This is where DROID deposits its signature files once it has downloaded them on my machine, yours may differ, and you may need to hunt around a little bit. You should see at least one XML file with the naming convention ‘DROID_SignatureFile_Vn.xml’ (where ‘n’ is the version number of the signature file). Open one of these files in a suitable XML viewer – if you don’t have specific XML editor/viewer, most modern browsers support structured views of XML file – try dragging the file into a new browser tab.

[1] http://sourceforge.net/projects/droid/


Mozilla Firefox
File Edit View History Bookmarks Tools Help
file:///C:/Documents%...ignatureFile_V45.xml
file:///C:/Documents and Settings/gattusoj/.droid6/signature_files/DROID_SignatureFile_V45.xml

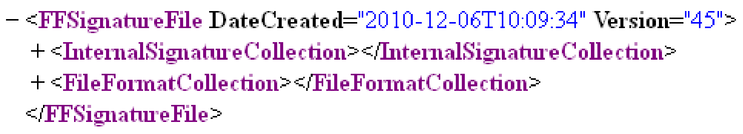
This XML file does not appear to have any style information associated with it. The document tree is shown below.

```
- <FFSignatureFile DateCreated="2010-12-06T10:09:34" Version="45">
  - <InternalSignatureCollection>
    - <InternalSignature ID="9" Specificity="Generic">
      - <ByteSequence Reference="BOFoffset">
        - <SubSequence MinFragLength="0" Position="1" SubSeqMaxOffset="0" SubSeqMinOffset="0">
            <Sequence>49492A00</Sequence>
            <DefaultShift>5</DefaultShift>
            <Shift Byte="00">1</Shift>
            <Shift Byte="2A">2</Shift>
            <Shift Byte="49">3</Shift>
          </SubSequence>
        </ByteSequence>
      </InternalSignature>
    - <InternalSignature ID="10" Specificity="Generic">
      - <ByteSequence Reference="BOFoffset">
        - <SubSequence MinFragLength="0" Position="1" SubSeqMaxOffset="0" SubSeqMinOffset="0">
            <Sequence>4D4D002A</Sequence>
            <DefaultShift>5</DefaultShift>
```

Figure 1: XML Viewed in a browser

I won't go into a full description of the signature file, but it's worth having a look around to find the important bits

The DROID signature-file has two main parts; the signature collection, and the fileformat collection.



```
- <FFSignatureFile DateCreated="2010-12-06T10:09:34" Version="45">
  + <InternalSignatureCollection></InternalSignatureCollection>
  + <FileFormatCollection></FileFormatCollection>
  </FFSignatureFile>
```

Figure 2: Two main sections of a DROID XML signature file

Part one is a list of signature 'patterns', this is in the first main XML block (<InternalSignatureCollection>). This section contains lists of various byte patterns found inside files.

The second part is the section that links a signature with a PUID record held in the PRONOM dataset. This section, <FileFormatCollection> links up with the above section, and includes file extension statements.

Have a look at a single format you recognise, starting in the <FileFormat> area (lower half of the XML file), and see how a single file format is declared with an ID, a name, a PUID, and a version, then there is an extension element (that is used to test the extension_mismatch condition), finally it references any internal signatures <InternalSignatureID> that relates to the format. If you use a find function to search for the relevant <InternalSignature> you'll see how it all fits together.
To understand more about how the signature is constructed TNA have written a very useful guide called 'Automatic Format Identification Using PRONOM and DROID'[2]

I highly recommend reading through – it can look quite complex, but once you get the hang of it, it will make sense.
It's also worth noting that you can see the specific signature details in PRONOM record by looking on the PRONOM website at your format of interest. This is a really handy reference for checking existing signatures, or when you are creating new ones.

[2] http://www.nationalarchives.gov.uk/aboutapps/fileformat/pdf/automatic_format_identification.pdf

1.2. HEX viewers

DROID essentially works by looking for a specific pattern, or set of patterns inside the binary representation of file. You don't need to read the binary digits (the noughts and ones) but you will need to look at the hexadecimal (hex) values that represent the binary content.

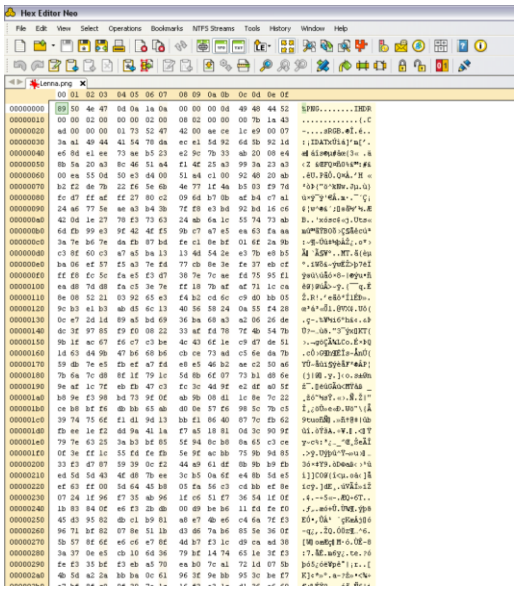
This sounds more complex that it actually is, and hopefully by the time you've read this section you'll have a good understanding of how this works.

To find these 'mythical' patterns, we need to open our files in a specific way.

We don't want to use the native applications for our files – this will give us the intended rendered view of the object. The view we want is the binary contents of the file. To do this, you'll need a HEX viewer. There are a number of freely available HEX viewers you can download.

Once you have installed a HEX viewer, simple open any file via the HEX viewer. This should give you a view that at first might be a little overwhelming, but once again, with a little practice you'll soon figure out what you are looking at. From the HEX view of an object you can start to see the important parts.[3]

Figure 3: Rendered view of lenna.png test file

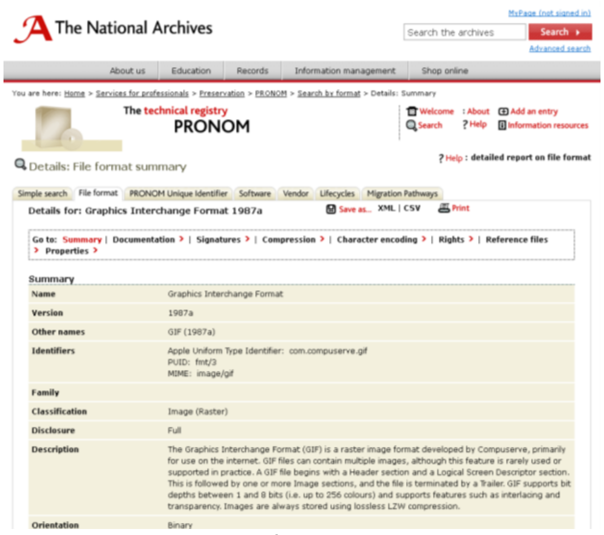


```
          00 01 02 03  04 05 06 07  08 09 0a 0b  0c 0d 0e 0f
00000000  89 50 4e 47  0d 0a 1a 0a  00 00 00 0d  49 48 44 52
00000010  00 00 02 00  00 00 02 00  08 02 00 00  00 7b 1a 43
00000020  ad 00 00 00  01 73 52 47  42 00 ae ce  1c e9 00 07
00000030  3a a1 49 44  41 54 78 da  ec e1 5d 92  6d 5b 92 1d
00000040  e6 8d e1 ee  73 ae b5 23  e2 9c 7b 33  ab 20 08 e4
00000050  8b 5a 20 a3  8c 46 51 a4  f1 4f 25 a3  99 3a 23 a3
00000060  00 ea 55 0d  50 e3 d4 00  51 a4 c1 00  92 48 20 ab
00000070  b2 f2 de 7b  22 f6 5e 6b  4e 77 1f 4a  b5 03 f9 7d
00000080  fc d7 ff af  ff 27 80 c2  09 6d b7 0b  af b4 c7 a1
00000090  24 a6 77 5e  ae a3 b4 3b  7f f8 e3 bd  92 bd 16 c6
000000a0  42 0d 1e 27  78 f3 73 63  24 ab 6a 1c  55 74 73 ab
000000b0  6d fb 99 e3  9f 42 4f f5  9b c7 a7 e5  ea 63 fa aa
000000c0  3a 7e b6 7e  da fb 87 bd  fe c1 8e bf  01 6f 2a 9b
000000d0  c3 8f 60 c3  a7 a5 ba 13  13 4d 54 2e  e3 7b e8 b5
000000e0  ba 06 ef 57  f5 a3 7e fd  77 cb 8e 3e  fe 37 eb cf
000000f0  ff f8 fc 5c  fa e5 f3 d7  38 7e 7c ae  fd 75 95 f1
00000100  ea d8 7d d8  fa c5 3e 7e  ff 18 7b af  af 71 1c ca
00000110  8e 08 52 21  03 92 65 e3  f4 b2 cd 6c  c9 d0 bb 05
00000120  9c b3 e1 b3  ab d5 6c 13  40 56 58 24  0a 55 f4 28
00000130  0c e7 2d 1d  89 a5 bd 69  36 ba 68 a3  a2 06 26 de
00000140  dc 3f 97 85  f9 f0 08 22  33 af fd 78  7f 4b 54 7b
00000150  9b 1f ac 67  f6 c7 c3 be  4c 43 6f 1e  c9 d7 de 51
00000160  1d 63 d4 9b  47 b6 68 b6  cb ce 73 ad  c5 6e da 7b
00000170  59 db 7e e5  fb ef a7 fd  e8 e5 46 b2  ae c2 50 a6
00000180  7b 6a 7c d8  8f 1f 79 1c  5d 8b 6f 07  73 b1 d8 6e
00000190  9e af 1c 7f  eb fb 47 c3  fc 3c 4d 9f  e2 df a0 5f
000001a0  b8 9e f3 98  bd 73 9f 0f  ab 9b 08 d1  1c 8e 7c 22
000001b0  ce b8 bf f6  db bb 65 ab  d0 0e 57 f6  98 5c 7b c5
000001c0  39 74 75 6f  f1 d1 9d 13  bb f1 86 40  87 7c fb 62
000001d0  fb ee 1e f2  dd 9a 41 1a  f7 a5 18 81  0d 3c 90 9f
000001e0  79 7e 63 25  3a b3 bf 85  5f 94 8c b8  8a 65 c3 ce
000001f0  0f 3e ff 1c  55 fd fe fb  5e 9f ac bb  75 9b 9d 85
00000200  33 f3 d7 87  59 39 0c f2  44 a9 61 df  8b 9b b9 fb
00000210  ed 5d 5d 43  4f d8 7b ee  3c b5 0a 6f  e4 8b 5d e5
00000220  ef 63 ff 00  5d 64 45 b8  05 fa 56 c3  cd bb ef 8e
00000230  07 24 1f 96  f7 35 ab 96  1f c6 51 f7  36 54 1f 0f
00000240  1b 83 84 0f  e6 f3 2b db  00 d9 be b6  11 fd fe f0
00000250  45 d3 95 82  db c1 b9 81  a8 e7 4b e6  c4 6a 7f f3
00000260  96 71 bf 82  07 8e 51 1b  d3 d6 7a b6  85 5e 36 0f
00000270  5b 57 8f 6f  e6 c6 e7 8f  4d b7 f3 1c  d9 ca ad 38
00000280  3a 37 0e e5  cb 10 6d 36  79 bf 14 74  65 1e 3f f3
00000290  fe f3 35 bf  f3 eb a5 70  ea b0 7c a1  72 1d 07 5b
000002a0  4b 5d a2 2a  bb ba 0c 61  96 3f 9e bb  95 3c be f7
```

Figure 4: Binary view of lenna.png test file

[3] http://upload.wikimedia.org/wikipedia/en/2/24/Lenna.png

Let's work through an example, starting with the PRONOM reference page, let's take a look in detail at fmt/3[4]


The National Archives

MyPage (not signed in) | Search the archives | Search | Advanced search

About us | Education | Records | Information management | Shop online

You are here: Home > Services for professionals > Preservation > PRONOM > Search by format > Details: Summary

The technical registry PRONOM

Welcome | About | Add an entry | Search | Help | Information resources

Details: File format summary

Help : detailed report on file format

Simple search | File format | PRONOM Unique Identifier | Software | Vendor | Lifecycles | Migration Pathways

Details for: Graphics Interchange Format 1987a — Save as... XML | CSV — Print

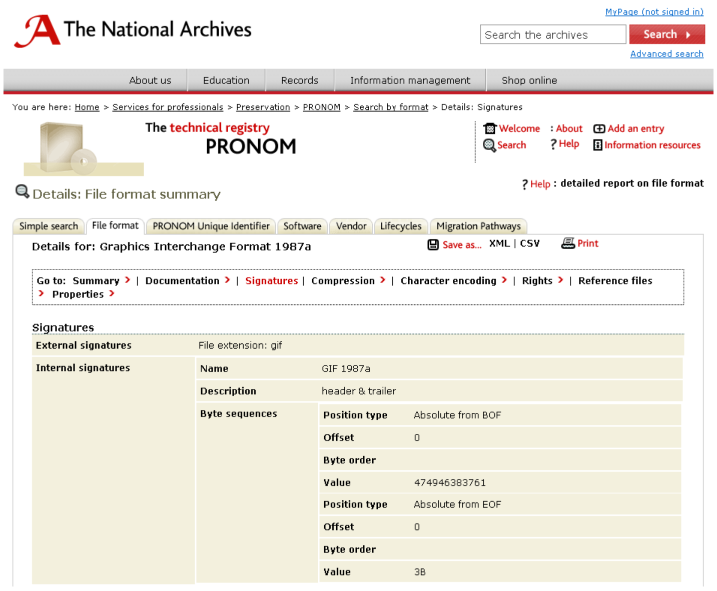
Go to: Summary | Documentation | Signatures | Compression | Character encoding | Rights | Reference files | Properties

Summary

Name	Graphics Interchange Format
Version	1987a
Other names	GIF (1987a)
Identifiers	Apple Uniform Type Identifier: com.compuserve.gif PUID: fmt/3 MIME: image/gif
Family	
Classification	Image (Raster)
Disclosure	Full
Description	The Graphics Interchange Format (GIF) is a raster image format developed by Compuserve, primarily for use on the internet. GIF files can contain multiple images, although this feature is rarely used or supported in practice. A GIF file begins with a Header section and a Logical Screen Descriptor section. This is followed by one or more Image sections, and the file is terminated by a Trailer. GIF supports bit depths between 1 and 8 bits (i.e. up to 256 colours) and supports features such as interlacing and transparency. Images are always stored using lossless LZW compression.
Orientation	Binary

Figure 5: PRONOM FMT/3 Page

Our first port of call will be the signature tab[5]:


The National Archives

MyPage (not signed in) | Search the archives | Search | Advanced search

About us | Education | Records | Information management | Shop online

You are here: Home > Services for professionals > Preservation > PRONOM > Search by format > Details: Signatures

The technical registry PRONOM

Welcome | About | Add an entry | Search | Help | Information resources

Details: File format summary

Help : detailed report on file format

Simple search | File format | PRONOM Unique Identifier | Software | Vendor | Lifecycles | Migration Pathways

Details for: Graphics Interchange Format 1987a — Save as... XML | CSV — Print

Go to: Summary | Documentation | Signatures | Compression | Character encoding | Rights | Reference files | Properties

Signatures

External signatures	File extension: gif		
Internal signatures	Name	GIF 1987a	
	Description	header & trailer	
	Byte sequences	Position type	Absolute from BOF
		Offset	0
		Byte order	
		Value	474946383761
		Position type	Absolute from EOF
		Offset	0
		Byte order	
		Value	3B

Figure 6: PRONOM FMT/3 Signature page

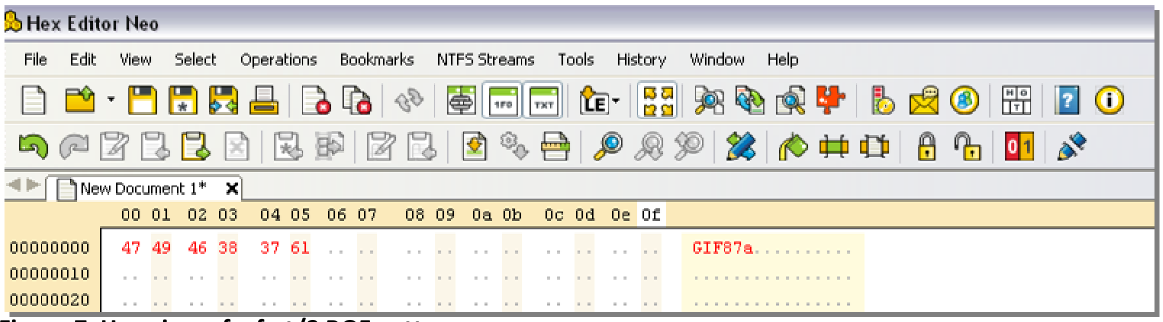
You can see there are two patterns in the signature. One at the Beginning Of the File (BOF) and one at the End Of the File (EOF).

Let's take a closer look at the BOF pattern. The PRONOM page tells us there is a 'value' of '474946383761' that has a '0 offset absolute from BOF'. In layman's terms, this means that the first set of bytes in the file, starting at the very beginning of the file have the above pattern.

The thing to note is that the pattern is actually a hexadecimal representation of what turns out to be readable text. To demonstrate this, open your HEX viewer, and enter the pattern as above, and see what happens in the text view of that pattern:

[4] http://www.nationalarchives.gov.uk/pronom/fmt/3

[5] http://www.nationalarchives.gov.uk/PRONOM/Format/proFormatSearch.aspx?status=detailReport&id=619&strPageToDisplay=signatures



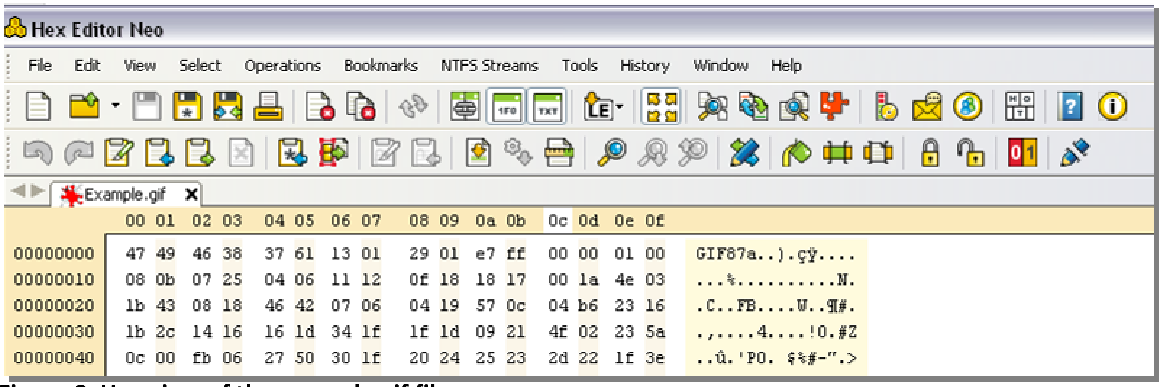
```
          00 01  02 03  04 05  06 07   08 09  0a 0b  0c 0d  0e 0f
00000000  47 49  46 38  37 61  .. ..   .. ..  .. ..  .. ..  .. ..   GIF87a..........
00000010  .. ..  .. ..  .. ..  .. ..   .. ..  .. ..  .. ..  .. ..   ................
00000020  .. ..  .. ..  .. ..  .. ..   .. ..  .. ..  .. ..  .. ..   ................
```

Figure 7: Hex view of a fmt/3 BOF pattern

As you can see from the screenshot, this pattern is actually something you can read and make sense of because in this case it's actually just the hex view of some text.

Repeat this process for the EOF pattern ('3b'). You'll notice that in the text view, this is not a character but what appears to be a punctuation mark. It's not really a punctuation mark, it's a reserved byte value that when encountered by a gif viewer tells the viewer that the end of the image has been reached, it just 'looks' like a punctuation mark when it's decoded as ASCII text.

Let's have a look at this in practise. Download the example.gif image from Wikipedia[6] and open it in your HEX viewer:



```
          00 01  02 03  04 05  06 07   08 09  0a 0b  0c 0d  0e 0f
00000000  47 49  46 38  37 61  13 01   29 01  e7 ff  00 00  01 00   GIF87a..).çÿ....
00000010  08 0b  07 25  04 06  11 12   0f 18  18 17  00 1a  4e 03   ...%..........N.
00000020  1b 43  08 18  46 42  07 06   04 19  57 0c  04 b6  23 16   .C..FB....W..¶#.
00000030  1b 2c  14 16  16 1d  34 1f   1f 1d  09 21  4f 02  23 5a   .,....4....!O.#Z
00000040  0c 00  fb 06  27 50  30 1f   20 24  25 23  2d 22  1f 3e   ..û.'P0. $%#-".>
```

Figure 8: Hex view of the example.gif file

While you have the file open, take at look for the EOF marker and confirm it is what you expect, and where you expect it to be.

Let's now take a look at the DROID signature file to confirm these patterns exist in the format we expect.

I'm using signature file 'DROID_SignatureFile_V55.xml' which I took from the C:\Documents and Settings\MY_USER_NAME\.droid6\signature_files location on my PC. It doesn't really matter which signature version you use, they will all have (mostly) the same details.

I opened the file in my browser by dragging it to a new tab.

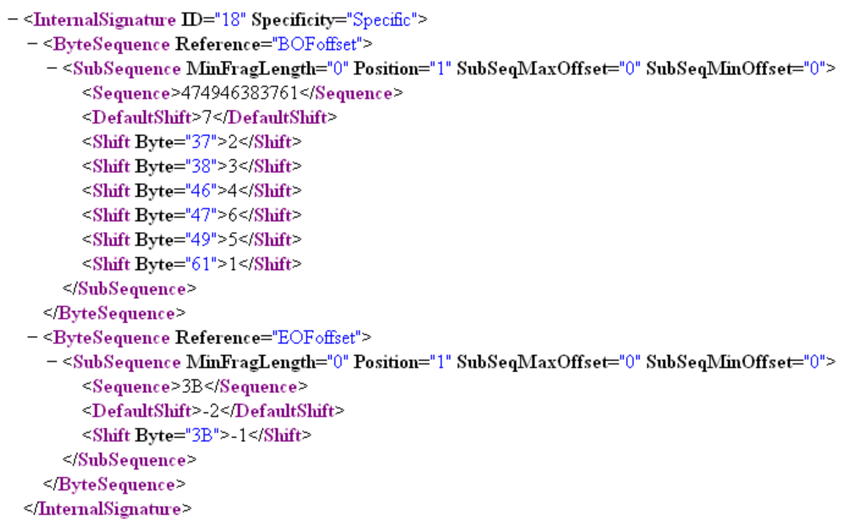
Let's search for our PUID of interest. Search for 'PUID="fmt/3"' in the XML document:

```
- <FileFormat ID="619" MIMEType="image/gif" Name="Graphics Interchange Format" PUID="fmt/3" Version="1987a">
    <InternalSignatureID>18</InternalSignatureID>
    <Extension>gif</Extension>
  </FileFormat>
```

Figure 9:XML for fmt/3 FileFormat ID

[6] http://upload.wikimedia.org/wikipedia/en/8/8f/Example.gif

This is the XML snippet that relates to our format of interest. This tells us that there the gif 1987a format has an internal signature, and an extension of '.gif'. Let's search for the relating internal signature in the same XML file using the following search term: <InternalSignature ID="18"



```
- <InternalSignature ID="18" Specificity="Specific">
  - <ByteSequence Reference="BOFoffset">
    - <SubSequence MinFragLength="0" Position="1" SubSeqMaxOffset="0" SubSeqMinOffset="0">
        <Sequence>474946383761</Sequence>
        <DefaultShift>7</DefaultShift>
        <Shift Byte="37">2</Shift>
        <Shift Byte="38">3</Shift>
        <Shift Byte="46">4</Shift>
        <Shift Byte="47">6</Shift>
        <Shift Byte="49">5</Shift>
        <Shift Byte="61">1</Shift>
      </SubSequence>
    </ByteSequence>
  - <ByteSequence Reference="EOFoffset">
    - <SubSequence MinFragLength="0" Position="1" SubSeqMaxOffset="0" SubSeqMinOffset="0">
        <Sequence>3B</Sequence>
        <DefaultShift>-2</DefaultShift>
        <Shift Byte="3B">-1</Shift>
      </SubSequence>
    </ByteSequence>
  </InternalSignature>
```

Figure 10: XML for fmt/3 Internal Signature

What you will find is some XML that looks complex, but really isn't that hard to follow. Let's try and unpack it. Firstly there are two sections, each called a subsection. The subsection starts with a byte sequence offset that tells DROID where it expects to find the pattern (in this case, either BOF or EOF).

The main parts of interest are the values you will find in the <shift> tags. Look closely at the values you can see, and compare them with the pattern we made in the HEX viewer.

You'll notice that they are of course the same. We said earlier the BOF pattern is 474946383761.
If we break this up into byte sized chucks we get: 47 49 46 38 37 61.

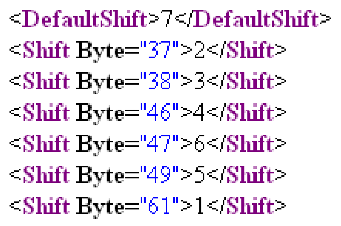
If we give these chunks, or bytes, an ordered number, we can make the following list:

Position	Byte Value
1	47
2	49
3	46
4	38
5	37
6	61

To write this correctly for DROID we have to count backwards towards the offset, meaning we need to reverse the order of the positions we just listed relative to their byte value. In simple terms, we simply need to reverse the Position list; because that's the order DROID expects the pattern to be presented:

Position	Byte Value
6	47
5	49
4	46
3	38
2	37
1	61

Now compare this list, with the values found in the XML:



```
<DefaultShift>7</DefaultShift>
<Shift Byte="37">2</Shift>
<Shift Byte="38">3</Shift>
<Shift Byte="46">4</Shift>
<Shift Byte="47">6</Shift>
<Shift Byte="49">5</Shift>
<Shift Byte="61">1</Shift>
```

Figure 11: Shift positions for fmt/3 BOF signature

I have reordered the values so they are easier to read in correct numerical order – DROID doesn't care for the order they are presented in, simply that it knows the byte position, and the expected value. If you want to explore a little more, have a look at the <DefaultShift> value, in this case '7' and see if you can identify what its purpose is.

(Essentially the <default shift> is the length of the byte pattern + 1, so in this case the pattern is 6 bytes long, so the shift is 7)

The National Archive have written a tool that make the creation of appropriate XML very simple – I wanted to walk you through the process by hand so it's a little more meaningful when we cover some steps shortly. This is also a very simple format. Much more complex patterns exist, and use some very useful notation from the regular expression world to help us build some complex patterns. The TNA paper describes this in very good detail.

Hopefully this has demystified some of this process – we have discovered how the broad format declaration, its file extension and internal signature(s) are linked together in the format signature XML. We have seen how this is a mirror of the data held in PRONOM. And we have seen how we can open a file in a HEX viewer and identify the parts of the file the internal signature refers to.

All that remains is for you to run the example.gif file through DROID, and confirm that DROID does in fact return fmt/3 as the PUID for this file...

2. Making your own signatures

What follows is a step-by-step description of the process we have used at NLNZ to try and figure out some of the patterns that relate to the objects we have collected over the years, but currently do not have a matching PUID/pattern in the PRONOM registry.

2.1. Step 1 – collect set of suitable objects

Amass as many examples as possible for the target format. Ideally they should come from a diverse range of content creators, and creating applications/systems to ensure good coverage of the pattern variations found in the target file.
Sometimes it's not possible to find more than a few examples. Smaller sets may result in 'narrower' or more specific signature definition than might be ideal.

Be very careful that you are looking at a collection of objects that are 'the same'. It's worth spending time digging around in the set to make sure that they are all the same type, and suitable for a group pattern.

You can look at a number of different things, including common file extensions, similar creation/modification dates in the metadata, notes from the original object creator, 'performance' or view etc, when the files are rendered/mounted in/opened with the native application or any other useful application (including HEX viewers, text viewers and image viewers, all of which might give you a view of an object that sheds a little more light on their history.

If you have a large enough set of objects that are the same, it's a very good idea to put some aside as a final test set. This means that you won't look at these objects until you have completed your signature creation process, and you'll use this subset to prove that the signature matches all the files of your target type, including these files that have not been used in the signature creation process.

2.2. Step 2 – Hex hex hex

Once I have a refined set of objects and established a high degree of confidence in their suitability to be a single format with a unique pattern(s), I look at the HEX of the set, searching for common patterns or strings. In the example above we saw that the GIF files all have a common BOF string. This would be an ideal situation as it's easy to find, and easy to confirm that all objects of this type have this pattern/string.

If there is no easily identifiable string, there is another tool that you can use to try and find matching strings.

Marco Pontello has written an excellent tool called TrIDScan. This supports his format ID tool, but is useful when creating DROID signatures none the less.

In essence TrIDScan allows you to give it your set of matching files and it will look for patterns that are common with every file in the set. If it finds one, it writes a small piece of XML that contains the patterns, and their offsets. From here you can use these patterns to construct your own DROID patterns (after you have returned the small piece of XML to Marco so he can add it to his knowledge base)

At this point it's worth commenting on specificity in our pattern creation. The ideal situation is that we can find a short and simple pattern that is unique to the format we are addressing. Let's say for example that we have a file set that has a BOF string of:

```
22 4c 6f 72 65 6d 20 69 70 73 75 6d 20 64 6f 6c 6f 72 20 73 69 74 20 61 6d 65 74 2c 20 63 6f
6e 73 65 63 74 65 74 75 72 20 61 64 69 70 69 73 69 63 69 6e 67 20 65 6c 69 74 2c 20 73 65
64 20 64 6f 20 65 69 75 73 6d 6f 64 20 74 65 6d 70 6f 72 20 69 6e 63 69 64 69 64 75 6e 74 20
75 74 20 6c 61 62 6f 72 65 20 65 74 20 64 6f 6c 6f 72 65 20 6d 61 67 6e 61 20 61 6c 69 71 75
61 2e 20 55 74 20 65 6e 69 6d 20 61 64 20 6d 69 6e 69 6d 20 76 65 6e 69 61 6d 2c 20 71 75
69 73 20 6e 6f 73 74 72 75 64 20 65 78 65 72 63 69 74 61 74 69 6f 6e 20 75 6c 6c 61 6d 63 6f
20 6c 61 62 6f 72 69 73 20 6e 69 73 69 20 75 74 20 61 6c 69 71 75 69 70 20 65 78 20 65 61 20
63 6f 6d 6d 6f 64 6f 20 63 6f 6e 73 65 71 75 61 74 2e 20 44 75 69 73 20 61 75 74 65 20 69 72
75 72 65 20 64 6f 6c 6f 72 20 69 6e 20 72 65 70 72 65 68 65 6e 64 65 72 69 74 20 69 6e 20 76
6f 6c 75 70 74 61 74 65 20 76 65 6c 69 74 20 65 73 73 65 20 63 69 6c 6c 75 6d 20 64 6f 6c 6f
72 65 20 65 75 20 66 75 67 69 61 74 20 6e 75 6c 6c 61 20 70 61 72 69 61 74 75 72 2e 20 45
78 63 65 70 74 65 75 72 20 73 69 6e 74 20 6f 63 63 61 65 63 61 74 20 63 75 70 69 64 61 74
61 74 20 6e 6f 6e 20 70 72 6f 69 64 65 6e 74 2c 20 73 75 6e 74 20 69 6e 20 63 75 6c 70 61 20
71 75 69 20 6f 66 66 69 63 69 61 20 64 65 73 65 72 75 6e 74 20 6d 6f 6c 6c 69 74 20 61 6e 69
6d 20 69 64 20 65 73 74 20 6c 61 62 6f 72 75 6d 2e
```

It maybe that we don't need to use the whole string as the pattern, because the odds of another file having the same byte pattern so long is very slim, so perhaps we can 'get away' with a shorter signature, which is more manageable. (Extra points to anyone who 'decodes' the hex above, and figures out why this particular string is actually a terrible example of a 'unique' string of hex!....).

The trade off here is always trying to balance succinctness (and complexity) with uniqueness. This quality of uniqueness is something that is very difficult to establish in a closed world, and it's only by testing our new signatures against our own diverse content collections, and by others doing the same that we can ensure that we are creating signatures that we trust to be unique and specific to the format of interest.

2.3. Step 3 – Identifying the format

On occasions we are fortunate to already have a good idea what the original format 'is'. This is perhaps not the place for a lengthy discussion on what comprises a unique format, so for conciseness let's assume a format definition as being a collection of digital objects that have a common structure and function, and can be shown through the use of a unique set of signature patterns and/or a common file extension.

It's worth also defining what I mean by 'knowing' what a format is. In this case, I mean having a name, description, or other identifiable features that allow us to describe this set as a useful and purposeful group of common things. Following the example above, the fmt/3 PUID covers a specific implementation of gif files that share a feature set, identifier, common file extension and render requirements. These differ from fmt/4 gif files, which have a different, but similarly unique set of features, identifiers, a common file extension and render requirements.

If we do not know the source of the original format we need to get our sleuthing hats on. There might be some human-readable text in the format files that indicate its creating application, or perhaps just a common string that can be researched for information on the internet. The two other main disciplines that might have some answers are the digital forensic and software engineering worlds. Both communities have various resources available online, and often even searching for the hex strings you can see results in some data from useful places – discussion forums, troubleshooting guides, or other informational sources.

As an example, take the hex value (search term: "47 49 46 38 37 61") we used earlier for the gif example, and see what your favourite search engine returns. If completed correctly, you should see mention or reference to the gif format we have been discussing.

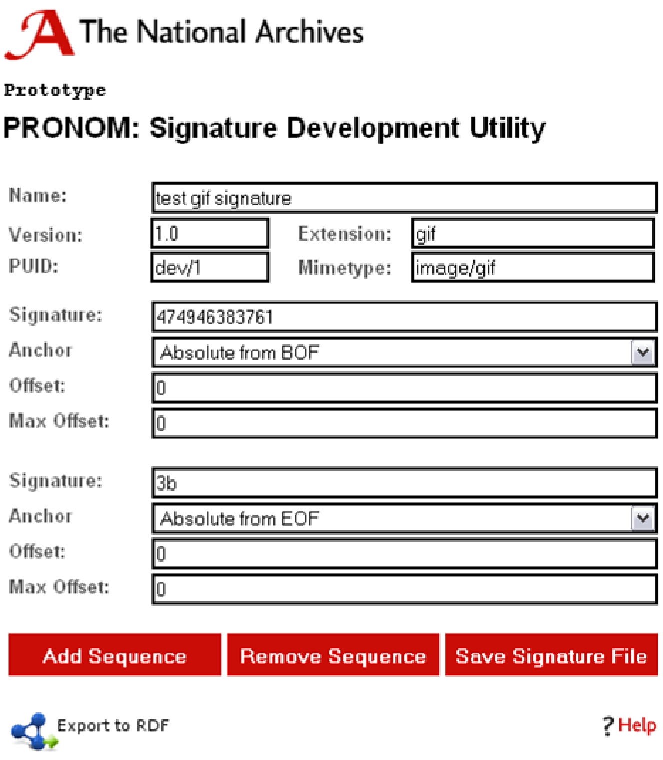
Take a record of any useful information you find so it can be added to the PRONOM record.

2.4. Step 4 – Writing the XML snippet

Once we have at least one pattern, we can use the TNA signature development tool to help us create a valid XML snippet.

At the time of writing this remains a test/demo tool, so the URL may have changed by the time you read this – it's worth contacting the digital preservation team at the TNA if this is the case to see if there is a new location for the tool.

Go to the tool page: http://test.linkeddatapronom.nationalarchives.gov.uk/sigdev/index.htm and populate the tool with your data. I have used the gif as an example:


The National Archives

Prototype

PRONOM: Signature Development Utility

Name:	test gif signature		
Version:	1.0	Extension:	gif
PUID:	dev/1	Mimetype:	image/gif

Signature:	474946383761
Anchor	Absolute from BOF
Offset:	0
Max Offset:	0

Signature:	3b
Anchor	Absolute from EOF
Offset:	0
Max Offset:	0

Figure 12: TNA Signature Development Tool

Save the signature file, using the button and put the signature file somewhere you can find it again.

Open the resulting XML and have a look at the new signature XML. You can see the two sections, and the two sub sequences as previously described:

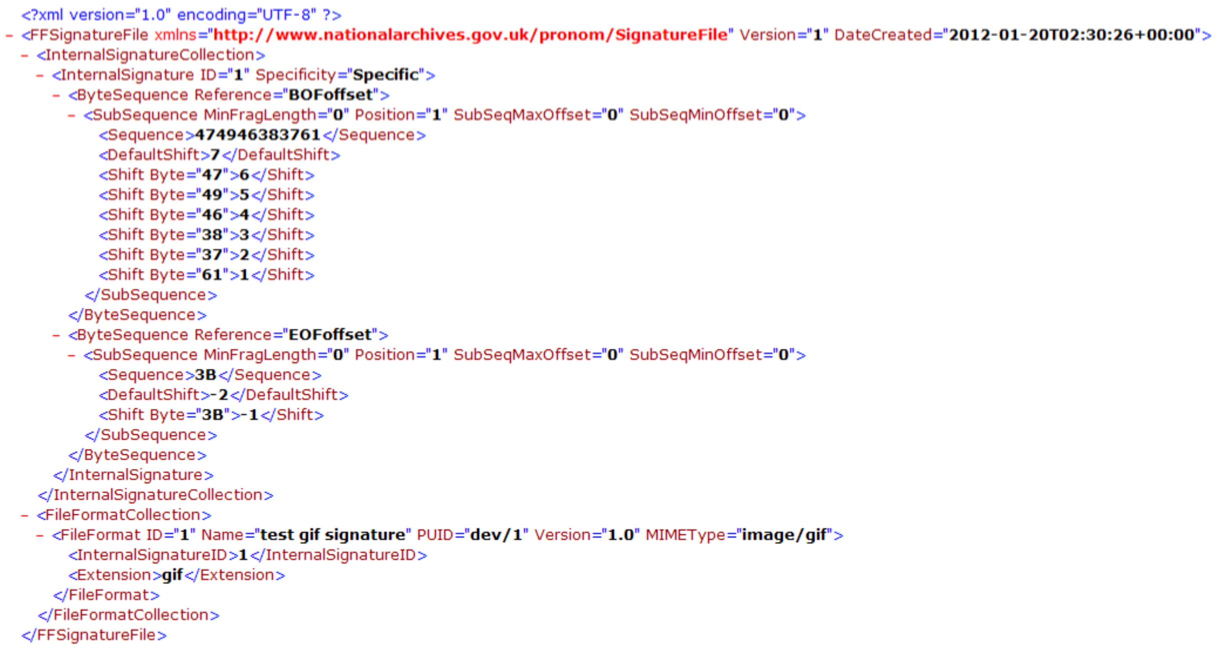
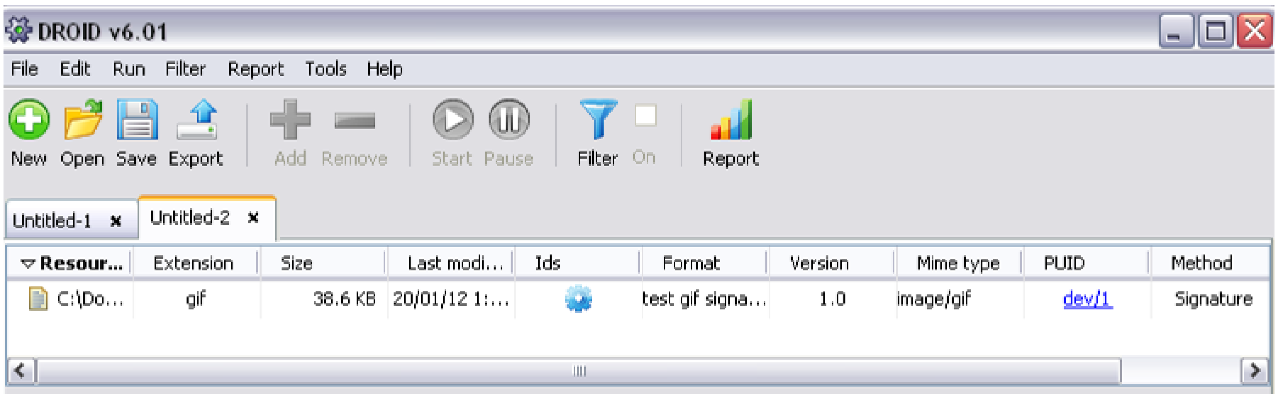

```
  <?xml version="1.0" encoding="UTF-8" ?>
- <FFSignatureFile xmlns="http://www.nationalarchives.gov.uk/pronom/SignatureFile" Version="1" DateCreated="2012-01-20T02:30:26+00:00">
  - <InternalSignatureCollection>
    - <InternalSignature ID="1" Specificity="Specific">
      - <ByteSequence Reference="BOFoffset">
        - <SubSequence MinFragLength="0" Position="1" SubSeqMaxOffset="0" SubSeqMinOffset="0">
            <Sequence>474946383761</Sequence>
            <DefaultShift>7</DefaultShift>
            <Shift Byte="47">6</Shift>
            <Shift Byte="49">5</Shift>
            <Shift Byte="46">4</Shift>
            <Shift Byte="38">3</Shift>
            <Shift Byte="37">2</Shift>
            <Shift Byte="61">1</Shift>
          </SubSequence>
        </ByteSequence>
      - <ByteSequence Reference="EOFoffset">
        - <SubSequence MinFragLength="0" Position="1" SubSeqMaxOffset="0" SubSeqMinOffset="0">
            <Sequence>3B</Sequence>
            <DefaultShift>-2</DefaultShift>
            <Shift Byte="3B">-1</Shift>
          </SubSequence>
        </ByteSequence>
      </InternalSignature>
    </InternalSignatureCollection>
  - <FileFormatCollection>
    - <FileFormat ID="1" Name="test gif signature" PUID="dev/1" Version="1.0" MIMEType="image/gif">
        <InternalSignatureID>1</InternalSignatureID>
        <Extension>gif</Extension>
      </FileFormat>
    </FileFormatCollection>
  </FFSignatureFile>
```

Figure 13: Example XML snippet for a new format

2.5. Step 5 – Testing testing testing

Now we have a signature file, let's upload this to DROID and test it. Fire up DROID – I'm using v6 – and go to the upload signature files menu (ctrl+shift+u), browse to your newly created signature file, and click on upload. Once it has uploaded test your new signature against your starting set. If things have worked out, you should see the new PUID you have assigned against all the files in your set. In my case, I ran the example.gif file through DROID, and was given a PUID match of dev/1 – as per my new signature file above.



Resour...	Extension	Size	Last modi...	Ids	Format	Version	Mime type	PUID	Method
C:\Do...	gif	38.6 KB	20/01/12 1:...		test gif signa...	1.0	image/gif	dev/1	Signature

Figure 14: DROID v6 result with only new xml snippet

If some of your files do not get correct matches, its back to the drawing board for signatures I'm afraid. If you have a reserved sunset (as per step 1) now would be a great time to break them out and see if they all get matches.

Assuming that this stage is okay (i.e. you only see matches, and no fails in the test set), extend your test pool to include some other files of a different type. As the signature only covers the single new format, you should only see matches against your new format type, and nothing else.

Assuming this step is okay it would be worthwhile amending an existing full XML signature file to include your new type – you will need to make sure there are no ID number clashes, for the two reference fields (fileFormatID and InternalSignatureID). You can edit the XML in a txt editor.

Upload your new full signature file, and re-run your first two tests. You should see that in the first test, running only your new format files, only complete matches against your new PUID, and in the second test you should see matches the same as you would have done before, apart from the new addition of your newly matched PUID.

Finally it's worth running the same test of a large collection of files, looking for false positives. If you've followed these steps carefully, this should be unlikely, but it's always worth making sure.

2.6. Step 6 – Submit to PRONOM

Once you have completed the testing it's time to submit your new signature file, and supporting data to PRONOM. There is the 'submit new format' form on the TNA/PRONOM website, its worth reading the notes page first: http://www.nationalarchives.gov.uk/PRONOM/submitinfo.htm