Digital Preservation Technical Paper:

1

Automatic Format Identification Using PRONOM and DROID

Document Control

Author: Adrian Brown, Head of Digital Preservation

Document Reference: DPTP-01

Issue: 2

Issue Date: 7 March 2006

Contents

1 Introduction

This document is one in a series of technical papers produced by the Digital Preservation Department of The National Archives (TNA), covering detailed technical issues related to the preservation and management of electronic records. This technical paper describes the methodology developed and implemented by TNA to automatically identify the formats in which digital objects are encoded.

For the purposes of this document, a format is defined as follows:

> *The internal structure and encoding of a digital object, which allows it to be processed, or to be rendered in human-accessible form. A digital object may be a file, or a bitstream embedded within a file.*

A distinction must be drawn between format *identification* and format *validation*. Identification simply ascertains the format in which an object purports to be encoded, whereas validation ensures that the object is fully conformant to the format specification. As such, validation actually provides the most secure form of identification. However, validation is a technically complex process and is most efficient if it is informed by prior identification of the format.

Identifying and validating the format in which a digital object is encoded is a fundamental prerequisite to accessing and managing the object. This knowledge is required both by human users and IT systems – without it, a digital object cannot be used or preserved. For the majority of the time, this requirement is not obvious – most people use a limited set of software tools, within a consistent IT environment which is able to correctly assign the correct software to a given object. Even so, at some point most users encounter the problem of attempting to access a file in a format which is unknown to them, and unrecognised by either the operating system or any available software. It is also not uncommon to encounter a file which appears to be in a known format, but cannot be opened by the relevant software.

However, it is managed information environments, such as Electronic Records Management Systems (ERMS), or digital preservation facilities, which impose the most rigorous requirements for identifying and validating formats. In these scenarios, it is essential to understand the precise format and version in which every stored object is encoded. It is not enough to know that an object is a Microsoft Word document, for example; the exact version must be identified. It is equally essential to ensure that stored objects are valid, and do conform to the appropriate specification. Invalid objects can arise through the use of poor-quality software tools, or as a result of accidental or deliberate corruption.

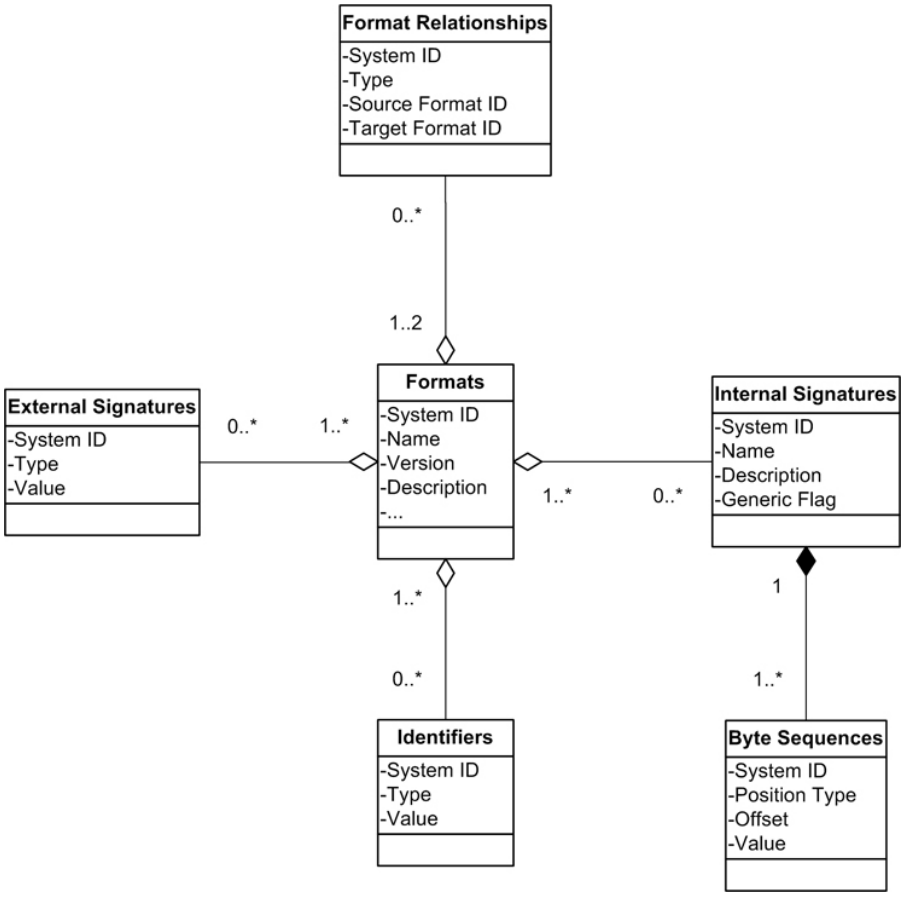
The presumption is made that any format identification or validation method must be automated. Although manual identification is possible, given sufficiently detailed knowledge of how a given format specification and structure is expressed in hexadecimal values, this is clearly not recommended as a practical approach under normal circumstances. This technical report describes an identification method developed by TNA, which uses automated analysis of the binary structure of a digital object, and comparison with predefined internal and external ‘signatures’ for specific formats. This approach has been implemented by TNA as a software application called DROID (Digital Record Object Identification), which uses signature information stored in the PRONOM technical registry. PRONOM and DROID are both freely available on the web at http://www.nationalarchives.gov.uk/pronom. This method is currently limited to identification;

full object characterisation, including validation and property extraction, is intended as a future enhancement.

2 The PRONOM format signature model

The signature information required to perform automatic format identification is stored in the PRONOM technical registry. This section describes how signatures are modelled within PRONOM.

A format signature is any collection of characteristics which may be used to indicate the format of a digital object. This signature may be external or internal to the actual object bitstream. The PRONOM technical registry contains detailed information about individual formats, including their associated signatures. A simplified version of the data model used by PRONOM to describe the relationships between formats and their signatures is illustrated in the following UML class diagram:



Each format may have multiple associated internal and external signatures, with each internal signature comprising one or more byte sequences. Multiple relationships may also be defined between formats. Formats may also be assigned PRONOM Unique Identifiers (PUIDs)[1], which can provide an unambiguous and persistent binding between the format identification of a given object, and the description of that format in PRONOM. Each type of signature is described in more detail in the following sections.

[1]: See Brown (2005)

2.1 External signatures

External signatures encompass all format indicators which are external to the object bitstream, such as Macintosh data forks and Windows file extensions.

In some operating systems, such as Microsoft DOS and Windows, an external signature is provided by the file extension. This indicates the format of the object, e.g. Mydoc.doc for a Microsoft Word document, and Mypic.tif for a TIFF image. However, the primary function of the extension is not to identify the format, but rather to indicate to the operating system the default software package which should be used to open the file. As a result, file extensions suffer from three disadvantages as a method of identification:

- Extensions are not necessarily unique to a single format. For example, the .wks extension is used for both Lotus 1-2-3 worksheets and Microsoft Works documents.

- They do not provide sufficient granularity to adequately identify format versions. For example, the .doc extension does not distinguish between a Word 6 document and a Word 2003 document, although these are significantly different formats.

- Extensions can be defined or altered by users. For example, a user of WordPerfect 5.1 might choose to distinguish letters and memos by using .let and .mem extensions respectively, overriding the default extension assigned by the system.

External signatures are best suited to providing a general indication of the file format, and should not be relied upon for definitive identification. PRONOM records known external signatures which are associated with a given format. However, in DROID, these are accorded much less weight than internal signatures in the identification process. External signatures are described in PRONOM as type/value pairs; currently file extensions are the only permitted type.

2.2 Internal signatures

Internal signatures encompass all format indicators which are contained within the object bitstream. By definition, a file format specification imposes a specific structure upon the content of the bitstream, which is consistent between all digital objects in that format. The characteristics of this structure may therefore be used as a signature for identifying the format. Some formats, such as PNG, include a signature specifically designed to allow identification; in other cases, an incidental signature can be derived from elements of the format structure. Longer signature sequences are clearly preferable to shorter ones, since they reduce the possibility of false identification through coincidental sequences appearing in a bitstream. Even a deliberate signature, such as PNG, may be enhanced using additional structural elements, to provide a more secure identification.

In PRONOM, an internal signature is composed of one or more byte sequences, each comprising a continuous sequence of hexadecimal byte values and, optionally, regular expressions. A signature byte sequence is modelled by describing its starting position within a bitstream and its value. The starting position can be one of two basic types:

- Absolute: the byte sequence starts at a fixed position within the bitstream. This position is described as an offset from either the beginning or the end of the bitstream. The byte sequence can therefore be located by moving to the specified offset, counting from either the Beginning of File (BOF) or End of File (EOF) position. If counting from the EOF position, the offset is to the final byte in the sequence.

- Variable: the byte sequence can start at any offset within the bitstream. The byte sequence can be located by examining the entire bitstream.

The value of the byte sequence is defined as a sequence of hexadecimal values, optionally incorporating any of the following regular expressions:

- **??:** wildcard matching any pair of hexadecimal values (i.e. a single byte).

  e.g.: 0x0A FF ?? FE would match 0x0A FF 6C FE or 0x0A FF 11 FE.

- ***:** wildcard matching any number of bytes (0 or more).

  e.g.: 0x0A FF * FE would match 0x0A FF 6C FE or 0x0A FF 6C 11 FE.

- **{n}:** wildcard matching *n* bytes, where *n* is an integer.

  e.g.: 0x1C 20 {2} 4E 12 would match 0x1C 20 FF 15 4E 12.

- **{m-n}:** wildcard matching between *m*-*n* bytes inclusive, where *m* and *n* are integers or '*'.

  e.g.: 0x03 {1-2} 4D would match 0x03 3C 4D or 0x03 3C 88 4D.

  e.g.: 0x03 {2-*} 4D would match 0x03 3C 88 4D or 0x03 3C 88 3F 4D.

- **(a|b):** wildcard matching one from a list of values (e.g. *a* or *b*), where each value is a hexadecimal byte sequence of arbitrary length containing no wildcards.

  e.g.: 0x0E (FF|FE) 17 would match 0x0E FF 17 or 0x0E FE 17.

- **[a:b]:** wildcard matching any sequence of bytes which lies lexicographically between *a* and *b*, inclusive (where both *a* and *b* are byte sequences of the same length, containing no wildcards, and where *a* is less than *b*). The endian-ness of *a* and *b* are the same as the endian-ness of the signature as a whole.

  e.g. 0xFF [09:0B] FF would match 0xFF 09 FF, 0xFF 0A FF or 0xFF 0B FF.

- **[!a]:** wildcard matching any sequence of bytes other than *a* itself (where *a* is a byte sequence containing no wildcards).

  e.g. 0xFF [!09] FF would match 0xFF 0A FF, but not 0xFF 09 FF.

- **[!a:b]:** wildcard matching any sequence of bytes which does not lie lexicographically between *a* and *b*, inclusive (where *a* and *b* are both byte sequences of the same length, containing no wildcards, and where *a* is less than *b*).

    e.g. 0xFF [!01:02] FF would match 0xFF 00 FF and 0xFF 03 FF, but not 0xFF 01 FF or 0xFF 02 FF.

Note: In the examples above, spaces are included between byte values for reasons of clarity, but are omitted in actual byte sequence values. The signature is processed left-to-right if the signature is measured relative to BOF and right-to-left if measured relative to EOF. The endian-ness of the signature is only relevant for sequences inside square brackets.

A byte sequence must contain a fixed subsequence of at least one byte between each occurrence of '*', or between the beginning or end of the sequence and an occurrence of '*'. Thus, sequences of the following form are not permitted:

[BOF] (a|b)*…

…*(a|b) [EOF]

…*(a|b)*…

The syntax of a byte sequence may be expressed in formal BNF notation as follows:

```
<byte_sequence> ::= <fixed_subsequence> | <byte_sequence>
<variable_wildcard> <fixed_subsequence>

<fixed_subsequence> ::= <chunk> | <fixed_subsequence> <chunk>

<chunk> ::= <byte_subsequence> | <wildcard>

<wild_card> ::= '??' | '{' <integer> '}' | '{' <integer> '-'
<integer> '}' | '(' <byte_subsequence> '|' <byte_subsequence>
')' | '[' <byte_subsequence> ':' <byte_subsequence> ']' | '[!'
<byte_subsequence> ':' <byte_subsequence> ']' | '[!'
<byte_subsequence> ']'

<variable_wildcard> ::= '*' | '{' <integer> '-' '*' '}'

<byte_subsequence> ::= <byte> | <byte_subsequence> <byte>

<byte> ::= <hexadecimal_digit> <hexadecimal_digit>

<hexadecimal_digit> ::= <digit> | <letter>

<integer> ::= <digit> | <integer> <digit>

<digit> ::= '0' | '1' | '2' | '3' | '4' | '5' | '6' | '7' | '8'
| '9'
```

```
<letter> ::= ‘A’ | ‘B’ | ‘C’ | ‘D’ | ‘E’ | ‘F’
```

The order in which byte sequences are recorded within a signature is arbitrary, and does not imply any order or prioritisation within the bitstream. For example, if a signature contains two variable-position sequences, these may appear in any order within a given bitstream. In cases where order is significant (e.g. Sequences 1 and 2 are both variable-position, but Sequence 2 must appear after Sequence 1 in the bitstream), this would be recorded by combining the relevant sequences into a single sequence, separated by the appropriate regular expressions. For the purposes of identification, a Boolean ‘And’ relationship is assumed between every sequence in a signature, i.e. a given object must match every sequence in the signature to match the signature.

2.2.1 Signature specificity

The level of granularity at which internal signatures can be defined may vary. In many cases, it is possible to define internal signatures which are specific to a single version of a given format. However, in other cases a single signature may be common to more than one version of a format (e.g. TIFF), or even to a class of formats (e.g. the OLE2 Compound Document Format). Signatures which are unique to one format record in PRONOM are termed ‘specific’ signatures, whereas those which are common to multiple format records are termed ‘generic’ signatures. No format may have both a generic and a specific internal signature. This distinction is reflected in the identification result: a match with a specific signature is recorded as a ‘positive specific’ identification, and a match with a generic signature is recorded as a ‘positive generic’ identification.

2.2.2 Format priority

It is possible for subtype and supertype relationships to exist between formats. For example, SVG can be considered as a subtype of XML. Since an SVG document is also an XML document, it would match the internal signatures for both formats. However, it is desirable that only the most specific identification be reported. To enable this, the PRONOM data model allows a ‘priority’ relationship to be defined between two formats. Where such a relationship exists, and a given object matches signatures for both formats, the lower priority identification is discarded.

2.2.3 Example byte sequences

The possible scenarios can be illustrated as follows (note that all offsets are calculated starting from 0):

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
1																					

Byte sequence 1 occurs at an absolute offset from the BOF. It would therefore be described as follows:

System ID: 1
Position Type: Absolute from BOF
Offset: 0

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
															2						

Byte sequence 2 occurs at an absolute offset from the EOF. It would therefore be described as follows:

System ID: 2
Position Type: Absolute from EOF
Offset: 2

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
				3																	

Byte sequence 3 occurs at a variable offset. It would therefore be described as follows:

System ID: 3
Position Type: Variable
Offset: n/a

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
				4a					**4b**												

Byte sequence 4 includes two subsequences. Sequence 4a appears at a variable offset, and 4b appears at a fixed relative offset to 4a. It would therefore be described as follows:

System ID: 4
Position Type: Variable
Offset: n/a
Value: *4a*{2}*4b*

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
				5a										**5b**							

Byte sequence 5 includes two subsequences. Sequence 5a appears at an absolute offset, and 5b appears at a variable relative offset to 5a. It would therefore be described as follows:

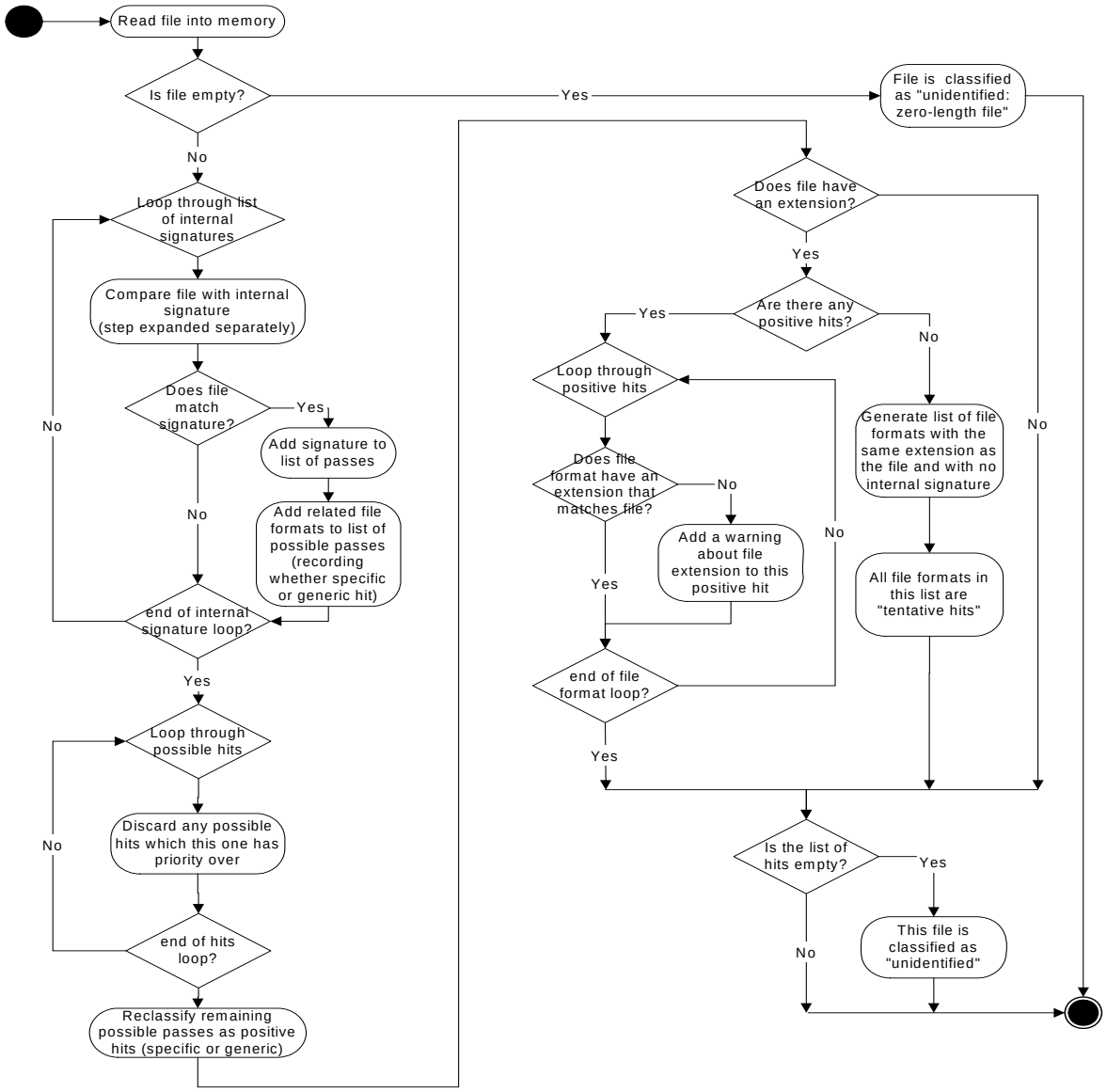
System ID: 5
Position Type: Absolute from BOF
Offset: 4
Value: 5*a*{*}5*b*

3 The DROID format identification process

The DROID software tool uses signature information stored in PRONOM to perform automatic format identification. This section describes the identification algorithm employed by DROID, and the formats of the XML files used by DROID to describe signatures and record the results of the identification process.

3.1 The format identification algorithm

The format identification process employed by DROID is illustrated in the following activity diagram:



If a file is too big to hold in memory in its entirety, the first step is skipped. Instead, the file is read into memory a chunk at a time. To be specific, whenever the algorithm needs to access byte *n* of the file, a check is made to see whether byte *n* is already in memory. If

not, the currently buffered portion of the file is discarded, and a new portion, containing byte *n* is loaded in its place. The size of the buffer used is currently 1000000 bytes.

The algorithm uses the word ‘file’ throughout but streams are identified using the same algorithm. They are read into memory at the start of the process. If this results in running out of memory then the memory contents are written to temporary file and the rest of the stream is appended on the end of that file. In that case, the algorithm then runs on that temporary file.

A file is first compared with the set of internal signatures, and any matches are classified as ‘positive’ identifications. These are further classified as ‘specific’ or ‘generic’ in accordance with the specificity rules defined in Section 2.2.1. If any matches have priority over other matches then the lower priority matches are discarded in accordance with the priority rules defined in Section 2.2.2. If the file matches one or more internal signatures then its file extension is checked against any allowed external signatures; if the file extension does not match any external signatures then a warning of a possible file extension mismatch is generated, but the identification result is not otherwise affected. If the file has no positive results against internal signatures then it is compared with the set of external signatures, and any matches are classified as ‘tentative’ identifications. If the file does not match any internal or external signatures then it is classified as a ‘negative’ identification.

The possible results of an identification process can be summarised as follows:

Status	Warning	Comments
Positive (Specific)		The file matches a specific internal signature
Positive (Specific)	Possible file extension mismatch	The file matches a specific internal signature but the file extension does not match any associated external signatures
Positive (Generic)		The file matches a generic internal signature
Positive (Generic)	Possible file extension mismatch	The file matches a generic internal signature but the file extension does not match any associated external signatures
Tentative		The file matches an external signature but no internal signatures
Negative		The file does not match any internal or external signatures

The pattern matching algorithm employed to undertake the internal signature identification is described in detail in Appendix C.

3.1.1 Identification example

PRONOM contains information on five formats. The first two ("Format A1" and "Format A2") each have their own specific internal signatures consisting of a single byte sequence. The third file format ("Format B") has no internal signature. The last two ("Format C1" and "Format C2") both share the same generic internal signature. All file formats can have an external signature in the form of the file extension "txt". They all also have their own specific extension. These are respectively "fa1", "fa2", "fb", "fc1", "fc2".

A set of files are processed by DROID using these signatures, with the following results:

- **aFile.fa1** passes the internal signature for "Format A1", but fails the others. It is classified as a "Positive Specific" hit on "Format A1".

- **bFile.fa1** passes the internal signature for "Format A2", but fails the others. It is classified as a "Positive Specific" hit on "Format A2". A warning is given, because the file extension is not associated with this format.

- **cFile.fa1** matches the internal signatures for both "Format A1" and "Format A2". It is therefore classified as a possible "Positive Specific" hit on both formats. However, "Format A2" is defined as having priority over "Format A1", so the hit on "Format A1" is discarded. The final output for this file is therefore a "Positive Specific" hit on "Format A2" with a warning about the file extension being wrong.

- **dFile.fa1** fails all internal signatures. Despite having a valid extension for "Format A1", it is not classified as a tentative hit as it has failed the internal signature. It is therefore classified as a "Negative" identification.

- **eFile.txt** fails all internal signatures. Despite having a valid extension for all file formats, it is only classified as a "Tentative" hit on "Format B" as it has failed the internal signatures for all other formats.

- **fFile.xxx** passes the internal signature for "Format A2". The file extension is unknown, so this is classified as a "Positive Specific" identification but with a warning about the file extension.

- **gFile.fb** passes the internal signature for "Format A2". Even though the file extension corresponds to "Format B", the comparison is never made because a positive identification has already been made on the internal signature for "Format A2". This file is classified as a "Positive Specific" identification on "Format A2" but with a warning about the file extension.

- **hFile.xxx** fails all internal signatures. The file extension is unknown, so this is classified as a "Negative" identification.

- **iFile.txt** passes the third internal signature and fails the other two. It is classified as a "Positive Generic" hit on "Format C1" and "Format C2" (no indication is made that the extension matches the other file formats).

- **jFile.fc1** passes the third internal signature and fails the other two. It is classified as a “Positive Generic” hit on “Format C1” and “Format C2”. The hit on “Format C2” also has a warning about the file extension.

- **kFile.txt** passes the all three internal signatures. It is classified as a “Positive Specific” hit on “Format A1” and “Format A2”. By the same argument as for “cFile.fa1”, the hit on “Format A1” is discarded. This file is also a “Positive Generic” hit on “Format C1” and “Format C2”.

3.2 The DROID signature file

The signature information contained in PRONOM must be exported in a form which can be used by the DROID tool to perform automatic format identification. A DROID signature file contains all the information required by DROID to identify formats, and takes the form of an XML document which complies with the schema described in Appendix A.1. The internal signatures contained in the signature file have been pre-processed in accordance with the method described in Appendix B. The resultant signature file contains the following information:

- A list of file formats and for each one, the internal system identifier, format name, format version, format PUID, and a collection of identifiers for all internal signatures associated with the file format, as well as the collection of external signatures. Any priority relationships over other formats are also recorded.

- A list of internal signatures and for each one, a unique identifier and a collection of byte sequences.

- Each byte sequence has a position type (absolute from BOF, absolute from EOF, or variable) and possibly a maximum offset. Each sequence is made up of a series of subsequences, each represented by its longest unambiguous byte sequence, its minimum offset, its shift distance function and a series of left and right sequence fragments (see Appendix B for further details).

- Each sequence fragment has a position number, an unambiguous byte sequence and minimum and maximum offsets (see Appendix B for further details).

3.2.1 Example signature file

The following example DROID Signature File describes the five example formats described in Section 3.1.1.

```
<?xml version=”1.0” encoding=”UTF-8”?>
<FFSignatureFile
xmlns=”http://www.nationalarchives.gov.uk/pronom/SignatureFile”
Version="3" DateCreated="2006-02-09T11:59:52">
      <!--The Internal Signatures-->
      <InternalSignatureCollection>
            <InternalSignature ID="15" Specificity="Specific">
                  <ByteSequence Reference="BOFoffset">
```

```
                        <SubSequence Position="1" SubSeqMinOffset="14"
SubSeqMaxOffset="14" MinFragLength="4">
                              <Sequence>B1B2B3B4</Sequence>
                              <DefaultShift>4</DefaultShift>
                              <Shift Byte="B1">3</Shift>
                              <Shift Byte="B2">2</Shift>
                              <Shift Byte="B3">1</Shift>
                              <Shift Byte="B4">0</Shift>
                              <LeftFragment Position="1" MinOffset="1"
MaxOffset="1">A1A2A3</LeftFragment>
                        </SubSequence>
                        <SubSequence Position="2" SubSeqMinOffset="0"
MinFragLength="0">
                              <Sequence>C1C2C3</Sequence>
                              <DefaultShift>3</DefaultShift>
                              <Shift Byte="C1">2</Shift>
                              <Shift Byte="C2">1</Shift>
                              <Shift Byte="C3">0</Shift>
                        </SubSequence>
                  </ByteSequence>
            </InternalSignature>
            <InternalSignature ID="16" Specificity="Specific">
                  <ByteSequence Reference="BOFoffset">
                        <SubSequence Position="1" SubSeqMinOffset="14"
SubSeqMaxOffset="14" MinFragLength="4">
                              <Sequence>B1B2B3</Sequence>
                              <DefaultShift>3</DefaultShift>
                              <Shift Byte="B1">2</Shift>
                              <Shift Byte="B2">1</Shift>
                              <Shift Byte="B3">0</Shift>
                              <LeftFragment Position="1" MinOffset="1"
MaxOffset="1">A1A2A3</LeftFragment>
                              <RightFragment Position="1" MinOffset="0"
MaxOffset="0">B4</RightFragment>
                              <RightFragment Position="1" MinOffset="0"
MaxOffset="0">B5</RightFragment>
                        </SubSequence>
                        <SubSequence Position="2" SubSeqMinOffset="5"
MinFragLength="2">
                              <Sequence>C1C2C3</Sequence>
                              <DefaultShift>3</DefaultShift>
                              <Shift Byte="C1">2</Shift>
                              <Shift Byte="C2">1</Shift>
                              <Shift Byte="C3">0</Shift>
                              <LeftFragment Position="1" MinOffset="1"
MaxOffset="1">01</LeftFragment>
                              <RightFragment Position="1" MinOffset="4"
MaxOffset="7">D1</RightFragment>
                              <RightFragment Position="2" MinOffset="2"
MaxOffset="2">F1F2F4F5</RightFragment>
                              <RightFragment Position="2" MinOffset="2"
MaxOffset="2">F1F3F4F5</RightFragment>
                        </SubSequence>
                  </ByteSequence>
            </InternalSignature>
            <InternalSignature ID="17" Specificity="Generic">
                  ........
                  ........
            </InternalSignature>
      </InternalSignatureCollection>
```

```
      <!--The File Formats-->
      <FileFormatCollection>
            <FileFormat ID="1" Name="Format A1" Version="V1.1" PUID="V1.1
of format A">
                  <InternalSignatureID>15</InternalSignatureID>
                  <Extension>txt</Extension>
                  <Extension>fa1</Extension>
            </FileFormat>
            <FileFormat ID="2" Name="Format A2" Version="V1.2" PUID="V1.2
of format A">
                  <InternalSignatureID>16</InternalSignatureID>
                  <Extension>txt</Extension>
                  <Extension>fa2</Extension>

      <HasPriorityOverFileFormatID>1</HasPriorityOverFileFormatID>
            </FileFormat>
            <FileFormat ID="3" Name="Format B" Version="V0.0" PUID="V0.0
of format B">
                  <Extension>txt</Extension>
                  <Extension>fb</Extension>
            </FileFormat>
            <FileFormat ID="4" Name="Format C1" Version="V1" PUID="V1 of
format C">
                  <InternalSignatureID>17</InternalSignatureID>
                  <Extension>txt</Extension>
                  <Extension>fc1</Extension>
            </FileFormat>
            <FileFormat ID="5" Name="Format C2" Version="V2" PUID="V2 of
format C">
                  <InternalSignatureID>17</InternalSignatureID>
                  <Extension>txt</Extension>
                  <Extension>fc2</Extension>
            </FileFormat>
      </FileFormatCollection>
</FFSignatureFile>
```

3.3 The DROID file collection file

The DROID file collection file contains the list of files selected for identification, and the results of a DROID identification process. It is generated by the DROID software and takes the form of an XML document. If saved prior to the identification process being performed, it contains the following information:

- The DROID version number
- The signature file version number
- The list of the files selected for identification

If saved following identification, it contains the following information:

- The DROID version number

- The signature file version number
- The list of the files submitted to the identification process
- For each file, a list of file format identifications
- For each file format identification, the name, version and PUID of the file format, the identification status (e.g. positive or tentative), and any relevant warnings or errors.

3.3.1 Example file collection file

The following example DROID File Collection File describes the results of the identification process described in Section 3.1.1.

```
<?xml version="1.0" encoding="UTF-8"?>
<FileCollection
xmlns="http://www.nationalarchives.gov.uk/pronom/FileCollection">
 <DROIDVersion>V1.1</DROIDVersion>
 <SignatureFileVersion>3</SignatureFileVersion>
 <DateCreated>2006-02-09T11:59:52</DateCreated>
 <IdentificationFile IdentQuality="Positive" >
   <FilePath>C:\testfiles\aFile.fa1</FilePath>
   <FileFormatHit>
     <Status>Positive (Specific Format)</Status>
     <Name>Format A1</Name>
     <Version>V1.1</Version
     <PUID>V1.1 of format A</PUID>
   </FileFormatHit>
 </IdentificationFile>
 <IdentificationFile IdentQuality="Positive" >
   <FilePath>C:\testfiles\bFile.fa1</FilePath>
   <FileFormatHit>
     <Status>Positive (Specific Format)</Status>
     <Name>Format A2</Name>
     <Version>V1.2</Version
     <PUID>V1.2 of format A</PUID>
     <IdentificationWarning>Possible file extension
mismatch</IdentificationWarning>
   </FileFormatHit>
 </IdentificationFile>
 <IdentificationFile IdentQuality="Positive" >
   <FilePath>C:\testfiles\cFile.fa1</FilePath>
   <FileFormatHit>
     <Status>Positive (Specific Format)</Status>
     <Name>Format A2</Name>
     <Version>V1.2</Version
     <PUID>V1.2 of format A</PUID>
     <IdentificationWarning>Possible file extension
mismatch</IdentificationWarning>
   </FileFormatHit>
 </IdentificationFile>
 <IdentificationFile IdentQuality="Not identified" >
   <FilePath>C:\testfiles\dFile.fa1</FilePath>
 </IdentificationFile>
```

```
 <IdentificationFile IdentQuality="Tentative" >
   <FilePath>C:\testfiles\eFile.txt</FilePath>
   <FileFormatHit>
     <Status>Tentative</Status>
     <Name>Format B</Name>
     <Version>V0.0</Version
     <PUID>V0.0 of format B</PUID>
   </FileFormatHit>
 </IdentificationFile>
 <IdentificationFile IdentQuality="Positive" >
   <FilePath>C:\testfiles\fFile.xxx</FilePath>
   <FileFormatHit>
     <Status>Positive (Specific Format)</Status>
     <Name>Format A2</Name>
     <Version>V1.2</Version
     <PUID>V1.2 of format A</PUID>
     <IdentificationWarning>Possible file extension
mismatch</IdentificationWarning>
   </FileFormatHit>
 </IdentificationFile>
 <IdentificationFile IdentQuality="Positive" >
   <FilePath>C:\testfiles\gFile.fb</FilePath>
   <FileFormatHit>
     <Status>Positive (Specific Format)</Status>
     <Name>Format A2</Name>
     <Version>V1.2</Version
     <PUID>V1.2 of format A</PUID>
     <IdentificationWarning>Possible file extension
mismatch</IdentificationWarning>
   </FileFormatHit>
 </IdentificationFile>
 <IdentificationFile IdentQuality="Not identified" >
   <FilePath>C:\testfiles\hFile.xxx</FilePath>
 </IdentificationFile>
 <IdentificationFile IdentQuality="Positive" >
   <FilePath>C:\testfiles\iFile.txt</FilePath>
   <FileFormatHit>
     <Status>Positive (Generic Format)</Status>
     <Name>Format C1</Name>
     <Version>V1</Version
     <PUID>V1 of format C</PUID>
   </FileFormatHit>
   <FileFormatHit>
     <Status>Positive (Generic Format)</Status>
     <Name>Format C2</Name>
     <Version>V2</Version
     <PUID>V2 of format C</PUID>
   </FileFormatHit>
 </IdentificationFile>
 <IdentificationFile IdentQuality="Positive" >
   <FilePath>C:\testfiles\jFile.fc1</FilePath>
   <FileFormatHit>
     <Status>Positive (Generic Format)</Status>
     <Name>Format C1</Name>
     <Version>V1</Version
     <PUID>V1 of format C</PUID>
   </FileFormatHit>
   <FileFormatHit>
     <Status>Positive (Generic Format)</Status>
     <Name>Format C2</Name>
```

```
     <Version>V2</Version
     <PUID>V2 of format C</PUID>
     <IdentificationWarning>Possible file extension
mismatch</IdentificationWarning>
   </FileFormatHit>
 </IdentificationFile>
 <IdentificationFile IdentQuality="Positive" >
   <FilePath>C:\testfiles\kFile.txt</FilePath>
   <FileFormatHit>
     <Status>Positive (Generic Format)</Status>
     <Name>Format A2</Name>
     <Version>V1.2</Version
     <PUID>V1.2 of format A</PUID>
   </FileFormatHit>
   <FileFormatHit>
     <Status>Positive (Generic Format)</Status>
     <Name>Format C1</Name>
     <Version>V1</Version
     <PUID>V1 of format C</PUID>
   </FileFormatHit>
   <FileFormatHit>
     <Status>Positive (Generic Format)</Status>
     <Name>Format C2</Name>
     <Version>V2</Version
     <PUID>V2 of format C</PUID>
   </FileFormatHit>
 </IdentificationFile>
</FileCollection>
```

4 Bibliography

Brown, A, 2004, *PRONOM 4 Information Model*, The National Archives

Brown, A, 2005, The PRONOM PUID Scheme: a scheme of persistent unique identifiers for representation information, *Digital Preservation Technical Paper*, **2** http://www.nationalarchives.gov.uk/aboutapps/pronom/pdf/pronom_unique_identifier_scheme.pdf [viewed 7 March 2006]

Horspool, R N, 1980, Practical fast searching in strings, in S*oftware - Practice and Experience*, **10**, 501-506
http://www-igm.univ-mlv.fr/~lecroq/string/node18.html [*viewed 2 September 2005*]

Tessella Support Services, 2006, *DROID: Software Requirements Document*

Tessella Support Services, 2006, *DROID: Architectural Design Document*

Appendices

1 XML schemas

1.1 Signature file schema

The XML schema definition for the signature file is as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:sf="http://www.nationalarchives.gov.uk/pronom/SignatureFile"
targetNamespace="http://www.nationalarchives.gov.uk/pronom/SignatureFile"
elementFormDefault="qualified">
     <xsd:annotation>
          <xsd:documentation xml:lang="en">
            The PRONOM File Format Signature Specification Schema

            Copyright The National Archives 2006.  All rights reserved.
          </xsd:documentation>
     </xsd:annotation>

     <xsd:element name="FFSignatureFile" type="sf:SignatureFileType">
          <xsd:key name="FormatIdKey">
              <xsd:annotation>
                   <xsd:documentation>
                   Define ID as key (ensuring they are also unique)
                        </xsd:documentation>
              </xsd:annotation>
              <xsd:selector xpath="sf:FileFormatCollection/sf:FileFormat"/>
              <xsd:field xpath="@ID"/>
          </xsd:key>
          <xsd:key name="SignatureIdKey">
              <xsd:annotation>
                   <xsd:documentation>
                   Define ID as key (ensuring they are also unique)
               </xsd:documentation>
              </xsd:annotation>
              <xsd:selector
xpath="sf:InternalSignatureCollection/sf:InternalSignature"/>
              <xsd:field xpath="@ID"/>
          </xsd:key>
          <xsd:keyref name="fileformat-haspriorityover-formatid"
refer="sf:FormatIdKey">
              <xsd:annotation>
                   <xsd:documentation>
                   Ensure file formats refer to other formats that exist
                        </xsd:documentation>
              </xsd:annotation>
              <xsd:selector
xpath="sf:FileFormatCollection/sf:FileFormat/sf:HasPriorityOverFileFormatID"/>
              <xsd:field xpath="*"/>
          </xsd:keyref>
          <xsd:keyref name="fileformat-to-signatureid"
refer="sf:SignatureIdKey">
              <xsd:annotation>
                   <xsd:documentation>
                   Ensure file formats refer to signatures that exist
```

```
                        </xsd:documentation>
                </xsd:annotation>
                <xsd:selector
xpath="sf:FileFormatCollection/sf:FileFormat/sf:InternalSignatureID"/>
                <xsd:field xpath="*"/>
           </xsd:keyref>
     </xsd:element>
     <xsd:complexType name="SignatureFileType">
          <xsd:all>
              <xsd:element name="FileFormatCollection">
                   <xsd:complexType>
                        <xsd:choice minOccurs="0" maxOccurs="unbounded">
                             <xsd:element name="FileFormat"
type="sf:FileFormatType"/>
                        </xsd:choice>
                   </xsd:complexType>
              </xsd:element>
              <xsd:element name="InternalSignatureCollection">
                   <xsd:complexType>
                        <xsd:choice minOccurs="0" maxOccurs="unbounded">
                             <xsd:element name="InternalSignature"
type="sf:InternalSignatureType"/>
                        </xsd:choice>
                   </xsd:complexType>
              </xsd:element>
          </xsd:all>
          <xsd:attribute name="Version" type="xsd:positiveInteger"
use="required"/>
          <xsd:attribute name="DateCreated" type="xsd:dateTime"
use="required"/>
     </xsd:complexType>
     <xsd:simpleType name="String50">
          <xsd:restriction base="xsd:string">
              <xsd:maxLength value="50"/>
          </xsd:restriction>
     </xsd:simpleType>
     <xsd:simpleType name="String100">
          <xsd:restriction base="xsd:string">
              <xsd:maxLength value="100"/>
          </xsd:restriction>
     </xsd:simpleType>
     <xsd:simpleType name="String150">
          <xsd:restriction base="xsd:string">
              <xsd:maxLength value="150"/>
          </xsd:restriction>
     </xsd:simpleType>
     <xsd:simpleType name="NonEmptyString">
          <xsd:restriction base="xsd:string">
              <xsd:minLength value="1"/>
          </xsd:restriction>
     </xsd:simpleType>
     <xsd:complexType name="FileFormatType">
          <xsd:choice minOccurs="0" maxOccurs="unbounded">
              <xsd:element name="InternalSignatureID"
type="xsd:nonNegativeInteger"/>
              <xsd:element name="Extension" type="sf:NonEmptyString"/>
              <xsd:element name="HasPriorityOverFileFormatID"
type="xsd:nonNegativeInteger"/>
          </xsd:choice>
          <xsd:attribute name="ID" type="xsd:nonNegativeInteger"
```

```
use="required"/>
          <xsd:attribute name="Name" type="sf:String100" use="required"/>
          <xsd:attribute name="Version" type="sf:String50" use="optional"/>
          <xsd:attribute name="PUID" type="sf:String150" use="optional"/>
     </xsd:complexType>
     <xsd:complexType name="InternalSignatureType">
          <xsd:choice minOccurs="0" maxOccurs="unbounded">
              <xsd:element name="ByteSequence" type="sf:ByteSequenceType"/>
          </xsd:choice>
          <xsd:attribute name="ID" type="xsd:nonNegativeInteger"
use="required"/>
          <xsd:attribute name="Specificity" use="required">
              <xsd:simpleType>
                   <xsd:restriction base="xsd:string">
                        <xsd:enumeration value="Generic"/>
                        <xsd:enumeration value="Specific"/>
                   </xsd:restriction>
              </xsd:simpleType>
          </xsd:attribute>
          <xsd:attribute name="Endianness" use="optional">
              <xsd:simpleType>
                   <xsd:restriction base="xsd:string">
                        <xsd:enumeration value="Big-endian"/>
                        <xsd:enumeration value="Little-endian"/>
                   </xsd:restriction>
              </xsd:simpleType>
          </xsd:attribute>
     </xsd:complexType>
     <xsd:complexType name="ByteSequenceType">
          <xsd:choice minOccurs="0" maxOccurs="unbounded">
              <xsd:element name="SubSequence" type="sf:SubSequenceType"/>
          </xsd:choice>
          <xsd:attribute name="Reference" use="optional">
              <xsd:simpleType>
                   <xsd:restriction base="xsd:string">
                        <xsd:enumeration value="BOFoffset"/>
                        <xsd:enumeration value="EOFoffset"/>
                        <xsd:enumeration value="NOoffset"/>
                   </xsd:restriction>
              </xsd:simpleType>
          </xsd:attribute>
     </xsd:complexType>
     <xsd:complexType name="SubSequenceType">
          <xsd:sequence>
              <xsd:element name="Sequence" type="sf:HexBytes"/>
              <xsd:element name="DefaultShift" type="xsd:integer"/>
              <xsd:choice minOccurs="0" maxOccurs="unbounded">
                   <xsd:element name="Shift" type="sf:ShiftType"/>
                   <xsd:element name="LeftFragment" type="sf:FragmentType"/>
                   <xsd:element name="RightFragment" type="sf:FragmentType"/>
              </xsd:choice>
          </xsd:sequence>
          <xsd:attribute name="Position" type="xsd:integer" use="required"/>
          <xsd:attribute name="SubSeqMinOffset" type="xsd:integer"
use="required"/>
          <xsd:attribute name="SubSeqMaxOffset" type="xsd:integer"
use="optional"/>
          <xsd:attribute name="MinFragLength" type="xsd:integer"
use="required"/>
     </xsd:complexType>
```

```
    <xsd:complexType name="ShiftType">
         <xsd:simpleContent>
             <xsd:extension base="xsd:integer">
                  <xsd:attribute name="Byte" type="sf:HexByte"
use="optional"/>
             </xsd:extension>
         </xsd:simpleContent>
    </xsd:complexType>
    <xsd:simpleType name="HexByte">
         <xsd:annotation>
             <xsd:documentation>
              A byte expressed as hexadecimal
           </xsd:documentation>
         </xsd:annotation>
         <xsd:restriction base="xsd:string">
             <xsd:pattern value="[0-9A-F]{2}"/>
         </xsd:restriction>
    </xsd:simpleType>
    <xsd:simpleType name="HexBytes">
         <xsd:annotation>
             <xsd:documentation>
              One or more bytes expressed as hexadecimal
           </xsd:documentation>
         </xsd:annotation>
         <xsd:restriction base="xsd:string">
             <xsd:pattern value="([0-9A-F]{2})+"/>
         </xsd:restriction>
    </xsd:simpleType>
    <xsd:simpleType name="HexByteRange">
         <xsd:annotation>
             <xsd:documentation>
              One or more bytes expressed as hexadecimal, with patterns.
              Eg. [A1:A4]
           </xsd:documentation>
         </xsd:annotation>
         <xsd:restriction base="xsd:string">
             <xsd:pattern value="(\[!?([0-9A-F]{2})+(:([0-9A-F]{2})+)?\]|[0-
9A-F]{2})+"/>
         </xsd:restriction>
    </xsd:simpleType>
    <xsd:complexType name="FragmentType">
         <xsd:simpleContent>
             <xsd:extension base="sf:HexByteRange">
                  <xsd:attribute name="Position" type="xsd:integer"
use="required"/>
                  <xsd:attribute name="MinOffset" type="xsd:integer"
use="required"/>
                  <xsd:attribute name="MaxOffset" type="xsd:integer"
use="required"/>
             </xsd:extension>
         </xsd:simpleContent>
    </xsd:complexType>
</xsd:schema>
```

1.2 File collection file schema

The XML schema definition for the file collection file is as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:fc="http://www.nationalarchives.gov.uk/pronom/FileCollection"
targetNamespace="http://www.nationalarchives.gov.uk/pronom/FileCollection"
elementFormDefault="qualified">
     <xsd:annotation>
          <xsd:documentation xml:lang="en">
            The PRONOM File Collection Schema

            Copyright The National Archives 2006.  All rights reserved.
       </xsd:documentation>
     </xsd:annotation>

     <xsd:element name="FileCollection" type="fc:FileCollectionType">
          <xsd:unique name="file-name-uniqueness">
               <xsd:selector xpath="fc:IdentificationFile"/>
               <xsd:field xpath="fc:Name"/>
          </xsd:unique>
     </xsd:element>
     <xsd:complexType name="FileCollectionType">
          <xsd:sequence>
               <xsd:choice minOccurs="0">
                    <xsd:sequence>
                         <xsd:element name="DROIDVersion"
type="xsd:string"/>
                         <xsd:element name="SignatureFileVersion"
type="xsd:positiveInteger"/>
                         <xsd:element name="DateCreated"
type="xsd:dateTime"/>
                    </xsd:sequence>
               </xsd:choice>
               <xsd:choice minOccurs="0" maxOccurs="unbounded">
                    <xsd:element name="IdentificationFile"
type="fc:IdentificationFileType"/>
               </xsd:choice>
          </xsd:sequence>
     </xsd:complexType>
     <xsd:complexType name="IdentificationFileType">
          <xsd:sequence>
               <xsd:element name="FilePath" type="xsd:string"/>
               <xsd:element name="Warning" type="xsd:string"
minOccurs="0"/>
               <xsd:choice minOccurs="0" maxOccurs="unbounded">
                    <xsd:element name="FileFormatHit"
type="fc:FileFormatHitType"/>
               </xsd:choice>
          </xsd:sequence>
          <xsd:attribute name="IdentQuality" type="fc:IdentQualityType"
use="optional"/>
     </xsd:complexType>
     <xsd:simpleType name="IdentQualityType">
          <xsd:restriction base="xsd:string">
               <xsd:enumeration value="Positive"/>
               <xsd:enumeration value="Tentative"/>
               <xsd:enumeration value="Not identified"/>
```

```
                <xsd:enumeration value="Error"/>
                <xsd:enumeration value="Not yet run"/>
           </xsd:restriction>
     </xsd:simpleType>
     <xsd:simpleType name="HitStatusType">
           <xsd:restriction base="xsd:string">
                <xsd:enumeration value="Positive (Specific Format)"/>
                <xsd:enumeration value="Positive (Generic Format)"/>
                <xsd:enumeration value="Tentative"/>
                <xsd:enumeration value="Positive"/>
           </xsd:restriction>
     </xsd:simpleType>
     <xsd:complexType name="FileFormatHitType">
           <xsd:all>
                <xsd:element name="Status" type="fc:HitStatusType"/>
                <xsd:element name="Name" type="xsd:string"/>
                <xsd:element name="PUID" type="xsd:string" minOccurs="0"/>
                <xsd:element name="Version" type="xsd:string"
minOccurs="0"/>
                <xsd:element name="IdentificationWarning" type="xsd:string"
minOccurs="0"/>
           </xsd:all>
     </xsd:complexType>
</xsd:schema>
```

2 Pre-processing signatures

2.1 Background for pattern matching algorithm

The pattern matching algorithm is based on the Boyer-Moore-Horspool (BMH) algorithm for string matching (Horspool, 1980). The BMH algorithm does not allow for wildcard characters such as defined above, and so a certain amount of pre-processing needs to be performed on the internal signatures before the algorithm can be applied. The signatures are stored in unprocessed form in the PRONOM registry, and pre-processing is automatically applied as part of the generation of a new signature file.

2.2 Pre-processing of the signature file

The following pre-processing is required on each byte sequence to be used in the pattern matching. To simplify the exposition, we will use the following sequence, defined with an offset of 10 bytes from the beginning of the file as a running example:

A1A2A3[A4:A5]??B1B2B3(B4|B5)*{5}01??C1C2C3{4-7}D1????F1(F2|F3)F4F5

2.2.1 Step 1. Split the byte sequence pattern into fragments to remove all '*'

- Split up each byte sequence pattern P into smaller fragments (P_1 - P_n) wherever a "*" is found (assume P_1 – P_n are in order left to right). Drop any wildcards of the form {n} or {n-m} which appear at the ends of the P_i.

In the example:

P_1 = A1A2A3[A4:A5]??B1B2B3(B4|B5)

P_2 = 01??C1C2C3{4-7}D1????F1(F2|F3)F4F5 (note that we have dropped the "{5}")

2.2.2 Step 2. Find the minimum and maximum subsequence offsests

- If the pattern is **not** defined relative to EOF:
    - For each sequence P_i, work out the minimum and (for P_1) maximum distance of the start of P_i from the end of P_{i-1} (or the start of the file, for P_1).
- Alternatively, if the pattern **is** defined relative to EOF:
    - For each sequence P_i, work out the minimum and (for P_n) maximum distance of the end of P_i from the start of P_{i+1} (or the end of the file, for P_n).

In the example:

P_1 has minimum and maximum subsequence offsets of 10 (recall that the pattern was defined with an offset of 10 bytes from BOF).

P_2 has a minimum subsequence offset of 5 (from the "{5}" we dropped).

2.2.3 Step 3. Find the longest unambiguous byte sequence in every fragment

- For each pattern fragment, pull out longest unambiguous byte sequence (i.e. not containing ??, {n}, {k-m}, (a|b), [!a:b]): P_1X – P_nX (if there is more than one possible longest byte sequence for P_iX, choose one arbitrarily.)

- If the pattern is **not** defined relative to EOF:

    - Work out the minimum offset of the start of P_iX from the start of P_i. This is the "minimum fragment length".

- Alternatively, if the pattern **is** defined relative to EOF:

    - Work out the minimum offset of the end of P_iX from the end of P_i. This is the "minimum fragment length".

In the example:

P_1X = B1B2B3 (although it could equally well have been A1A2A3). The minimum fragment length is 5 (the length of "A1A2A3[A4:A5]??").

P_2X = C1C2C3. The minimum fragment length is 2 (the length of "01??").

2.2.4 Step 4. Split the fragments into remaining unambiguous byte sequences

- For each P_i, split up the remainder of the sequence (i.e. the part not in P_iX) to the left and the right of P_iX according to any occurrences of ? or {n} or {k-m}. This creates arrays of objects P_iL_j (to the left of P_iX) and P_iR_j (to the right of P_iX) where each of these objects contains one or more unambiguous sequences (i.e. if "|" occurs in sequence, then list all possibilities). Unlike in the previous step, these sequences may contain occurrences of the [:] wildcards, and are therefore not technically unambiguous.

- For each subsequence P_iL_j, calculate the minimum and maximum offsets of the end of P_iL_j from the start of P_iL_{j+1} (or the start of P_iX, for the rightmost P_iL_j).

- Similarly, for each subsequence P_iR_j, calculate the minimum and maximum offsets of the start of P_iR_j from the end of P_iR_{j-1} (or the end of P_iX, for the leftmost P_iR_j).

In the example:

P_1L_1 = A1A2A3[A4:A5], with a minimum and maximum offset of 1 (the "??")

P_1R_1 = B4 or B5, with a minimum and maximum offset of 0 (since P_1R_1 follows directly on from P_1X).

P_2L_1 = 01, with a minimum and maximum offset of 1 (the "??")

P_2R_1 = D1, with a minimum offset of 4, and a maximum offset of 7 (corresponding to "{4-7}").

P_2R_2 = F1F2F4F5 or F1F3F4F5, with a minimum and maximum offset of 2 (corresponding to "????").

2.2.5 Step 5. Calculate the 'shift distance': the minimum distance between each byte and the end (or start) of the longest unambiguous byte sequence in its fragment.

- For each distinct byte, *b*, the "shift distance" $D_i(b)$ is equal to the minimum distance from the end of pattern P_iX to the occurrence of that byte in P_iX (unless the byte sequence is defined relative to EOF, in which case it is from the start of P_iX). Any bytes which do not occur in P_iX are given a shift distance equal to the length of P_iX + 1.

In the example, the shift distances for P_1 are given by:

D_1(B3) = 1

D_1(B2) = 2

D_1(B1) = 3

D_1(<*all other bytes*>) = 4

The shift distances for P_2 are defined similarly.

2.3 Pre-processing glossary

Term	Meaning
P	A byte sequence pattern as contained in the internal signature
P_i	A byte sequence pattern fragment created by splitting the pattern so that it does not contain a '*'
P_iX	The longest P_i in P.
Minimum fragment length for P_iX	If the pattern is **not** defined relative to EOF, this is the minimum offset of the start of P_iX from the start of P_i. If the pattern **is** defined relative to EOF, this is the minimum offset of the end of P_iX from the end of P_i.
P_iL_j	A byte sequence pattern fragment to the left of P_iX
P_iR_j	A byte sequence pattern fragment to the right of P_iX
Minimum and maximum offsets	For each subsequence P_iL_j, these are the minimum and maximum number of bytes between the end of P_iL_j from the start of P_iL_{j+1} (or the start of P_iX, for the rightmost P_iL_j). Similarly, for each subsequence P_iR_j, these are the minimum and maximum number of bytes between the start of P_iR_j from the end of P_iR_{j-1} (or the end of P_iX, for the leftmost P_iR_j).
D_i(*byte*)	The 'shift distance' of that byte. This is defined as the minimum distance from the end of pattern P_iX to the occurrence of that byte in P_iX (unless the byte sequence is defined relative to EOF, in which case it is from the start of P_iX).

3 The pattern matching algorithm

The direction in which the pattern matching is carried out is determined by whether the byte sequence is relative to BOF, EOF or neither. The default in the following algorithm is that it is carried out from left to right from the beginning of the file, but if byte sequence is relative to EOF, then the pattern matching will be carried out from right to left, starting at the end of the file. The latter is described below by text in brackets: (/…).

1. We begin by trying to find P_1X (/P_nX). To this end, commence the search at the beginning (/end) of file F, at an offset of the minimum subsequence offset plus the minimum fragment length. This is the earliest (/latest) point in the file at which P_1X (/P_nX) may occur. Take a "window" on F of the same length as P_1X (/P_nX) and compare it with sequence P_1LX (/P_nX).

2. If the window and sequence don't match, then get the "shift distance" associated with the first byte to the right (/left) of the window in F. Shift the window forwards (/backwards) by that many bytes.

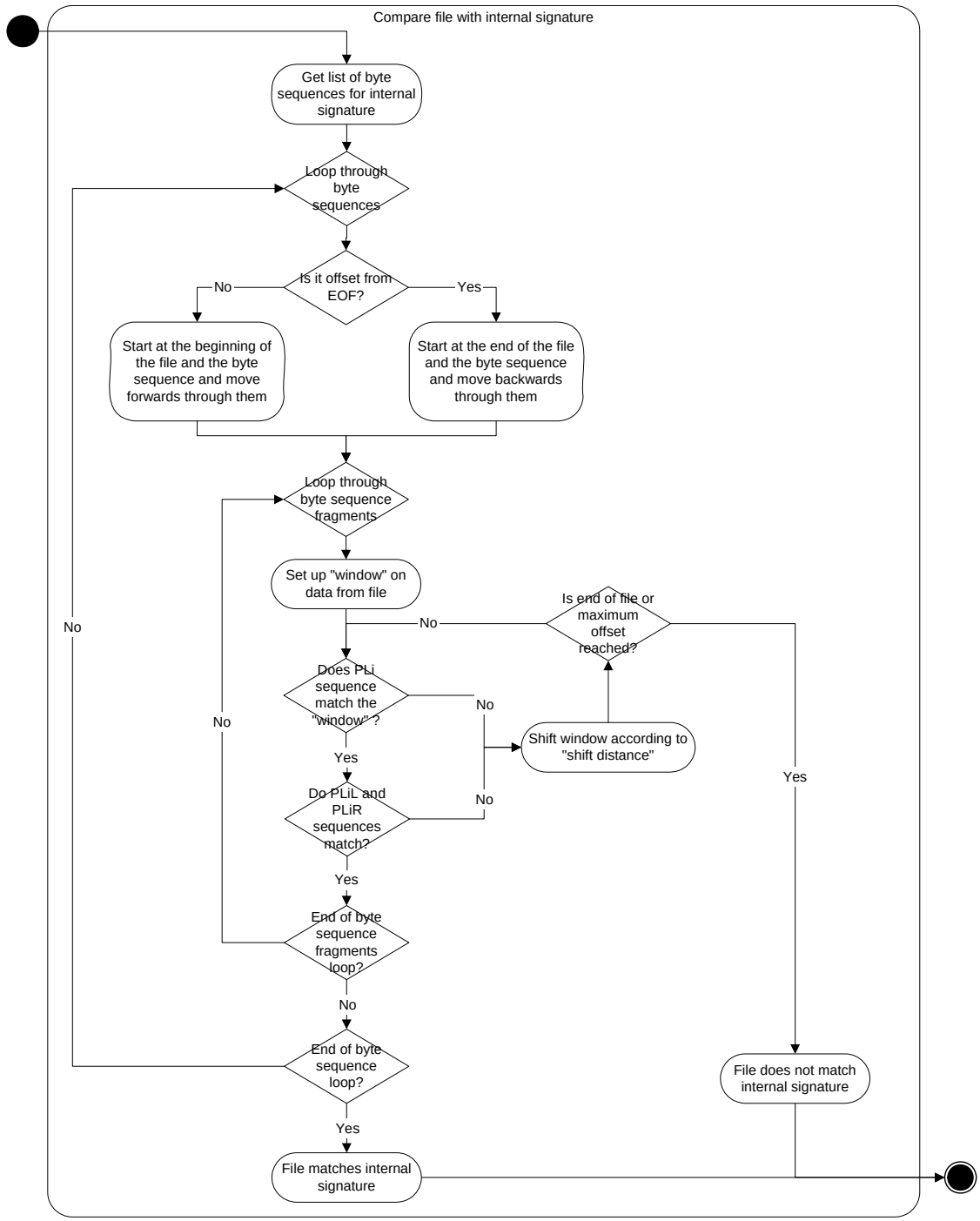
3. Now repeat step 2 until either a match is found (move on to next step) or until either the end (/beginning) of the file or the maximum offset is reached (byte sequence fails).

4. Check for matches to the P_1L_i and P_1R_i ($/P_nL_i$ and P_nR_i) to see whether a match for the whole of P_1 ($/P_n$) can be found. For each sequence, start from smallest possible offset and stop as soon as a match is found so that we end up with the shortest possible sequence in F that matches the pattern. If matches are found for all these sequences, then record the location of the rightmost (/leftmost) byte for match of P_1 ($/P_n$) as this will determine where the search for the next pattern fragment starts. If no match is found for P_1 ($/P_n$), then search for next possible occurrence of P_1X ($/P_nX$) as in steps 2 and 3 until either the end of the file or the maximum offset is reached.

5. Now that we have found a match for the whole of P_1, repeat steps 1 to 4 for the remaining patterns P_2 to P_n. In each case however, do not begin searching at the start (/end) of the file, but at the rightmost (/leftmost) byte of the last pattern found, as recorded in Step 4 (plus (/minus) the minimum subsequence offset for the pattern). Continue until all patterns have been found (i.e. positive match) or until the end (/start) of the file is reached (i.e. no match).

Note that because we search for the patterns in order (from left to right, in the default case), and always find the earliest occurrence of each pattern, there is never any need to backtrack using this algorithm. If we can't find pattern P_3, say, we know that this has nothing to do with the placement of patterns P_1 and P_2.

An activity diagram for the pattern matching is given below (this corresponds to the 'comparison of the file with an internal signature' step in the main activity diagram in section 3.1).


Compare file with internal signature
Get list of byte sequences for internal signature
Loop through byte sequences
Is it offset from EOF?
No
Yes
Start at the beginning of the file and the byte sequence and move forwards through them
Start at the end of the file and the byte sequence and move backwards through them
Loop through byte sequence fragments
Set up "window" on data from file
Is end of file or maximum offset reached?
No
Yes
Does PLi sequence match the "window" ?
No
Shift window according to "shift distance"
Yes
Do PLiL and PLiR sequences match?
No
Yes
End of byte sequence fragments loop?
No
No
End of byte sequence loop?
No
Yes
File does not match internal signature
File matches internal signature