Extensions of the UNIX File Command and Magic File for File Type Identification

William Underwood

Technical Report ITTL/CSITD 09-02

September 2009

Computer Science and Information Technology Division
Information Technology and Telecommunications Laboratory
Georgia Tech Research Institute
Georgia Institute of Technology

The Army Research Laboratory (ARL) and the National Archives and Records Administration (NARA) sponsor this research under Army Research Office Cooperative Agreement W911NF-06-2-0050. The findings in this paper should not be construed as an official ARL or NARA position unless so indicated by other authorized documentation.

ABSTRACT

File format identification is a core requirement for digital archives. The UNIX file command is among the most promising technologies for file type identification. This report describes extensions to the file command and magic file that enhance their utility for file format identification in archival systems.

A File Format Library (database) has been created to manage information about file formats. This information includes file format name, MIME type, PRONOM Universal Identifier and file signature tests. There is a one-to-one correspondence between file formats and file signature tests. Precedence relations between file signature tests are explicitly expressed in the database. Published specifications for file formats are also collected in the library and are used to determine file signatures for the formats. When specifications have not been published for a file format, samples for files in those formats have been collected and analyzed to determine possible file signatures. File signature tests have been created for more than 800 file formats. Sample files for more than 500 of the file formats in the library have been created or collected for testing of the file signatures. These examples are included in the library

The Library includes links to file format software resources that are needed in archival processing of digital records. These include: file viewers/players, archive extractors, file format converters, password recovery software and repairers for damaged files.

The File Format Library supports the creation of a magic file from the file signature tests in the Library. The GTRI File Type Identifier is a graphical user interface to the file command and the magic file created from the File Format Library. The file command and magic tests have been applied to examples of 500+ file formats from the File Format Library. These tests have led to refinement of the file signature tests and discovery of the precedence relationships among file signature tests.

The National Archives (TNA) of the UK provides a public registry of file format information (PRONOM). This information includes file signature patterns expressed as regular expressions. TNA also provides a tool (DROID) that uses these file signature patterns for file format identification. This approach to file type identification is also promising and seems to be primarily limited by the small number of file signature patterns in the PRONOM registry. GTRI is collaborating with TNA to enhance the content of the registry and the performance of the DROID file format identifier.

TABLE OF CONTENTS

1. Introduction

Automated file format identification is a necessary feature for the ingestion of digital objects into an archive. Such identification is needed to insure that the files received from a creator have the expected file formats so that the archive is able to preserve the files and make them available to the public. Knowledge of the file formats is necessary to insure that viewers/players are available for the files, for conversion of legacy file formats into standard, current or persistent object file formats, for extraction of files from archive files, and for repair of damaged files.

The file command and magic file available in the Linux and BSD flavors of UNIX is probably the most widely used tool for file format identification. The tests for identifying file formats in the magic file have been and remain the largest repository of information on file signatures in the world. However, the file command and magic file lack some features for file format identification that are required by digital archives.

The primary objective of the research described in this report is to identify the most promising technology for reliable file format identification and to advance this technology to meet the needs of the National Archives and Records Administration. The specific purpose of this report is to describe extensions made to the UNIX file command and magic file to improve the management of file format information and to increase the reliability of file type identification. These extensions include a File Format Library for managing file format information including file signature criteria that can be used to identify file formats. The library is also a repository for file format specifications and software for viewing/playing files, extracting files from archive files, recovering passwords, and repairing damaged files. It also contains sample files for the file formats in the library.

Section 2 of this report discusses the concepts of file types and file type identifiers. Section 3 briefly summarizes features of the file command and magic file and discusses some of their limitations. Section 4 describes a File Format Library that supports the management of file format information and that supports the creation of magic files used by the file command. Section 5 describes a graphical user interface for file type identification that is based on the file command and a magic file created from the File Format Library. Section 6 describes related research and development. Section 7 summarizes the results.

2. File Format Signatures and External File Identifiers

2.1 Basic Concepts

In the context of data storage and transmission, a *file* is a sequence of bits in which a data representation or computer instructions internal to a computer program has been encoded according to a file format so that it can be stored on a storage medium or transmitted over a network communication link. When the resulting file is read by a computer operating system, it is either decoded and executed by the computer, or passed to a computer program and decoded according to the file format to create a copy of the original internal data representation.

An *executable file* is a serialization of encoded computer instructions. A *script file* is a file that contains instructions for an interpreter or a virtual machine.

A *data file* is an external data representation in a sequence of bits of an internal data representation that can be stored on a storage medium or transmitted across a communication network. When the resulting file is reread according to the file format, it can be used to create a copy of the original internal data representation.

In object-oriented programming, *serialization* is the process of encoding an object into an architecture independent serial format for storage or transmission across a communication network. When the resulting series of bits is decoded according to the serialization format, it can be used to create a semantically identical copy of the original object. Such methods of serialization result in persistent objects that because of their architecture independence are not subject to obsolescence of computer platforms (hardware and operating systems). Examples of such formats include the Hierarchical Data Format (HDF), Comma Separated Values (CSV), and JavaScript Object Notation (JSON).

A *file type* (or *file format class*) is class of files with the same file format. A *file format signature* is invariant data in a file format that can be used to identify the file type (or format) of a file. In the UNIX operating system (including flavors such as BSD, Linux and Solaris), file signatures are referred to as *magic numbers*. In contrast to file format signatures, a *file signature* is a checksum or hash code of a file that can be used to check the integrity of the file

2.1 External Format Identifiers

A unique identifier is needed for file formats that is external to the file but can be linked to the file so that file signatures do not need to be checked every time a file is accessed. File name extensions and metadata stored in the operating system are two approaches that are used. MS-DOS and Windows file names use a file name extension to distinguish different file types. However, file extensions alone are often not enough to discriminate file types. For instance, file extensions such as DOC are ambiguous, since there are several applications that create files with that extension but have different file formats. Furthermore, there are WordPerfect document files that do not have the .DOC extension recommended by the WordPerfect manual. Instead, the document creator avails himself of the filename plus the filename extension to create a longer mnemonic filename. These extended names sometimes result in an extension used for another file type. For instance, SPEECH.COM is a user-created WordPerfect document file discovered in the Bush Presidential e-record collection that contains a speech to the Commonwealth Club. However, the .COM extension is also used to represent a MSDOS compressed executable file.

An alternative way of identifying file formats was developed by Apple Computer for the Macintosh OS Hierarchical File System. Each program installed has an associated *creator* code. This is a 3-4 letter code that tells the MacOS Finder which program created a file in the file system. Each time an application writes a file to the file system, a creator code as well as a file type code are stored as part of the directory entry for the file. The *file* type code is also a 3-4 letter code that tells the MacOS Finder the format of the file. The combination of creator and file type codes is referred to as an *OSTyp*e. Both codes are needed because the same file type could be created by different applications.

A Uniform Type Identifier (UTI) is a string added to Mac OS X 10.4 operating system for uniquely identifying "typed" classes of items. For example, com.apple.pict is the UTI for the Apple Quickdraw PICT format and com.adobe.pdf is the UTI for Adobe PDF. UTIs are used to identify the types of files and folders, clipboard data, bundles, aliases, symbolic links and streaming data. It was developed by Apple to eliminate problems associated with inferring a file's content from its file name extension, MIME type, or Mac OSType code [Apple 2007]. UTIs support multiple inheritance, allowing multimedia files to be identified as not as single type (as in MIME), but as all the types a file contains.

Multipurpose Internet Mail Extensions (MIME) media types were created to configure mail clients to view files that contain multiple formats. MIME was extended to configure browsers for a similar purpose. Each MIME media type consists of a type and a subtype separated by a slash, which uniquely identifies the application that created a file. When one person sends an email message with an attachment, the MIME media type for the application that created the attachment is included in the email message. The person receiving g the email can read the attachment if he/she has the mime type associated with the application that created the file. MIME media types are registered with the Internet Assigned Numbers Authority (IANA) [RFC2048].

The MIME standard initially defined seven media types [RFC2046] with *Model* being added in 1997 [RFC2077].

- Text—Textual information that requires a graphical display device to display the text, e.g., plain text and html.
- Image—Graphical data that requires a graphical display or printing device to view the information, e.g., g3fax, gif or jpeg image formats.
- Audio—Audio data that requires an audio output device such as speakers e.g., au and wav files.
- Video—Video data that requires a display device to display time-varying images possibly with color and coordinated sound, e.g., mpeg or mov files.
- Application—Data that does not fit into the other categories that is either (1) intended to be processed by an application program, e.g., PostScript, word processing or spreadsheet data, or (2) binary data that is not intended to be interpreted and displayed but just transferred. The subtype of *application* will often be the name or include part of the name of the application for which the data are intended. The *octet-stream* subtype is used to indicate that a body contains arbitrary binary data.
- Multipart— Data consisting of multiple entities of independent data types. Four subtypes were initially defined: the *mixed* subtype specifying a generic mixed set of parts, *alternative* for representing the same data in multiple formats, *parallel* for parts intended to be viewed simultaneously, and *digest* for multipart entities in which each part has a default type of *message/rfc822*.
- Message—Used for various types of messages.

- Model— A behavioral or physical representation within a given domain, e.g., mesh, iges and vrml.

There are three registration trees listed on the IANA application for a MIME media type— Vendor, IETF and Personal. RFC 2048 states the following guidelines regarding Vendor and IETF trees.

> The vendor tree is used for media types associated with commercially available products. A registration may be placed in the vendor tree by anyone who has need to interchange files associated with the particular product. However, the registration formally belongs to the vendor or organization producing the software or file format.
>
> Registrations in the vendor tree will be distinguished by the leading facet *vnd*. That may be followed, at the discretion of the registration, by either a media type name from a well-known producer (e.g., vnd.microsoft) or by an IANA-approved designation of the producer's name which is then followed by a media type or product designation (e.g., vnd.microsoft.excel).
>
> The IETF tree is intended for types of general interest to the Internet Community. Registration in the IETF tree requires approval by the IESG and publication of the media type registration as some form of RFC. Media types in the IETF tree are normally denoted by names that are not explicitly faceted, i.e., do not contain period characters.

The format of MIME Types is media type/subtype. It is possible to experimentally extend the subtype names that are not registered with IANA by prefixing them with *x*- [RFC1521].

After the media type and subtype names, can occur a set of parameters, specified in an *attribute=value* notation. The ordering of parameters is not significant.

A *charset* parameter is be used to indicate the character set of the file for *text* subtypes. The *octet-stream* subtype of type *application* is used to indicate that a body contains arbitrary binary data. One of the optional parameters for this subtype is *type* which is the general type or category of binary data. This is intended as information for the human recipient rather than for any automatic processing. A *codecs* parameter is used for *audio* and *video* media types to indicate the coder-decoder for encoding analog signals to digital and decoding digital to analog signals [RFC4281, RFC5334].

The PRONOM Persistent Universal Identifier (PUID) is an extensible scheme of persistent, unique and unambiguous identifiers for file formats in the PRONOM registry [Brown 2006b]. PRONOM, operated by The National Archives of the UK, was the first and remains, to date, the only operational public file format registry in the world. The PUID for file formats is of the form *fmt*/*identifier* where *identifier* is a sequence of digits or lowercase letters.

A PUID of the type *x-fmt* can be assigned to formats that have not yet been assigned an *fmt* identifier. PUID types prefixed by *x*- are used to provide temporary, private or experimental identifiers for that type. These may be used, for example, in the File Format Library as PUIDs for file formats that have not yet been formally assigned a *fmt* identifier by PRONOM. However, the PRONOM registry has 500 or so PUIDs of the type x-fmt. Those file formats are primarily file formats for which there are not internal file signature tests.

MIME Type and Uniform Type Identifiers indicate type-subtype relations and thus are more descriptive than PUIDs. In cases that a human must assign or interpret a file format identifier, a descriptive identifier such as *application/postscript; version=1.0* is preferable to the PUID *x-fmt/91* for the same file type.

3. The File Command and Magic File

3.1 The File Command

File is a UNIX program for determining the file type of a file [OpenBSD 2009]. The original version of the UNIX *file* command originated in Unix Research Version 4 in 1973. All checks for file type were internal to the *file* program. The current version of the file command derives from a version created by Ian Darwin circa 1986-87 and first introduced in Unix System V. The most significant change in that version was to specify the tests for file types in a file external to the file program. There have been many contributors to the evolution of the file command. Since 1990, the primary developer and maintainer of the file command has been Christos Zoulas [OpenBSD 2009].

In outline, the file command's procedure for determining file types is:

1. Check for the empty file or file system files (socket, symbolic link, named pipes (FIFO)).
2. Use tests indicated in an external Magic file to check for file types in particular formats that have invariant data at some location in the file.
3. Check if the file is a text file and if so, indicate the character set (ASCII, ISO-8859-x, ISO 8-bit ASCII, UTF-8, UTF-16, EBCDIC)
    a. Check for the language of a text file (e.g., troff(1), C-program)
    b. Check for tar(1) files
    c. Check for EMX application type
    d. Check for Compound Document Files (CDF)
    e. Check for elf file
4. If none of the checks above succeed, the file type is indicated as *data*.

The file command is probably the most widely used tool for identifying file types. Versions of the file command have been ported to the Windows operating system.

3.2 Magic File

In the UNIX operating system, the system and various application programs distinguish among executable files by checking for a so-called magic number at the beginning of a file. Many other files types, including file formats used in other operating systems, now have a magic number somewhere in the file format.

The *file(1)* command identifies the file type of a file using among other tests, a test of whether a file matches certain patterns specified in a *magic file*. These patterns, referred to as magic numbers, are file signatures. The magic file for version 4.21 of the file command contained tests for approximately 2000 file types.

The magic file is an ASCII text file. The criteria for identifying file types are represented by a series of one or more tests. A test is specified in a line with 4 fields [OpenBSD 2009b]

1. The offset from the beginning of the file at which to begin the test. The initial location of a file is byte 0.

2. The type of data to match, e.g., byte, short, long, string, regex, search.

3. The value (or pattern) to be compared with the value from the file.

    a. If the type of data to match is regex, the pattern is in extended POSIX regular expression syntax and is tested against line *n+1*, where *n* is the given offset.
    b. If the type of data to match is search, the value is a string to search for beginning at the given offset.

4. The message to be printed if the comparison succeeds. If the this field contains a *printf(3)* format specification, the value from the file is printed according to this specification. The fourth field may be empty, if additional tests are needed to identify the file type.

Right angle brackets (>) are used to indicate additional tests that are necessary to identify a file type. A line beginning with a numerical constant is considered to be a level 0 test. The number of right angle brackets preceding a numerical constant indicates the level of the test. If a test at level *n* succeeds, all tests at level *n+1* are performed, and the messages printed if the tests succeed, until a test with level n (or less) occurs.

Blank lines are allowed in the magic file, but ignored. Lines beginning with the number sign (#) are comment lines.

Example: Identification with a single test

```
20      lelong  0xFDC4A7DC      Zoo Archive
```

Example: Identification with a series of tests

```
0       leshort 0xEA60
>7      byte    <0x0A
>>8     byte    <100
>>>8    byte    !1
>>>>8   byte    !2
>>>>>10         byte    2       ARJ Archive
```

Example: Regular expression used for identifying XyWrite Document

```
0       byte    0xAE
>0      regex   (\xAE[A-Z0-9,.]+\xAF)+      XyWrite - Note Bene Document
```

Example: Identification using the search operator

```
0       string          PK\003\004
>4      search/256      META-INF/MANIFEST.MF        JAVA Archive
```

Sometimes the location of invariant data that is needed to identify a file type is not at a constant offset, but is pointed to by a value at a constant location. To locate such information, the magic file provides indirect offsets. “If the first character following the last > is a (then the string following the (is considered an indirect offset. That means that the number after the parenthesis is used as an offset in the file. The value at that offset is read, and is used again as an offset in the file.” [OpenBSD 2009b]

Example: Identification using indirect offsets.

```
0       string  MZ
>0x1E string    PKLITE
>>0x24E         string  PKSFX\ for\ Windows         Zip Self-extracting Archive
>(4.s*512)      long    x
>>&(2.s-517)    byte    x
>>>&0string     PK\3\4 Zip Self-extracting Archive
```

3.3 Limitations of File and Magic

Each new version of the *file(1)* man page contains a section titled BUGS. In version 5.03, there are no bugs listed, but first comment is:

> There must be a better way to automate the construction of the magic file from all the glop in Magdir. What is it?

This comment refers to the fact that sections of the magic file are stored in 112 folders in the Magic directory that have folder names indicating a kind of file format, for example, animation, apple, archive audio, etc. The folders are ordered alphabetically. Each folder contains a single text file with tests for a number of file types. This supposedly facilitates locating tests that need to be modified or where new tests should be placed. Placing the file signature tests in a database table is a better approach to managing this information. The magic file could then be generated from the ordered tests in the table.

The placing of tests in this file structure is also complicated by the fact that some file signature tests must precede other tests and these precedence relationships are only indicated by comments in the sections of the magic file. Creating a precedence relationship for the tests in a database table is one approach to solving this problem.

A sequence of magic tests is used to identify versions of a file format or even different file formats. Editing of file format signatures would be easier if there were a one-to-one correspondence of file signature tests to file formats.

For many file types, the current magic file extracts and outputs more metadata than is necessary to identify the file type. It is an important feature to be able to extract this metadata, but the two functions would be better supported by distinct magic files for file type identification and for file type identification with metadata extraction.

The tests for the language of a text file are keyword based, unreliable and embedded in C-program code that is not as easily modified as the magic tests. The tests and output descriptions for tar(1) files, EMX application types, CDF files and elf files are also embedded in C-programs and not easy to modify.

Some file formats only have invariant data that could be used as a file signature at the end of a file. There needs to be a capability to test for invariant data at an offset from the end-of-file.

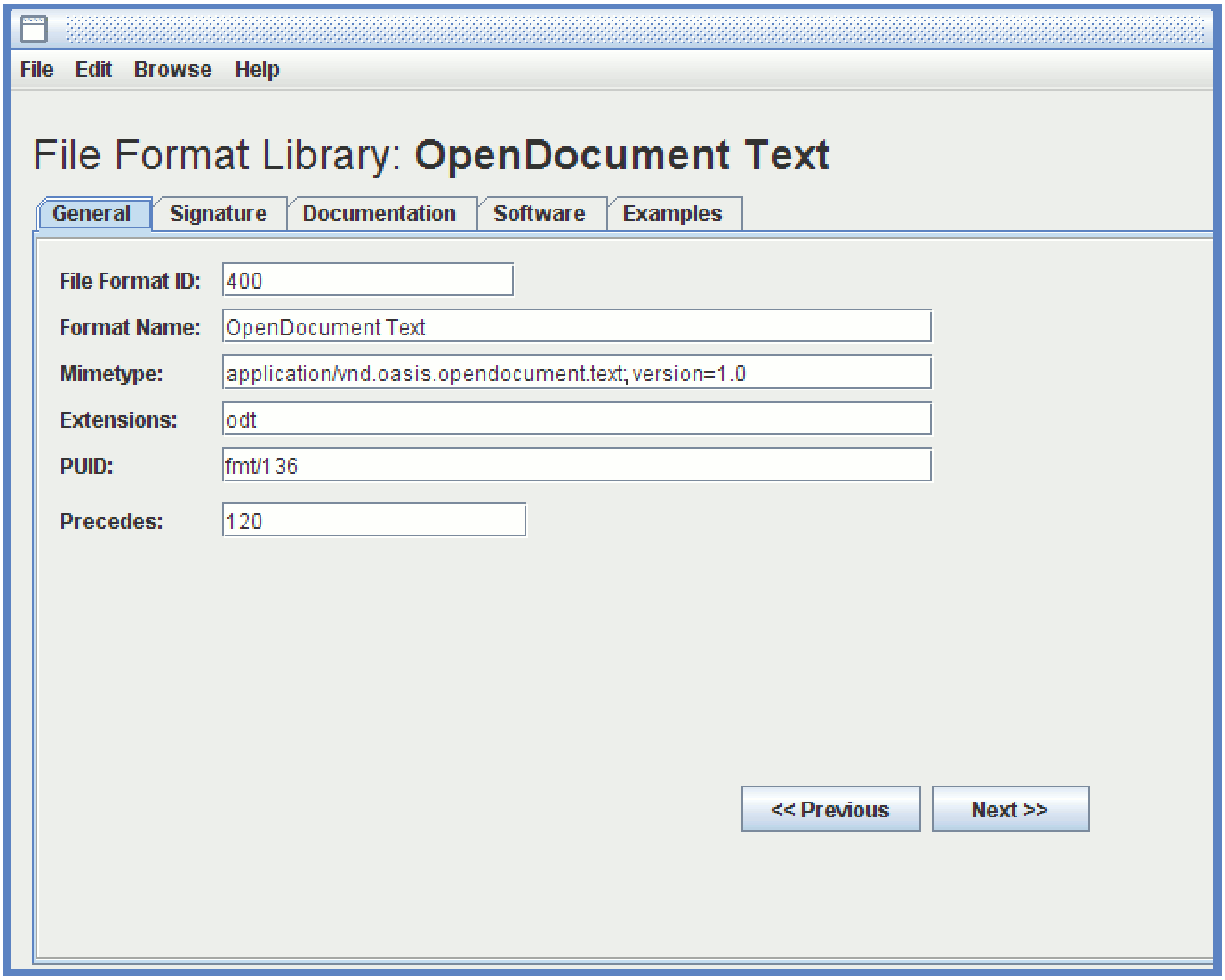
Many software manufacturers consider the specifications for the file formats of their software applications to be proprietary and do not publish them. For example, IBM did not publish the file format specifications for IBM's DisplayWrite 4 documents. Consequently, many of the criteria for identifying file types that are in the magic file are created by individuals who have analyzed examples of the files of a particular file type. Many of these tests are inadequate to correctly identify the file type. This can be addressed by obtaining more file format specifications, analysis of additional examples, and extensive testing with samples of file types.

There are file formats that occur in digital archives for which there are not file signature tests in the magic file. This is largely because the file command was developed for use on UNIX platforms, but many of the file formats accessioned into government archives were created on DOS and Windows platforms.

4. A File Format Library

In this section, extensions of the file command and magic file are described that incorporate features that overcome the limitations discussed in the previous section. Primary among these is the development of a database management application for managing information about file formats including file signature criteria that can be used to identify file formats. This database application is referred to as the GTRI File Format Library. The library is also a repository for file format specifications, software for viewing/playing files, extracting files from archive files, recovering passwords, and repairing damaged files. It also contains sample files.

The File Format library is created using Java and the MySQL RDBMS [Sun 2009]. Figure 1 shows the user interface to the File Format Library.



File Format Library: **OpenDocument Text**

File Format ID: 400

Format Name: OpenDocument Text

Mimetype: application/vnd.oasis.opendocument.text; version=1.0

Extensions: odt

PUID: fmt/136

Precedes: 120

Figure 1. General Tab of the File Format Library.

The *File Format ID* is an internal sequential integer that uniquely identifies a file format in the library. The *Format Name* is the name assigned to the format by the creator or the name by which it is commonly known. There are difficulties in naming a file format if the creator did not assign a name to the format or the format is commonly known by more than one name. There is no standard for naming file formats. If there are versions of the format, the format name should also include a version number.

The value of *MIME Type* is the registered MIME Type for this file format (or the application creating the file format), if one is registered with IANA. If the file format does not have a registered MIME type, a MIME type is created for the format in the form media-type/x-application_name where *application_name* is the name of the product creating the file format. For example, if one were creating a MIME type for an application called Smart Calendar, it would be defined as application/x-smartcalendar.

MIME types have parameters such as *charset*, *type* and *codecs*. Additional parameters have been added to the MIME types in the File Format Library to indicate the *version* of the file format (1, 1.1, iv) and *encoding* of files (encrypted compressed, and rle). For Windows bitmaps (image/bmp), the parameter *colors* (16, 16bit, 24bit, 256, 32bit) has been added.

For executable programs, for example a Windows 32-bit executable, the MIME type includes a *type* parameter, for example,

application/octet-stream; type=win32-exe

The value of extensions is a list of filename extensions commonly used for files in this file format. This value of this attribute can be null.

In the GTRI File Format Library, if there is a PRONOM *fmt* or *x-fmt* identifier for the file format, it has been entered in the PUID field. If there is no PRONOM PUID, the PUID attribute value is null. To have unique identifiers for all file formats, formats without PUIDs could be assigned an identifier such as x-fmt/gtri*numeric_identifer*.

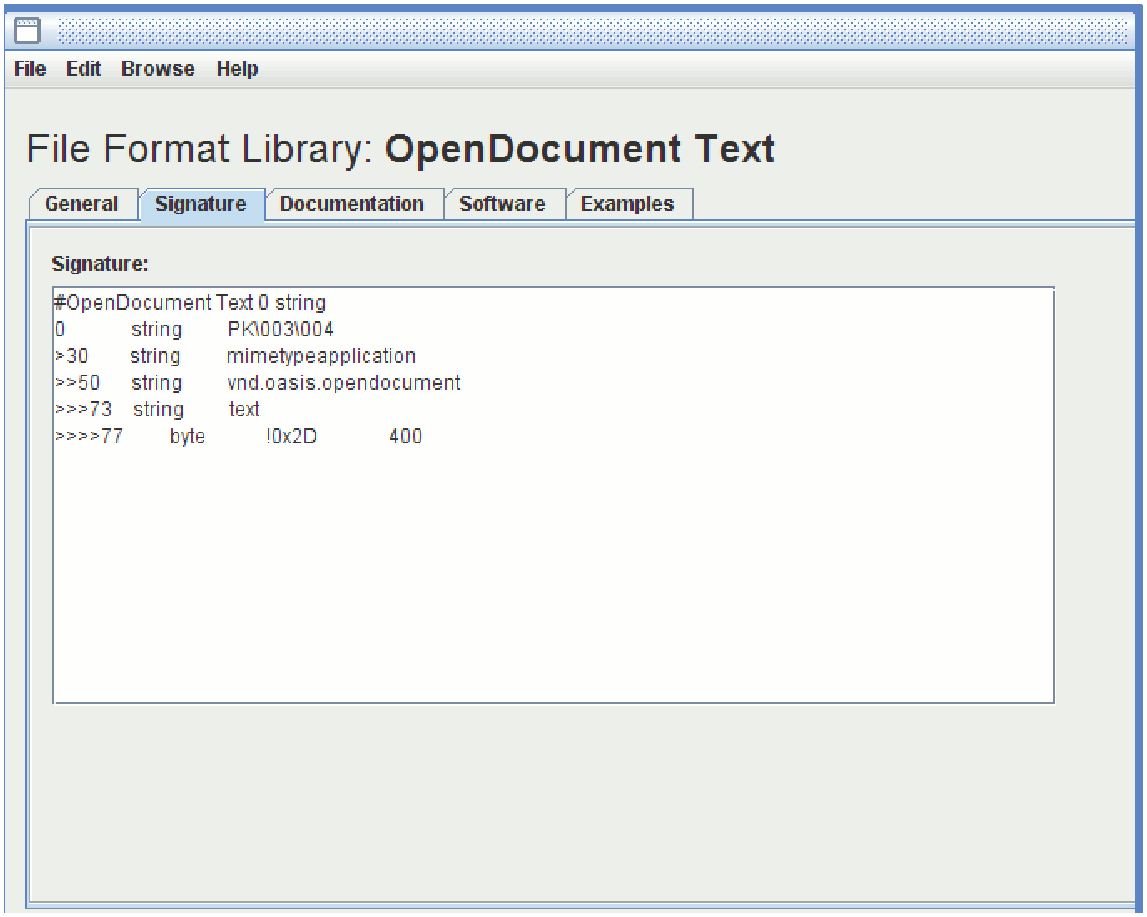
The value for *Precedes* is a list of File Format Ids. The interpretation is that the file signature (magic number) tests for the current File Format must precede the tests for the file formats whose ids are the values of Precedes. Precedence relationships are necessary because some tests for file formats must be performed before others, or the file format will be incorrectly identified. For example, the OpenDocument Text format must be recognized before the Zip file format, because the former is a special case of the latter.

In the future, the attributes of a file format might be extended to include:

Platform (OS, hardware)
Digital Object Class (e.g., 3D model, image, video, audio)
Description (History, relationship to other formats, etc.)
Release date/supported until date
MacOS Creator/Type codes
MacOS X Uniform Type Identifier (UTI)

The magic file released with the file command often uses a sequence of magic statements to characterize multiple file formats and in addition extracts additional metadata for a file type. The file signature tests that have been created for the File Format Library contain criteria for individual file formats and do not extract additional metadata. In the future, an attribute could be added to file format whose values were tests for technical metadata as well as identification of file type.

File signature tests have been created for more than 800 file formats. Figure 2 shows the magic tests characterizing the *OpenDocument Text* file format.



Signature:

```
#OpenDocument Text 0 string
0        string      PK\003\004
>30      string      mimetypeapplication
>>50     string      vnd.oasis.opendocument
>>>73    string      text
>>>>77        byte        !0x2D          400
```

Figure 2. Signature Tab of the File Format Library

The Documentation Tab shows the citation for the file format specification or similar documentation and contains a link to a copy of the specification. File format specifications have been collected for more than 500 of the file formats in the library.

The Software Tab contains the names, sources, and copies of viewers/players for 400 file formats. It also includes the names sources and copies of some file format converters, repairers for damaged file formats and extractors for archive file formats.

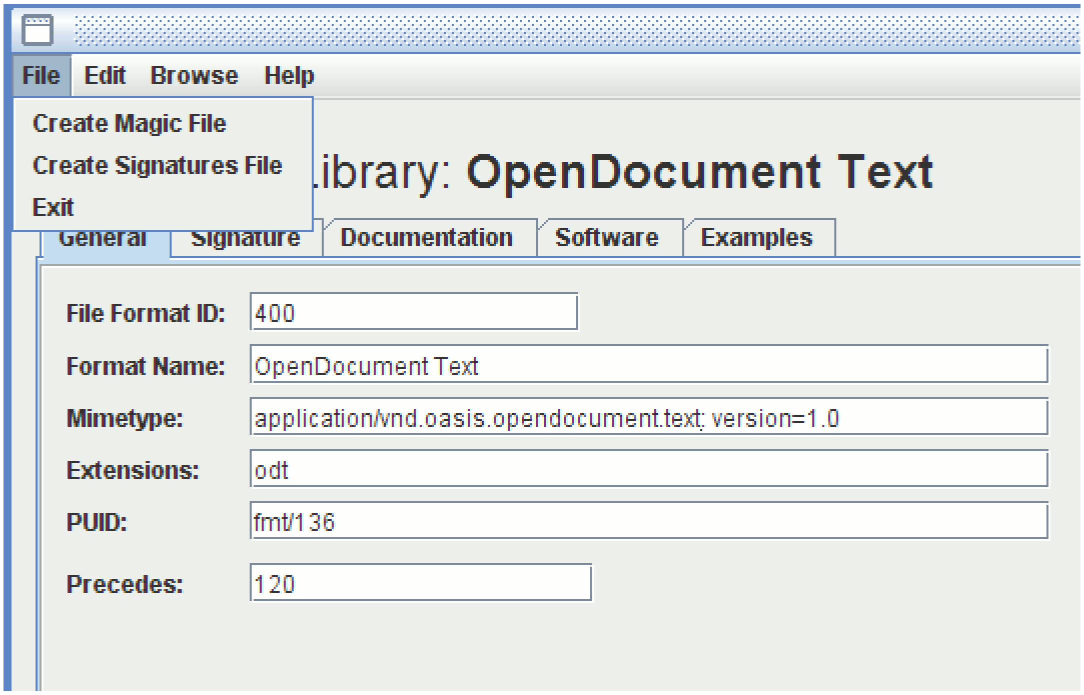
The Examples tab contains links to sample files as well as file title, file name, creator, source and intellectual property rights (public domain, GFDL, Creative Commons, copyright with permission to redistribute). Example files have been collected for more than 500 of these file formats. The file format examples have been used to verify the correctness of the signature tests.

Public domain files are materials believed to be out of copyright, either because of the expiration of the original copyright, or because the materials have been explicitly released into the public domain by their creator(s). This includes files created by the US Government or its employees that have been released to the public, because these are not copyrightable under Section 105 of the US Code, Title 17, and thus are in the public domain. These files also include files that contain material no longer in copyright, or that the GTRI research staff have created and put in the public domain. In all cases, the origin of the file and the name of the creator are provided.

File formats in the library can be located by a search function under the Edit pull down menu. Formats can be located by system format id, file format name, MIME type, file name extension or PUID.

The *File* pull-down menu shown in Figure 3 allows the user to create either:

1. A traditional magic file in which the output is the name of the file format, or
2. A signature file that is a magic file in which the output of magic tests is a file format id, and a tab-delimited file containing File format ID, File Format Name, MIME Type, extensions and PUID.



File Format ID: 400

Format Name: OpenDocument Text

Mimetype: application/vnd.oasis.opendocument.text; version=1.0

Extensions: odt

PUID: fmt/136

Precedes: 120

Figure 3. Creating a Magic File or Signatures File

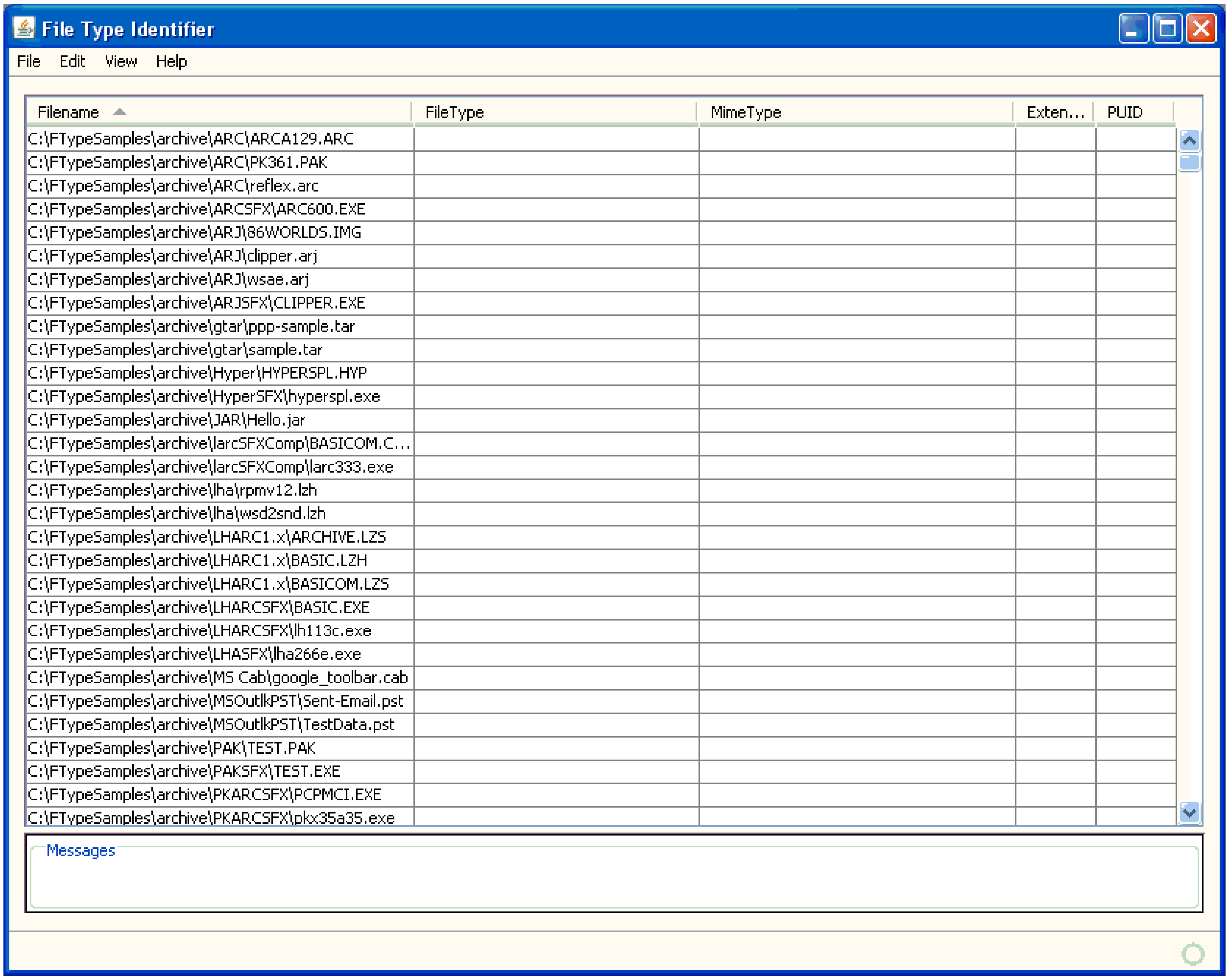
The magic file itself is automatically generated from the signature tests in the File Format Library and the precedence relations between tests are observed.

The output of the file command when using the Signature File is a File Format ID, not the name of the file format. This enables the File Format Identifier described in the next section to relate the File Format ID to metadata such as File Format Name, MIME type, file name extensions, and PRONOM Unique identifiers (PUIDs).

5. GTRI File Type Identifier

In this section a file type identifier is described that uses the file command and the signature file produced from the File Format Library. The tool is written in Java and operates in both the Windows and Linux operating systems.

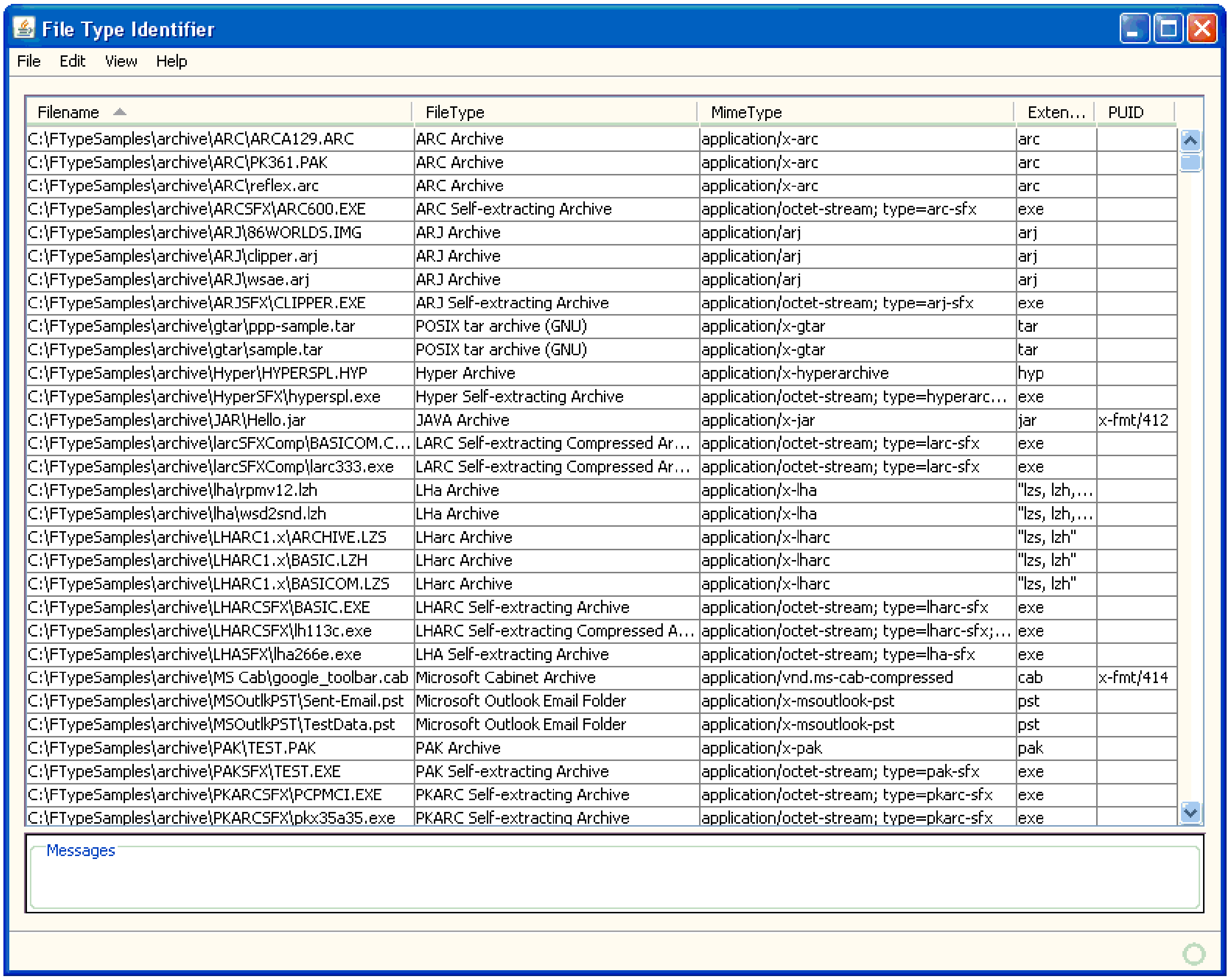
Figure 4 shows the graphical user interface of the File Type Identifier and the paths and file names of files whose formats are to be identified. The files are samples of file formats from the File Format Library.



Filename	FileType	MimeType	Exten...	PUID
C:\FTypeSamples\archive\ARC\ARCA129.ARC				
C:\FTypeSamples\archive\ARC\PK361.PAK				
C:\FTypeSamples\archive\ARC\reflex.arc				
C:\FTypeSamples\archive\ARCSFX\ARC600.EXE				
C:\FTypeSamples\archive\ARJ\86WORLDS.IMG				
C:\FTypeSamples\archive\ARJ\clipper.arj				
C:\FTypeSamples\archive\ARJ\wsae.arj				
C:\FTypeSamples\archive\ARJSFX\CLIPPER.EXE				
C:\FTypeSamples\archive\gtar\ppp-sample.tar				
C:\FTypeSamples\archive\gtar\sample.tar				
C:\FTypeSamples\archive\Hyper\HYPERSPL.HYP				
C:\FTypeSamples\archive\HyperSFX\hyperspl.exe				
C:\FTypeSamples\archive\JAR\Hello.jar				
C:\FTypeSamples\archive\larcSFXComp\BASICOM.C...				
C:\FTypeSamples\archive\larcSFXComp\larc333.exe				
C:\FTypeSamples\archive\lha\rpmv12.lzh				
C:\FTypeSamples\archive\lha\wsd2snd.lzh				
C:\FTypeSamples\archive\LHARC1.x\ARCHIVE.LZS				
C:\FTypeSamples\archive\LHARC1.x\BASIC.LZH				
C:\FTypeSamples\archive\LHARC1.x\BASICOM.LZS				
C:\FTypeSamples\archive\LHARCSFX\BASIC.EXE				
C:\FTypeSamples\archive\LHARCSFX\lh113c.exe				
C:\FTypeSamples\archive\LHASFX\lha266e.exe				
C:\FTypeSamples\archive\MS Cab\google_toolbar.cab				
C:\FTypeSamples\archive\MSOutlkPST\Sent-Email.pst				
C:\FTypeSamples\archive\MSOutlkPST\TestData.pst				
C:\FTypeSamples\archive\PAK\TEST.PAK				
C:\FTypeSamples\archive\PAKSFX\TEST.EXE				
C:\FTypeSamples\archive\PKARCSFX\PCPMCI.EXE				
C:\FTypeSamples\archive\PKARCSFX\pkx35a35.exe				

Figure 4. Graphical User Interface to the GTRI File Type Identifier

When *Identify* is selected from the *File* pull-down menu, the file command is executed using the File Signature File created from the signature tests in the File Format Library. The result shown in part in Figure 5 includes the File Type Name, MIME Type, possible filename extensions and the PUID for each of the sample files.



Filename	FileType	MimeType	Exten...	PUID
C:\FTypeSamples\archive\ARC\ARCA129.ARC	ARC Archive	application/x-arc	arc	
C:\FTypeSamples\archive\ARC\PK361.PAK	ARC Archive	application/x-arc	arc	
C:\FTypeSamples\archive\ARC\reflex.arc	ARC Archive	application/x-arc	arc	
C:\FTypeSamples\archive\ARCSFX\ARC600.EXE	ARC Self-extracting Archive	application/octet-stream; type=arc-sfx	exe	
C:\FTypeSamples\archive\ARJ\86WORLDS.IMG	ARJ Archive	application/arj	arj	
C:\FTypeSamples\archive\ARJ\clipper.arj	ARJ Archive	application/arj	arj	
C:\FTypeSamples\archive\ARJ\wsae.arj	ARJ Archive	application/arj	arj	
C:\FTypeSamples\archive\ARJSFX\CLIPPER.EXE	ARJ Self-extracting Archive	application/octet-stream; type=arj-sfx	exe	
C:\FTypeSamples\archive\gtar\ppp-sample.tar	POSIX tar archive (GNU)	application/x-gtar	tar	
C:\FTypeSamples\archive\gtar\sample.tar	POSIX tar archive (GNU)	application/x-gtar	tar	
C:\FTypeSamples\archive\Hyper\HYPERSPL.HYP	Hyper Archive	application/x-hyperarchive	hyp	
C:\FTypeSamples\archive\HyperSFX\hyperspl.exe	Hyper Self-extracting Archive	application/octet-stream; type=hyperarc...	exe	
C:\FTypeSamples\archive\JAR\Hello.jar	JAVA Archive	application/x-jar	jar	x-fmt/412
C:\FTypeSamples\archive\larcSFXComp\BASICOM.C...	LARC Self-extracting Compressed Ar...	application/octet-stream; type=larc-sfx	exe	
C:\FTypeSamples\archive\larcSFXComp\larc333.exe	LARC Self-extracting Compressed Ar...	application/octet-stream; type=larc-sfx	exe	
C:\FTypeSamples\archive\lha\rpmv12.lzh	LHa Archive	application/x-lha	"lzs, lzh,...	
C:\FTypeSamples\archive\lha\wsd2snd.lzh	LHa Archive	application/x-lha	"lzs, lzh,...	
C:\FTypeSamples\archive\LHARC1.x\ARCHIVE.LZS	LHarc Archive	application/x-lharc	"lzs, lzh"	
C:\FTypeSamples\archive\LHARC1.x\BASIC.LZH	LHarc Archive	application/x-lharc	"lzs, lzh"	
C:\FTypeSamples\archive\LHARC1.x\BASICOM.LZS	LHarc Archive	application/x-lharc	"lzs, lzh"	
C:\FTypeSamples\archive\LHARCSFX\BASIC.EXE	LHARC Self-extracting Archive	application/octet-stream; type=lharc-sfx	exe	
C:\FTypeSamples\archive\LHARCSFX\lh113c.exe	LHARC Self-extracting Compressed A...	application/octet-stream; type=lharc-sfx;...	exe	
C:\FTypeSamples\archive\LHASFX\lha266e.exe	LHA Self-extracting Archive	application/octet-stream; type=lha-sfx	exe	
C:\FTypeSamples\archive\MS Cab\google_toolbar.cab	Microsoft Cabinet Archive	application/vnd.ms-cab-compressed	cab	x-fmt/414
C:\FTypeSamples\archive\MSOutlkPST\Sent-Email.pst	Microsoft Outlook Email Folder	application/x-msoutlook-pst	pst	
C:\FTypeSamples\archive\MSOutlkPST\TestData.pst	Microsoft Outlook Email Folder	application/x-msoutlook-pst	pst	
C:\FTypeSamples\archive\PAK\TEST.PAK	PAK Archive	application/x-pak	pak	
C:\FTypeSamples\archive\PAKSFX\TEST.EXE	PAK Self-extracting Archive	application/octet-stream; type=pak-sfx	exe	
C:\FTypeSamples\archive\PKARCSFX\PCPMCI.EXE	PKARC Self-extracting Archive	application/octet-stream; type=pkarc-sfx	exe	
C:\FTypeSamples\archive\PKARCSFX\pkx35a35.exe	PKARC Self-extracting Archive	application/octet-stream; type=pkarc-sfx	exe	

Figure 5. Display of File Types Identified by the File Type Identifier.

The reason that only two PUIDs are shown in Fig. 5 is that the File Type Identifier recognizes many more types of archive files than are registered in the PRONOM registry.

The parameters of the *file* command are used to exclude the file command's procedural tests. These include tests for the character set of text files, the language of a text file, tar files, EMX application type, compound document files and elf files. There are several reasons for temporarily excluding these procedural tests. First, many of these tests result not in file type identification but additional description or metadata extraction.

Secondly, the recognition of character sets should be possible with a finite state acceptor. Finite state acceptors are equivalent in recognition power to regular expressions, and the magic language supports tests of the content of lines of text using regular expressions. An extension of the regular expressions tests to multiple lines might accomplish the recognition of character sets.

Thirdly, the file command’s procedural tests for recognition of languages (e.g., English, PL/1, Pascal, Java code, HTML) are based on keywords and are unreliable. Magic tests using regular expressions should be explored as an alternative.

Even if procedures with greater expressive power than magic tests are required to identify some file types, these procedures could be included as procedural tests in the File Format Library. An option to compile them as part of the file command could be added to the File Format Library.

6. Related Research

Tresh et al [1995] developed a vector space classifier for identifying file types. It identifies 47 classes of UNIX files using 206 features. The performance of the vector space classifier was compared, with regard to speed, accuracy and extensibility, to discriminant analysis, a decision tree classifier [Quinlan 1993], a rule-based classifier [Hong 1994], and a decision table classifier [UNIX file command and magic file]. Tresh did not present detailed performance numbers, but concluded that their vector space classifier outperformed the other four methods with regard to speed, accuracy and extensibility. Discriminant analysis was very slow because of the extensive computations performed, had high error rates, and was not incrementally extensible. The vector space, decision tree and rule-based classifiers provided comparably fast training and classification. The vector space, decision tree, and rule-based classifiers had low error rates in approximately the same range (2.1 to 5%). The decision table [file command and magic file] classifier had error rates up to 30%. The decision tree, rule-based, and decision table classifiers were judged not to be easily extensible because extension to new classes required rebuilding the trees, rules, tables from scratch, whereas the feature centroid matrix of the vector space classifier could be incrementally extended with coefficients for newly added classes, without having to recompute the existing centroid coefficients.

The version of the file command and magic file used in this experiment was probably version 1.29-1.33. The current file command (and magic file) reliably recognizes all 47 of the classes of files that were used in this experiment.

Another approach to identifying file formats is by the byte frequency distributions of files. File contents are a sequence of bytes and a byte has 256 unique values. Thus, counting the occurrence of the byte values in files of a particular file format might be a way to build representative models for file types. McDaniel [2001, 2003] investigated this method and two extended methods, byte frequency cross-correlation and byte header-trailer profiles. The number of occurrences of a byte value is normalized by dividing them by the number of occurrences of the most frequent byte value. The model for a file signature, which they term a *file fingerprint*, is the average of the normalized number of occurrences for each byte value for each sample in a file type. Models of a file type are compared to the byte frequency distributions of unknown file types by computing the Manhattan distance, which is just the sum of the differences of corresponding bytes in the distribution. The methods were tested on 30 file types with 4 examples per file type. The byte frequency distribution method had an accuracy of 27.5%, the byte frequency cross-correlation method an accuracy of 45.83% and the byte header-trailer method an accuracy of 95.83%.

Li et al [2005] investigate a method for file type identification that is based on n-grams. Since they only consider 1-grams, the method is also a variety of the byte frequency distribution method. Their method differs from McDaniel's in several ways. First, they normalize the number of occurrences of a byte value by the length of the file rather than the number of occurrences of the most frequent byte value. Further, they include in their model the standard deviation of the byte values for a file type. They also considered modeling the 1-grams of only fixed portions of the file, e.g., the first 20 bytes of a file, which they term truncation sizes. They tested their models on 8 file types. The average accuracy of file type classification was evaluated for 1-gram models of the entire file as well as truncation sizes from the beginning of the file of 20, 200, 500 and 1000 bytes. The classification accuracy results for a truncation size of 20 bytes was the best. 98.9-100%. The classification accuracy for 1-gram models of the entire file was 77.1-98.9%.

The number of file types considered in each of the investigations summarized above is too small and at too high a level of granularity. The number of file types that need to be considered in experiments and comparisons of methods is on the order of 1000-2000 file types. Furthermore, one needs to distinguish between versions of a file format because of differences in format between versions. None of the investigations considered file types in their experiments that were different from the file types in the training sets, so could not conclude that the file type of a file was unknown, that is, not among the file types that could be identified.

As discussed in section 2, PRONOM is a public file format registry maintained by The National Archives (TNA) of the UK. DROID is a File Format Identifier distributed by TNA. DROID identifies the file format of a file by file extension (an external signature) or by internal file signature pattern. An internal signature pattern is composed of one or more signature byte sequences. A signature byte sequence is modeled by describing its starting position within a bitstream and its value.

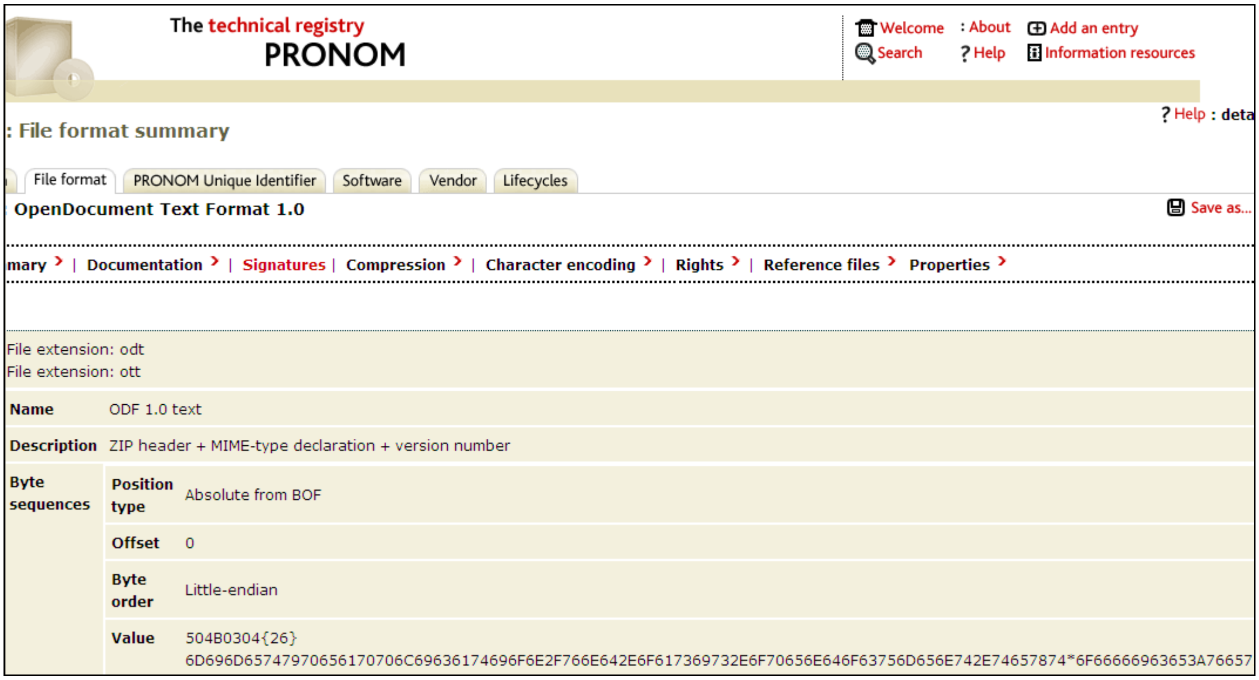
The starting position can be one of two basic types:

- Absolute: the byte sequence starts at a fixed position described as an offset from either the beginning or the end of the bitstream.

- Variable: the byte sequence can start at any offset within the bitstream.

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

The file signature patterns are maintained in the PRONOM registry. Figure 4 shows an example of the metadata and the file signature pattern for the Open Document Text file format whose magic test was shown in Figure 2.



: File format summary

OpenDocument Text Format 1.0

File extension: odt
File extension: ott

Name	ODF 1.0 text	
Description	ZIP header + MIME-type declaration + version number	
Byte sequences	Position type	Absolute from BOF
	Offset	0
	Byte order	Little-endian
	Value	504B0304{26} 6D696D65747970656170706C69636174696F6E2F766E642E6F617369732E6F70656E646F63756D656E742E74657874*6F66666963653A76657

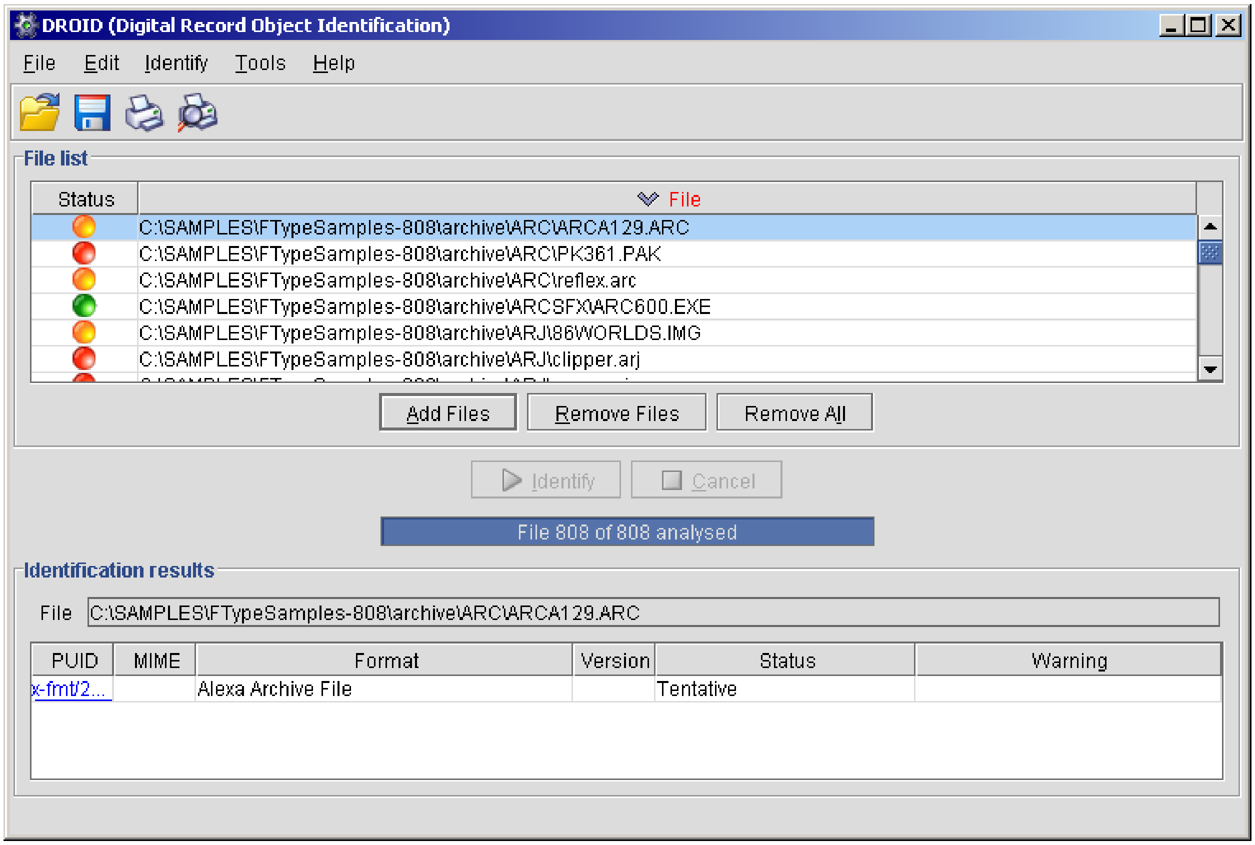
Figure 6. File Signature for Open Document Text in the PRONOM Registry

DROID, the TNA tool for file signature identification, matches the regular expression patterns for file signatures against the bytes in a file to determine the file type of a file. It uses a fast string matching algorithm that does not allow the notation of regular expressions. Brown [2006a] describes an algorithm for preprocessing the regular expressions into a form that can be used by the string matching algorithm.

Figure 7 shows the graphical user interface of DROID. In this example, DROID was applied to the sample files of more than 500 file types from the File Format Library.

Figure 7 shows the results for the first half dozen files in the sample. One can scroll down to see other files that were processed. A green status bullet indicates that the identification is definite, because it was identified by file signature pattern. A red status bullet indicates that the file format could not be identified. An orange status bullet indicates that the file format identification is tentative because it was identified by filename extension. When a file path and file name is highlighted, descriptive information for the indicated file is shown in the lower window.

In Figure 7, the files named ARCA129.ARC and reflex.arc are tentatively identified as Alexa Archive Files. This is incorrect, because these are actually ARC archives. The tentative identification of the file named WORLDS.IMG as a GEM Image is also incorrect, because this is actually an ARJ archive. The reason for the tentative identifications is that the files are identified by their file extension. DROID could not identify the formats for the files named PK361.PAK and clipper.arj because the formats for PAK and ARJ Archives are not in the PRONOM Registry and thus not in the Signature file used by DROID. DROID positively identifies the file named ARC600.EXE as x-fmt/410, a Windows New Executable. It does this by matching the file signature pattern for x-fmt/410 with the contents of the file. DROID is correct, but not specific enough, because this file is actually an ARC Self-extracting Archive, which contains other files that might be records.



Status	File
🟠	C:\SAMPLES\FTypeSamples-808\archive\ARC\ARCA129.ARC
🔴	C:\SAMPLES\FTypeSamples-808\archive\ARC\PK361.PAK
🟠	C:\SAMPLES\FTypeSamples-808\archive\ARC\reflex.arc
🟢	C:\SAMPLES\FTypeSamples-808\archive\ARCSFX\ARC600.EXE
🟠	C:\SAMPLES\FTypeSamples-808\archive\ARJ\86WORLDS.IMG
🔴	C:\SAMPLES\FTypeSamples-808\archive\ARJ\clipper.arj

File 808 of 808 analysed

File C:\SAMPLES\FTypeSamples-808\archive\ARC\ARCA129.ARC

PUID	MIME	Format	Version	Status	Warning
x-fmt/2...		Alexa Archive File		Tentative	

Figure 7. DROID's User Interface

The reason for DROID's poor performance on these particular files is that the PRONOM registry doesn't include many archive and self-extracting archive file formats. Version 21 of DROID's Signature file dated 9-8-2009 includes 611 File types, 442 identified by file extension and 169 by internal file signature.

The following table compares technical features of DROID and the GTRI File Type identifier.

DROID	**GTRI File Type Identifier**
Matches sequences of hex values at offsets	Matches a variety of data types at offsets
Regular expressions on hex values from absolute or variable positions in the file	Regular expressions on strings in lines
Efficient substring search	Less efficient substring search, but indirect offsets increase efficiency
Identifies all possible signature matches and then selects the one of highest priority	Preorders signature tests and stops search when test succeeds
Includes offsets from EOF	Lacks offsets from EOF

7. Conclusion

File format identification is a core requirement for digital archives. The UNIX file command is among the most promising technologies for file type identification. However, there are a number of factors that limit its utility for use in archival systems. These limitations include:

1. Difficult to locate and edit file signature tests
2. Precedence among tests is by the order of tests in the magic file
3. Sequence of magic tests are often for several file formats
4. Output includes technical metadata a well as file type
5. Some identification is by procedures rather than file signatures
6. Lacks capability to test for file signatures at an offset from the end-of-file
7. File command and magic file have not been rigorously tested
8. There are file formats that occur in digital archives for which there are not file signature tests in the magic file

A File Format Library (database) has been created to manage information about file formats. This information includes file format name, MIME type, PRONOM Universal Identifier and file signature tests. There is a one-to-one correspondence between file formats and file signature tests. Precedence relations between file signature tests are explicitly expressed in the database. Published specifications for file formats are also collected in the library and are used to determine file signatures for the formats. When specifications have not been published for a file format, examples of the files in the format have been collected and analyzed to determine possible file signatures. File signature tests have been created for more than 800 file formats. Sample files for more than 500 of the file formats in the library have been created or collected for testing of the file signatures. These examples are included in the Library

The Library includes links to file format software resources that are needed in archival processing of digital records. These include:

- File viewers/players
- Archive extractors
- File format converters
- Password recovery software
- Repairers for damages files

The File Format Library supports the creation of a magic file from the file signature tests in the Library. The GTRI File Type Identifier is a graphical user interface to the file command and the magic file created from the File Format Library. The file command and magic tests have been applied to examples of 500+ file formats from the File Format Library. These tests have led to refinement of the file signatures test and discovery of the precedence relationships among file signature tests.

The extensions to the file command and magic file described in this paper can provide the reliable file format identification needed by NARA, if

1. the file types that the National Archives and Records Administration has already acquired are identified,
2. file signature tests for these files are defined,
3. samples of these file types are acquired for use in testing of these file types, and
4. the File Format Library is kept up to date with file signature tests for new file formats.

The National Archives (TNA) of the UK provides a public registry of file format information (PRONOM). This information includes file signature patterns expressed as regular expressions. TNA also provides a tool, DROID, that uses these file signature patterns for file format identification. This approach to file type identification is also promising and seems to be primarily limited by the small number of file signature patterns in the PRONOM registry. GTRI is collaborating with TNA to enhance the content of the registry and the performance of the DROID file format identifier.

The NARA Transcontinental Persistent Archives Prototype (TPAP) is a collaborative research testbed in which the challenges inherent in preserving, protecting, and providing access to the electronic records are being addressed. TPAP is based on the integrated Rule Oriented Data System (iRODS), a second generation data grid system providing a unified view and seamless access to distributed digital objects across a wide area network.[1] GTRI is developing for TPAP an archival service based on then GTRI File Type Identifier.

[1] http://www.renci.org/focus-areas/humanities-arts-and-social-science/nara-tpap

References

[Apple 2007] Apple. Uniform Type Identifiers Overview. *Apple Developer Connection Reference Library*, 2007.

[Brown 2006a] A. Brown, Automatic Format Identification Using PRONOM and DROID, DTTP-01, The National Archives, March 2006.

[Brown 2006b] A. Brown. The PRONOM PUID Scheme: A scheme of persistent unique identifiers for representation information, DTTP-02, The National Archives, July 2006.

[Hong 1994] S.J. Hong. R-MINI: A heuristic algorithm for generating minimal rules from examples. *Proceedings of the 3rd Pacific Rim International Conference on Artificial Intelligence (PRICAI-94)*, August 1994.

[Li et-al 2005] W. Li, K. Wang, S. J. Stolfo and B. Herzog. Fileprints: identifying file types by n-gram analysis, *Proceeding of the 2005 IEEE Workshop on Information Assurance*, 2005

[McDaniel 2001] M. McDaniel, Automatic File Type Detection Algorithm, Masters Thesis, James Madison University, 2001

[McDaniel & Heydari 2003] M. McDaniel and M. H. Heydari, Content Based File Type Detection Algorithms, *36th Annual Hawaii International Conference on System Sciences*, 2003

[OpenBSD 2009a] OpenBSD Reference Manual, file(1) http://www.openbsd.org/cgi-bin/man.cgi?query=file&sektion=1

[OpenBSD 2009b] OpenBSD Programmer's Manual magic(5) http://www.openbsd.org/cgi-bin/man.cgi?query=magic&sektion=5&apropos=0&manpath=OpenBSD+Current&arch=

[Quinlan 1993] J.R. Quinlan. C4.5: *Programs for Machine Learning*. Morgan Kaufman, San Mateo, CA, 1993.

[RFC1521] N. Borenstein & N. Freed. MIME (Multipurpose Internet Mail Extensions) Part One: Mechanisms for Specifying and Describing the Format of Internet Message Bodies. RFC1521, Sept. 1993

[RFC2045] N. Freed and N. Borenstein, Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies. RFC 2045, Nov. 1996.

[RFC2046] N. Freed and N. Borenstein, Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types, RFC 2046, Nov. 1996

[RFC 2048] Multipurpose Internet Mail Extensions (MIME) Part Four: Registration Procedures, Nov 1996. www.iana.org/assignments/media-types/

[RFC2077] S. Nelson and C. Parks. The Model Primary Content Type for Multipurpose Internet Mail Extensions, RFC 2077, January 1997.

[RFC4281] R. Gellens, D. Singer & P. Frojdh. The Codecs Parameter for "Bucket" Media Types. RFC 4281, Nov. 2005.

[RFC5334] I. Goncalves & C. Montgomery. Ogg Media Types RFC 5334, September 2008. www.ietf.org/rfc/rfc5334.txt

[Sun 2009] Sun Microsystems, Inc. *MySQL 5.0 Reference Manual*, 2009.

[Tresh et al 1995] M. Tresh, N. Palmer and A. Luniewski. Type Classification of Semi-Structured Documents *Proceedings of the 21st VLDB Conference*, Zurich, 1995, pp. 263-274.