filemagic Documentation

Release 1.6

Aaron Iles

February 04, 2014

Contents

filemagic provides a Python API for *libmagic*, the library behind Unix *file* command. It enables the Python developer to easilty test for file types from the extensive identification library that is shipped with *libmagic*.

> "Any sufficiently advanced technology is indistinguishable from magic."
>
> —Arthur C. Clark, 1961

CHAPTER 1

Features

- Simple, Python API.
- Identifies named files or strings.
- Return a textual description, mime type or mime encoding.
- Provide custom magic files to customize file detection.
- Support for both Python2 and Python3.
- Support for both CPython and PyPy.

CHAPTER 2

Table of Contents

2.1 Guide to using filemagic

2.1.1 Background

libmagic is the library that commonly supports the *file* command on Unix system, other than Max OSX which has its own implementation. The library handles the loading of *database* files that describe the magic numbers used to identify various file types, as well as the associated mime types. The library also handles character set detections.

2.1.2 Installation

Before installing *filemagic*, the *libmagic* library will need to be availabile. To test this is the check for the presence of the *file* command and/or the *libmagic* man page.

```
$ which file
$ man libmagic
```

On Mac OSX, Apple has implemented their own version of the file command. However, *libmagic* can be installed using homebrew

```
$ brew install libmagic
```

After *brew* finished installing, the test for the *libmagic* man page should pass.

Now that the presence of *libmagic* has been confirmed, use pip to install filemagic.

```
$ pip install filemagic
```

The `magic` module should now be availabe from the Python shell.

```
>>> import magic
```

The next section will describe how to use the `magic.Magic` class to identify file types.

2.1.3 Usage

The `magic` module uses ctypes to wrap the primitives from *libmagic* in the more user friendly `magic.Magic` class. This class handles initialization, loading databases and the release of resources.

```
>>> import magic
```

To ensure that resources are correctly released by `magic.Magic`, it's necessary to either explicitly call `close()` on instances, or use `with` statement.

```
>>> with magic.Magic() as m:
...     pass
...
```

`magic.Magic` supports context managers which ensures resources are correctly released at the end of the `with` statements irrespective of any exceptions.

To identify a file from it's filename, use the `id_filename()` method.

```
>>> with magic.Magic() as m:
...     m.id_filename('setup.py')
...
'Python script, ASCII text executable'
```

Similarily to identify a file from a string that has already been read, use the `id_buffer()` method.

```
>>> with magic.Magic() as m:
...     m.id_buffer('#!/usr/bin/python\n')
...
'Python script, ASCII text executable'
```

To identify with mime type, rather than a textual description, pass the `MAGIC_MIME_TYPE` flag when creating the `magic.Magic` instance.

```
>>> with magic.Magic(flags=magic.MAGIC_MIME_TYPE) as m:
...     m.id_filename('setup.py')
...
'text/x-python'
```

Similarily, `MAGIC_MIME_ENCODING` can be passed to return the encoding type.

```
>>> with magic.Magic(flags=magic.MAGIC_MIME_ENCODING) as m:
...     m.id_filename('setup.py')
...
'us-ascii'
```

2.1.4 Memory management

The *libmagic* library allocates memory for its own use outside that Python. This memory needs to be released when a `magic.Magic` instance is no longer needed. The preferred way to doing this is to explicitly call the `close()` method or use the `with` statement, as described above.

Starting with version 1.4 `magic.Magic` this memory will be automatically cleaned up when the instance is garbage collected. However, unlike CPython, some Python interpreters such as PyPy, Jython and IronPython do not have deterministic garbage collection. Because of this, *filemagic* will issue a warning if it automatically cleans up resources.

2.1.5 Unicode and filemagic

On both Python2 and Python3, `magic.Magic`'s methods will encode any unicode objects (the default string type for Python3) to byte strings before being passed to *libmagic*. On Python3, returned strings will be decoded to unicode using the default encoding type. The user **should not** be concerned whether unicode or bytes are passed to

`magic.Magic` methods. However, the user **will** need to be aware that returned strings are always unicode on Python3 and byte strings on Python2.

2.1.6 Reporting issues

The source code for *filemagic* is hosted on Github. Problems can be reported using Github's issues tracking system.

filemagic has been tested against *libmagic* 5.11. Continuous integration is provided by Travis CI. The current build status is .

2.2 Command Line Invocation

filemagic can be invoked from the command line by running the `magic.command` module as a script. Pass `-h` or `--help` to print usage information.

```
$ python -m magic.command --help
Usage: python -m magic [options] file ...

Options:
  -h, --help            show this help message and exit
  -m PATHS, --magic=PATHS
                        A colon separated list of magic files to use
  --json                Format output in JSON
```

One or more files can be passed to be identified. The textual description, mimetype and encoding type will be printed beneath each file's name.:

```
$ python -m magic.command setup.py
setup.py
    Python script, ASCII text executable
    text/x-python
    us-ascii
```

The output can also be rendered in machine parseable JSON instead of the simple textual description of above..

```
$ python -m magic.command --json setup.py
{
  "setup.py": {
    "textual": "Python script, ASCII text executable",
    "mimetype": "text/x-python",
    "encoding": "us-ascii"
  }
}
```

The `magic.command` module is not intended to be a replacement for the *file* command.

2.3 The filemagic API

Importing the `magic` module provides access to all *filemagic* primitives. Most importantly the `Magic` class.

2.3.1 Exceptions

If something goes with *libmagic*, an exception will be raised.

exception `magic.api.`**`MagicError`**(*errno*, *error*)
`errno` is the numerical error code returned by *libmagic*. `error` is the textual description of that error code, as supplied by *libmagic*.

`MagicError` inherits from `EnvironmentError`.

2.3.2 Classes

The `Magic` class supports context managers, meaning it can be used with the `with` statement. Using the `with` statement is the recommended usage as failing to call `close()` will leak resources. See *Usage* for guidance.

class `magic.`**`Magic`**([*paths*, *flags*])
Instances of this class provide access to *libmagics*'s file identification capabilities. Multiple instances may exist, each instance is independant from the others.

To supply a custom list of magic database files instead of letting libmagic search the default paths, supply a list of filenames using the paths argument. These filenames may be unicode string as described in *Memory management*.

By default *flags* is `magic.MAGIC_MIME_TYPE` which requests default behaviour from *libmagic*. This behaviour can be controlled by passing alternative *Constants* for flags.

`id_filename`(*filename*)
Identify a file from a given filename. The file will be opened by *libmagic*, reading sufficient contents to complete the identification.

`id_buffer`(*buffer*)
Identify a file from the contents of a string or buffer.

`close`()
Release any resources held by *libmagic*. This will be called automatically when a context manager exists.

`list`()
Prints a list of magic entries to standard out. There is no return value. It's mostly intended for debugging.

`consistent`
This property will be `True` if the magic database files loaded by libmagic are consistent.

This class encapsulates the low level ctypes api from `magic.api` that interfaces directly with *libmagic*. It's not expected that the user would want to do this.

If you do not know if *libmagic* is available, refer to the *Installation* section of the guide.

2.3.3 Constants

`magic.`**`MAGIC_NONE`**
Default flag for `magic.Magic` that requests default behaviour from *libmagic*.

`magic.`**`MAGIC_MIME_TYPE`**
Supply to `magic.Magic` constructor to return mime type instead of textual description.

`magic.`**`MAGIC_MIME_ENCODING`**
Supply to `magic.Magic` constructor to return mime encoding instead of textual description.

CHAPTER 3

Issues

If you encounter problems, please refer to *Reporting issues* from the guide.