mfr

Release 0.24.2

Feb 25, 2018

Contents

Release v0.24.2. (*Installation*)

mfr (short for "modular file renderer") is a Python package for rendering files to HTML.

CHAPTER 1

Ready to dive in?

Go to the *Quickstart tutorial*, check out some *examples* for code, or see the *Overview* to understand its architecture.

CHAPTER 2

Guide

2.1 Installation

mfr is actively developed on Github.

You can clone the public repo:

```
git clone https://github.com/CenterForOpenScience/modular-file-renderer.git
```

Or download one of the following:

- tarball
- zipball

Make sure that you have installed pspp and are using python 3.5 or greater.

Install the versions of `setuptools` and `invoke` found in the requirements.txt file:

```
pip install setuptools==30.4.0
pip install invoke==0.13.0
```

Install requirements:

```
invoke install
```

Or for some nicities (like tests):

```
invoke install --develop
```

Start the server:

```
invoke server
```

2.2 Quickstart

```
with open('mycode.img') as filepointer:
    handler = mfr.detect(filepointer)  # returns a handler object
    if handler:
        render_result = handler.render(filepointer)
    else:
        content = '<p>Cannot render file.</p>'
        render_result = mfr.RenderResult(content=content)
```

You can also use `mfr.render` to perform detection and rendering simultaneously. If no valid handler for a file is available, a `ValueError` is raised.

This example is equivalent to above.

```
with open('mycode.img') as filepointer:
    try:
        render_result = mfr.render(filepointer)
    except ValueError:  # No valid handler available
        render_result = mfr.RenderResult('<p>Cannot render file.</p>')
```

RenderResult objects contain the resultant html as content. Any javascript or css assets are contained in a dictionary. To display assets with jinja, simply iterate through the lists.

```
{% for stylesheet in render_result.assets.css %}
    <link rel="stylesheet" href={{ stylesheet }}/>
{%  endfor %}

{% for javascript in render_result.assets.js %}
    <script type="text/javascript" src={{ javascript }}/>
{%  endfor %}

{{ render_result.content|safe }}
```

2.2.1 Configuration

In order to use mfr in a web application, you will need to configure mfr with certain facts about the application, e.g. the base URL from which static files are served and the folder where to store static assets.

Configuration is stored on a `mfr.config`, which can be modified like a dictionary.

```
import mfr
import mfr_code_pygments

mfr.config['STATIC_URL'] = '/static'
mfr.config['STATIC_FOLDER'] = '/path/to/app/static'

# Filehandlers can be registered this way
mfr.config['HANDLERS'] = [mfr_code_pygments.Handler]
```

Note: The `mfr.config` shares the same API as Flask's config, so you can also load configuration values from files or Python objects.

```
import mfr
import mfr_code_pygments

# Equivalent to above
class MFRConfig:
    STATIC_URL = '/static'
    STATIC_FOLDER = '/path/to/app/static'
    HANDLERS = [mfr_code_pygments.Handler]

mfr.config.from_object(MFRConfig)
mfr.config['STATIC_URL']  # '/static'
```

2.2.2 Using Static Files

Many renderers require static files (e.g. CSS and Javascript). To retrieve the static files for a file renderer, the object has a 'assets_url' that serves as the base path.

2.2.3 Next Steps

That's it for the quickstart. For more complete examples, check out the *examples* page.

2.3 Examples

2.3.1 Example Usage with Flask

Below is an example Flask application that uses mfr.

app.py

```
from flask import Flask, url_for, send_from_directory, render_template

import mfr
import mfr_image
import os

app = Flask(__name__)

@app.route('/view/<filename>')
def view_file(filename):
    with open(os.path.join('path/to/uploads/', filename)) as fp:
        # Get first available handler for the file
        handler = mfr.detect(fp)
        if handler:
            # some renderers, e.g. the image renderer, require a src argument
            src = url_for('serve_file', filename=filename)
            render_result = handler.render(fp, src=src)
            return render_template('view_file.html', render_result=render_result)
        else:
            return 'Cannot render {filename}.'.format(filename=filename)

@app.route('/files/<filename>')
```

(continues on next page)

(continued from previous page)

```
def serve_file(filename):
    return send_from_directory(app.config['FILES_DIR'], filename)


def main():
    # Configure MFR with correct static URL and folder
    mfr.config.update({
        'STATIC_URL': app.static_url_path,
        'STATIC_FOLDER': app.static_folder,
        # Register handlers through config
        'HANDLERS': [mfr_image.Handler]
    })
    app.run(debug=True)

if __name__ == '__main__':
    main()
```

view_file.html

```
{% for stylesheet in render_result.assets.css %}
    <link rel="stylesheet" href={{ stylesheet }}/>
{%  endfor %}
{% for javascript in render_result.assets.js %}
    <script type="text/javascript" src={{ javascript }}/>
{%  endfor %}

{{ render_result.content|safe }}
```

2.4 Overview

Modular File Renderer (MFR) is a Python web application that provides a single interface for displaying many different types of files in a browser. If an iframe's `src` attribute is an MFR render url, MFR will return the html needed to display the image, document, object, etc.

There are three main categories of modules in MFR: *Handlers*, *Providers*, and *Extensions*. Handlers are the user-facing endpoints of MFR, accepting HTTP requests and returning either HTML or the file. Providers are responsible for knowing how to fetch the metadata and content for a file, given a URL to it. Extensions convert files and construct HTML to make specific file types renderable in a browser.

2.4.1 Handlers

In MFR, **handlers** are the classes that handle the web requests made to MFR. The two most important handlers are the Render handler, which handles reequests to the `/render` endpoint, and the Export handler, which handles requests to the `/export` endpoint. There are also endpoints for handling static assets, but those will not be described here. See *mfr.server.app* and *mfr.server.core.ExtensionsStaticFileHandler* for those.

Base handler

The **base handler** extracts the `url` query parameter from the request, constructs an appropriate MFR Provider object, then asks the Provider to fetch the file metadata.

Render handler

The **Render handler** will construct an appropriate renderer using the Extension module that is mapped to the file's extension. Some renderers require the file contents be inserted inline (ex. code snippets needing syntax highlighting). Those will download the file via the Provider. Others will only need a url to the file, which the Extension renderer will be responsible for inserting. The output from the renderer will be cached if caching is enabled.

Export handler

The **Export handler** takes the `url` to the file and a `format` query parameter, and constructs an Extension exporter to convert the file into the requested format. For example, most browsers can't render `.docx` files directly, so the Extension exporter will convert it to a PDF. The Export handler can also cache results if caching is enabled.

2.4.2 Providers

The **Provider** is responsible for knowing how to take a url to a file and get both the content and metadata for that file.

Base provider

Does little except verifying that the url is hosted at a supported domain.

HTTP provider

Naive provider that infers file metadata (extension, type, etc.) from the url. Downloads by issuing GET request against the url.

OSF provider

Open Science Framework -aware provider that can convert the given url into a WaterButler url. WaterButler is a file action abstraction service that can be used to fetch metadata and download file contents. The OSF provider also knows how to pass through OSF credentials, to enforce read-access restrictions.

2.4.3 Extensions

Extensions are the modules that generate the HTML needed to render a given file type. They may also provide exporters if the file's native type is unrenderable and needs to be converted to another format suitable for browsers. Extension renderers inherit from *mfr.core.extension.BaseRenderer* and exporters inherit from *mfr.core.extension.BaseExporter*.

2.5 Code

2.5.1 Core

mfr.core.exceptions

exception mfr.core.exceptions.**DownloadError**(*message*, **args*, *download_url: str = ''*, *response: str = ''*, ***kwargs*)

Bases: *mfr.core.exceptions.ProviderError*

The MFR related errors raised from a *mfr.core.provider* and relating to downloads should inherit from DownloadError

exception mfr.core.exceptions.**DriverManagerError**(*message*, **args*, *namespace: str = ''*, *name: str = ''*, *invoke_on_load: bool = None*, *invoke_args: dict = None*, ***kwargs*)

Bases: *mfr.core.exceptions.PluginError*

exception mfr.core.exceptions.**ExporterError**(*message*, **args*, *exporter_class: str = ''*, ***kwargs*)

Bases: *mfr.core.exceptions.ExtensionError*

The MFR related errors raised from a *mfr.core.extension* and relating to exporting should inherit from ExporterError

exception mfr.core.exceptions.**ExtensionError**(*message*, **args*, *extension: str = ''*, ***kwargs*)

Bases: *mfr.core.exceptions.PluginError*

The MFR related errors raised from a *mfr.core.extension* should inherit from ExtensionError

exception mfr.core.exceptions.**MakeExporterError**(**args*, ***kwargs*)

Bases: *mfr.core.exceptions.UnsupportedExtensionError*

The MFR related errors raised from `mfr.core.utils.make_exporter` should inherit from MakeExporterError

exception mfr.core.exceptions.**MakeProviderError**(*message*, **args*, *code: int = 500*, ***kwargs*)

Bases: *mfr.core.exceptions.DriverManagerError*

Thrown when MFR can't find an applicable provider class. This indicates programmer error, so `code` defaults to `500`.

exception mfr.core.exceptions.**MakeRendererError**(**args*, ***kwargs*)

Bases: *mfr.core.exceptions.UnsupportedExtensionError*

The MFR related errors raised from a `mfr.core.utils.make_renderer` should inherit from MakeRendererError

exception mfr.core.exceptions.**MetadataError**(*message*, **args*, *metadata_url: str = ''*, *response: str = ''*, ***kwargs*)

Bases: *mfr.core.exceptions.ProviderError*

The MFR related errors raised from a *mfr.core.provider* and relating to metadata should inherit from MetadataError

exception mfr.core.exceptions.**PluginError**(*message*, **args*, *code=500*, ***kwargs*)

Bases: `waterbutler.core.exceptions.PluginError`

The MFR related errors raised from a plugin should inherit from PluginError

as_html()

exception mfr.core.exceptions.**ProviderError**(*message*, **args*, *provider: str = ''*, ***kwargs*)

Bases: *mfr.core.exceptions.PluginError*

The MFR related errors raised from a *mfr.core.provider* should inherit from ProviderError

exception mfr.core.exceptions.**RendererError**(*message*, **args*, *renderer_class: str = ''*, ***kwargs*)

Bases: *mfr.core.exceptions.ExtensionError*

The MFR related errors raised from a *mfr.core.extension* and relating to rendering should inherit from RendererError

exception mfr.core.exceptions.**SubprocessError**(*message*, **args*, *code: int = 500*, *process: str = ''*, *cmd: str = ''*, *returncode: int = None*, *path: str = ''*, ***kwargs*)

Bases: *mfr.core.exceptions.ExporterError*

The MFR related errors raised from a *mfr.core.extension* and relating to subprocess should inherit from SubprocessError

exception mfr.core.exceptions.**TooBigToRenderError**(*message*, **args*, *requested_size: int = None*, *maximum_size: int = None*, *code: int = 400*, ***kwargs*)

Bases: *mfr.core.exceptions.ProviderError*

If the user tries to render a file larger than a server specified maximum, throw a TooBigToRenderError.

exception mfr.core.exceptions.**UnsupportedExtensionError**(**args*, *code: int = 400*, *handler_type: str = ''*, ***kwargs*)

Bases: *mfr.core.exceptions.DriverManagerError*

When make_renderer and make_exporter fail, it's usually because MFR doesn't support that extension yet. This error inherits from DriverManagerError (since it's the DriverManager that trips this) and includes a handler_type argsument

mfr.core.extension

class mfr.core.extension.**BaseExporter**(*ext*, *source_file_path*, *output_file_path*, *format*)

Bases: object

export()

class mfr.core.extension.**BaseRenderer**(*metadata*, *file_path*, *url*, *assets_url*, *export_url*)

Bases: object

cache_result

file_required

Does the rendering html need the raw file content to display correctly? Syntax-highlighted text files do. Standard image formats do not, since an <img> tag only needs a url to the file.

render()

mfr.core.provider

class mfr.core.provider.**BaseProvider**(*request*, *url*)

Bases: object

Base class for MFR Providers. Requires `download` and `metadata` methods. Validates that the given file url is hosted at a domain listed in *mfr.server.settings.ALLOWED_PROVIDER_DOMAINS*.

NAME

download()

metadata()

class mfr.core.provider.**ProviderMetadata**(*name*, *ext*, *content_type*, *unique_key*, *download_url*)

Bases: object

serialize()

mfr.core.utils

mfr.core.utils.**make_exporter**(*name*, *source_file_path*, *output_file_path*, *format*)

Returns an instance of *mfr.core.extension.BaseExporter*

Parameters

- **name** (*str*) – The name of the extension to instantiate. (.jpg, .docx, etc)
- **source_file_path** (*str*) –
- **output_file_path** (*str*) –
- **format** (*str*) –

Return type *mfr.core.extension.BaseExporter*

mfr.core.utils.**make_provider**(*name*, *request*, *url*)

Returns an instance of *mfr.core.provider.BaseProvider*

Parameters

- **name** (*str*) – The name of the provider to instantiate. (osf)
- **request** –
- **url** (*dict*) –

Return type *mfr.core.provider.BaseProvider*

mfr.core.utils.**make_renderer**(*name*, *metadata*, *file_path*, *url*, *assets_url*, *export_url*)

Returns an instance of *mfr.core.extension.BaseRenderer*

Parameters

- **name** (*str*) – The name of the extension to instantiate. (.jpg, .docx, etc)
- **file_path** (*str*) –
- **url** (*str*) –
- **assets_url** (*str*) –
- **export_url** (*str*) –

Param *mfr.core.provider.ProviderMetadata* metadata:

Return type *mfr.core.extension.BaseRenderer*

mfr.core.utils.**sizeof_fmt**(*num*, *suffix='B'*)

2.5.2 Extensions

AudioRenderer

class mfr.extensions.audio.**AudioRenderer**(*metadata*, *file_path*, *url*, *assets_url*, *export_url*)

Bases: *mfr.core.extension.BaseRenderer*

TEMPLATE = <mako.template.Template object>

cache_result

file_required
Does the rendering html need the raw file content to display correctly? Syntax-highlighted text files do. Standard image formats do not, since an <img> tag only needs a url to the file.

render()

CodePygmentsRenderer

class mfr.extensions.codepygments.**CodePygmentsRenderer**(**args*, ***kwargs*)
Bases: *mfr.core.extension.BaseRenderer*

DEFAULT_LEXER
alias of pygments.lexers.special.TextLexer

TEMPLATE = <mako.template.Template object>

cache_result

file_required
Does the rendering html need the raw file content to display correctly? Syntax-highlighted text files do. Standard image formats do not, since an <img> tag only needs a url to the file.

render()

DocxRenderer

class mfr.extensions.docx.**DocxRenderer**(**args*, ***kwargs*)
Bases: *mfr.core.extension.BaseRenderer*

TEMPLATE = <mako.template.Template object>

cache_result

file_required
Does the rendering html need the raw file content to display correctly? Syntax-highlighted text files do. Standard image formats do not, since an <img> tag only needs a url to the file.

render()

ImageExporter

ImageRenderer

IpynbRenderer

class mfr.extensions.ipynb.**IpynbRenderer**(**args*, ***kwargs*)
Bases: *mfr.core.extension.BaseRenderer*

TEMPLATE = <mako.template.Template object>

cache_result

file_required
Does the rendering html need the raw file content to display correctly? Syntax-highlighted text files do. Standard image formats do not, since an <img> tag only needs a url to the file.

render()

PdbRenderer

Note: This module requires jquery on the page in which it is loaded.

class mfr.extensions.pdb.**PdbRenderer**(*metadata*, *file_path*, *url*, *assets_url*, *export_url*)

Bases: *mfr.core.extension.BaseRenderer*

TEMPLATE = <mako.template.Template object>

cache_result

file_required

Does the rendering html need the raw file content to display correctly? Syntax-highlighted text files do. Standard image formats do not, since an <img> tag only needs a url to the file.

render()

PdfRenderer

RstRenderer

class mfr.extensions.rst.**RstRenderer**(**args*, ***kwargs*)

Bases: *mfr.core.extension.BaseRenderer*

TEMPLATE = <mako.template.Template object>

cache_result

file_required

Does the rendering html need the raw file content to display correctly? Syntax-highlighted text files do. Standard image formats do not, since an <img> tag only needs a url to the file.

render()

TabularRenderer

class mfr.extensions.tabular.**TabularRenderer**(*metadata*, *file_path*, *url*, *assets_url*, *export_url*)

Bases: *mfr.core.extension.BaseRenderer*

TEMPLATE = <mako.template.Template object>

cache_result

file_required

Does the rendering html need the raw file content to display correctly? Syntax-highlighted text files do. Standard image formats do not, since an <img> tag only needs a url to the file.

render()

VideoRenderer

class mfr.extensions.video.**VideoRenderer**(*metadata*, *file_path*, *url*, *assets_url*, *export_url*)

Bases: *mfr.core.extension.BaseRenderer*

TEMPLATE = <mako.template.Template object>

cache_result

file_required
Does the rendering html need the raw file content to display correctly? Syntax-highlighted text files do. Standard image formats do not, since an <img> tag only needs a url to the file.

render()

2.5.3 Providers

mfr.providers.http

class mfr.providers.http.**HttpProvider**(*request*, *url*)
Bases: *mfr.core.provider.BaseProvider*

Basic MFR provider. Infers file metadata (extension, type) from the url. Downloads by issuing a GET to the url.

NAME = 'http'

download()

metadata()

mfr.providers.osf

class mfr.providers.osf.**OsfProvider**(*request*, *url*)
Bases: *mfr.core.provider.BaseProvider*

Open Science Framework (https://osf.io) -aware provider. Knows the OSF ecosystem and can request specific metadata for the file referenced by the URL. Can correctly propagate OSF authorization to verify ownership and permisssions of file.

NAME = 'osf'

UNNEEDED_URL_PARAMS = ('_', 'token', 'action', 'mode', 'displayName')

download()
Download file from WaterButler, returning stream.

metadata()
Fetch metadata about the file from WaterButler. V0 and V1 urls must be handled differently.

2.5.4 Server

mfr.server.app

mfr.server.app.**make_app**(*debug*)

mfr.server.app.**serve**()

mfr.server.app.**sig_handler**(*sig*, *frame*)

mfr.server.handlers

class mfr.server.handlers.core.**BaseHandler**(**args*, ***kwargs*)
Bases: *mfr.server.handlers.core.CorsMixin*, tornado.web.RequestHandler, raven.contrib.tornado.SentryMixin

Base class for the Render and Export handlers. Fetches the file metadata for the file indicated by the `url` query parameter and builds the provider caches. Also handles writing output and errors.

NAME

bytes_written = 0

log_exception(*typ*, *value*, *tb*)
Override to customize logging of uncaught exceptions.

By default logs instances of *HTTPError* as warnings without stack traces (on the `tornado.general` logger), and all other exceptions as errors with stack traces (on the `tornado.application` logger).

New in version 3.1.

on_finish()
Called after the end of a request.

Override this method to perform cleanup, logging, etc. This method is a counterpart to *prepare*. `on_finish` may not produce any output, as it is called after the response has been sent to the client.

prepare()
Builds an MFR provider instance, to which it passes the the `url` query parameter. From that, the file metadata is extracted. Also builds cached waterbutler providers.

write_error(*status_code*, *exc_info*)
Override to implement custom error pages.

`write_error` may call *write*, *render*, *set_header*, etc to produce output as usual.

If this error was caused by an uncaught exception (including HTTPError), an `exc_info` triple will be available as `kwargs["exc_info"]`. Note that this exception may not be the "current" exception for purposes of methods like `sys.exc_info()` or `traceback.format_exc`.

write_stream(*stream*)

class mfr.server.handlers.core.**CorsMixin**
Bases: object

options()

set_default_headers()

class mfr.server.handlers.core.**ExtensionsStaticFileHandler**(*application*, *request*, ***kwargs*)
Bases: tornado.web.StaticFileHandler, *mfr.server.handlers.core.CorsMixin*

Extensions static path definitions

get(*module_name*, *path*)

initialize()
Hook for subclass initialization.

A dictionary passed as the third argument of a url spec will be supplied as keyword arguments to initialize().

Example:

```
class ProfileHandler(RequestHandler):
    def initialize(self, database):
        self.database = database

    def get(self, username):
        ...

app = Application([
    (r'/user/(.*)', ProfileHandler, dict(database=database)),
    ])
```

class mfr.server.handlers.export.**ExportHandler**(**args*, ***kwargs*)
Bases: *mfr.server.handlers.core.BaseHandler*

ALLOWED_METHODS = ['GET']

NAME = 'export'

get()
Export a file to the format specified via the associated extension library

prepare()
Builds an MFR provider instance, to which it passes the the `url` query parameter. From that, the file metadata is extracted. Also builds cached waterbutler providers.

class mfr.server.handlers.render.**RenderHandler**(**args*, ***kwargs*)
Bases: *mfr.server.handlers.core.BaseHandler*

ALLOWED_METHODS = ['GET']

NAME = 'render'

get()
Return HTML that will display the given file.

prepare()
Builds an MFR provider instance, to which it passes the the `url` query parameter. From that, the file metadata is extracted. Also builds cached waterbutler providers.

class mfr.server.handlers.status.**StatusHandler**(*application*, *request*, ***kwargs*)
Bases: tornado.web.RequestHandler

get()
List information about modular-file-renderer status

CHAPTER 3

Project info

3.1 Contributing guidelines

3.1.1 In general

- PEP 8, when sensible.
- Test ruthlessly. Write docs for new features.
- Even more important than Test-Driven Development–*Human-Driven Development*.
- If you add an extension to setup.py, add it to supportedextensions.md.
- Please update AUTHORS.rst when you contribute.

3.1.2 In particular

Setting up for development

Clone the repo:

```
$ git clone https://github.com/CenterForOpenScience/modular-file-renderer.git
$ cd modular-file-renderer
```

Configure development environment and install the development dependencies.

Note: It is recommended that you use a virtualenv with virtualenvwrapper during development. Python 3.5 or greater, R, and pspp are required.

```
# For Mac OS X: Install the latest version of python3.5
$ brew install python3
$ brew install r pspp

# Linux users, probably the same thing but with apt-get
# If someone wants to update this guide, please do.

$ pip install virtualenv
$ pip install virtualenvwrapper
$ mkvirtualenv --python=`which python3` mfr
$ pip install setuptools==30.4.0
$ pip install invoke==0.13.0
```

Lastly, install mfr in development mode.

```
$ invoke install -d
$ invoke server
```

Running tests

To run all tests (requires pytest)

```
$ invoke test
```

You can also use pytest directly.

```
$ py.test
```

Writing tests

Unit tests should be written for all rendering code.

Tests should be encapsulated within a class and written as functions, like so:

```
# in test_myformat.py

from mfr_something import render


def test_render_html():
    with open('testfile.mp4') as fp:
        assert render.render_html(fp) == '<p>rendered testfile.mp4</p>'
```

There are a few pytest fixtures to help you mock files. You can use them by simply including them as parameters to your test functions. For example, the `fakefile` fixture is a fake file-like object whose name and content you can set to any value.

The above test can be rewritten like so:

```
# in test_myformat.py

from mfr_something import render

def test_render_html(fakefile):
    assert render.render_html(fakefile) == '<p>rendered testfile.mp4</p>'
```

Manual Local Testing

To make sure a new renderer is functioning properly, it's recommended that you try to render a file of that type locally.

First, change the default provider to HTTP (in */mfr/server/settings.py*), then update the provider domain in the `ALLOWED_PROVIDER_DOMAINS` whitelist (a space-separated string):

```
PROVIDER_NAME = config.get('PROVIDER_NAME', 'http')
ALLOWED_PROVIDER_DOMAINS = config.get('ALLOWED_PROVIDER_DOMAINS', 'http://
↪localhost:8000/')
```

Because the MFR is passed a url to render, you also need to be running an http server.

From a directory with a file you want to render:

```
python -m SimpleHTTPServer 8000
```

Or for python 3

```
python3 -m http.server 8000
```

With both the SimpleHTTPServer and the MFR server running, go to

```
http://localhost:7778/render?url=http://localhost:8000/[filename].[ext]
```

Writing A File Format Package

There are two main pieces of a file format package are

- Your custom rendering and/or exporting code
- Your `FileHandler`

Rendering Code

Renderers are simply callables (functions or methods) that take a file as their first argument and return

Here is a very simple example of function that takes a filepointer and outputs a render result with an HTML image tag.

```
def render_img_tag(filepointer):
    filename = filepointer.name
    content = '<img src="{filename}" />'.format(filename=filename)
    return RenderResult(content)
```

You can also write renderers as methods.

```
# in mfr_video/render.py

class VideoRenderer(object):

    def render_html5_tag(self, fp):
        content = '<video src="{filename}"></video>'.format(filename=fp.name)
        return RenderResult(content)

    def render_flash(self, fp):
        # ...
        pass
```

The FileHandler

A file handler is responsible for using your custom rendering and exporting code to actually render and export a file. When you call `mfr.detect`, you receive a list of `FileHandler` classes.

Your FileHandler **must** define a `detect` method which, given a file object, returns whether or not it can handle the file.

Your FileHandler class should be named Handler and should be defined in your 'mfr_format/__init__.py' file.

```
# in mfr_image/__init__.py

from mfr import FileHandler, get_file_extension

# Your custom code
from mfr_image.render import render_img_tag
from mfr_image.export import ImageExporter


class Handler(FileHandler):
    renderers = {
        'html': render_img_tag,
    }

    exporters = {
        'png': ImageExporter().export_png,
        'jpg': ImageExporter().export_jpg,
        # ...
    }

    def detect(self, fp):
        return get_file_extension(fp.name) in ['.jpg', '.png', ]  # and so on
```

Organization

Each package has its own directory. At a minimum, your package should include:

- `__init__.py`: Where your `FileHandler <mfr.core.FileHandler>`` subclass will live.
- `render-requirements.txt`: External dependencies for rendering functionality.
- `export-requirements.txt`: External dependencies for export functionality.

Apart from those files, you are free to organize your rendering and export code however you want.

A typical extension plugin directory structure might look like this:

```
modular-file-renderer
├── mfr
│       ├── __init__.py
│       └── extensions
│               ├── __init__.py
│               └── custom-plugin
│                       ├── __init__.py
│                       ├── render.py
│                       ├── export.py
│                       ├── settings.py
│                       ├── static
```

(continues on next page)

(continued from previous page)

```
│                         │       ├── css
│                         │       └── js
│                         ├── templates
│                         │       └── viewer.mako
│                         └── libs
│                                 ├── __init__.py
│                                 └── tools.py
├── tests
│       ├── __init__.py
│       └── extensions
│               ├── __init__.py
│               └── custom-plugin
│                       ├── __init__.py
│                       └── test_custom_plugin.py
├── setup.py
├── README.md
└── requirements.txt
```

Documentation

Contributions to the documentation are welcome. Documentation is written in reStructured Text (rST). A quick rST reference can be found here. Builds are powered by Sphinx.

To build docs:

```
$ pip install -r doc-requirements.txt
$ cd docs
$ make html
$ open _build/html/index.html
```

The `-b` (for "browse") automatically opens up the docs in your browser after building.

3.2 Credits

3.2.1 Contributors

- Steven Loria @sloria
- Austin Macdonald @asmacdo
- Alex Schiller @alexschiller
- Kurtis Jungersen @kmjungersen
- Nezar Abdennur @nvictus
- Elijah Hamovitz @Hamms
- Tom Baxter @TomBaxter
- Joshua Carp @jmcarp
- Eric Bower @neurosnap
- Stefanie Schirmer @linse
- Meli Lewis @meli-lewis

- Barrett Harber @binoculars
- Lyndsy Simon @lyndsysimon
- Jeffrey Spies @JeffSpies
- Matt Frazier @mfraezz
- Casey Rollins @caseyrollins
- Michael Haselton @icereval
- Megan Kelly @megankelly
- Chris Seto @chrisseto
- Fitz Elliott @felliott
- Erin Braswell @erinspace
- Rafael de Lucena Valle @rafaeldelucena
- Matthew Keitelman @zamattiac
- John Tordoff @Johnetordoff
- Addison Schiller @AddisonSchiller
- Longze Chen **'@cslzchen https://github.com/cslzchen'_**
- Jonathon Love **'@jonathon-love https://github.com/jonathon-love'_**

3.3 Change Log

```
*********
ChangeLog
*********

0.24.2 (2018-02-09)
===================
- Fix: Update UNOCONV_BIN setting to reflect the default pip install path.

0.24.1 (2018-02-09)
===================
- Fix: Specifically install LibreOffice 6.0.1.1 and install unoconv 0.8.2.

0.24.0 (2018-02-01)
===================
- Feature: MFR now renders tiff files containing multiple images!  The file will be␣
↪exported to a
multi-page PDF, with one image per page.  (thanks, @AddisonSchiller!)
- Feature: Matlab data files (.mat) are now rendered using the tabular formatter. ␣
↪(thanks,
@AddisonSchiller!)
- Feature: The 3D-object renderer now supports STEP (.step and .stp) files.  (thanks,
@AddisonSchiller!)
- Fix: Don't send invalid payloads to MFR's metrics-tracking service.  (thanks,␣
↪@AddisonSchiller!)
- Fix: Don't reload the entire page when clicking on a tab header in the tabular␣
↪renderer.
- Code: Upgrade Pillow dependency.  (thanks, @AddisonSchiller!)
```

(continues on next page)

(continued from previous page)

```
- Code: Upgrade MFR's pym.js version to latest.  (thanks, @johnetordoff!)
- Code: Upgrade Codepygments dependency to get the newest code highlighters.  (thanks,
@AddisonSchiller!)
- Code: Support rendering videos, PDFs, PDBs, and 3D objects in a local development␣
↪environment.
(thanks, @TomBaxter!)
- Docs: Add a guide for using MFR with the OSF via docker-compose.  (thanks,␣
↪@AddisonSchiller!)

0.23.1 (2018-01-25)
===================
- Fix: Update the Jamovi renderer to handle images with spaces in their name. ␣
↪(thanks,
@jonathon-love!)

0.23.0 (2018-01-19)
===================
- Feature: Add a renderer for zip files!  Viewing a zip file in MFR will list the␣
↪names, modified
dates, and sizes for all files in the zipball.  (thanks, @johnetordoff!)
- Feature: Layered image files from Adobe Photoshop (*.psd) are now rendered by the␣
↪image renderer.
(thanks, @AddisonSchiller!)
- Feature: Stata dataset files (.dta) are now supported by the tabular file renderer,␣
↪similar to
csv, tsv, and excel spreadsheets.  (thanks, @AddisonSchiller!)
- Feature: Add a renderer for .omv files created in the Jamovi open statistical␣
↪spreadsheet program.
(thanks, @jonathon-love!)
- Feature: Add two new endpoints (/renderers and /exporters) to provide a␣
↪programmatic view of MFR's
file type support.  Both endpoints return a JSON object mapping extensions to the␣
↪renderer or
exporter that handles them.  (thanks, @johnetordoff!)
- Feature: Add a static list of supported file types (`supportedextensions.md`) to␣
↪the project
repository.  Includes a test to warn developers when they've failed to document a␣
↪newly-added file
type.  (thanks, @AddisonSchiller!)
- Feature: Add basic sandboxing to iframes.  (thanks, @AddisonSchiller!)
- Fix: Update the MathJax CDN url in the IPython notebook renderer.  (thanks,␣
↪@AddisonSchiller!)
- Fix: Support uppercased extensions (e.g. .CSV) for tabular files.  (thanks,␣
↪@AddisonSchiller!)
- Fix: Links in a pdf inside an iframe now open in a new tab instead of the iframe. ␣
↪(thanks,
@AddisonSchiller!)
- Code: Catch OSF metadata request failures earlier and report them accurately. ␣
↪(thanks,
@TomBaxter!)
- Code: Tidy up and reduce layers in the Dockerfile  (thanks, @binoculars!)

0.22.0 (2017-10-10)
===================
- Feature: Added support for rendering ~50 new plain-text file types via codepygments,
↪ including
FASTA files, ImageJ macros, and Turtle files. (thanks, @AddisonSchiller!)
```

(continues on next page)

(continued from previous page)

```
- Feature: Add a unique request ID to MFR's response headers to help with tracking␣
↪down errors.
- Feature: Add a new task to clean the export and render caches. (thanks, @icereval!)
- Fix: Error more gracefully when missing required query parameters.
- Fix: Make scrollbars visible on IE11 when content overflows the iframe.
- Fix: Fix ALLOWED_PROVIDER_DOMAINS example in MFR docs. (thanks, @jonathon-love for␣
↪reporting!)
- Code: Teach MFR to listen for a SIGTERM signal and exit immediately upon receiving␣
↪it. This
bypasses the 10 second wait for shutdown when running it in Docker.
- Code: Add code-coverage checking via coveralls.io. (thanks, @abought!)
- Code: Add Python 3.6 to travis testing matrix. (thanks, @abought!)
- Code: Add a Pull Request template for GitHub. (thanks, @cslzchen!)

0.21.2 (2017-09-13)
===================
- Fix: Update jQuery onload invocation to be compatible with jQuery 3.  (thanks,␣
↪@sloria!)

0.21.1 (2017-07-20)
===================
- Fix: Quiet some overly-verbose error logging.

0.21.0 (2017-04-07)
===================
- Feature: Turn on scrolling for the MFR iframe to support wide JASP files.
- Fix: Fix rendering of Google Drawing (.gdraw) files by directing them to the image␣
↪renderer
rather than the unoconv renderer.

0.20.1 (2017-03-02)
===================
- Fix: Cast OSF file size metadata to an int before comparing to our maximum␣
↪supported file size
limit.  Some providers return file size as a string instead of int.

0.20.0 (2017-03-01)
===================
- The "(thanks, @johnetordoff!)" release
- Feature: The tabular spreadsheet renderer now recognizes date fields as dates and␣
↪will format
them as such. (thanks, @johnetordoff!)
- Feature: Don't even try to render tabular files larger than 100Mb. Neither the␣
↪server nor the
browser wants that. (thanks, @johnetordoff!)
- Feature: Render a better error message when encountering a csv file with a single␣
↪field larger
than ~128kb.  The underlying library can't handle that, so it's polite to let the␣
↪user know.
(thanks, @johnetordoff!)
- Feature: MFR will now render .m4v files using the <video> tag. (thanks,␣
↪@johnetordoff!)
- Fix: Improve tooltip language on 3d object-renderer. (thanks, @johnetordoff!)
- Code: Start depending on upstream xlrd instead of our own fork. (thanks,␣
↪@johnetordoff!)

0.19.1 (2017-02-21)
```

(continues on next page)

(continued from previous page)

```
==================
- Fix: explicitly depend on IPython to fix ipynb rendering.  nbconvert and nbformat␣
↪have an
undeclared dependency on it. (thanks, @johnetordoff!)

0.19.0 (2017-02-02)
===================
- Feature: MFR errors are now categorized and have error-specific metadata added to␣
↪then.  If Keen
logging is enabled, the error metadata will be logged there for future investigation.
- Fix: The 3D object renderer now imposes a maximum zoom-out limit, so objects can't␣
↪be shrunk to
infinitesimalness. (thanks, @johnetordoff!)
- Fix: MFR docs are once again building on readthedocs.org! (thanks, @johnetordoff!)
- Code: Update MFR to use invoke 0.13.0. If you have an existing checkout, you will␣
↪need to
upgrade invoke manually: pip install invoke==0.13.0 (thanks, @johnetordoff!)
- Docs: MFR has been verified to work with python 3.5.3 and 3.6.0. From now on, the␣
↪docs will
mention which python versions MFR has been verfied to work on. (thanks, @johnetordoff!
↪)

0.18.3 (2017-01-11)
===================
- Fix: Increase max codepygments render size to 200kb from 64kb.

0.18.2 (2017-01-04)
===================
- Happy New Year!
- Fix: Be more ruthless about fixing setuptools breakage in Dockerfile. (thanks,␣
↪@cwisecarver!)

0.18.1 (2016-12-13)
===================
- Pin setuptools to v30.4.0 to avoid package-namespace-related breakage.

0.18.0 (2016-10-31)
===================
- HALLOWEEN RELEASE!
- Feature: Add configurable size limit to text renderer to avoid dumping 100s of MB␣
↪of text into
the user's browser. (thanks, @TomBaxter!)
- Fix: Pad MFR 404 errors to >512 bytes.  IE discards 404 messages smaller than that␣
↪and
substitutes its own 404 page. (thanks, @alexschiller!)

0.17.0 (2016-10-11)
===================
- Feature: WaterButler accepts configuration from the environment, overriding any␣
↪file-based
configuration.  This helps MFR integrate nicer in a docker-compose environment.␣
↪(thanks, @icereval!)
- Fix: Fix pdf presentation mode on Safari. (thanks, @darioncassel!)
- Fix: Fix aiohttp crashing on gzipped HEAD requests.
- Fix: Fix incorrect WB API usage metrics.
- Code: Bump raven dependency to 5.27.0.
```

(continues on next page)

(continued from previous page)

```
0.16.0 (2016-09-13)
===================
- Feature: MFR now does .sav conversion via pspp-convert instead of rpy2.
- Fix: Update ipython notebook renderer to use split-out nbconvert and nbformat␣
↪libraries.

0.15.0 (2016-08-25)
===================
- Feature: add analytics to MFR requests.  MFR now keeps track of requests, handlers,␣
↪renderers,
and exporters, with more to come!
- Fix: Better error handling for a number of edge cases. (thanks, @TomBaxter!)
- Docs: many fixes to doc build, layout, and formatting. (thanks, @TomBaxter!)
- Docs: add overview of MFR architecture to docs

0.14.0 (2016-06-24)
===================
- Feature: The Dockerfile now sets up unoconv for you.
- Fix: Character encodings for text files are now detected with the chardet library. ␣
↪Inspired
by a file that was both valid ISO-8859-1 and UTF-16 at the same time.

0.13.0 (2016-06-17)
===================
- Avoid an unnecessary lookup when MFR's OSF provider gets a WaterButler V1 url for
downloading. (thanks, @pattisdr!)
- Update the install docs to pin invoke to 0.11.1.

0.12.3 (2016-06-13)
===================
- Pin some dependencies and update our travis config to avoid spurious build failures.

0.12.2 (2016-06-13)
===================
- Add a Dockerfile to simplify running MFR in dev environments.
- Pin invoke to v0.11.1.  Our tasks.py is incompatible with v0.13.

0.12.1 (2016-05-31)
===================
- When an invalid provider is passed to MFR, HTML escape the url in the error message.

0.12.0 (2016-05-24)
===================
- MFR now requires python-3.5!  Make sure to set the SERVER_DEBUG flag to false in␣
↪your server
config to avoid the hated "throw() takes 2 positional arguments but 4 were given"␣
↪error.
- Tabular files now sort numish columns numerically!  (thanks, @mfraezz!)
- MFR correctly sets the Access-Control-Allow-Origin header when it receives a␣
↪request with
an Authorization header but no cookie. IOW it can now be used outside the OSF!␣
↪(thanks,
@samchrisinger!)
- Text files now wrap on Safari and Chrome. (thanks, @zamattiac!)

0.11.1 (2016-04-19)
===================
```

(continues on next page)

(continued from previous page)

```
- Require admin to set a whitelist of permitted provider domains, to avoid spamming␣
↪other sites
with requests.

0.11.0 (2016-04-08)
==================
- IPython notebooks are now styled and look **much** better. (thanks, @erinspace!)
- The OSF (https://osf.io) has moved to Open Sans as the default font, so we shall do␣
↪the same for
Markdown, ReStructred Text, and IPython notebooks. (thanks, @mfraezz!)
- COS is hiring! http://cos.io/jobs (thanks, @AndrewSallans!)
- Update copyright date. (thanks, monotonic progression of time!)

0.10.2 (2016-03-21)
==================
- Pin WaterButler version to v0.18, the last version using python-3.4.

0.10.1 (2016-03-14)
==================
- Fix bug in text encoding detector that was causing utf-8 to always detect as cp-
↪1252.

0.10.0 (2016-02-11)
==================
- Markdown and ReStructuredText files are now rendered in their formatted
display. (thanks, @TomBaxter!)
- ...oh, and they're styled, too! (thanks, @erinspace!)
- Update MFR install instructions. (thanks, @rafaeldelucena!)

0.9.6 (2016-02-01)
==================
- A few helpful tips for the user when rendering 3D objects

0.9.5 (2016-01-31)
==================
- Remove Sentry Ravent Client patch, latest version no longer requires
patching

0.9.3/4 (2016-01-30)
==================
- Updated Sentry Raven Client to send release w/ Tornado patch and tests

0.9.2 (2016-01-30)
==================
- Fix to provide IE10 support for JSC3D

0.9.1 (2016-01-30)
==================
- Fix to prevent JSC3D CTM Loader from removing download url query parameters

0.9.0 (2016-01-30)
==================
- Support for 3D model file formats (.stl, .3ds, .ctm, .obj) via jsc3d,
https://github.com/humu2009/jsc3d
- Support for .scad (OpenSCAD) syntax highlighting

0.8.4 (2016-01-27)
```

(continues on next page)

(continued from previous page)

```
==================
- Exclude .ico files from image scaling

0.8.3 (2016-01-26)
==================
- Support maximum dimensions for images
- Images larger than maximum dimensions are scaled down

0.8.2 (2015-11-13)
==================
- Remove invoke from requirements

0.8.1 (2015-11-13)
==================
- Manually garbage collect exceptions to work around issue in python 3.4

0.8.0 (2015-10-22)
==================
- Support word breaks in <pre> tags in Firefox.
- Codepygments extension: remove dependency on bootstrap css; add minimal
  styling to maintain look.

0.7.1 (2015-10-09)
==================
- Unpin numpy to avoid error in pandas.

0.7.0 (2015-10-08)
==================
- Add support for rendering JASP files
- Add support for searching tabular renderers

0.6.0 (2015-09-17)
==================
- Add support for cookie based authentication for pdfs and pdbs

0.1.0 (2015-06-07)
==================
- service oriented architecture (SOA)
- plugin system for providers (file resources)
- plugin system for extensions (renderers and exporters)
- embeddable rendering via iframe widget
- initial unit tests
```

3.4 License

```
                                 Apache License
                           Version 2.0, January 2004
                        http://www.apache.org/licenses/

   TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

   1. Definitions.
```

(continues on next page)

(continued from previous page)

```
"License" shall mean the terms and conditions for use, reproduction,
and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by
the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all
other entities that control, are controlled by, or are under common
control with that entity. For the purposes of this definition,
"control" means (i) the power, direct or indirect, to cause the
direction or management of such entity, whether by contract or
otherwise, or (ii) ownership of fifty percent (50%) or more of the
outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity
exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications,
including but not limited to software source code, documentation
source, and configuration files.

"Object" form shall mean any form resulting from mechanical
transformation or translation of a Source form, including but
not limited to compiled object code, generated documentation,
and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or
Object form, made available under the License, as indicated by a
copyright notice that is included in or attached to the work
(an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object
form, that is based on (or derived from) the Work and for which the
editorial revisions, annotations, elaborations, or other modifications
represent, as a whole, an original work of authorship. For the purposes
of this License, Derivative Works shall not include works that remain
separable from, or merely link (or bind by name) to the interfaces of,
the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including
the original version of the Work and any modifications or additions
to that Work or Derivative Works thereof, that is intentionally
submitted to Licensor for inclusion in the Work by the copyright owner
or by an individual or Legal Entity authorized to submit on behalf of
the copyright owner. For the purposes of this definition, "submitted"
means any form of electronic, verbal, or written communication sent
to the Licensor or its representatives, including but not limited to
communication on electronic mailing lists, source code control systems,
and issue tracking systems that are managed by, or on behalf of, the
Licensor for the purpose of discussing and improving the Work, but
excluding communication that is conspicuously marked or otherwise
designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity
on behalf of whom a Contribution has been received by Licensor and
subsequently incorporated within the Work.
```

(continues on next page)

(continued from previous page)

```
   2. Grant of Copyright License. Subject to the terms and conditions of
      this License, each Contributor hereby grants to You a perpetual,
      worldwide, non-exclusive, no-charge, royalty-free, irrevocable
      copyright license to reproduce, prepare Derivative Works of,
      publicly display, publicly perform, sublicense, and distribute the
      Work and such Derivative Works in Source or Object form.

   3. Grant of Patent License. Subject to the terms and conditions of
      this License, each Contributor hereby grants to You a perpetual,
      worldwide, non-exclusive, no-charge, royalty-free, irrevocable
      (except as stated in this section) patent license to make, have made,
      use, offer to sell, sell, import, and otherwise transfer the Work,
      where such license applies only to those patent claims licensable
      by such Contributor that are necessarily infringed by their
      Contribution(s) alone or by combination of their Contribution(s)
      with the Work to which such Contribution(s) was submitted. If You
      institute patent litigation against any entity (including a
      cross-claim or counterclaim in a lawsuit) alleging that the Work
      or a Contribution incorporated within the Work constitutes direct
      or contributory patent infringement, then any patent licenses
      granted to You under this License for that Work shall terminate
      as of the date such litigation is filed.

   4. Redistribution. You may reproduce and distribute copies of the
      Work or Derivative Works thereof in any medium, with or without
      modifications, and in Source or Object form, provided that You
      meet the following conditions:

      (a) You must give any other recipients of the Work or
          Derivative Works a copy of this License; and

      (b) You must cause any modified files to carry prominent notices
          stating that You changed the files; and

      (c) You must retain, in the Source form of any Derivative Works
          that You distribute, all copyright, patent, trademark, and
          attribution notices from the Source form of the Work,
          excluding those notices that do not pertain to any part of
          the Derivative Works; and

      (d) If the Work includes a "NOTICE" text file as part of its
          distribution, then any Derivative Works that You distribute must
          include a readable copy of the attribution notices contained
          within such NOTICE file, excluding those notices that do not
          pertain to any part of the Derivative Works, in at least one
          of the following places: within a NOTICE text file distributed
          as part of the Derivative Works; within the Source form or
          documentation, if provided along with the Derivative Works; or,
          within a display generated by the Derivative Works, if and
          wherever such third-party notices normally appear. The contents
          of the NOTICE file are for informational purposes only and
          do not modify the License. You may add Your own attribution
          notices within Derivative Works that You distribute, alongside
          or as an addendum to the NOTICE text from the Work, provided
          that such additional attribution notices cannot be construed
          as modifying the License.
```

(continues on next page)

(continued from previous page)

```
   You may add Your own copyright statement to Your modifications and
   may provide additional or different license terms and conditions
   for use, reproduction, or distribution of Your modifications, or
   for any such Derivative Works as a whole, provided Your use,
   reproduction, and distribution of the Work otherwise complies with
   the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise,
   any Contribution intentionally submitted for inclusion in the Work
   by You to the Licensor shall be under the terms and conditions of
   this License, without any additional terms or conditions.
   Notwithstanding the above, nothing herein shall supersede or modify
   the terms of any separate license agreement you may have executed
   with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade
   names, trademarks, service marks, or product names of the Licensor,
   except as required for reasonable and customary use in describing the
   origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or
   agreed to in writing, Licensor provides the Work (and each
   Contributor provides its Contributions) on an "AS IS" BASIS,
   WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
   implied, including, without limitation, any warranties or conditions
   of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A
   PARTICULAR PURPOSE. You are solely responsible for determining the
   appropriateness of using or redistributing the Work and assume any
   risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory,
   whether in tort (including negligence), contract, or otherwise,
   unless required by applicable law (such as deliberate and grossly
   negligent acts) or agreed to in writing, shall any Contributor be
   liable to You for damages, including any direct, indirect, special,
   incidental, or consequential damages of any character arising as a
   result of this License or out of the use or inability to use the
   Work (including but not limited to damages for loss of goodwill,
   work stoppage, computer failure or malfunction, or any and all
   other commercial damages or losses), even if such Contributor
   has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing
   the Work or Derivative Works thereof, You may choose to offer,
   and charge a fee for, acceptance of support, warranty, indemnity,
   or other liability obligations and/or rights consistent with this
   License. However, in accepting such obligations, You may act only
   on Your own behalf and on Your sole responsibility, not on behalf
   of any other Contributor, and only if You agree to indemnify,
   defend, and hold each Contributor harmless for any liability
   incurred by, or claims asserted against, such Contributor by reason
   of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

APPENDIX: How to apply the Apache License to your work.
```

(continues on next page)

(continued from previous page)

```
    To apply the Apache License to your work, attach the following
    boilerplate notice, with the fields enclosed by brackets "[]"
    replaced with your own identifying information. (Don't include
    the brackets!)  The text should be enclosed in the appropriate
    comment syntax for the file format. We also recommend that a
    file or class name and description of purpose be included on the
    same "printed page" as the copyright notice for easier
    identification within third-party archives.

Copyright [yyyy] [name of copyright owner]

Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at

    http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License.
```

Python Module Index

m

Index

U

V

W