XStatic Documentation

Release 1.0.0

Thomas Waldmann

Sep 18, 2018

Contents

CHAPTER 1

What is XStatic

1.1 The Idea

XStatic is a packaging standard to package external (often 3rd party) static files as a python package, so they are easily usable on all operating systems, with any package management system or even without one.

Many python projects need to use some specific data files, like javascript, css, java applets, images, etc.

Sometimes these files belong to YOUR project (then you may want to package them separately, but you could also just put them into your main package).

But in many other cases, those files are maintained by someone else (like jQuery javascript library or even much bigger js libraries or applications) and you definitely do not really want to merge them into your project.

So, you want to have static file packages, but you don't want to get lots of stuff you do not want. Thus, stuff required by XStatic file packages (especially the main, toplevel XStatic package) tries to obey to be a MINIMAL, no-fat thing.

We won't "sell" you any web framework or other stuff you don't want. Maybe there will be optional XStatic extensions for all sorts of stuff, but they won't be required if you just want the files.

By having static files in packages, it is also easier to build virtual envs, support linux/bsd/... distribution package maintainers and even windows installs using the same mechanism.

1.2 Pros

- can be put on PyPI (Python Package Index) and can get discovered there
- can be fetched and installed from PyPI automatically by just requiring it in your main project's setup
- you do not need to add 3rd party files to your repository or your distribution archives
- supports development / virtualenv / windows installs (where no other package management tools are available)
- less work for linux distribution package maintainers, you already have split your stuff into separate packages, so they don't need to

- outsource some work to other people. there are lots of people needing these static packages, so you don't need to maintain them all yourself.
- additionally to the files, you'll get some metadata (like version info, name, CDN URLs (if any).
- we can use version number of the package to reflect the version of the packaged static stuff and use the packaging system to require some specific version, or some specific minimum/maximum version.
- security updates are easier, the static file packages can be updated separately from your main package.

1.3 Cons

- it creates a little management overhead for the developer, especially if the xstatic file package you need does not exist yet - but packaging is very easy.

CHAPTER 2

Using XStatic

2.1 Using XStatic

The XStatic package does only offer the most fundamental functions for dealing with static files (and this is very much the point of XStatic: being low-fat).

The only bit of code is in XStatic.main, class XStatic.

When you instantiate an object of this class, it'll read the uppercase attributes from the xstatic module you give to it and make them available as lowercase instance attributes.

E.g. (we use the xstatic-jquery package as example, see also the code example below):

- xstatic.pkg.jquery.NAME -> xs.name
- xstatic.pkg.jquery.BASE_DIR -> xs.base_dir

Thus, you have all the metadata that came with the xstatic-jquery package easily available.

2.1.1 Example code to setup local file serving

```
from xstatic.main import XStatic
# names below must be package names
mod_names = [
    'jquery', 'bootstrap', 'font_awesome',
]
pkg = __import__('xstatic.pkg', fromlist=mod_names)
serve_files = {}
for mod_name in mod_names:
    mod = getattr(pkg, mod_name)
    xs = XStatic(mod, root_url='/static', provider='local', protocol='http')
    serve_files[xs.name] = xs.base_dir

# now, serve_files has the mapping name -> base_dir for all the xstatic
```

(continues on next page)

(continued from previous page)

```
# packages you want to use. you can use it in your python code to set
# up the static file serving.
```

In this example, we wanted to use the local static files we got within the xstatic-* packages.

For some packages there is also a CDN available, you can use it by giving the appropriate provider (not 'local') and protocol (see the xstatic-* package metadata about which cdnnames and protocols are available for the package):

:: xs = XStatic(mod, provider='cdnname', protocol='https') print xs.base_url

Note: base_url is often a str (as you maybe have expected). But it also can be a dict (which maps relative pathes to full urls) - we needed that for some CDNs where one can not just compute the full url from base url + relative path.

The Xstatic class also has a simple url_for(relative_path) method which computes the full url - for local URLs as well as for CDN URLs.

2.2 Packaging for XStatic

It's easy, no rocket-scientist needed.

We suggest you just take XStatic-jQuery package as a template and do these steps:

- Copy XStatic-jQuery to XStatic-FooBar (replace "FooBar" by the official name [display_name] of the project you are packaging).
- Rename xstatic/pkg/jquery package directory to xstatic/pkg/foobar (use simple all-lowercase name, only a..z - this must be a valid python package name [name]).
- Remove xstatic/pkg/foobar/data/* and place FooBar project's static files there.
- Edit xstatic/pkg/foobar/__init__.py and update all information there appropriately (see the comments there and also the hints below). Most stuff from there will get reused by setup.py.
- Edit setup.py:
  - You need to change the "from xstatic.pkg import ... as xs" appropriately to import your package.
  - Review the rest of it, but most stuff should be fine as it just reuses stuff from XStatic metadata.
- Edit MANIFEST.in and change the recursive-include statement there to refer to your files (xstatic/pkg/foobar), so that your static files will be included in the package created later.
- Edit README.txt and replace references to jQuery with FooBar. This file's content will also be shown as long description on PyPi. Please note that this file is written in reST markup, so that PyPi can generate your project's page from it.
- If you use Mercurial, update .hgignore so it ignores: ^XStatic_FooBar.egg-info/
- Review all the stuff you did, make sure you did not forget anything, make sure there is no jquery reference left.
- Run python setup.py sdist.
- Look into build/... - there should be your XStatic-FooBar package now.
- Test it locally:
  - E.g. use pip install XStatic-FooBar-1.0.0.tar.gz to install it.
  - Use it from your project, does it work?
- If you are happy with it and you think the package is also useful for many other Pythonistas, register and upload it to PyPi: python setup.py sdist register upload

2.3 Misc. Hints

2.3.1 Names

There are 2 names involved and you should follow these rules:

- package name (== metadata NAME): simple, all lowercase name. E.g. foobar or jquery. If you would have to use "-": please replace it by "_". Minus is not valid in Python package names, so use underscore so that you can use same name for your package directory / package name.
- DISPLAY_NAME (metadata): the name as the upstream project itself spells it, e.g. jQuery or FooBar. No spaces.

Note: if you are not packaging original files, but modified files, then you must use a name that makes this fact obvious.

2.3.2 Version Numbers

VERSION - as you are just repackaging another project, you should use the upstream version number (or at least something closely related to it).

Some projects do not have good version numbers, make the best of it:

E.g. upstream version: 2010-12-31, XStatic-FooBar version: 2010.12.31

BUILD - as you maybe do not get packaging right on the first try, you'll want to enumerate your builds: 0, 1, 2, ...

PACKAGE_VERSION - is automatically computed from VERSION . BUILD.

2.3.3 Which files to put into your package?

It is suggested that you only package files that are useful for Python projects, because XStatic packages will be only used by them. No need for PHP, ASP, Java, etc. related files.

If you package files that are somehow "compiled/compressed" versions, we suggest you only package the files needed for production usage, not the source code.

If the original download archive has the files needed for production in some subdirectory, we suggest you strip the directory hierarchy and just put the production files/directories into xstatic/pkg/foobar/data/.

2.3.4 CDN locations

If your files are available via a public CDN (Content Distribution Network), you can give the URLs via the locations metadata.

If you do not have a CDN for the files, just use locations = {}.

2.3.5 Licensing

You should put your XStatic-FooBar package under same license as the upstream FooBar package. This avoids licensing complications and is also appropriate because you only added a little metadata anyway.

2.4 Notes for Linux (or other OS) Package Maintainers

If you are maintaining packages for some other packaging system, like .deb or .rpm, this section is for you.

When designing XStatic stuff, we had YOU in mind! :)

But not only you, we also had in mind that there is no packaging system on Windows and that developers or virtualenv users rather like setuptools, distribute and pip.

Because of this, we did not want to rely on any mechanism that might be not available in some scenario - thus, after files are installed, we only use Python features (like importing from a installed python package, using __file__ to find out the path/filename, etc.).

You, as a package maintainer are interested in avoiding duplication, so that if you need to do a security update, you only need to fix in one place.

XStatic-* packages support this. If you do not want to heavily patch some Python software that uses XStatic ressource packages, you can alternatively just package the XStatic resource packages for your package system.

In case that would add duplication (because you already have a package that provides the same static files), you can simply remove the static files below data/ from the XStatic ressource package and adjust the path/filename so it points to the files provided by that other package.

E.g. for the XStatic-jQuery package, change:

```
BASE_DIR = join(dirname(__file__), 'data')
```

To:

```
BASE_DIR = '/usr/share/javascript/jquery'
```

Of course you need to make sure that the files at the location you point to are the same as the ones the XStatic ressource package provides below the data/ directory.

In your package dependencies for your repackaged XStatic ressource package you would then just require (depend on) the package providing these files.

CHAPTER 3

Indices and tables

- genindex
- modindex
- search