FREE eBook

LEARNING plone

Free unaffiliated eBook created from
Stack Overflow contributors.

#plone

Table of Contents

About

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: plone

It is an unofficial and free plone ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official plone.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Chapter 1: Getting started with plone

Remarks

This section provides an overview of what plone is, and why a developer might want to use it.

It should also mention any large subjects within plone, and link out to the related topics. Since the Documentation for plone is new, you may need to create initial versions of those related topics.

Examples

Installation or Setup

Detailed instructions on getting plone set up or installed.

Read Getting started with plone online: https://riptutorial.com/plone/topic/8657/getting-started-with-plone

Chapter 2: Create add-on

Examples

Overview file-structure

Minimum skeleton

```
mynamespace.myaddon     | The container-directory of the add-on.

    setup.py            | Register this directory to the Python-interpreter of
                        | the ZOPE-instance.

    mynamespace         | The namespace-directory of the add-on.

        __init__.py     | Makes this directory a Python-module and contains
                        | the lines for the namespace-magic.

        myaddon         | The base-directory of the add-on.

            __init__.py | Empty file, make this directory a Python-module.

            myfile.py   | Example-file: If it contained a method called
                        | 'myMethod', it would be importable of any other
                        | registered egg like this:
                        |    from mynamespace.myaddon.myfile import myMethod
```

Optional components

The 'mynamespace.myaddon/mynamespace/myaddon'-directory can contain:

```
configure.zcml  | Register all following files and directories, here.

setuphandlers.py | Make an addon available to ZOPE-instance.

profiles        | Make an addon installable to Plone-site and generally
                | hold GenericSetup-xml-files.

skins           | The "old-school"-way, where resources like templates,
                | stylesheets, etc., live.

browser         | The "new-school"-way, where resources live.

content         | Holds the Python-files for defining custom content-types.

subscriber.py   | Subscribers listen to events, describe here what to do
                | when an event happens, e.g.: "An item has been added to the
                | site, notify admin about it via e-mail-notifica."
```

A tale about creating a Plone add-on

Preamble

TL;TR: For illustration-purposes the lines of this document beginning with 'TL;TR:' are followed by commandline refering to Plone-helper-scripts of the egg 'adi.devgen'. If you execute the command given after 'TL;TR:', it will create all the files explained of the following chapter. There are many alternative helper-script-tools, however in case you want to use this tool, you can install it quickly like this of the commandline:

```
pip install adi.devgen
```

Structure of a minimal Plone-Add-on

TL;TR: `devgen addBase mynamespace.myaddon`

Just like Plone consists of more than 200 Python-eggs, a Plone-Add-On is also a Python-egg and an egg is a namespaced Python-module. Any egg will be available to the Python-interpreter of ZOPE. To make an egg installable in a Plone-site, it also needs a so called 'profile'. The following will explain these terms.

Python-module

A Python module is a directory which contains at least one file which must be named `__init__.py`. The name is a flag for the Python-interpreter to recognize this directory as a module, meaning its path can be imported of other Python-scripts, e.g. assuming we have a directory called 'myaddon', which contains a file called 'myfile' and that file contains a defintion called 'myDefinition', you can import the definition of another Python-file like this:

```
from myaddon.myfile import myDefinition
```

Python-egg

A Python-egg is a namespaced Python-module, a namespace can be any name, but it should better not be taken already, in case you want to share your add-on with the world.

The reason for namespacing is foremost to exclude the possibility that two or more modules could have the same name (in this example called 'myaddon'), as it would result in conflicts when a Python-script tries to import the path, being not unique anymore.

Here is an overview of the directory-structure and files involved, before we continue to explain their purposes in detail furtheron:

```
mynamespace.myaddon
    setup.py
    mynamespace
```

```
        __init__.py
        myaddon
            __init__.py
            myfile.py
```

To namespace an add-on firstly create a directory named 'mynamespace.myaddon'.

This directory must contain the 'setup.py'-file, it makes the egg registrable to the Python-interpreter of the ZOPE-instance, this is further explained in the next chapter[TODO].

Then, in that directory, create another directory named after your namespace: 'mynamespace' Now, we make this directory an importable module-path, by adding the keyword-named file `__init__.py` in it. It also must contain the following lines, which make the namespace-magic happen, explained in the last paragraph:

```
[TODO: namespace-magic-lines]
```

Phew almost done, lastly in this new directory we put the module of the previous chapter: The directory 'myaddon', containing the '**init**.py' and 'myfile.py' with the 'myDefinition'.

Now we have an registrable egg for the ZOPE-instance and could reference its methods of any other registered egg, using this path-notation:

```
from mynamespace.myaddon.myfile import myDefinition
```

You might see the magic, the first two directories are omitted, it is *not*:

```
from mynamespace.myaddon.mynamespace.myaddon.myfile import myDefinition
```

Python-interpreter

[TODO:

- Explain setup.py
- Add a previous chapter about buildout (=installing Plone) for reference.]

Making an addon installable within a Plone-site

A 'profile' will make the add-on show up in the add-ons-controlpanel of a site, so an admin can (de-)activate it there for the site.

Additionally, as a Plone-site is always a child of ZOPE-instance and a ZOPE-instance can contain several sites, we might not want to have unintendendly components of our add-on installed in other Plone-sites, therefore we bind them to a 'profile'.

TL;TR: `devgen addProfile mynamespace.myaddon`

This leads us to the really interesting parts, the ZOPE-Component-Architecture, controlled by the

files ending with '.zcml', which stands for 'ZOPE-Component- Markup-Language'. With it you can register a profile, bind views to interfaces and much more. In fact, it deserves an own major chapter and a subchapter for each of the directories called "profile", "skins", "browser" and "content".

Read Create add-on online: https://riptutorial.com/plone/topic/8751/create-add-on

Credits

S. No	Chapters	Contributors
1	Getting started with plone	Community
2	Create add-on	Ida Ebkes