Table of Contents

How to create an OpenAlea Package

Introduction

This tutorial describes how to create an OpenAlea package.

We distinguish:

Pure python packages
Extended python packages with C++ or shared library

Before starting, ensure you have a working development environment.

If you want to start directly with a real example, download the starter package.

Pure Python package

Layout

The standard layout for a pure Python package is :

```
your_package/
  AUTHORS.txt: List of project contributors.
  ChangeLog.txt: Modifications file by file at the developer level.
  LICENSE.txt: Package license terms.
  NEWS.txt: What are the modifications from the last release at the user level (
functional)?
  README.txt: General presentation of the package. Build and install
instructions.
  TODO.txt: What's next?
  setup.py : basic setup file for pure python package.

  #namespace is openalea, vplants, etc...
  src/namespace/your_package/
    __init__.py : python package initialisation file.
    your_modules.py : python modules.

  test/  : test subdirectory which contains test files.
  doc/ : documentation subdirectory
```

Installation system

To install OpenAlea packages, we use setuptools python installation system.

Adapt the `setup.py` as you do for a standard pure python package

OpenAlea package must installed in the *openalea namespace*

You can also use OpenAlea.Deploy in order to create the openalea namespace (if needed).

```
      import sys, os
      from setuptools import setup, find_packages
      from os.path import join as pj


      # Package name: MODIFY IT with the name of your package.
      name = 'xxxxxxx'

      namespace = 'openalea'
      pkg_name = namespace + '.' + name

      version = '0.0.0'

      # Short description of the package (1 line).
      description = 'xxxxxxxx xxxxxxxxxxx xxxxxxxxxx'
      long_description =
'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx'

      # Author
      author = 'XXXXXXXXXX XXXXXXXX'
      author_email = 'xxx.xxxx@xxxx.xx'
```

```
    # URL
    url = 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx'

    # License: License for the template package.
    # Please, choose an OpenSource license for distribution of your package.
    license = 'LGPL' # or 'GPL' or 'Cecill' or  'Cecill-C'

    setup(name = "your",
          version = version,
          description = description,
          long_description = long_description,
          author = author,
          author_email = author_email,
          license = license,
          namespace_packages = [namespace],
          create_namespaces = True,
          packages = [pkg_name],
          packages =  [ 'openalea.' + x for x in find_packages('src') ],
          package_dir = { 'openalea.xxxx':  pj('src','xxxx'),  },
    )
```

Extended Python package (with C++ and shared library)

The starter package is a simple package to help developers to create their own package with C++.

We recommend to download it and adapt it for your own project.

Requirements

The following packages are required. You can specify them as dependencies in your `setup.py`.

OpenAlea.Deploy
OpenAlea.SconsX
Boost.Python for the C++ Wrappers
If you have developed your package on Linux and on Windows with MinGW, you have some work to adapt your code for Visual Studio. You need to export symbols explicitly by adding a macro before the name of all your classes. See the starter package as an example.

Layout

The standard layout for a extended Python package is :

```
your_package/
  AUTHORS.txt: List of project contributors.
  ChangeLog.txt: Modifications file by file at the developer level.
  LICENSE.txt: Package license terms.
  NEWS.txt: What are the modifications from the last release at the user level
(functional)?
```

```
  README.txt: General presentation of the package. Build and install
instructions.
  TODO.txt: What's next?

  setup.py : basic setup file for pure python package.
  setup.cfg : external configuration for setup.py.
  SConstruct : scons script.
  options.py : external scons configuration
  src/
    cpp/ : contains C++ files (*.cpp) + a SConscript
    wrapper/ : contains boost.python wrappers (*.cpp) + a SConscript
    include/ : contains header files (*.h)
    namespace/your_package/  : contains the pythons files (*.py)
      __init__.py : python package initialization file.
      your_modules.py : python modules.
  test/  : test subdirectory which contains test files.
  doc/ : documentation subdirectory
```

Build system

Starter build is based on **SCons** and **SConsX**.

Adapt the `SConstruct` and `SConscript` files for your needs (replace the directory name `pkg_name` and replace source and target files).

SConstruct

This script determine the platform specific option by creating **SConS** environments. These options can be override in the `option.py` file.

```
# -*-python-*-
from openalea.sconsx import config, environ
import os
pj= os.path.join

Config= config.Config
ALEAConfig= config.ALEAConfig
ALEAEnvironment= config.ALEAEnvironment

name='starter'

SConsignFile()

options = Options( 'options.py', ARGUMENTS )

wrapper_conf= ALEAConfig(name,['boost_python', 'alea'])
cpp_conf= ALEAConfig(name, [])
opt_conf= ALEAConfig(name, ['boost_python', 'alea'])

# Set all the common options for the package
# TODO: Have a configure stage.
```

```
# Fill the options from file option.py or cmd line args.
opt_conf.UpdateOptions( options )

opt_env= Environment( options= options )
opt_conf.Update( opt_env )

# Generate Help available with the cmd scons -h
Help(options.GenerateHelpText(opt_env))


# Set build directory
prefix= opt_env['build_prefix']
BuildDir( prefix, '.' )


cpp_env= ALEAEnvironment( cpp_conf, 'options.py', ARGUMENTS )
wrapper_env= ALEAEnvironment( wrapper_conf, 'options.py', ARGUMENTS )
# Build stage
SConscript( pj(prefix,"src/cpp/SConscript"),
            exports={"env": cpp_env} )
SConscript( pj(prefix,"src/wrapper/SConscript"),
            exports={"env":wrapper_env} )

Default("build")
```

cpp/SConscript

This script define what are the library to build

```
# -*-python-*-
import os, re

Import( "env" )

# 1. Select and install the headers

include_dir = str(env.Dir( "../include" ).srcnode())
h_pattern = re.compile( r".*\.(h|hpp)$" )
includes = [ "../include/%s" % ( s, ) for s in os.listdir(include_dir) if
h_pattern.match( s ) ]

env.ALEAIncludes( "starter", includes )


# Build shared libraries

# 2. Build first library
sources= "scene_object.cpp"
target= "libsceneobject"

# Add defines to export symbols on Windows
DEFINES= list(env['CPPDEFINES'])
```

```
DEFINES.append('SCENEOBJ_DLL')

# Build the library
lib1 = env.ALEALibrary( target, sources, CPPDEFINES=DEFINES)

# 3. Build the second library
env2= env.Copy()
env2.AppendUnique(LIBS= ["libsceneobject"])

sources= ["scene_container.cpp"]
target= "libscenecontainer"

DEFINES= list(env['CPPDEFINES'])
DEFINES.append('SCENECONT_DLL')

lib2 = env2.ALEALibrary( target, sources, CPPDEFINES=DEFINES)
```

wrapper/SConscript

This script define what are the wrappers to build.

```
# -*-python-*-

Import( "env" )

py_dir = '../starter'

# Build wrappers as shared libraries
# First wrapper
env1=env.Copy()

sources= ["sceneobject_wrap.cpp", "export_scene_object.cpp"]
target= "_sceneobject"
lib1 = ["libsceneobject"]

env1.AppendUnique(LIBS=lib1)
env1.ALEAWrapper( py_dir, target, sources )

# Second wrapper
env2=env.Copy()

sources= ["scenecontainer_wrap.cpp", "export_scene_container.cpp"]
target= "_scenecontainer"
lib2 = ["libscenecontainer"]

env2.AppendUnique(LIBS=lib2)

env2.ALEAWrapper( py_dir, target, sources )
```

Installation system

Installation system uses **OpenAlea.Deploy** setuptools extension. Adapt the `setup.py` to your need.

In addition to the Pure Python package setup.py, we have the following lines :

```
...
build_prefix = 'build-scons'
...
setup()
    ...
    include_package_data = True,
    zip_safe= False,

    lib_dirs = { 'lib' : pj(build_prefix, 'lib'), },
    inc_dirs = { 'include' : pj(build_prefix, 'include') },
    share_dirs = { 'share' : 'share' },
    #postinstall_scripts = ['',],

    # Dependencies
    setup_requires = ['openalea.deploy'],
    dependency_links = ['http://openalea.gforge.inria.fr/pi'],
    #install_requires = [],
```

Create Distribution

You can create source and binary distribution directly with your `setup.py` configuration:

You can create a source distribution :

```
python setup.py sdist
```

You can create also a binary platform-dependent egg distribution :

```
python setup.py bdist_egg
```