

---

# **PyLunch Documentation**

***Release 0.2***

**Florent Aide**

February 20, 2009



# CONTENTS

<b>1</b>	<b>Foreword</b>	<b>1</b>
<b>2</b>	<b>Requirements</b>	<b>3</b>
<b>3</b>	<b>Sections</b>	<b>5</b>
3.1	Installation from source . . . . .	5
3.2	Basic Hands-on . . . . .	6
3.3	TurboGears2 support . . . . .	9



# FOREWORD

First of all let's say that we have decided from the beginning to have a scope as focused as possible. This means our goal is to support super easy deployment of python programs on the windows platform.

Being long time users of py2exe and before that of MacMillan's installer, we still needed two important features to fully be happy with the Windows platform:

- python eggs support
- portable application for windows

PyLunch is a windows solution for easily packaging python applications. Using this launcher you won't have to worry about creating complicated py2exe setup.py files. Just create your application as if you released an egg to the pypi or your own private index, tell PyLunch the name of your application, the index to use if it is a private one and your're done!

PyLunch even permits to create windows services to run WSGI application with the paster WSGI server.

We have tested and documented how to create a windows service out of a TurboGears2 application. Pylons applications should be exactly the same and should work without problem either.



# REQUIREMENTS

We use py2exe to generate our launcher and paster as our wsgi support server. So obviously we rely on the following tools:

- [Python 2.5](#) (we did not try yet 2.6 or 3)
- [py2exe](#)
- [pywin32](#)
- [pastescript](#)

In pylunch we have added helper functions to bundle GTK so that it becomes extra easy to deploy GUI applications using pygtk. This of course means you have any requirement you need on the build machine.

With the bundled application produced by pylunch you won't need to install GTK system wide on the target machine.





## SECTIONS

### 3.1 Installation from source

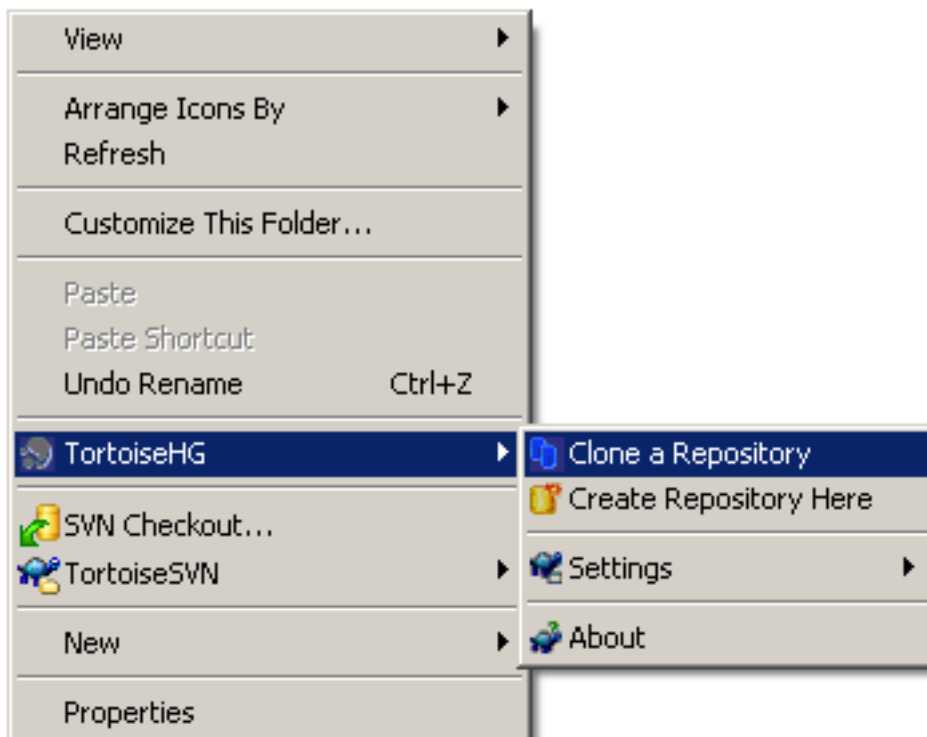
For those of you who want to build pylunch itself from source, here are the steps.

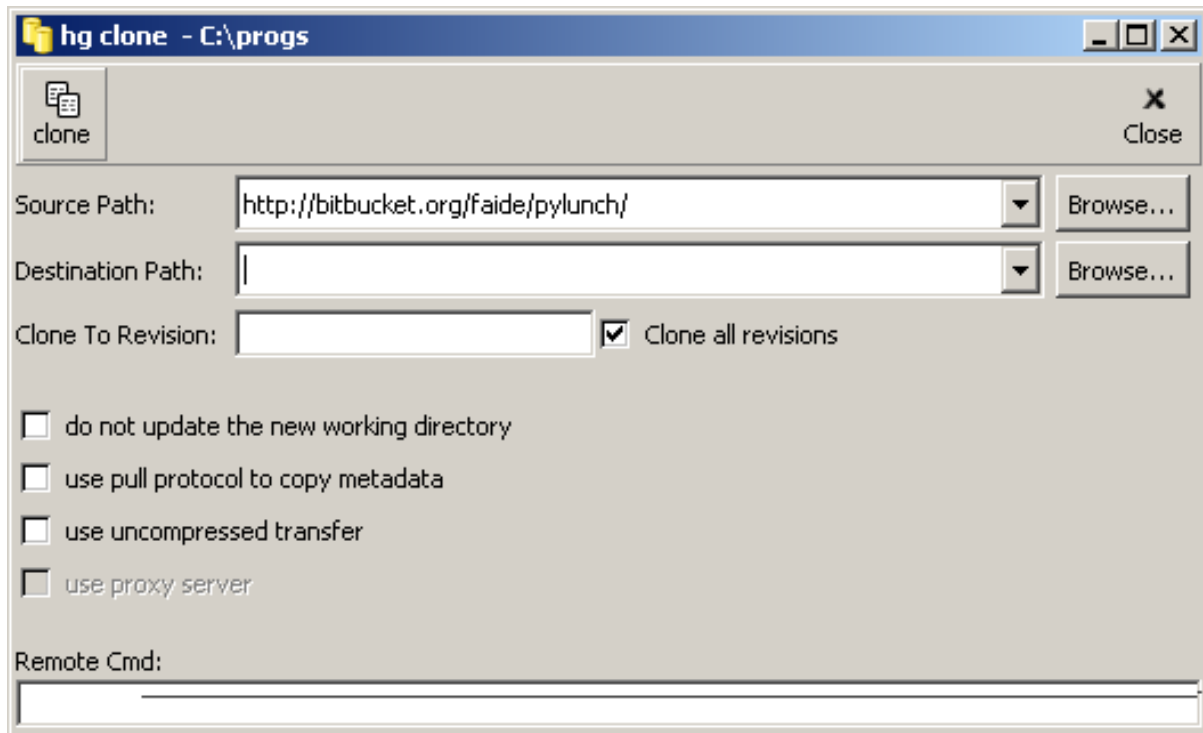
First you should clone the Mercurial repository of pylunch itself:

```
hg clone http://bitbucket.org/faide/pylunch/
```

Since most of you will be doing this work on windows, I recommend you install the most useful [TortoiseHG](#) program that will give you a nice GUI front-end to mercurial nicely integrated into the Windows explorer.

Here is how it looks using TortoiseHG:





Once you have cloned the source repository you should install all the prerequisites, evidently Python but also:

- `py2exe`
- `pywin32`
- `setuptools`
- `pastescript`

PasteScript should be easy\_installed in the python you'll be using to create your executables:

```
easy_install pastescript
```

Once this is done you are ready to begin the packaging of your application.

## 3.2 Basic Hands-on

Ok, so you now have a binary version of the launchers in a nice directory with all the dlls and resource files, and now you wonder: How do I package a python application with *this*?

### 3.2.1 A sample application

Let's create a small demo application just so that you see the whole process.

We will show you how to create a complete application in python from scratch to a zipped egg ready to be used...

You installed the pastescript requirement right? Then now is the time to use it, go in a CMD shell, change directory to a place you want to create your demo application and:

```
paster.exe create demoapp
```

This will prompt you with some questions, you'll find a transcript just below:

```
$ paster.exe create demoapp
Selected and implied templates:
  pastescript#basic_package  A basic setuptools-enabled package

Variables:
  egg:      demoapp
  package:  demoapp
  project:  demoapp
Enter version (Version (like 0.1)) ['']: 0.1
Enter description (One-line description of the package) ['']: demo app
Enter long_description (Multi-line description (in reST)) ['']: demo app that does nothing
Enter keywords (Space-separated keywords/tags) ['']: demo
Enter author (Author name) ['']: Florent Aide
Enter author_email (Author email) ['']: my@email.com
Enter url (URL of homepage) ['']:
Enter license_name (License name) ['']: MIT
Enter zip_safe (True/False: if the package can be distributed as a .zip file) [False]: True
Creating template basic_package
Creating directory .\demoapp
  Recursing into +package+
    Creating .\demoapp\demoapp\
    Copying __init__.py to .\demoapp\demoapp\__init__.py
    Copying setup.cfg to .\demoapp\setup.cfg
    Copying setup.py_tmpl to .\demoapp\setup.py
Running c:\Python25\python.exe setup.py egg_info
```

The result is a new `demoapp` directory with a small app inside. Our first task will be to add some minimum code to make sure we see something on the screen when we test it with pylunch.

Create a new file named `core.py` in the `demoapp/demoapp` directory along-side the lone `__init__.py` file that sits there and add the following content to it:

```
import time

def main():
    print "demo app running 1"
    time.sleep(1)
    print "demo app running 2"
    time.sleep(1)
    print "demo app running 3"
    time.sleep(1)
```

The important point here is to make sure you have at least one function somewhere in your program that takes **no** argument and serves as your entry point.

### 3.2.2 Preparing Pylunch for you application

Go back to the pylunch directory and edit the `pylunch.cfg` file (a sample is give in the sources as a template).

Edit the *product* by putting *demoapp* you like, and also set a *version* number, vendor and copyright can be omitted if you want they are used for the windows installer autocreation which is another part of the docs :)

This *product* variable will determine the executable name that will be produced. The exe can be renamed without consequence later on.

For the moment please leave the *embark\_gtk* flag to *false*. If you set this to yet, pylunch will search your system for the gtk dlls and environment and copy them to the right place in the distribution folder. This is in case you have an application using pyGTK and you have GTK install on the system and properly installed. We don't need this in our example.

If you have PIL and require it, set the *embark\_PIL* flag to *True*. If you don't know what is PIL or don't need it, then leave the flag to *False*.

Leave the *makensis\_path* empty for the moment. If you have an NSIS script for the *\_resulting\_* application produced by pylunch, then you'll be able to auto build the installer using NSIS commands directly during the build process.

In the *additionnal\_libs* section, you'll find the *main\_lib* setting. This is the most important one: Here you need to provide either the full path to an application directory that contains a proper *setup.py* that can be used by *easy\_install* or you can alternatively give a name like "MyApp == 0.5".

If you provide an application string you'll need to provide also the *index\_url* information to help pylunch find your application and it's requirements.

In our example we wil give the full path to our application we just created. We won't need an index. Here is what we will put in *additional\_libs*:

```
main_lib = c:/progs/demoapp
```

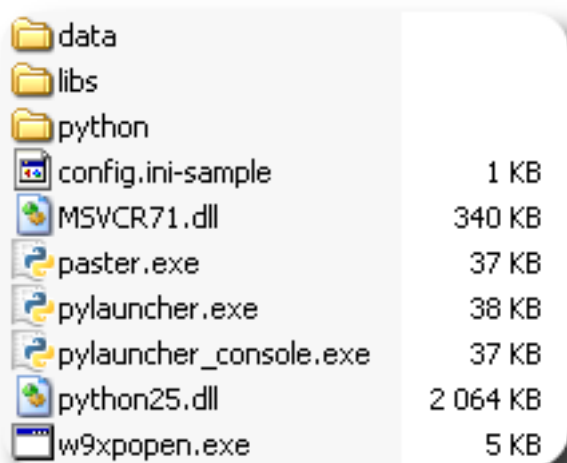
If you created your application in another directory, please change accordingly.

### 3.2.3 Building our application

To build the portable application you need to run the following command:

```
python setup.py py2exe
```

At this point you should have a new directory called *pylunch\_dist*.



data	
libs	
python	
config.ini-sample	1 KB
MSVCR71.dll	340 KB
paster.exe	37 KB
pylauncher.exe	38 KB
pylauncher_console.exe	37 KB
python25.dll	2 064 KB
w9xpopen.exe	5 KB

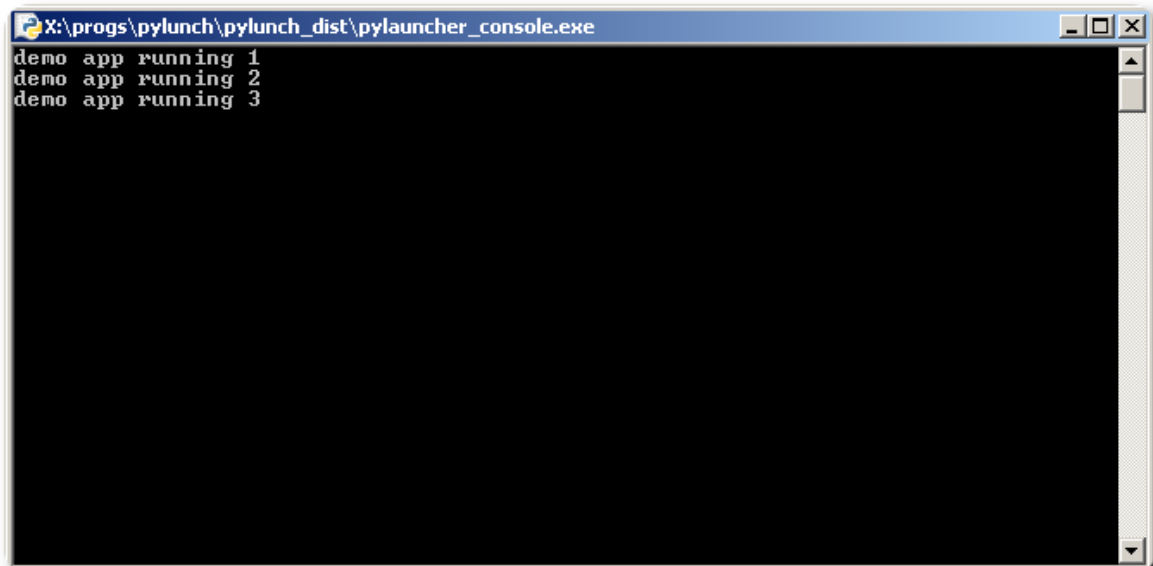
The last thing you need to do is to copy the *config.ini-sample* file that can be found at the root of your launcher directory to a file named *config.ini* and edit it.

Find the variable named *name* in the *application* section. This should be the last one of the file. You name to point the launcher to the entry point function we created earlier in the demo application:

```
[application]

; the application name should be in the form of a module
; and a function name separated by a sharp sign (#)
; the function will be called without any argument and will
; be responsible to scan the command line for options
; to setup its logging functions and load its config
name = demoapp.core#main
```

Now double click on the *demoapp\_console.exe* file. If everything went well you should see your prints running on the screen.



You're done with the first example of the pylunch python launcher :)

### In case of problem

If nothing happens, open-up a CMD console and run the same executable, you should see some trace-back. Read it, and try to correct the error...

## 3.3 TurboGears2 support

We will describe here a method to bundle a [TurboGears2](#) application into a portable windows application. At the end of this document you will have a windows service ready to launch your application on a server without human intervention.

So let's pretend you have a [TurboGears2](#) application that you developed and which is working correctly. Let's pretend that this application of yours is called *supertg* and that you want to deploy *supertg* on a sales-person laptop.

In this document I will quickstart a sample [TurboGears2](#) application. We suppose that you already have installed your TurboGears development environment and all is working.

### 3.3.1 Create a demonstration application

Open up a CMD console, and quickstart a new [TurboGears2](#) application named *supertg*:

```
paster quickstart -p supertg supertg
```

Once the application is created you should go inside the directory and test that it works normally.

If you are sure all you dependencies are declared you can begin to bundle your application.

### Preparing Pylunch for you application

Go back to the pylunch directory and edit the pylunch.cfg file (a sample is give in the sources as a template).

Edit the *product* by putting *supertg* you like, and also set a *version* number, vendor and copyright can be omitted if you want they are used for the windows installer autocreation which is another part of the docs :)

This *product* variable will determine the executable name that will be produced. The exe can be renamed without consequence later on.

For the moment please leave the *embark\_gtk* flag to *false*. If you set this to yet, pylunch will search your system for the gtk dlls and environment and copy them to the right place in the distribution folder. This is in case you have an application using pyGTK and you have GTK install on the system and properly installed. We don't need this in our example.

If you have PIL and require it, set the *embark\_PIL* flag to *True*. If you don't know what is PIL or don't need it, then leave the flag to *False*.

Leave the *makensis\_path* empty for the moment. If you have an NSIS script for the *\_resulting\_* application produced by pylunch, then you'll be able to auto build the installer using NSIS commands directly during the build process.

In the *additionnal\_libs* section, you'll find the *main\_lib* setting. This is the most important one: Here you need to provide either the full path to an application directory that contains a proper setup.py that can be used by *easy\_install* or you can alternatively give a name like "MyApp == 0.5".

If you provide an application string you'll need to provide also the *index\_url* information to help pylunch find your application and it's requirements.

In our example we wil give the full path to our application we just created. We won't need an index. Here is what we will put in *additional\_libs*:

```
main_lib = c:/progs/supertg
```

If you created your application in another directory, please change accordingly.

### Building our application

To build the portable application you need to run the following command:

```
python setup.py py2exe
```

At this point you should have a new directory called `pylunch_dist`.

## Configuring the service

To control the service *you need to be logged as an administrative account or to run a shell with an administrative account*.

You have an exe called *paster-service-control.exe* you should run it with a command like this one:

```
paster-service-control.exe -c c:\deployment\supertg\development.ini -l c:\deployment\sup
```

the `-c` option must point to your *development.ini* or *production.ini* configuration file for the supertg application we created in the beginning of this document. The file can be anywhere you want.

the `-l` should point to a directory where the service can create some log files. Those logs files are not the ones that you configure for your application. They are just in case something is written to stdout and is caught.

## Installing the service

To install the service you need to be logged as an administrative account or to run a shell with an administrative account. Then proceed with this command:

```
paster-service -install
```

This will add your service to the control panel. Go to the control panel and start the service... The service is named by the name you have chosen in the config.ini file that is alongside the paster-service.exe, so if you did not change this it should be named pasterWSGI.

It is important to note that each service must have a different name under windows, so if you need to run two such services at the same time, copy the pylunch\_dist directory, and change the service name inside the config.ini file.

Please also note that the services use the registry keys that are written by the *paster-service-control.exe*. If you change the name of the service inside the configuration you will need to also change the registry keys by running the *paster-service-control.exe* again.

For information, the registry keys are as follows:

```
HKEY_LOCAL_MACHINE\SOFTWARE\pylunch\pasterservice\pasterWSGI
```

where the last part is the service name you have chosen in the config file.

Then you can find the two config parameters that the service will use:

- ConfigFilePath
- LoggingDir

