mxPyRun - One-file easy-to-install Python interpreter

Install : Build : Contribute : Copyright & License : History : Home Version 0.8.0

Introduction

CGI Programming with Python

If you are in the CGI scripting business, then you know how hard it can sometimes be getting the SysAdmins to install Python for you. I ran into such situations a few times. Fortunately it's not a big problem, if you can get a grip on a pre-compiled binary for the machine the ISP is running.

Since installing a complete package with many files through a FTP-only connection is not exactly fun, we need something different here. This were the freeze tool can help: with it you can wrap the interpreter together with the whole standard library and the builtin modules into one single file. All that remains is to FTP that binary to the ISP and off you go.

Ok, so much for the theory. Now where do you get that pre-compiled binary from ? That's where this campaign starts...

Other Usages

Note that the you can also use the provided setup files to create your own custom one-file interpreter for easy installation of Python at your site.

This may be an interesting alternative for those that don't want several different installations around. In that case, you may want to edit the included `makepyrun.py` script and the `Modules/Setup` file (which was adapted from the Python source distribution) to produce a custom version that suits your special needs.

Another interesting feature of the one-file interpreter is that startup time is reduced: all the libs are already available when the interpreter starts and thus no searching along filesystem paths is necessary.

Notes

The encodings, email and distutils packages from Python 2.1.x were not included in the pyrun file in versions using mxPyRun 0.5.0 and below. If you need codecs other than 'utf-8' and 'ascii', you'll have to copy the encodings package from the Python distribution to the same directory where you place the pyrun binary.

All standard packages except the test package are included starting with mxPyRun 0.7.0 and up.

Importing shared modules was fixed in mxPyRun 0.5.0. mxPyRun now includes a patched version of the Python freeze tools.

If you want to use mxPyRun to compile versions of Python 2.3 and above, you'll need to edit the Modules/Setup and change the line "new newmodule.c" to "#new newmodule.c".

To compile on FreeBSD, you need to remove all occurences of "unset PYTHONPATH;" in Makefile.cgi before starting the compilation. Thanks to Oleg Broytmann for pointing out this work-around.

Installing the one-file Python Interpreter

To get your scripts to run on the ISP machine, you'll have to upload the interpreter binary via FTP and make it executable (I normally do this by uploading a small shell script into cgi-bin and then let the http daemon execute it... not exactly a very clean way, but what else can you do if you don't have telnet access ?).

To find out which machine your ISP is using, use the `uname -a` command (which you can either call directly using a telnet session or encapsuled in a small CGI script). Then go to the binaries section and download the needed interpreter file for that platform. The mxPyRun archive includes a small Perl script `uname.cgi` (posted by Thomas A. Bryan to comp.lang.python in the thread "using python in www site", May 1999) that helps you find out the machine type. To run it, upload the script to your `cgi-bin` directory and then point your web-browser at it.

The interpreter binaries are gzipped to reduce network load. You can either gunzip them on the ISP's machine, or before uploading them using, e.g.

```
gunzip -N pyrun-2.1.2-Linux-2.4.2-i586-unknown.gz
```

This will turn the archive file back into a plain binary called `pyrun`.

You may also want to further compress the executable using a compressor such as UPX before uploading it. This can reduce the size of the executable from around 1.8MB to just about 600KB (for WinNT).

Unix Target:

Here is a step-by-step guide for installing a binary on your ISP account via FTP only. Note that you need a `cgi-bin` directory (or similar, e.g. `cgi-local`) that executes `.cgi` files as CGI scripts to have this work.

1. Create a new directory in your account's home dir, say pythons/, using FTP (ftp myhost.isp.com; mkdir pythons)

2. Find out the path to that directory (still in FTP: cd pythons; pwd)
3. Upload the un'gzipped pyrun binary (still in FTP: binary; put pyrun);
4. Write a simple shell script install.cgi that sets the execute flags (chmod 755) on

   ```
   /...full path to pyrun as noted in step 2/pyrun
   ```

   and put it into the cgi-bin dir, e.g.

   ```
   #!/bin/sh
   chmod 755 /...full path to pyrun as noted in step 2/pyrun
   ```

5. Use your browser to execute that script, e.g. point it to http://myhost.isp.com/cgi-bin/install.cgi
6. Delete the install.cgi shell script again (we don't want keep this around for anyone to play with)
7. Add the line

   ```
   #!/...full path to pyrun as noted in step 2/pyrun
   ```

   as first line to all your Python CGI scripts
8. Upload the CGI scripts to the cgi-bin dir (they usually have to have an `.cgi` extension to become executable)

This should be all the steps you need to get Python up and running on your ISP account.

Note that you should not install the interpreter in the `cgi-bin` directory of your account: use some other secret directory. Installing it in `cgi-bin` would compromise security as it permits executing any known Python script on the ISP's machine.

Windows NT Target:

These are some notes I received from *Florent Heyworth* about installing and using the pyrun interpreter on WinNT with IIS (slightly edited):

In order to get the Unix shebang "#!" functionality on NT 4.0 running IIS 3.0 one should do the following (I tested this on Winnt 4.0 with IIS 3.0 although (I hope) the principle should hold true for IIS 4.0):

To make your scripts executable in the unix shebang style (note that you don't need the shebang "#!" in the script) you have to have access to the registry and enter the following string values in the registry node:

```
HKEY_LOCAL_MACHINE\SYSTEM\ControlSet001\Services\W3SVC\Parameters\Script Map

key=.py value=path_to_executable %s %s
key=.pyc value=path_to_executable %s %s
key=.pyo value=path_to_executable %s %s
```

as in:

```
.py c:\inetpub\scripts\pyrun.exe %s %s
.pyc c:\inetpub\scripts\pyrun.exe %s %s
.pyo c:\inetpub\scripts\pyrun.exe %s %s
```

It is unlikely that a user will have access to the ISP host's registry however I include a .reg file taken from my own registry which you can supply to your ISP so that they can change the paths where appropriate (if they want to- applicable to IIS 3.0 only as I don't have a IIS 4.0 installation handy) and merge it in their registry. Note that merging (typically double-clicking) on the file will NOT overwrite existing keys (unless they already have values for .pyc,.py and .pyo files).

Note that the scripts should be placed in an area with only "execute" access and that the IIS 3.0 service should be stopped and then restarted. It's unfortunate that these steps can only be followed if the ISP "plays the game" - there are some hacks that one could use to go around that but it is not wise to circumvent one's provider... In the case where you don't have access to the host's registry or you don't want to involve him/her then your best best is including the full path to the script as in:

```
http://localhost/scripts/pyrun.exe?testboot.py
```

One of the easiest ways to get at the form variables is as in the following test case:

\\localhost\inetpub\wwwroot\testit.html:

```
"""
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<html>
<head>
        <title>PyRun test</title>
</head>

<body>

<FORM ACTION="../scripts/testboot.py" METHOD="POST">
<INPUT TYPE="Text" NAME="TextInput">
<INPUT NAME="Submit" VALUE="testboot.py" TYPE="SUBMIT">
</FORM>

</body>
</html>
"""
```

and \\localhost\inetpub\scripts\testboot.py:

```
"""
import cgi

print 'Content-Type: text/html\n'

print '<H2>Directory:</H2>'
import os
l = os.listdir('.')
for i in l:
    print '<TT> %s </TT><BR>' % i

# display executable passed to (hopefully) pyrun.exe
print '<H2>%s</H2>' %sys.argv

# get all parameters passed in the form
form=cgi.FieldStorage()
cgi.print_form(form)
```

```
sys.exit(0)
"""
```

Note that there is one small problem which is why I have uploaded a new version of pyrun.exe:

The windows initialisation prints to STDOUT "initialising (module name)" where module name is eg. "Pythoncom" or whatever [the code calls GetProcAddress for each of the possible Win32 extensions (which may not be available on the destination machine) and as such if there are not present will simply not initialize them].

This will not cause the CGI itself to fail, although one should not rely on these modules being present because if they were there is no way to get them to initialize as they'll typically be in an unreachable directory like winnt\system32 or the standard Python installation root.

What could cause the CGI to fail in the end (Netscape is here less forgiving than IE) is that the code printed out to STDOUT "initializing (modulename)" will produce an invalid header response from Netscape and Netscape will try to get the user to download the response instead of displaying it in the browser. It is sufficient to just comment out the output code in dll_frozenmain.c.

I recommend renaming the PyRun.exe to PyRun.cgi and removing any "read","write" permissions in the directory. Additionally 8-) one has to remember to make the first line of the script one wants to execute print "Content-type: text/html"... But then we all know that 8-)

Building the one-file Python Interpreter

I have put together a small zip file (see below for the URL), which contains all necessary setup and config files. Please also see the Notes section for OS specific hints.

Here is what you have to do to compile a pyrun one-file- does-it-all Python interpreter (alas, I don't know how this is done on WinXX, these descriptions apply to Unix installations -- which is what most ISPs use anyway):

1. Download the latest Python source distribution (if you don't already have it).

   Please always use a fresh copy of the distribution. This ensures that all default modules are built static (i.e. the code is included into the binary rather than referenced as shared library) and a common setup is available in all provided binaries.

2. Download the mxPyRun 0.7.0 setup files for Python 2.1.x and above (or mxPyRun 0.2.3 for Python 1.5.2).

3. Untar/zip the Python source distribution to some temporary directory.

4. Cd to the Python-x.x.x/ dir.

5. Unzip the mxPyRun archive into the current directory; this will not overwrite any distribution files, BTW.

6. Run 'make -sf Makefile.cgi'.

   Note that the build process uses a temporary subdir (default is /tmp/pycgi) as build target. That subdir is created if not existant and removed after successful completion.
   You can change this target by passing a parameter TARGETDIR=/my/tmp/dir/pyrun to the make program, e.g. 'make -sf Makefile.cgi TARGETDIR=/my/tmp/dir/pyrun'.

7. Wait a few minutes, drink some coffee, go for walk, you know what I mean...

E.g. a typical build session would look something like this (here for Python 1.5.2):

```
/home/lemburg> cd tmp/

lemburg/tmp> tar xfz /downloads/python/py152.tgz

lemburg/tmp> cd Python-1.5.2/

tmp/Python-1.5.2> unzip /downloads/python/mxPyRun-0.2.3.zip
Archive:  /downloads/python/mxPyRun-0.2.3.zip
  inflating: ./Makefile.cgi
  inflating: ./README.mxPyRun
  inflating: PyRun/makepyrun.py
  inflating: PyRun/mxPyRun.html
 extracting: PyRun/nop.py
  inflating: PyRun/testboot.py
  inflating: PyRun/testmod.py

tmp/Python-1.5.2> make -sf Makefile.cgi
creating cache ./config.cache
checking MACHDEP... linux2
checking CCC... CCC= g++
[...]
creating config.h
making Makefile in subdirectory .
making Makefile in subdirectory Parser
making Makefile in subdirectory Objects
making Makefile in subdirectory Python
making Makefile in subdirectory Modules
[...]
Creating directory /tmp/pycgi//lib
Creating directory /tmp/pycgi//lib/python1.5
Creating directory /tmp/pycgi//lib/python1.5/config
Creating directory /tmp/pycgi//include
Creating directory /tmp/pycgi//include/python1.5
./install-sh -c -m 644 ./Include/Python.h /tmp/pycgi//include/python1.5
[...]
./install-sh -c -m 644 ./Include/tupleobject.h /tmp/pycgi//include/python1.5
making Makefile in subdirectory .
making Makefile in subdirectory Parser
making Makefile in subdirectory Objects
making Makefile in subdirectory Python
making Makefile in subdirectory Modules
Now run "make" in ../../PyRun to build the target: pyrun
-----------------------------------------------------------------------

Please send the file pyrun-1.5.2-Linux-2.0.35-i586-unknown.gz to

  pyrun@egenix.com

so that I can move the file to a central archive on our web-server
for everyone to download.
```

```
Thank you !

--------------------------------------------------------------------------

Finished. The interpreter is called: ./pyrun.
```

If all goes well, you'll find an executable file 'pyrun' in the Python distribution directory. This file contains the interpreter and the standard lib. It executes the first command line argument as Python file, e.g.

```
#!/path/to/pyrun
print 'Hello world!'
```

will work. It does *not* except any flags though, thus

```
#!/path/to/pyrun -v
print 'Oops'
```

will not do, but then you normally don't need flags for CGI scripts. The environment flags like PYTHONVERBOSE and PYTHONINSPECT still work as usual, though, so you can use those to get the modified behaviour.

Note that the setup mechanism has only been tested on Linux, so please verify that the resulting binary does work as expected.

Making your contribution available to others

PLEASE NOTE: Due to time contraints we cannot actively host mxPyRun binaries anymore.

We may restart this effort again in a few months. If you want to contribute new files to the project, please send in URLs to you mxPyRun binaries and we'll place them on this page as references. Thank you.

Copyright & License

you agree to be bound by the terms and conditions of the eGenix.com Public License Agreement and the Python 2.2 License.

History & Future

Things that still need to be done:

- Test the mechanism on all available platforms...
- Provide as many compiled binaries as we can.

Changes from 0.7.0 to 0.8.0:

- Ported mxPyRun to Python 2.5, 2.6 and 2.7.
- Enhanced the error reporting and command line interface.
- Added options to run byte-code and modules on the PYTHONPATH directly.

Changes from 0.6.0 to 0.7.0:

- Added all standard packages.

Changes from 0.5.0 to 0.6.0:

- Added the encodings package.

Changes from 0.4.0 to 0.5.0:

- Added patched freeze tool to enable loading shared modules with pyrun. Thanks to Martin v. Löwis for hinting me to the solution.

Changes from 0.2.3 to 0.4.0:

- Updated to Python 2.1.x.

Changes from 0.2.2 to 0.2.3:

- Updated links to starship.python.net.
- Added note about UPX. Thanks to Neil Hodgson for bringing it to my attention.

Changes from 0.2.1 to 0.2.2:

- Minor cosmetics in the Makefile.
- Added note to also mail othe configuration details.
- Changed the shell script hacks for platform and Python version to real Python scripts.
- Added several new contributed binaries to the list.

Changes from 0.2.0 to 0.2.1:

- Added a way to configure the temporary target directory used by the Makefile.
- Added targets to auto-generate uploadable and properly versioned pyrun executables.
- Added more documentation for the building process.

Changes from 0.1 to 0.2.0:

- Made the setup code independent of the used Python version. It should work with all version starting with Python 1.5.
- Simplified the make process.

Version 0.1 was the initial release.
