pycall Documentation

Release 2.2

Randall Degges

April 05, 2015

Contents

Welcome to pycall's documentation. This documentation is divided into several different parts, and can be read through like a book. If you're relatively new to telephony, you may want to start at the *Foreword*, and read through to the end; otherwise, you can jump right into the *Usage* docs.

If you want to contribute code to the project, please read the *entire* *Developing* section before diving into the code.

CHAPTER 1

Foreword

Thanks for checking out pycall. pycall is a simple python wrapper for creating Asterisk call files. If you're familiar with the Asterisk PBX system, and are building an application that needs to automatically place outbound calls, pycall may be just what you need!

1.1 What is Asterisk?

Asterisk is a popular open source PBX (phone system) which many businesses and hobbyists use for placing and receiving phone calls. Asterisk is also used for a wide variety of other purposes, all relating to telephony and communications.

If you've never used Asterisk, head over to the Asterisk website for more information.

1.2 What Are Call Files?

Since pycall is a python library for "creating and using Asterisk call files", you may be wondering what a call file is. Call files are specially formatted text files that Asterisk uses to initiate outbound calls automtically.

In a nutshell, Asterisk call files allow developers to automatically generate calls and launch programs through Asterisk, without any sort of complex network API.

To learn more about call files, head over to the VoIP Info call files page.

1.3 Why Should I Use pycall?

There are lots of reasons why you should use pycall. I could just be vague and leave it at that, but you want to *real* reasons right?

- **Simple**

  pycall makes building telephony applications easy.

- **Flexible**

  pycall gives you a lot of flexibility. If you're Asterisk environment is setup in a non-standard way, pycall won't mind.

- **Well Documented**

  pycall has great documentation (but you already know that by now, right?) and active development.

1.4 Target Audience

pycall is intended for usage by telephony developers who are building Asterisk applications. It has sensible defaults, and is simple to implement into any application.

If you're familiar with python programming, and want to write some telephony software, give pycall a try and you won't be disappointed!

CHAPTER 2

Installation

pycall can be installed just like any other python program, through pypi. To install it, simply run:

```
$ pip install pycall
```

If you'd like to install the latest development version of pycall (not recommended), then download the latest release and run:

```
$ python setup.py install
```

If you're totally unfamiliar with installing python packages, you should probably read the official tutorial before continuing.

CHAPTER 3

Usage

Integrating pycall into your project is quick and easy! After reading through the sections below, you should be able to integrate pycall into your project, and understand what it can and cannot be used for.

3.1 Preparation

The rest of this guide assumes you have the following:

1. A working Asterisk server.
2. Some sort of PSTN (public switch telephone network) connectivity. Regardless of what sort of PSTN connection you have (SIP / DAHDI / ZAPTEL / ISDN / etc.), as long as you can make calls, you're fine.

For simplicity's sake, I'm going to assume for the rest of this guide that you have a SIP trunk named *flowroute* defined.

3.2 Hello, World!

pycall allows you to build applications that automate outbound calling. In the example below, we'll call a phone number specified on the command line, say "hello world", then hang up!

```
import sys
from pycall import CallFile, Call, Application

def call(number):
        c = Call('SIP/flowroute/%s' % number)
        a = Application('Playback', 'hello-world')
        cf = CallFile(c, a)
        cf.spool()

if __name__ == '__main__':
        call(sys.argv[1])
```

Just save the code above in a file named *call.py* and run it with python!

```
$ python call.py 18002223333
```

Assuming your Asterisk server is setup correctly, your program just placed a call to the phone number *18002223333*, and said "hello world" to the person who answered the phone!

3.2.1 Code Breakdown

1. First we imported the pycall classes. The `CallFile` class allows us to make call files. The `Call` class stores information about a specific call, and the `Application` class lets us specify an Asterisk application as our `Action`. Every call file requires some sort of action (what do you want to do when the caller answers?).
2. Next, we build a `Call` object, and specify the phone number to call in standard Asterisk format. This tells Asterisk who to call.

Note: In this example we made a call out of a SIP trunk named *flowroute*, but you can specify any sort of dial string in its place. You can even tell Asterisk to call multiple phone numbers at once by separating your dial strings with the & character (eg: *Local/1000@internal&SIP/flowroute/18002223333*).

---

3. Then we build an `Application` object that tells Asterisk what to do when the caller answers our call. In this case, we tell Asterisk to run the Playback command, and pass the argument 'hello-world' to it.

Note: The name 'hello-world' here refers to one of the default Asterisk sound files that comes with all Asterisk installations. This file can be found in the directory */var/lib/asterisk/sounds/en/* on most systems.

---

4. Finally, we create the actual `CallFile` object, and run its `spool()` method to have Asterisk make the call.

3.3 Scheduling a Call in the Future

Let's say you want to have Asterisk make a call at a certain time in the future–no problem. The `spool()` method allows you to specify an optional datetime object to tell Asterisk when you want the magic to happen.

In this example, we'll tell Asterisk to run the call in exactly 1 hour:

```
import sys
from datetime import datetime
from datetime import timedelta
from pycall import CallFile, Call, Application

def call(number, time=None):
        c = Call('SIP/flowroute/%s' % number)
        a = Application('Playback', 'hello-world')
        cf = CallFile(c, a)
        cf.spool(time)

if __name__ == '__main__':
        call(sys.argv[1], datetime.now()+timedelta(hours=1))
```

Note: If you specify a value of *None*, the call file will be ran immediately.

Just for the heck of it, let's look at one more code snippet. This time we'll tell Asterisk to run the call file at exactly 1:00 AM on December 1, 2010.

```
import sys
from datetime import datetime
from pycall.callfile import CallFile

def call(number, time=None):
        c = Call('SIP/flowroute/%s' % number)
        a = Application('Playback', 'hello-world')
        cf = CallFile(c, a)
```

```
        cf.spool(time)

if __name__ == '__main__':
        call(sys.argv[1], datetime(2010, 12, 1, 1, 0, 0))
```

3.4 Setting Call File Permissions

In most environments, Asterisk is installed and ran as the user / group 'asterisk', which often poses a problem if your application doesn't run under the 'asterisk' user account.

pycall recognizes that this is a frustrating problem to deal with, and provides three mechanisms for helping make permissions as painless as possible: the `user` attribute, the `NoUserError`, and the `NoUserPermissionError`.

- The `user` attribute lets you specify a system username that your call file should be ran as. For example, if your application is running as 'root', you could say:

  ```
  cf = CallFile(c, a, user='asterisk')
  ```

  and pycall would chown the call file to the 'asterisk' user before spooling.

- If you specify the `user` attribute, but the user doesn't exist, pycall will raise the `NoUserError` so you know what's wrong.
- Lastly, if your application doesn't have the proper permissions to change the ownership of your call file, pycall will raise the `NoUserPermissionError`.

As an example, here we'll change the call file permissions so that Asterisk can actually read our call file:

```
import sys
from pycall import CallFile, Call, Application

def call(number):
        c = Call('SIP/flowroute/%s' % number)
        a = Application('Playback', 'hello-world')
        cf = CallFile(c, a, user='asterisk')
        cf.spool(time)

if __name__ == '__main__':
        call(sys.argv[1])
```

Note: If you run this code on a system that doesn't have Asterisk installed, you will most likely get a `NoUserError` since pycall won't be able to find the 'asterisk' user that it's trying to grant permissions to.

3.5 Adding Complex Call Logic

Most applications you write will probably be a bit more complex than "hello world". In the example below, we'll harness the power of the `Context` class to instruct Asterisk to run some custom dial plan code after the caller answers.

```
from pycall import CallFile, Call, Context

c = Call('SIP/flowroute/18002223333')
con = Context('survey', 's', '1')
```

```
cf = CallFile(c, con)
cf.spool()
```

For example purposes, let's assume that somewhere in your Asterisk *extensions.conf* file there exists some dial plan in a context labeled *survey*.

After the caller answers our call, Asterisk will immediately jump to the dial plan code we've specified at *survey,s,1* and start executing as much logic as desired.

3.6 Setting a CallerID

A lot of the time, you'll want to force Asterisk to assume a specific caller ID when making outbound calls. To do this, simply specify a value for the `callerid` attribute:

```
c = Call('SIP/flowroute/18002223333', callerid="'Test User' <5555555555>'")
```

Now, when Asterisk makes your call, the person receiving the call (depending on their phone and service type) should see a call coming from "Test User" who's phone number is 555-555-5555!

3.7 Passing Variables to Your Dial Plan

Often times, when building complex applications, you'll want to pass specific data from your application to Asterisk, so that you can read the information later.

The example below will pass some information to our Asterisk dial plan code, so that it can use the information in our call.

```
from pycall import CallFile, Call, Context

vars = {'greeting': 'tt-monkeys'}

c = Call('SIP/flowroute/18882223333', variables=vars)
x = Context('survey', 's', '1')
cf = CallFile(c, x)
cf.spool()
```

And somewhere in our *extensions.conf* file...

```
[survey]
exten => s,1,Playback(${greeting})
exten => s,n,Hangup()
```

As you can see, our dial plan code can now access the variable 'greeting' and its value.

3.8 Track Your Calls with Asterisk Account Codes

Asterisk call files allow you to specify that a certain call should be associated with a certain account. This is mainly useful for logging purposes. This example logs the call with the 'randall' account:

```
c = Call('SIP/flowroute/18002223333', account='randall')
```

Note: For more information on call logs, read the CDR documentation.

3.9 Specify Call Timing Values

pycall provides several ways to control the timing of your calls.

1. `wait_time` lets you specify the amount of time to wait (in seconds) for the caller to answer before we consider our call attempt unsuccessful.
2. `retry_time` lets you specify the amount of time to wait (in seconds) between retries. Let's say you try to call the number 1-800-222-3333 but they don't answer, Asterisk will wait for `retry_time` seconds before calling the person again.
3. `max_retries` lets you specify the maximum amount of retry attempts (you don't want to call someone forever, do you?).

Using these attributes is simple:

```
c = Call('SIP/flowroute/18002223333', wait_time=10, retry_time=60,
                max_retries=2)
```

3.10 Archiving Call Files

If, for some reason, you want to archive call files that have already been spooled with Asterisk, just set the `archive` attribute to *True*:

```
cf = CallFile(..., archive=True)
```

and Asterisk will copy the call file (with a status code) to the archive directory (typically */var/spool/asterisk/outgoing_done*).

3.11 Dealing with Non-Standard Asterisk Installs

If your Asterisk server isn't installed with the defaults, chances are you need to make some changes. pycall provides a ton of flexibility in this regard, so you should have no problems getting things running.

3.11.1 Specifying a Specific Name for Call Files

If you need to name your call file something special, just specify a value for both the `filename` and `tempdir` attributes:

```
cf = CallFile(..., filename='test.call', tempdir='/tmp')
```

Note: By default, pycall will randomly generate a call file name.

3.11.2 Specifing a Custom Spooling Directory

If you're Asterisk install doesn't spool to the default */var/spool/asterisk/outgoing* directory, you can override it with the `spool_dir` attribute:

```
cf = CallFile(..., spool_dir='/tmp/outgoing')
```

CHAPTER 4

Developing

Contributing code to pycall is easy. The codebase is simple, clear, and tested. All ideas / patches / etc. are welcome.

This section covers everything you need to know to contribute code, please give it a full read before submitting patches to the project.

4.1 Project Goals

The main goal of the pycall library is to provide a *simple* python wrapper for generating Asterisk call files. I'd like to keep the project and codebase small and concise. It would be easy to add features to pycall that fall outside of its humble goals–however, please resist that urge :)

Before submitting patches that add functionality, please consider whether or not the patch contributes to the overall project goals, or takes away from them.

4.2 Code Organization

pycall is broken up into several python modules to keep the code organized. To start reading code, check out the *callfile.py* module. This contains the main `CallFile` class that most users interact with.

Each `CallFile` requires an `Action` to be specified. The idea is that a `CallFile` should represent the physical call file, whereas an `Action` should represent the behavior of the call file (what actions should our call file perform if the call is answered?).

Actions can be either applications (`Application`) or contexts (`Context`). Both applicatoins and contexts correspond to their Asterisk equivalents.

Each `CallFile` must also specify a `Call` object. `Call` objects specify the actual call information– what number to call, what callerid to use, etc.

If there are errors, pycall will raise a custom `PycallError` exception. The errors are very descriptive, and always point to a solution.

4.3 Tests

pycall is fully tested. The project currently makes use of a full unit test suite to ensure that code works as advertised. In order to run the test suite for yourself, you need to install the python-nose library, then run *python setup.py nosetests*. If you'd like to see the coverage reports, you should also install the coverage.py library.

All unit tests are broken up into python modules by topic. This is done to help keep separate test easy to work with.

If you submit a patch, please ensure that it doesn't break any tests. If you submit tests with your patch, it makes it much easier for me to review patches and integrate them.

4.4 Code Style

When submitting patches, please make sure that your code follows pycall's style guidelines. The rules are pretty simple: just make your code fit in nicely with the current code!

pycall uses tabs instead of spaces, and uses standard PEP8 formatting for everything else.

If in doubt, just look at pre-existing code.

4.5 Documentation

One of pycall's goals is to be well documented. Users should be able to quickly see how to use the library, and integrate it into their project within a few minutes.

If you'd like to contribute documentation to the project, it is certainly welcome. All documentation is written using Sphinx and compiles to HTML and PDF formats.

4.6 Development Tracker

pycall is proudly hosted at Github. If you'd like to view the source code, contribute to the project, file bug reports or feature requests, please do so there.

4.7 Submitting Code

The best way to submit code is to fork pycall on Github, make your changes, then submit a pull request.

If you're unfamiliar with forking on Github, please read this awesome article.