mod_python

About

This module allows one to write FreeSWITCH applications using Python.

Install & Configure

Building mod_python

Install python-dev package on Debian/Ubuntu:

```
apt-get install python python-dev
```

Enable compilation in modules.conf:

- In the FreeSWITCH source folder, edit modules.conf and uncomment languages/mod_python

Recompile and install FreeSWITCH:

```
make install all
```

The configure script will try to detect your existing python version. If it cannot find it or it does not support multi-threading, it will print a warning message.

You can specify an argument to configure to make it use a particular version:

```
./configure --with-python=... (eg, --with-python=/usr/bin/python2.7)
```

⚠ Make current auto fixes the makefile in case it doesn't get generated (correctly, or at all). make python-reconf to regenerate it.

Enabling mod_python

Open up **conf/autoload_configs/modules.conf.xml** and add an entry:

```
<load module="mod_python"/>
```

Finding Python modules

There are two different ways to tell the python interpreter how to find python modules. If you don't do either of these, the embedded Python interpreter will have *no way* to find your python scripts.

Assuming you have:

```
<action application="python" data="foo.bar"/>
```

This is telling python to load the **bar** *module* that lives in the **foo** *package*.

Copy or Symlink to site-packages directory, i.e if the source file is in /usr/src/foo/bar.py:

```
cd /path/to/python/site-packages
ln -s /usr/src/foo .
```

The same can be done via copying. Don't forget, the foo package directory will need an __init__.py.

Adding to PYTHONPATH environment variable, if the file is in /usr/src/foo/bar.py, add the following to your system environment startup

```
export PYTHONPATH=$PYTHONPATH:/usr/src
```

Don't forget, the foo package directory will need an __init__.py.

In the shell where freeswitch is started, this environment variable will need to be defined.

Invoking mod_python applications

To call a Python application from dialplan, you should probably be familiar with the Dialplan. You simply call it as an application similar to:

```
<action application="python" data="foo.bar"/>
```

The module is **bar**, in the **foo** package. See the Finding python modules section to tell the embedded python interpreter how to find this module.

If your module (say, test.py) is not in any package directory, then you would instead use:

```
<action application="python" data="test"/>
```

In both cases, you need to leave off the .py file extension otherwise it will not work. It expects a fully qualified module name only.

You would put this in your 'dialplan' if using the XML dialplan module with Freeswitch. Don't forget your 'condition' tags and all that goodness.

It's possible to call a Python script from the CLI with the python command:

```
 freeswitch> python foo.bar
```


if you invoke it this way, your python **handler()** function will be called with no arguments.

Python module specification

Your python module must define a function called handler that takes two arguments: session and args.

```
def handler(session, args):
    pass
```

The session is the main interface to Freeswitch, which wraps a Freeswitch session, and the args are any args passed to the handler script.

API

Supports the same API as mod_lua

Sample Python Scripts

Hello World via call

In this example, if you copied that module to your site-packages directory and called it from an extension in the dialplan, you'd be using the `handler` function.

Here is a copy of the python_example module that comes in the FreeSWITCH source code. See latest version here.

FreeSWITCH's mod_python usage examples

```
import freeswitch

""" FreeSWITCH's mod_python usage examples.   This module uses the default names looked up by mod_python, but
most of these names can be overridden using <modname>::<function> when calling the module from FreeSWITCH.   """


def handler(session, args):
    """ 'handler' is the default function name for apps.   It can be overridden with <modname>::<function>
`session` is a session object `args` is a string with all the args passed after the module name   """
    freeswitch.consoleLog('info', 'Answering call from Python.\n')
    freeswitch.consoleLog('info', 'Arguments: %s\n' % args)

    session.answer()
    session.setHangupHook(hangup_hook)
    session.setInputCallback(input_callback)
    session.execute("playback", session.getVariable("hold_music"))


def hangup_hook(session, what, args=''):
    """ Must be explicitly set up with session.setHangupHook(hangup_hook).   `session` is a session object.
`what` is "hangup" or "transfer". `args` is populated if you pass extra args to session.setInputCallback().
"""
    freeswitch.consoleLog("info", "hangup hook for '%s'\n" % what)


def input_callback(session, what, obj, args=''):
    """ Must be explicitly set up with session.setInputCallback(input_callback).   `session` is a session
object. `what` is "dtmf" or "event". `obj` is a dtmf object or an event object depending on the 'what' var.
`args` is populated if you pass extra args to session.setInputCallback().   """
    if (what == "dtmf"):
        freeswitch.consoleLog("info", what + " " + obj.digit + "\n")
    else:
        freeswitch.consoleLog("info", what + " " + obj.serialize() + "\n")
    return "pause"


def fsapi(session, stream, env, args):
    """ Handles API calls (from fs_cli, dialplan HTTP, etc.).   Default name is 'fsapi', but it can be
overridden with <modname>::<function>   `session` is a session object when called from the dial plan or the
string "na" when not. `stream` is a switch_stream. Anything written with stream.write() is returned to the
caller. `env` is a switch_event. `args` is a string with all the args passed after the module name.   """
    if args:
        stream.write("fsapi called with no arguments.\n")
    else:
        stream.write("fsapi called with these arguments: %s\n" % args)
    stream.write(env.serialize())


def runtime(args):
    """ Run a function in a thread (eg.: when called from fs_cli `pyrun`).   `args` is a string with all the
args passed after the module name.   """
    print args + "\n"


def xml_fetch(params):
    """ Bind to an XML lookup.   `params` is a switch_event with all the relevant data about what is being
searched for in the XML registry.   """
    xml = ''' <?xml version="1.0" encoding="UTF-8" standalone="no"?> <document type="freeswitch/xml"> <section
name="dialplan" description="RE Dial Plan For FreeSWITCH"> <context name="default"> <extension name="generated"
> <condition> <action application="answer"/> <action application="playback" data="${hold_music}"/> </condition>
</extension> </context> </section> </document> '''

    return xml
```

Hello World - Dialplan API

```
<action application="set" data="foo=${python(my_script)}"/>
```

So foo channel variable is set to the output of the my_script python script.

The script will be called with a magic object called "stream" which has the method write, and the anything written to this method will be the script's output. So for example

```
 def fsapi(session, stream, env, args):
    stream.write("hello")
```

will cause the foo variable to be set to "hello".

Run something in a thread using API

Usage of the API in Python is conceptually identical to usage in other supported languages.

The example scripts below employ the API, along with the 'runtime' function described in python_example.py to run a job in a thread.

This approach provides a means of implementing non-blocking code without employing mod_event_socket and may be suitable/useful for cleanup or post-processing.

This example includes two modules. The first module is a virtual copy of the default example script with a couple of notable differences.

```
import os
from freeswitch import *

# WARNING: known bugs with hangup hooks, use with extreme caution
def hangup_hook(session, what):

    consoleLog("info","hangup hook for %s!!\n\n" % what)
    return

def input_callback(session, what, obj):
    if (what == "dtmf"):
        consoleLog("info", what + " " + obj.digit + "\n")
    else:
        consoleLog("info", what + " " + obj.serialize() + "\n")
    return "pause"

def handler(session, args):
    session.answer()
    session.setHangupHook(hangup_hook)
    session.setInputCallback(input_callback)
    session.streamFile("/my/test/audio.wav")
    session.hangup() #hangup the call
    #Now run another python script in a thread. If we
    # don't do it this way all subsequent work will block
    # the hangup message from being sent to the client
    new_api_obj = API()
    new_api_obj.executeString(
         "pyrun foo.postprocessing " +
         session.getVariable('caller_id_number'))
```

The second module, "postprocessing", handles our post-processing needs and for convenience resides in the same package, "foo":

```
import os, sys, time
from freeswitch import *

# everything after the command (in this case pyrun) and
# the module name (in this case foo.postprocessing) will
# be interpreted as a string and handed to our 'runtime'
# function where it will be accessible via the argument 'arg1'
def runtime(arg1):
    time.sleep(10) # this is just to test that we are actually
                   # running in a separate thread.
    consoleLog( "info", "Caller: %s hung up 10s ago!\n" % arg1 )
```

When running the above example, the client should receive a hangup immediately after streamFile returns. 10 seconds later the "Caller: xxxx hung up 10s ago!" message should be printed to the console.

Fetch Effective Caller Name from CSV file using Python

The purpose of this script is simply to associate foreign caller-id without caller_id_name to a static caller_id_name using a csv file, and at the same time it shows how to work with Python and CSV files inside FreeSWITCH

First you must add something like that to the generic dialplan for your local extensions (personally i use *Local_Extension* in **dialplan/default.xml**:

```
<action application="set" data="caller=${caller_id_number}"/>
<action application="python" data="setCallerName"/>
```

It's pretty self-explanatory, it create a variable which will be used in the python script that we call on the second line. **setCallerName** is the name of your script in **${FS_ROOT}/scripts/**

Here the code in **${FS_ROOT}/scripts/setCallerName.py** :

```
import csv
from freeswitch import *

def handler(session,args):
    caller = session.getVariable("caller") # We use the variable we set previously in our Dialplan
    csv_reader = csv.reader(open("<filepath>","rb")) # <filepath> must be replace with the path of the .csv
file (watch out for relative path, if not sure use full path ie. /opt/freeswitch/file.csv)
    portfolio_list = []
    portfolio_list.extend(csv_reader)
    for data in portfolio_list:
        if (data[0] == caller): # We compare first column to the variable
            session.execute("set","effective_caller_id_name="+data[1]) # If equal, then set the session
variable to second column
```

The CSV should look something like this :

```
"7001","Remote User"
"7002","Remote User2"
```

FAQ

Does each script spawn a python interpreter?

No. A single python interpreter is spawned at module startup and used for the lifetime of the freeswitch process.

Are there thread safety issues?

Each thread swaps in its "thread state" before executing python code and then swaps it out when finished. Also during blocking calls into freeswitch, a thread will swap out its thread state in order to not block other threads, and then swap it in after the blocking call to freeswitch has finished.

I changed a module I'm importing, and nothing happened

Answer: assume you are importing a module called baz, change your entry point module to:

```
import baz
reload(baz)
```

How do I pass arguments to the script?

This is possible using channel variables. In the dialplan:

```
<extension name="foo">
  <condition field="destination_number" expression="^123$">
    <action application="set" data="foo=bar"/>
    <action application="python" data="mypackage.myscript"/>
  </condition>
</extension>
```

and in the python script:

```
foo = session.getVariable("foo")
freeswitch.consoleLog("info", "foo: %s\n" % foo)
```

Can I test scripts using python shell?

No, it will fail as follows when you try to import the python module:

```
 >>> import freeswitch
 Traceback (most recent call last):
   File "<stdin>", line 1, in ?
   File "/usr/lib/python2.4/site-packages/freeswitch.py", line 7, in ?
     import _freeswitch
 ImportError: No module named _freeswitch
```

Can it serve configuration (like Lua)?

Yes, this has been added but not documented. The default hook for serving a config is xml_fetch as stated in the python_example.py script. However, the interpreter complains with a 'takes exactly 1 argument (2 given)' message if xml_fetch is defined to accept only one param. Altering the definition to accept 2 params solves the problem. However, consoleLog invariably shows param2 to contain nothing...

```
from freeswitch import *

def xml_fetch( param1, param2 ):
    xml = ''' <?xml version="1.0" encoding="UTF-8" standalone="no"?> <document type="freeswitch/xml"> <section
name="dialplan" description="RE Dial Plan For FreeSWITCH"> <context name="default"> <extension name="generated"
> <condition field="destination_number" expression="^9992$">> <action application="answer"/> <action
application="playback" data="${hold_music}"/> </condition> </extension> </context> </section> </document> '''
    return xml
```

and you will need to create/edit a python.conf.xml to be something like:

```
<nowiki>
<configuration name="python.conf" description="Python Configuration">
  <settings>
    <param name="xml-handler-script" value="mypackage.mymodule"/>
    <param name="xml-handler-bindings" value="dialplan"/>
  </settings>
</configuration>
</nowiki>
```

Gotchas

String Substitution in Functions

The session function recordFile cannot contain anything but strings in the <filename> parameter. Attempting to use sql rows or other tuples as part of a string substitution will give you a

```
NotImplementedError: Wrong number of arguments for overloaded function 'CoreSession_recordFile'.
```

For example:

```
record_file = row[0] + '-' + row[1] + '.wav'
session.recordFile(record_file,120,500,2)
```

will throw the error mentioned above even though when printing the variable record_file will result in the expected value. Instead the values row[0] and row [1] need to be wrapped in the str() function when assigning a value to record_file:

```
first_name = str(row[0])
last_name = str(row[1])
record_file = first_name + '-' + last_name + '.wav'
session.recordFile(record_file,120,500,2)
```

should give proper behavior.

Bridge and transfer the call

If you want to originate call and bridge them directly..

```
call_addr='sofia/internal/200@host_ip'
session.execute("bridge", call_addr)
```

Troubleshooting

Cannot import freeswitch

Copy freeswitch.py from the python directory installed by freeswitch to /usr/lib/python2.X/{dist,site}-packages

NOTE: if you are getting this error trying to test on the python shell .. you will never get past it. The only way to test python IVR scripts is to define the script in the dialplan and call the number. There is no way to currently do any testing with mod_python scripts outside the context of an IVR running in freeswitch.

Message: 'module' object has no attribute 'EventConsumer_node_set'

If you see:

```
Message: 'module' object has no attribute 'EventConsumer_node_set'
```

You upgraded your freeswitch but obviously you forgot to update freeswitch.py within the /usr/lib/pythonX.Y/site-packages/ (X and Y are the python's version)

CoreSession_streamfile() takes exactly 3 arguments (4 given)

If you see

```
TypeError: CoreSession_streamfile() takes exactly 3 arguments (4 given)
```

it could mean that you are trying to use a dtmf callback that is a bound method of an object -- don't do that! The dtmf callback function should always be at the module scope and not take ("self") as an argument.

Error calling DTMF callback - wrong # of arguments

Same as above. You may be trying to use a bound instance method to a class. (eg, takes self as first argument). This will not work. Instead what you can do is to have a nested method for your dtmf handler that can access instance's **self** name.

'consoleLog', argument 2 of type 'char *'

If you see error messages:

```
TypeError: in method 'console_log', argument 2 of type 'char *'
```

You just need to call str() on the variable before passing to console_log, which cannot deal with Unicode strings at the present time.

Channels are not being cleaned up

This should not happen, if it does please report a bug with detailed instructions on how to reproduce. This has surfaced and been fixed a few times.

Avoid module-level global variables

If you find yourself using the **globals** keyword -- redesign your script. Concurrent calls will also be looking at the same variables, and things might not work as you expected. (These aren't "thread safety" issues, per se, since the Python GIL ensures only one thread can run python code at any given time, but just be aware that multiple threads can see/access these variables).

Build error: Python.h: No such file or directory

If you see this:

```
 freeswitch_python.h:5:20: error: Python.h: No such file or directory
```

You need to install the python-dev package. You should also double-check the src/mod/languages/mod_python/Makefile to make sure it's using the version of python you are expecting. If not, you can edit the Makefile manually.

Hangup hook + SQLAlchemy crashes switch

It's not clear if it only happens in conjunction with SQLAlchemy, but removing the hangup hook definitely fixed the problem. Hangup hooks are buggy, please avoid or use with extreme caution.

mod_python error: mod_python.c:293 Error calling python script

If you see this:

```
 [ERR] mod_python.c:293 Error calling python script
 Message: expected string or Unicode object, NoneType found
  or
 Message: expected string or Unicode object, bool found
```

Check your dtmf_process function. If you used "session.setInputCallback()" function, check your callback function. It should return **one of these strings: "true", "false" or "pause"**. Forgetting to return, or using `return True` or `return False` (as a boolean) will cause the error above.

See https://jira.freeswitch.org/browse/FS-1414