Part 4: Twisted Plugin

Creating our `twistd` command line plugin for easy deployment.

twistd Plugin Setup

First, within your `network_project` directory, create the following directory, and file within that directory

```
(NetworkProj) $ mkdir twisted
(NetworkProj) $ mkdir twisted/plugins
(NetworkProj) $ touch twisted/plugins/talkbackbot_plugin.py
```

The `twisted` directory should be in the same level as the `talkback` directory, within `network_project`. Go ahead and open up `talkbackbot_plugin.py` within your text editor.

To setup our plugin, we need a way to parse our settings configuration. For this, we use `ConfigParser` from Python's standard library:

```
from ConfigParser import ConfigParser

# <--snip-->
```

Next, we have a bunch of Twisted import statements to create our plugin (don't get scared!):

```
# <--snip-->

from twisted.application.service import IServiceMaker, Service
from twisted.internet.endpoints import clientFromString
from twisted.plugin import IPlugin
from twisted.python import usage, log
from zope.interface import implementer

# <--snip-->
```

And last, we'll import our talkback bot and quote picker function:

```python
# <--snip-->

from talkback.bot import TalkBackBotFactory
from talkback.quote_picker import QuotePicker

# <--snip-->
```

Again, notice the order of imports per Python's style guide (http://www.python.org/dev/peps/pep-0008/) grouped by standard library, third-party libraries/modules, and our own written modules, each group of import statements in alphabetical order.

Scaffolding for talkbackbot_plugin.py

We'll first want to leverage Twisted's `usage` module to parse our configuration:

```python
# <--snip-->

class Options(usage.Options):

# <--snip-->
```

Next, the actual class that constructs our application using Twisted's `Service` class to start and stop our application:

```python
# <--snip-->

class TalkBackBotService(Service):

    def __init__(self, endpoint, channel, nickname, realname, quotesFilename,
                 triggers):

    def startService(self):
        """Construct a client & connect to server."""

    def stopService(self):
        """Disconnect."""

# <--snip-->
```

To go along with our `TalkBackBotService`, we create a Maker class (similar to having our bot Factory class to create our bot) that constructs our service.

```
# <--snip-->

class BotServiceMaker(object):
    tapname = "twsrs"
    description = "IRC bot that provides quotations from notable women"
    options = Options

    def makeService(self, options):
        """Construct the talkbackbot service."""

# <--snip-->
```

Lastly, we construct an object which calls our `BotServiceMaker`:

```
# <--snip-->

serviceMaker = BotServiceMaker()
```

Let's first approach our `BotServiceMaker`.

BotServiceMaker class

First, a few settings for our class:

```
# <--snip-->

tapname = "twsrs"
description = "IRC bot that provides quotations from notable women"
options = Options

# <--snip-->
```

The `tapname` is the short string name for our plugin; this is the subcommand of `twistd`. The `description` is the short summary of what the plugin does. And the `options` variable refers to our `Options` class that we will code out in a bit.

Next, our `makeService` function:

```
# <--snip-->

def makeService(self, options):
    """Construct the talkbackbot service."""
    config = ConfigParser()
    config.read([options['config']])
    triggers = [
        trigger.strip()
        for trigger
        in config.get('talkback', 'triggers').split('\n')
        if trigger.strip()
    ]

    return TalkBackBotService(
        endpoint=config.get('irc', 'endpoint'),
        channel=config.get('irc', 'channel'),
        nickname=config.get('irc', 'nickname'),
        realname=config.get('irc', 'realname'),
        quotesFilename=config.get('talkback', 'quotesFilename'),
        triggers=triggers,
    )

# <--snip-->
```

First, we instantiate `ConfigParser()`, and read from our `options` parameter that we pass in to grab `'config'` in our options. This is essentially grabbing and reading our `settings.ini` file. Next, we create a list comprehension for `triggers`. We strip the null characters for every trigger we find in our settings.ini file. Looking at the file, we are able to pull out only the triggers with the `config.get('talkback', 'triggers')` function:

```
# <--snip-->

[talkback]

# <--snip-->

triggers =
    that's what she said
```

The `.split('\n')` means that each quote is separated by a new line.

After we setup our triggers, we then return our instantiated `TalkBackBotService` class with the parameters grabbed from our `config` variable:

```
# <--snip-->

    return TalkBackBotService(
        endpoint=config.get('irc', 'endpoint'),
        channel=config.get('irc', 'channel'),
        nickname=config.get('irc', 'nickname'),
        realname=config.get('irc', 'realname'),
        quotesFilename=config.get('talkback', 'quotesFilename'),
        triggers=triggers,
    )
```

One final bit that I didn't detail in the scaffolding: Twisted makes use of Zope's interfaces (http://docs.zope.org/zope.interface/). Earlier, we imported `implementer` from `zope.interface`. The way we will use `implementer` is a Python decorator (http://simeonfranklin.com/blog/2012/jul/1/python-decorators-in-12-steps/), and with Twisted, it is considered an interface:

```
# <--snip-->

@implementer(IServiceMaker, IPlugin)
class BotServiceMaker(object):

# <--snip-->
```

Rather than having `BotServiceMaker` inherit from both `IServiceMaker` and `IPlugin`, we use `@implementer` as a marker saying "this class implements these interfaces". You can read more about Twisted's interfaces here (http://twistedmatrix.com/documents/current/core/howto/components.html).

Options class

This is pretty simple: we need to tell our Twisted application about the options it can handle:

```
# <--snip-->

class Options(usage.Options):
    optParameters = [
        ['config', 'c', 'settings.ini', 'Configuration file.'],
    ]

# <--snip-->
```

This gives us two flags: `--config` and `-c` that we could include when we run `twistd twsrs` (remember that `twsrs` is the `tapname` for our service):

```
$ twistd twsrs --config=/path/to/settings.ini
$ twistd twsrs -c /path/to/settings.ini
```

We also feed it a default value, in this case, `settings.ini`. If you were not to include a config flag, the application would look for `settings.ini` in the current directory (same directory that the `README.md`, `settings.ini.EXAMPLE`, `quotes.txt` files live).

TalkBackBotService class

Our `BotServiceMaker.makeService` method returns an instance of `TalkBackBotService` with parameters grabbed from our configuration, definied in `settings.ini`. Now let's implement our `TalkBackBotService` class.

We'll first create a private variable `_bot` with value `None` (private is denoted with a leading `_`, and while it's not meant to be publically accessible, it isn't enforced).

We also initialize the class:

```
# <--snip-->

def __init__(self, endpoint, channel, nickname, realname, quotesFilename,
             triggers):
    self._endpoint = endpoint
    self._channel = channel
    self._nickname = nickname
    self._realname = realname
    self._quotesFilename = quotesFilename
    self._triggers = triggers

# <--snip-->
```

This `__init__` function gets called when we return `TalkBackBotService` from `BotServiceMaker.makeService` method with our settings from our parsed configuration.

Next, we'll define `startService` method, which is a part of the `Service` base class we inherit from:

```python
# <--snip-->

def startService(self):
    """Construct a client & connect to server."""
    from twisted.internet import reactor

    def connected(bot):
        self._bot = bot

    def failure(err):
        log.err(err, _why='Could not connect to specified server.')
        reactor.stop()

    quotes = QuotePicker(self._quotesFilename)
    client = clientFromString(reactor, self._endpoint)
    factory = TalkBackBotFactory(
        self._channel,
        self._nickname,
        self._realname,
        quotes,
        self._triggers,
    )

    return client.connect(factory).addCallbacks(connected, failure)

# <--snip-->
```

Our `startService` method has a few interesting things going on. We first have an import statement nested in it: `from twisted.internet import reactor`. Ashwini Oruganti (http://twitter.com/_ashfall_), a contributor to Twisted, wrote up a great blog post (http://ashfall.github.io/blog/2013/06/15/the-twisted-reactor-part-1/) detailing why we nest this import statement within `startService` method:

> If you import `twisted.internet.reactor` without first installing a specific reactor implementation, then Twisted will install the default reactor for you. The particular one you get will depend on the operating system and Twisted version you are using. For that reason, it is general practice not to import the reactor at the top level of modules to avoid accidentally installing the default reactor. Instead, import the reactor in the same scope in which you use it.

Within `startService` method, we define `connected(bot)`, which assigns our private variable we defined earlier, `_bot`, to the passed-in parameter, `bot`.

We also define `failure(err)` within `startService` to log that we could not connect to a specific service, along with the error message the failure gave us. We then stop our reactor upon calling `failure`.

Next, we instantiate the `QuotePicker` class with our quote file with defining `quotes`. This pulls in all our quotes within `quotes.txt` file.

Now we need to define a `client` that basically constructs an endpoint based on a string with `clientFromString` function. The `clientFromString` takes in the `reactor` that we imported, and the endpoint, which is grabbed from the endpoint string defined in our `settings.ini` file. The `reactor` Twisted's event loop driving your Twisted applications. More about Twisted's `reactor` object is detailed in its howto documentation (http://twistedmatrix.com/documents/current/core/howto/reactor-basics.html).

We then create a `factory` variable that instantiates `TalkBackBotFactory` defined in `bot.py` which passes in the appropriate parameters.

Last, we return `client`, defined by our endpoint, and connect to our endpoint with the `factory` variable. We also add `addCallbacks` which take a pair of functions of what happens on success and on failure (our `connected` and `failure` functions).

The last function we define in our `TalkBackBotService` class is `stopService`:

```python
# <--snip-->

def stopService(self):
    """Disconnect."""
    if self._bot and self._bot.transport.connected:
        self._bot.transport.loseConnection()

# <--snip-->
```

It is a deferred (http://twistedmatrix.com/documents/current/core/howto/defer.html) (a callback which we put off until later) that is triggered when the service closes our connection between the client and server (if `_bot` is not `None`, and if the bot is connected).

Near the home stretch!

Constructing BotServiceMaker

At the very end of our plugin module, we have to include: `serviceMaker = BotServiceMaker()` to construct an object which provides the relevant interfaces to bind to `IPlugin` and `IServiceMaker`.

Completed talkbackbot_plugin.py

```
from ConfigParser import ConfigParser

from twisted.application.service import IServiceMaker, Service
from twisted.internet.endpoints import clientFromString
from twisted.plugin import IPlugin
from twisted.python import usage, log
from zope.interface import implementer

from talkback.bot import TalkBackBotFactory
from talkback.quote_picker import QuotePicker


class Options(usage.Options):
    optParameters = [
        ['config', 'c', 'settings.ini', 'Configuration file.'],
    ]


class TalkBackBotService(Service):
    _bot = None

    def __init__(self, endpoint, channel, nickname, realname, quotesFilename,
                 triggers):
        self._endpoint = endpoint
        self._channel = channel
        self._nickname = nickname
        self._realname = realname
        self._quotesFilename = quotesFilename
        self._triggers = triggers

    def startService(self):
        """Construct a client & connect to server."""
        from twisted.internet import reactor

        def connected(bot):
            self._bot = bot

        def failure(err):
            log.err(err, _why='Could not connect to specified server.')
            reactor.stop()

        quotes = QuotePicker(self._quotesFilename)
        client = clientFromString(reactor, self._endpoint)
        factory = TalkBackBotFactory(
            self._channel,
            self._nickname,
            self._realname,
            quotes,
            self._triggers,
        )

        return client.connect(factory).addCallbacks(connected, failure)

    def stopService(self):
        """Disconnect."""
        if self._bot and self._bot.transport.connected:
            self._bot.transport.loseConnection()


@implementer(IServiceMaker, IPlugin)
class BotServiceMaker(object):
    tapname = "twsrs"
    description = "IRC bot that provides quotations from notable women"
    options = Options

    def makeService(self, options):
        """Construct the talkbackbot service."""
        config = ConfigParser()
        config.read([options['config']])
        triggers = [
            trigger.strip()
            for trigger
            in config.get('talkback', 'triggers').split('\n')
            if trigger.strip()
        ]

        return TalkBackBotService(
            endpoint=config.get('irc', 'endpoint'),
            channel=config.get('irc', 'channel'),
            nickname=config.get('irc', 'nickname'),
            realname=config.get('irc', 'realname'),
            quotesFilename=config.get('talkback', 'quotesFilename'),
            triggers=triggers,
        )

# Now construct an object which *provides* the relevant interfaces
# The name of this variable is irrelevant, as long as there is *some*
# name bound to a provider of IPlugin and IServiceMaker.

serviceMaker = BotServiceMaker()
```

← Part 3: Bot Module (/networks/part-3/)

Part 5: Testing the Bot → (/networks/part-5/)