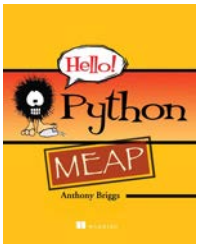
MANNING PUBLICATIONS



Hello! Python
By Anthony Briggs

In this article from chapter 10 of Hello! Python, author Anthony Briggs Talks about extending an application using the Twisted framework to make it interactive over the Internet.

You may also be interested in…

Writing a Simple Chat Server

We'll build a simple chat server so we can log in and play a game with other people via the Internet. Normally these games are referred to as MUDs, which stands for multiuser dungeons. Depending on the person creating them, MUDs can range from fantasy hack-and-slash to science fiction, and players can compete or cooperate to earn treasure, points or fame.

We'll use a framework called Twisted, which contains libraries for working with many different networking protocols and servers.

Getting started

The first step is to install Twisted and get a test application running. Twisted comes with installers for Windows (figure 1) and Macintosh, which are available from the Twisted homepage at http://twistedmatrix.com/. If you're using Linux, there should be packages available through your package manager.

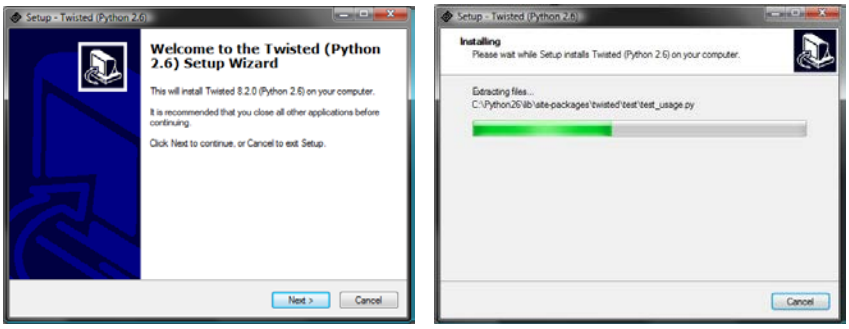


Figure 1 Installing Twisted on Windows

The installer will pop up a window as it compiles things but, once you see the window on the right in figure 1, Twisted is installed!

The only other thing that you need is a telnet application. Most operating systems come with one built in, and there are many free ones that you can download. I normally use PuTTY, which is available for Windows.

Our application

We'll now write our simple chat server. It's a little more complex than Hello World, but we can extend this program and use it for gaming. Open a new file and save it as something like `chat_server.py`.

Let's start with the first part of our application, the protocol for our chat server. In Twisted terminology, a protocol refers to the part of your application that handles the low level details—opening connections, receiving

data, and closing connections when we're finished. You can do this in Twisted by subclassing its existing networking classes.

Listing 1 A simple chat server protocol

```
from twisted.conch.telnet import StatefulTelnetProtocol          #1

class ChatProtocol(StatefulTelnetProtocol):                      #1
    def connectionMade(self):                                    #2
        self.ip = self.transport.getPeer().host                  #3
        print "New connection from", self.ip                     #3
        self.msg_all(                                            #3
            "New connection from %s" % self.ip,                  #3
            sender=None)                                         #3
        self.factory.clients.append(self)                        #3

    def lineReceived(self, line):                                #4
        line = line.replace('\r', '')
        print ("Received line: %s from %s" %                     #4
               (line, self.ip))                                  #4
        self.msg_all(line, sender=self)                          #4

    def connectionLost(self, reason):                            #5
        print "Lost connection to", self.ip                      #5
        self.factory.clients.remove(self)                        #5

    def msg_all(self, message, sender):                          #6
        self.factory.sendToAll(                                  #6
            message, sender=sender)                              #6

    def msg_me(self, message):                                   #6
        message = message.rstrip() + '\r'                        #6
        self.sendLine(message)                                   #6
```

#1 ChatProtocol is like Telnet

For our chat server, we'll be using Twisted's `StatefulTelnetProtocol`. It takes care of the low-level line parsing code, which means that we can write our code at the level of individual lines and not have to worry about whether we have a complete line or not.

#2 Override connectionMade

We're customizing our protocol by overriding the built-in `connectionMade` method. This will get called by Twisted for each connection the first time that it's made.

#3 A new connection

We're just taking care of a bit of housekeeping here—storing the client's IP address and informing everyone who's already connected of the new connection. We also store the new connection so that we can send it broadcast messages in the future.

#4 Handle data

The Telnet protocol class provides the `lineReceived` method, which gets called whenever a complete line is ready for us to use (whenever the person at the other end hits the return key). In our chat server, all we need to do is send whatever's been typed to everyone else who's connected to the server. The only tricky thing that we need to do is to remove any line feeds; otherwise, our lines will overwrite each other when we print them.

#5 Close the connection

If the connection is lost for some reason—either the client disconnects or is disconnected by us, `connectionLost` will be called so that we can tidy things up. In our case, we don't really need to do much, just remove the client from our list of connections so that we don't send them any more messages.

#6 Convenience methods

To make our code easier to follow, I've created the `msg_all` and `msg_me` methods, which will send out a message to everyone and just us, respectively. `msg_all` takes a `sender` attribute, which we can use to let people know who the message is coming from.

So that takes care of how we want our program to behave—how do we link it in to Twisted? We use what Twisted refers to as a Factory, which is responsible for handling connections and creating new instances of `ChatProtocol` for each one. You can think of it as a switchboard operator—as people connect to your server, the Factory creates new Protocols and links them together—something like figure 1.

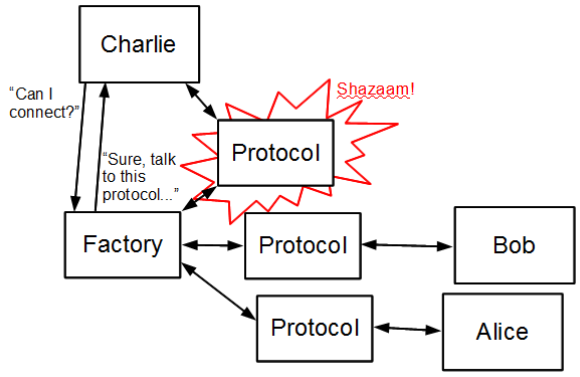


Figure 1 A factory creating protocols

So how do we do this in Twisted? Easy. See listing 2.

Listing 2 Connecting up our protocol

```
from twisted.internet.protocol import ServerFactory      #1
from twisted.internet import reactor                     #3
...
class ChatFactory(ServerFactory):                        #1
    protocol = ChatProtocol                              #1

    def __init__(self):
        self.clients = []                                #2

    def sendToAll(self, message, sender):                #2
        message = message.rstrip() + '\r'                #2
        for client in self.clients:                      #2
            if sender:                                   #2
                client.sendLine(                         #2
                    sender.ip + ": " + message)          #2
            else:                                        #2
                client.sendLine(message)                 #2

print "Chat server running!"                             #3
factory = ChatFactory()                                  #3
reactor.listenTCP(4242, factory)                         #3
reactor.run()                                            #3
```

#1 A ChatFactory?

A Factory is object-oriented terminology for something that creates instances of another class. In this case, it'll create instances of `ChatProtocol`.

#2 Talking to everyone

The `ChatFactory` is the natural place to store the data shared between all of the `ChatProtocol` instances. The `sendToAll` method is responsible for sending a message to each of the clients specified within the clients list. As you saw in listing 1, the client protocols are responsible for updating this list whenever they connect or disconnect.

#3 Wiring everything together

The final step is to let Twisted know about our new protocol and factory. We do this by creating an instance of `ChatFactory`, binding it to a particular port with the `listenTCP` method and then starting Twisted with a call to

its main loop, `reactor.run()`. I've chosen 4242 as the port to listen to. It doesn't matter too much which one you use, as long as it's something above 1024 so that it doesn't interfere with the existing network applications.

If you save that program and run it, you should see the message "Chat server running!" If you connect to your computer via telnet on port 4242 (usually by typing `telnet localhost 4242`), you should see something like figure 2.

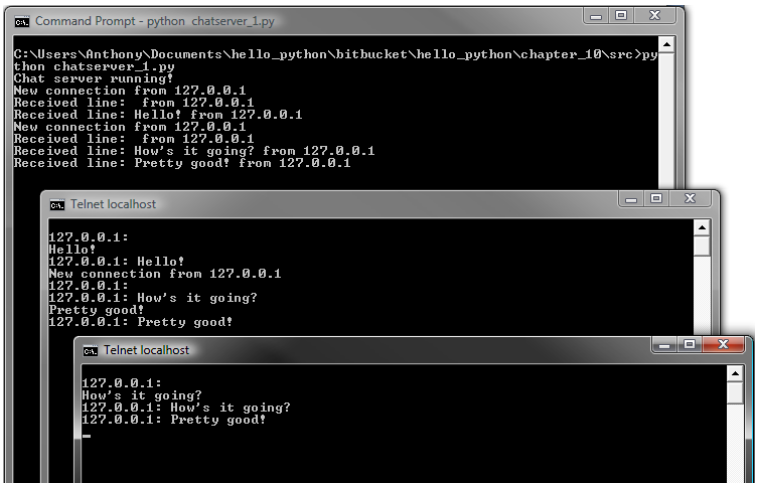


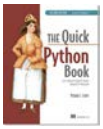
Figure 2 Our chat server is running

It may not seem like much, but we've already got the basic functionality of our MUD server going. Now we're ready to start connecting a game to the network.

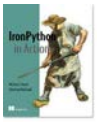
Summary

We learned how to set up a chat server so we can log on and interact with other game players on the Internet. We first set up the protocol for our chat server and then we linked it to Twisted, which contains libraries for working with many different networking protocols and servers.

Here are some other Manning titles you might be interested in:



The Quick Python Book, Second Edition
Vernon L. Ceder



IronPython in Action
Michael J. Foord and Christian Muirhead

Hello World!
Warren D. Sande and Carter Sande

Last updated: February 2, 2012