LINUX VOICE TUTORIAL

PYTHON: BUILD DYNAMIC WEB PAGES

BEN EVERARD

Keep your websites up to date by harnessing live data piped from Python's web services.

WHY DO THIS?

- Keep your websites up to date with the latest information
- Pull data from across the web and feed it into your programs
- Discover the powerful combination of Python and Tornado

HTML is one of the greatest developments in computing. It's so simple, anyone with a text editor can knock up a simple web page with text, pictures, and links to other sites. This simplicity gives the web the potential to grow to encompass almost the whole of humanity. However, its original developers intended it for content that doesn't change very often. Every change to a page of HTML needs the file to be modified and resaved, which is fine for some purposes; but sometimes you need something a little more dynamic. In this tutorial we're going to look at four different methods for including constantly changing content in your website.

Since we've got a lovely new magazine, we're going to create a lovely new website to help us keep track of everything that's going on. The skeleton code for this website is:

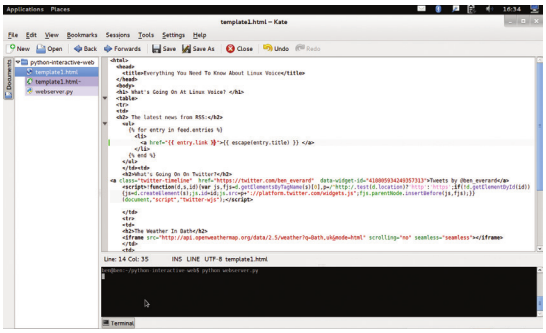
```
<html>
<head>
<title>All About Linux Voice</title>
</head>
<body>
<h1>All About Linux Voice</h1>
<table>
```

Useful Tornado template

Tornado templates are based on Python, but they have their own simple language. Here are a few of the most useful commands:

- **{% set var_x = a_value %}** Sets local variable **var_x** to the value **a_value**.
- **{% if condition_1 %} … {% elif condition_2 %} … {% else %} … {% end %}** An if statement. elif and else are optional.
- **{% while condition_1 %} … {% end %}** A normal while loop.
- **{% import a_module %}** Import the Python module **a_module**.
- **{% include a_template %}** Copy the contents of **a_template** to this point in the file.

There are full details of the template syntax at **www.tornadoweb.org/en/stable/template.html**. In general, it's best to do as much of the processing as possible in the web server, and use the template just to display the data. You can use the included commands to create various components that you can combine in different ways on different pages.



A good text editor will highlight different parts of the code, so you can see what part does what.

```
<tr>
<td>Data1</td>
<td>Data2</td>
</tr>
<tr>
<td>Data3</td>
<td>Data4</td>
</tr>
</table>
```

If you haven't come across HTML before, everything is kept between pairs of angular bracketed tags that describe what the content is. For example, **<h1>** marks the start of heading 1 (the biggest heading), and **</h1>** tells the browser that we've finished the heading. The **<table></table>** tags describe a table, **<tr></tr>** describe a table row, and **<td></td>** describe a table cell. The skeleton code can be found on the coverdisc or at **linuxvoice.com** as **lv-skeleton.html**.

In this skeleton, Data1 to 4 are the places we'll put four different pieces of dynamic content.

As a British magazine, the most important thing to us is obviously the weather, and this changes a lot. If we kept looking out of the window, and updating our website every time the weather changed, we'd have no time to make tea, let alone a magazine. Fortunately, though, we don't have to. The first, and easiest, method of including dynamic content we'll look at is an iframe. These enable you to embed other websites inside yours. In this case, we'll embed a weather forecast. You can put in any website, though it's best to do it with one designed for the purpose, otherwise it's unlikely to look good. For our purposes, **openweathermap.com** provides exactly what we need. The website **http://api.openweathermap.org/data/2.5/weather?q=Bath,uk&mode=html** is a

compact forecast for the beautiful city of Bath, designed for embedding.

In the skeleton, you can change **Data1** to the following:

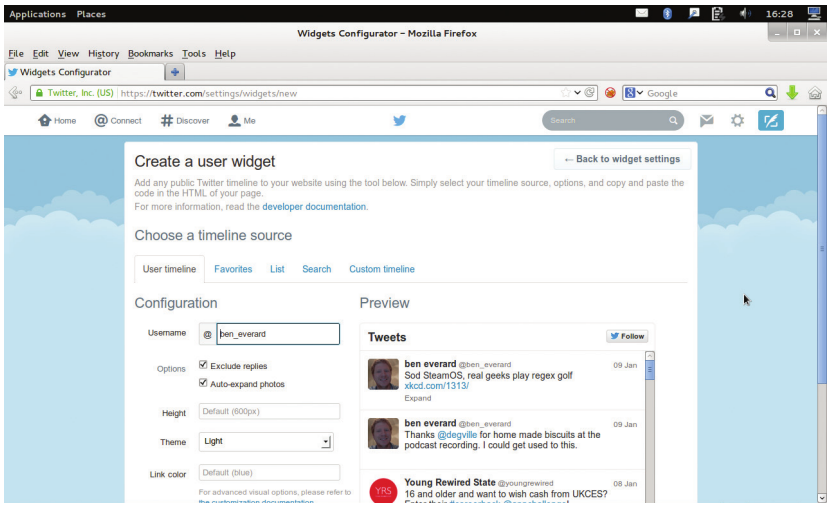
```
<h2>The Weather In Bath</h2>
<iframe src="http://api.openweathermap.org/data/2.5/weather?q=Bath,uk&mode=html" scrolling="no" seamless="seamless"></iframe>
```

This will embed the weather forecast in our website. The **scrolling** value tells the browser that we don't want a scroll bar on the iframe, and **seamless** tells it that it should be integrated into our page seamlessly. Not all browsers recognise these, so it will appear slightly different on different platforms.

Keepin' Tweetin'

Iframes are the most basic way to grab data and serve it in a web page. For simple things like weather forecasts they work great, but sometimes they're a bit lacking. Some data providers provide 'widgets' that you can put in your page. These are generally small chunks of HTML, usually with some JavaScript to grab data and display it in a useful way. For our Linux Voice website, we'll add a widget that grabs the Linux Voice Twitter feed.

You can create Twitter widgets for any Twitter account. On the Twitter web page, just go to the cog menu icon, then Settings > Widgets > Create New. By default it'll set it to the currently logged-in account, but you can change this to whatever you like. We also changed ours to have a height of 300. Once you've entered the details and hit Create, the appropriate code will be displayed below the preview. You just need to copy and paste it in place of **Data2**. The code



You can create Twitter widgets to show anyone's tweets, but if they post something inappropriate, it will be displayed on your site as well.

we used was:

```
<a class="twitter-timeline" href="https://twitter.com/LinuxVoice" data-widget-id="419158898222698496">Tweets by @LinuxVoice</a><script>!function(d,s,id){var js,fjs=d.getElementsByTagName(s)[0],p=/^http:/.test(d.location)?'http':'https';if(!d.getElementById(id)){js=d.createElement(s);js.id=id;js.src=p+"://platform.twitter.com/widgets.js";fjs.parentNode.insertBefore(js,fjs);}}(document,"script","twitter-wjs");</script>
```

Of course, you can create your own.

Don't worry about trying to understand this script (unless you're a JavaScript masochist) as it's computer generated and not meant to be human readable. Save the file and refresh your web browser and you should now have the weather and the latest news from Twitter all without having to handle any of it yourself. There are a few options on the Create Widget Twitter page to help you control the look and feel of this datastream, so see which settings work best with your page.

Get more control

The problem with the two previous methods is that they pull everything from the other website, so as well as the data you get the other site's formatting too. Sometimes this isn't a problem and the simplicity is worth it. Other times you may find that you want more control over how the content is displayed, or even the ability to process it before putting it on the screen. Another risk in putting content from other places on your website is that they could maliciously alter your page using JavaScript. It's unlikely that either Twitter or OpenWeatherMap would do this deliberately, but if hackers managed to break into the main system, they could use this to attack all the web pages that pull data from there.

"You may find that you want more control over how the content is displayed"

Therefore, it's better if you don't just put other people's content directly into your site, but process the data and produce HTML that uses the raw data. For this we're going to use Python.

Make it more dynamic

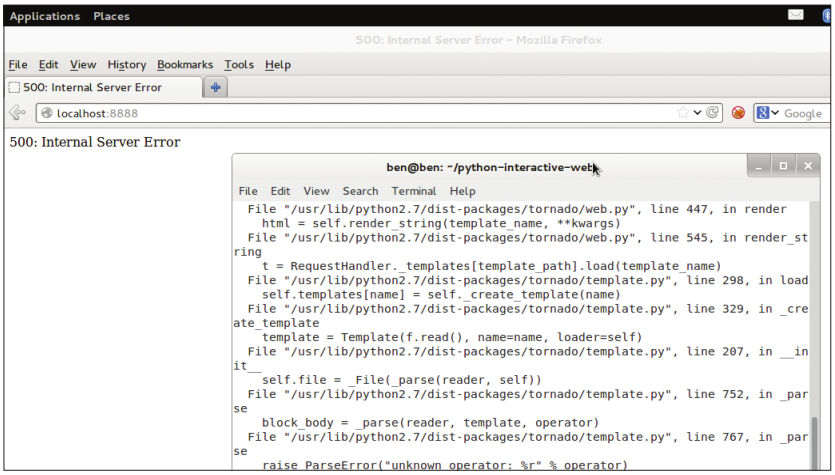
Server-side processing is great for keeping a site updated, but it has one fatal flaw: it only updates the information every time the website is loaded. Sometimes you need to keep a page's information fresh even if the user leaves it loaded.

The simplest solution is simply to tell the browser to keep refreshing the page. This is incredibly simple – just add the following tag inside the **<head></head>** tags of the template:

```
<meta http-equiv="refresh" content="60" >
```

The **content** value is the number of seconds after loading you want it to refresh.

This method is a little crude, but it will work. A more advanced technique is to keep a connection open between the browser and the server and continue to send data between them. There are ways of doing this using HTTP, but a better solution is to use websockets. These require both code on the server and JavaScript running in the browser in order to work properly, and they're a bit beyond the scope of this tutorial, but you can find out how to use them on the Tornado website at **www.tornadoweb.org/en/stable/websocket.html**.



If there's a problem with the template, the site won't load. You'll get Python errors, but they aren't usually very helpful.

The Tornado module contains a web browser that lets you modify templates by passing more information to them. To start with, you'll need to make sure you have the appropriate Python modules installed. We'll be using Tornado and Feedparser (as well as some modules from the Python standard library). These are available through the PIP (Python Install Python) package manager for Python, but it'll be easier to keep them up to date if you install them through your distro's package manager. On Debian-based systems you can do this with:

```
sudo apt-get install python-tornado python-feedparser
```

Once this is done, you just need a simple Python program to serve the website. We've called this code **webserver-start.py** and it's on the DVD and website.

```
import tornado.ioloop
import tornado.web
class MainHandler(tornado.web.RequestHandler):
  def get(self):
    self.render("lv.html")
application = tornado.web.Application([
  (r"/", MainHandler),])
if __name__ == "__main__":
        application.listen(8888)
        tornado.ioloop.IOLoop.instance().start()
```

We won't go into everything that's going on here (you can learn more about Tornado from the excellent documentation at **www.tornadoweb.org/en/stable**), but simply put, this starts a web server on port 8888 of localhost. It has a class called **MainHandler**, which is used every time someone visits the root of this web server (ie **r"/"** in the above code). The method get of this class is called every time someone sends a GET HTTP request to this address, and it renders the template **lv.html**. (Make sure the HTML file you created before is called **lv.html**). As long as you save this in the same directory as **lv.html**, you can run it from a terminal in the same directory with:

```
python webserver-start.py
```

Once that's running, you can point a web browser to **http://localhost:8888** and it'll display the same page as before. The difference is that it's now a Tornado template, which has more power than regular HTML, and you can pass it data from the Tornado server.

A Yen a Mark a Buck or a Pound

As Linux Voice does a lot of business in the USA, changes in the exchange rate between the Dollar and the Pound make a difference to our income. Keeping tabs on this is important, so the next bit of data we pull in will be the latest exchange rate.

www.openexchangerates.org operates a service that enables you to grab the latest exchange rate data (you'll need to register for an API key before you can use it though). There are various levels, but the free one is suitable for our needs, and you can sign up for it here: **https://openexchangerates.org/signup/free**.

The data comes in JSON (JavaScript Object Notation) format. While this was designed for JavaScript, it also works really well with Python.

There are a couple of Python modules that help us get and access the data: **urllib2** and **json**. The code to grab and access the data is:

```
import urllib2
import json
def getRate() :
        url = "https://openexchangerates.org/api/latest.json?app_id=YOUR_API_KEY"
        req = urllib2.Request(url)
        response=urllib2.urlopen(req)
        return json.loads(response.read().decode("UTF-8"))['rates']['GBP']
```

This piece of code needs to go into the **webserver-start.py** file between **import tornado.web** and **class MainHandler**. Change **YOUR_API_KEY** to the one you got when you signed up for the service.

urllib grabs and opens the resource, then the **json** module converts it into a Python dictionary. This has the key rates, which is another dictionary, and the key GBP returns the Dollars–Pounds exchange rate. You then need to pass the latest data across to the template by changing the line **self.render("lv.html")** to:

```
self.render("lv.html", rate = getRate())
```

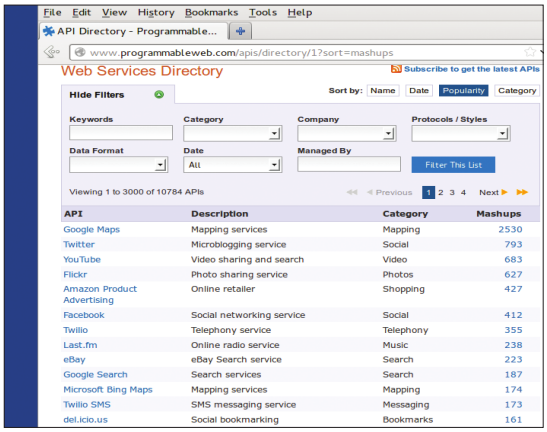
Data sources

There are loads of places you can get information for dynamic websites. OpenWeatherMaps provide JSON-encoded weather data for forecasts as well as current weather. Twitter also has an API that's easy to use through a module such as **python-twitter** (**http://code.google.com/p/python-twitter**).

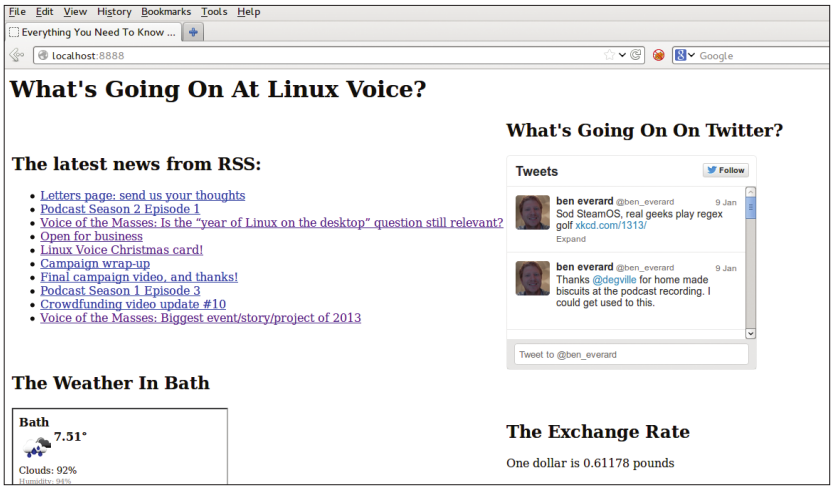
In addition to the ones we've looked at here, these are some more that you may find useful:

- **Facebook Graphs API** (**https://developers.facebook.com/docs/graph-api**)
- **IPInfoDB** (**http://ipinfodb.com/ip_location_api_json.php**) enables you to check the location of an IP address.
- **The BBC** (among others) publishes an RSS feed of the latest news. It also has a few APIs to help you access information about what's happening.
- **Reddit** can be browsed through JSON. For an example, take a look at **www.reddit.com/r/linux/hot.json**. For more information see **www.reddit.com/dev/api**.
- **StopForumSpam** hosts a database of known spammer IPs that you can use to vet visitors, though there are some restrictions on use. Take a look at **www.stopforumspam.com/usage**

These are just a few examples; there is a huge range of data sources available. Many offer free access, but some are only for paying customers.



You'll find a comprehensive list of useful data sources at www.programmableweb.com.



We focused on simplicity so it's not pretty to look at, but there are plenty of CSS and HTML tricks to sort that out.

This created the global variable rate that you can access in the HTML template with **{{ rate }}**. Change **Data3** to:

```
<h2>The Exchange Rate</h2>
One dollar is {{rate}} pounds
```

After you make any changes to either the webserver code or the template, they won't take effect until you restart the web server (a simple Ctrl+C to stop it, then re-running **python websever.py** does this). You can then refresh the website in the browser. If everything's worked correctly, you should see the exchange rate displayed. You can use this method to put whatever you want into the web page. This could be things you've just pulled from a database, or information about the computer that you're running on as well as data grabbed from other sources.

Going Loopy

Tornado templates can do far more than just display the values that are passed to them. They can also include bits of Python code that can manipulate the data. The final piece of our datastream will demonstrate this. We'll pull in the latest posts from the Linux Voice website using RSS and the **feedparser** module. This works a bit like the **json** module in that it pulls in data and converts it into a Python dictionary. However, unlike in the previous example, this time we'll pass the entire dictionary to the template and process it there. You'll need to add the line

```
import feedparser
```

To the start of **webserver.py**, then change the get method of MainHandler to:

```
lv_feed = feedparser.parse('http://www.linuxvoice.com/feed/')
self.render("lv.html", feed = lv_feed, rate = getRate())
```

You don't need to use **urllib2** to get the document with RSS, as the feedparser module handles everything.

On our page, we want to loop through every entry in the RSS file and display the post title as a link to the post on **linuxvoice.com**. In the template, you can include Python code inside **{% %}** brackets. Indentation doesn't work; instead, blocks of code are ended using **{% end %}.** This is done with the following code (in place of **Data4**):

```
<h2> The latest news from RSS:</h2>
<ul>
{% for entry in feed.entries %}
<li>
<a href="{{entry.link}}">{{ escape(entry.title) }} </a>
</li>
{% end %}
</ul>
```

The final version of **webserver.py** is on the website and DVD as **webserver-final.py**.

**
** creates an unordered (ie bullet pointed) list, and **
** tag items in the list.

This for loop repeats every line between it and the end, including the HTML lines. This will then create a new list item for every item in **feed.entries**. The **escape()** function just adds escape characters to the text before passing them across so they display correctly in the browser.

As you can see, any data that you can access with Python, you can display on a website. Tornado templates give you complete control over how these are displayed. If you're already running a website with Apache, it isn't easy to incorporate this last technique into it, though you could do something similar with PHP or even JavaScript. If you're using Nginx as a web server, you could set it up to reverse-proxy a Tornado server for some pages while retaining the speed of Nginx for simpler pages.

XML

We've looked at JSON, HTML and RSS for data sources, but they're not the only options. XML is also a common format for data on the web, though it can be a little more complex than the others. It's often done using ElementTree. As the name suggests, this converts the XML into a tree from which you can then extract the information you need.

Ben Everard is the co-author of *Learning Python With Raspberry Pi* – coming soon to an Amazon near you.