Using the Echo Nest API, I was able to write the beginning portions of a JavaScript library that creates requests and parses responses for various Echo Nest API calls. Here I will be presenting a total of six library functions, which correspond to the three API calls needed to request the information that was previously displayed about your favorite artist.

**Artist Hotttnesss**

Hotttnesss is a parameter defined by Echo Nest as “how hottt an artist currently is”. [http://developer.echonest.com/docs/v4/artist.html#hotttnesss]<a href=></a> A blog post on Music Machinery mentions that hotttnesss is an indicator of how much buzz there is around a current artist. This could correspond to how frequently their name is mentioned in blog posts, reviews, play counts, etc. [http://musicmachinery.com/2009/12/09/a-rising-star-or/]<a href=>Hottt or Nottt</a>

Hotttnesss is a value between 0 and 1, where the higher the value the more buzz the artist has. Significantly popular artists like David Guetta have a hotttnesss around 0.7, where artists like Sleepy have a hotttnesss closer to 0.3. Values outside of [0.3, 0.7] are very rare. Even The Beatles have a hotttnesss of 0.776.

getArtistHotttnesss

This function will create the string sent to Echo Nest during a request. The only parameter it accepts is the artist name, although it does expect Echo Nest developer credentials to be supplied via an external file. The string returned from this function can immediately be used in the request like so:

<code>

// example usage

var hotttString = getArtistHotttnesss(context.favArtist);

request(hotttString, function(hotttError, hotttResponse, hotttBody){

//callback

});

</code>

parseArtistHotttnesss

The response from a request for hotttnesss holds several values, but generally the value we care about is the hotttnesss value itself. parseArtistHotttnesss returns that value if the request was successful. Otherwise, it returns the error message from the response status. It takes the entire response object.

<code>

// example usage

var hotttString = getArtistHotttnesss(context.favArtist);

request(hotttString, function(hotttError, hotttResponse, hotttBody){

if(!hotttError && hotttResponse.statusCode < 400){

context.hotttnesss = parseArtistHotttnesss(hotttBody);

}

});

</code>

INSERT ACTUAL LIBRARY CODE HERE

**Artist Images**

Echo Nest doesn’t have an API call for requesting a single image. Instead, it has a call for requesting several images. The hottter the artist, generally the more images are available.

And since we are talking about making requests to and from webservers, the response doesn’t contain jpegs or pngs; instead, it contains web links to said jpegs and pngs. Getting the resulting picture to display requires that the server hosting the image is up and running. (While utilizing this particular call, I have noticed that the first sets of images returned by Echo Nest are frequently hosted on a server that doesn’t respond and may very well not be running at all. If developing a professional website, I would put effort into protecting against this sort of behavior, but as my focus is on the content of this HowTo, and less so on developing a pristine browsing experience, we will all have to suffer with broken images.)

getArtistImages

As it sounds, getArtistImages builds the request string for requesting images of a particular artist. In order to do so, it takes in the artist, a quantity and a starting index as parameters. Along with the request payload, Echo Nest will provide information about how many artist images it has. The start value indicates from where in this entire list to start pulling images.

It returns an array of length quantity where each element is an object containing information about a single image.

parseArtistImages

parseArtistImages takes the response body from the images request. If an error was received, it reports the error; otherwise it returns an array of image objects. Image objects contain a url to an image file (whether or not the server on the receiving side of that url responds seems to be questionable as noted above) and some licensing information for that image. If an image doesn’t contain a url (dynamic data can do weird things, see the Warning page) parseArtistImages will write a warning to the console and skip the image to deal with the rest.

Together the two functions work as so:

<code>

// example usage

var imagesString = getArtistImages(context.favArtist, 1, 0);

request(imagesString, function(imagesError, imagesResponse, imagesBody){

if(!imagesError && imagesResponse.statusCode < 400){

context.images = parseArtistImages(imagesBody);

}

});

</code>

INSERT ACTUAL LIBRARY CODE HERE

Artist Biographies

Similarly to images, Echo Nest doesn’t have a call for requesting a single artist biography. Instead, there is a call for getting artist biographies, from which one or more biographies can be extracted.

getArtistBiographies

Let’s take a look at how to request some biographies. The getArtistBigraphies takes the artist, a quantity and a start index, just like getArtistImages did. The parameters represent the same concepts: the artist whose biographies we’d like to get, the number of biographies to get, and the start index so we can let Echo Nest know from where in the entire pile of biographies we’d like to pull.

parseArtistBiographies

Like parseArtistImages, parseArtistBiographies is expected to parse through a payload that contains a varying number of biographies. It creates an array of biography objects, where the objects contain the text of the biography, a site for where the biography came, a url for obtaining more information and licensing information for the biography. Some biography objects also contain another item: truncated. If the object contains this item, its value will always be true. Truncated indicates that the biography has been abbreviated (usually significantly) but more information is available at the url.

parseArtistBiographies sorts the biographies found in the response by whether or not the biography has been truncated. Full biographies are placed at the beginning of the returned array, whereas truncated ones are pushed to the end of the array. This would make it easy for someone using my library to display a full biography with only the first element returned by this method. If a user wanted to display a truncated biography, they would find the length of the array and display the element at length-1.

Just like the other request/response dual methods, obtaining artist biographies works like so: