Creating High Definition Visuals for RadioDNS

Nick Piggott - 2019-01-24

Summary

Vehicles can display very high definition visuals, but have a variety of screen resolutions / dimensions. Using virtual browser screen capture techniques combined with responsive templates defined using HTML5 allows JPG images to be quickly generated at arbitrary resolutions, and delivered to the head unit.

Main Concepts of RadioVIS

The RadioVIS standard (TS 101 499 v3.1.1) segments visuals delivery into two stages:

- Signalling - a simple text protocol over an “always open” TCP/IP connection that tells the head unit to load and/or show images (JPG or PNG), along with one associated link URL
- Transfer - the head unit retrieves the image using HTTP, providing its screen resolution details in the request

Implementing Signalling

Signalling can be provided using two protocols:

- STOMP - a simple text protocol using a message broker like ActiveMQ, where the client opens a persistent socket connection to the broadcaster’s server which stays open. The client subscribes to a topic, which defines which radio station(s) it wants to receive messages from.
- COMET - an HTTP protocol (which may be able to traverse network proxies where STOMP can’t), where the client will request a URL but the server will wait until a new messages is to be sent before responding and closing the socket. The client then immediately re-requests the URL to be able to receive the next message.

The signalling is used to tell the client when to load and display images. Usually the server is connected to the playout system of the radio station, which will generate a message when the on-air item changes. This cascades to the signalling server, and an instruction distributed to all connected clients (subscribed to that station):

```
trigger-time: now
link: (associated link URL)
SHOW <url of image to display>
```

An example of a RadioVIS signalling server implemented in node.js is available at https://github.com/peteredhead/node-radiovis

Other examples are in the Visuals section of https://radiodns.org/developers/libraries/

The nature of the signalling means that this process needs to be “always on”, awaiting new connections from clients.

Implementing High Definition Visuals

Visuals can be generated by requests by clients, which means they lend themselves well to on-demand scripting like Amazon Lambda.

The incoming HTTP request from the client will specify the URL (which should be unique for each version of a visual).

The URL can be used to do all or any of these things:

- Select a specific template
- Provide variables to populate that template (background image information, artist title)
- Provide a reference to a metadata collection on the server that can be used to populate the template

The x, y and dpi resolution are provided by the client in custom HTTP headers: Display-Height, Display-Width, Display-PPI. The Display-PPI will give an approximate value of the physical size of the screen.

Designing the Template

The template should be designed in HTML5 and CSS, using responsive techniques to allow for re-sizing from 200px x 200px up to 1920px x 1080px. Changes might involve the number of elements on screen, their positioning, and the size of text.

In general, any “hero image”, such as an artist photo, station logo, or album cover is most easily included in the design as a background element with appropriate cropping / positioning / tiling directives.

Creating the image

The image (JPG or PNG) can be created using a headless / virtual browser. A good candidate is Puppeteer (https://pptr.dev/).

Create an instance of Puppeteer, and set its viewport dimensions to the those requested by the client. Load the template URL into the view and when it has finished rendering, take a screenshot.

The example code to do this is shown at https://www.scrapehero.com/how-to-take-screenshots-of-a-web-page-using-puppeteer/ in the final section (“Taking a screenshot at a particular viewport”).

Caching the image

You should consider storing / caching the screenshot image so that you can serve it immediately to subsequent client requests without having to regenerate the image each time.

You might consider using a transparent cache like varnish or CloudFront to achieve this.

Suggested Cloud implementation

1. Deploy a medium-resource cloud instance to run the signalling server (node-radiovis). This will accept connections from clients, and receive update information from radio stations.
2. Deploy on-demand scripts to load the templates and render them to images.
3. Deploy caching for the templates (and assets) and for the generated images.