



**D&D Figurines (Tech Studio Project)**

Using

NFC Tools, NFC Tags & PhoneGap Build

By

Lauren Harvey

# Step #1 – Layout & Concept

This concept of this app is to load text data onto an NFC tag (built into a figurine) and when scanned by an NFC-compatible device, it loads that data into a pre-built template in a custom-made app.

Of course, in order to load data into a pre-built template, a template must be created. The template for the app was very simple – all of the required information needed in D&D would be structured in a table format (without using tables, of course) which can be seen below:



The table above is an image representing the figurine being tagged.

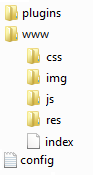
Once the layout is designed, it must be saved as index.html.

## Stumbles, Breakthroughs & Victories:

While the technical creation of the table and app in general was simple, there were issues with formatting everything so that regardless of the device, all of the tables would still look the same. On top of this, it was a time-consuming effort to determine which factors were important to include in the app and which could be avoided. That being said, I made significant breakthroughs in learning about nesting divs and how to lay things out in ways specific to my needs.

# Step #2 – App

The next step is incredibly important, which involves building the app itself. I chose to create the app using PhoneGap Build because I had some previous experience with it. With this build, you need two things – a Github repository for reference and a file stack with at the very least an index.html file. A general file template will look something like this:



Mine is only slightly different and can be found at my Github address:

<https://github.com/laurenhrv/NFC-Figurines>

Taking the layout created in step #1, save it as index.html and add any JS, CSS or img files into the corresponding folders. ‘Res’ folder includes any icons needed for the app, ‘plugins’ includes any extensions you would like to add to your project and ‘config’ is a general PhoneGap xml configuration file that can be found online.

Take this file set up and upload it to a Github repository.

“Stumbles, Breakthroughs & Victories” for this step can be found at the end of the next step.

# Step #3 – NFC

This step involves making my app compatible with NFC tags. This could be done in several different ways depending on how you create your app, but I will share how I was successful.

It’s assumed at this point that you have NFC tags available for testing. I purchased plain white NFC stickers from <http://nfctags.tagstand.com>.

My first step, which relates to the previous step as well, was to download Cordova, which is the new term for PhoneGap. I didn’t have this on my home computer so I had to make the steps to download it properly. This involves:

1. Installing the most recent Java Dev Kit
2. Opening the NodeJS command prompt and type the following command:

*npm install -g cordova*

1. This installs Cordova onto the system.

Once everything is set up, it is a lot easier to install NFC capabilities into your app. More detailed step-by-step instructions can be found at the following Github URL:

https://github.com/chariotsolutions/phonegap-nfc

The link above is the official NFC repository reference for Adobe PhoneGap Build and includes all of the installation instructions, including how to change the config.xml file and how to use NodeJS to install the NFC plugin.

This also includes references to NFC-compatible listeners that you can use in the JavaScript in your app.

I used the following simple listener within the onDeviceReady function in the JavaScript file:

*nfc.addNdefListener (*

*function (nfcEvent) {*

*var tag = nfcEvent.tag,*

*ndefMessage = tag.ndefMessage;*

*alert(JSON.stringify(ndefMessage));*

*}*

*}*

This adds an event listener to the action of scanning an NFC tag with your phone. Whatever is on the phone, assuming it’s a text-based message, it takes that text and shares it as an alert. This shows that the NFC capabilities are working properly.

## Stumbles, Breakthroughs & Victories:

The process of creating an app, and having it compatible with NFC, was by far the most difficult process of the entire project. I searched high and low all over the Internet for any advice on how to create an app that was compatible with NFC, and I foolishly didn’t make the first step of figuring out how to create an app in the first place. All I knew from my previous experiences with PhoneGap Build was that my file stack needed to have an index.html file, but every example of an NFC-compatible app that I found referenced online didn’t have one. In fact, all of the examples I found had no familiar aspects. Of course, I realized that these were fully processed app builds (as indicated by the AndroidManifest documents in them which are usually generated by the building process) and therefore wouldn’t help me with my project. Eventually, and with some trial-and-error, I was able to find the answer through the PhoneGap documentation, which brought me to some template files and the NFC plugin Github shown above. Finally, here was some code in a format I recognized! I was able to put some things together and the first major breakthrough I had was when I loaded my app onto my phone and it registered that an NFC tag was present. From there, it was only a matter of manipulating JavaScript and HTML, which I was familiar with.

The other major ongoing stumble in this project was the process of making changes. Because my app had NFC capabilities and I didn’t have an NFC reader that I could plug into my computer, every single time I made a change I had to go through a long process of updating my work. First, I had to update the code in my Github. Then, I had to pull the latest Github update to PhoneGap Build to build my app and create the APK file to download. Third, I had to somehow transfer that APK file to my phone, which I usually did via dropbox. Finally, I had to uninstall the app from my device and reinstall it using the new file. I had to uninstall it every time because if not the app had a tendency to duplicate all of the code, and I’d end up with a strange looking app.

# Step #4 Data

Once the layout is all formatted, the proper files are in the proper places, the NFC plugin is installed properly, all of the code is uploaded to Github and everything is installed successfully as an app on a device, then it’s finally time to get into the actually data of the project.

I downloaded the app NFC Tools as my NFC writer, as my own app only reads data. I highly recommend NFC Tools to anyone playing around with NFC tags, even though it’s mostly geared towards fun, gimmicky projects than development, but it serves this purpose.

What I had to do was write down all of the data I wanted for one character in a single block of text. From there, in the NFC Tools app I went to “Create Record” and added a text/plain record and added the text of my character. I then wrote that data to the tag. You can add a “Text record” specifically, but I found that it can cause some issues when you take shortcuts, so I recommend adding a “Custom Record” instead.

Repeat this for however many characters you are making.

## Stumbles, Breakthroughs & Victories:

The biggest obstacle I found in this section was that the size of my text file was too big to hold on my NFC sticker, which had a limit of approx. 868 bytes. Luckily my data was around that size also, so I didn’t have to cut down on too much and the app still works like it should. I can imagine that with more writing capabilities and data it could become too much for the tag.

# Step #5 – Javascript

If the NFC tags are working and the layout is complete, it’s time to make everything all work together.

In my layout, I gave every div with text that I wanted affected by NFC a specific “id”. For example: *<div id="name">Name</div>*

This shows the word “Name” in the app, but I want it to say the character’s name, “Seria” instead.

Here is the JavaScript added to the NFC event listener that performs this function:

*var name = document.getElementById("name");*

*if (nfc.bytesToString(ndefMessage[0].payload).substring(0, 5) == "Seria") {*

*name.innerHTML = nfc.bytesToString(ndefMessage[0].payload).substring(0, 5);*

*}*

This scans the data found on the NFC tag, checks that the first five letters are “Seria”, and if so, takes the first 5 letters of the text and replaces the word “Name” with “Seria”. The if statement exists so that if the first letters are the name of a different character, that character’s name would appear instead.

Every block in the template has a code like the one above, picking out specific substrings from the block of text and inputting it into the exact div that it needs to be. This way, as soon as you scan a character, only the information you want is shown.

Also added: *document.getElementById("image").src = "img/fig-apa.png";*

[to change the main image depending on which character is shown]

## Stumbles, Breakthroughs & Victories:

I originally had a very convoluted way of creating this function. It looked a little like this:

*var a = “data”;*

*var name = document.getElementByID(“name”);*

*a = nfc.bytesToString(ndefMessage[0].payload).substring(0, 5);*

*name.innerHTML = a;*

I realized pretty quickly that this was a lot of redundancy and cleaned it up. Otherwise, the JavaScript aspect was relatively simple and straight forward.

# Step #6 – Figurines

This is the fun part – I spent a long day creating all of my figurines from scratch since they are all custom characters. True, you could go out and buy figurines, or you could get them 3D printed, but I created the figurines out of aluminum wire and baking clay and painted them.

Once you put the NFC stickers on to the bottom of the figurines and update the app, the project is complete!

