Lauren Wood, Ashley Gentles, Max Hasbrouck
Election Systems - Fall 2016

# Ballot Printing Documentation and Review

**Objectives going in**:
Our project is to build a tool that makes studying and verifying the printing options of the STAR system easy. What that entails is creating a tool to generate any viable print format that a ballot could reasonably fall under, given a file of candidates voted for, and a configuration file listing the desired format (font size, number of columns, etc.). With this tool the human factors team can generate a large number of ballots very quickly, conforming to a variety of standards that they wish to test. Additionally this tool could be used to sample potential ballot formats for STAR whenever a change is required.

**Step by step progress**:
Our first steps involved researching how best to build this tool, including deciding the language we would use, finding the external tools that might already be available, and figuring out how our tool was meant to be used. We started by talking to one of the eventual users of our tool, Claudia Z. Acemyan, in order to get a sense of what functionality was desired and which goals should be prioritized. Once we had the objectives, we transitioned to determining how best to build our product. We chose Python as our language, as it fit the skillset of the team, and it allowed us to rapidly develop versions of our product. Additionally, Python gave us the access to a number of libraries that seemed like they would be useful for our project. In particular, we found an open-source PDF generation tool called ReportLab that provided us a variety of options for creating our ballot PDFs.

Once we had our basic approach planned out we began by implementing a simple ballot generator that had hard-coded values for each of the races and for all of the file configurations. It used a simple default PDF template from ReportLab to generate a single page, and then added consecutive rows of race data to that page until there were no more races or the page ran out of room. Following this initial version, we worked to make the tool more flexible so that it could adapt to more races and to candidate/race names of different lengths.This included transitioning the row based approach to a column based approach, where races were consecutively added to a column so that the length of each candidate name did not decide the height of the entire row. Additionally we began supporting multiple pages, switching the ReportLab framework we were using to allow for repeated template usage with different content.

The next stage involved making the tool more practically useful, by abstracting the settings of ballot creation. This included setting up a configuration file that could control the font and column specifications of the PDF, and a file that would contain the race data that could be read in and parsed by our code. Throughout this stage we also added error reporting for a variety of problems that might occur in the code, and handled several edge cases that we discovered in the course of designing the product. This included making the number of elements in a column

dependent on the height of each column and the paper size. Our final changes enabled support of valid barcode entries, to ease the future transition to STAR.

**Current product**:
At this point in the project we have a tool that can generate a ballot PDF given any set of "selected" candidates and a config file. We can successfully gather the boundaries from the config, grab the race data from the candidate file, and then add the races to each PDF page. The config file supports selecting the size of the paper, the font and the font size, the number of columns, and the location of the file with candidate data. The PDF generator takes this info and then adjusts the number of candidates it can fit on a page based off of the font size and the number of columns desired, with excess text wrapping and excess races moving to the next page. The generator supports all valid Unicode characters.

**Important Notes**:
We modified the upc.py file in pybarcode to eliminate the line that adds a checksum to the barcode. When we used our own barcode scanner to read the barcode, it was not expecting the checksum and read it in exactly as it was created, so we eliminated that line in the code.
The information about the specific election is also hard coded in because the type of information that needs to be there is not currently defined well. You can easily take in an extra input to modify that.

**Possible future goals**:
A large possible future goal for this ballot printing project is integrating it into STAR. This would be beneficial because election officials could potentially change the format of the ballot depending on how many races there are. Another alternative option is to have the program determine what the format should be based on the number of races, determined by preset information that human factors testing would provide. If STAR is used outside of the U.S., then more paper sizes could be included.

**How to use this tool**:
1) Set permissions of the "printer" file to allow the tool to run
    a) From the command line this can be done by navigating to the folder containing the tool, and typing "chmod u+x printer"
2) Locate the file that contains the data on the races
3) Open the config file (config.cfg) and on the line below the [Races] header set "filename = <location of the races file>"
4) In the config file, set the type of paper you would like to generate a result for (the standard is Letter)
5) In the config file set the font type and font size that are desired
    a) Note that the font types are limited to the 14 basic PDF fonts
6) In the config file set the number of columns that you would like the candidate names to be displayed in
7) Save the changes you made to the config file

8) From the command line type “./printer <name of output PDF file you want made> <number for barcode>”
9) If no errors are displayed, then the races have been successfully printed to the file. If the barcode number input is too long, it will truncate it and print error message but the output will still be created, if the number is too short it will fill the beginning with 0’s.

**How the code is structured**:
Our code reads in the configuration file to determine how to format the text and columns, and then loops through candidates, adding them to appropriately sized sections that are added to the columns. When each column fills up, it moves onto the next column. When all the columns in the loop are full, it moves onto the next page and starts over. At the end of each page (or when the candidates have all been added) we add a header and a footer with information and the barcodes. The 2 barcodes on any given page are the same, and end in the page number. So each page has its own barcode based on the input and the page number.