

Honour-Roll CSV Part

*Technical Report and User Guide*

A report on the design and implementation of the Honour-Roll CSV Part developed in the BBIS environment for the purpose of retrieving data from an Excel spreadsheet/CSV file and populating the honour-roll lists on the UWTYR website.

Written by: Yael Sprikut

8/13/2015

Table of Contents

[Introduction 2](#_Toc427249952)

[Part Overview 2](#_Toc427249953)

[The Guide 4](#_Toc427249954)

[Summary 4](#_Toc427249955)

[CSV File/Excel Sheet 4](#_Toc427249956)

[BBIS Widget 6](#_Toc427249957)

[PART I: The Source Code 6](#_Toc427249958)

[PART II: Exceptions 9](#_Toc427249959)

[Alphabetized Anchor List 10](#_Toc427249960)

[Future Application and Conclusion 10](#_Toc427249961)

# Introduction

The following report is designed to serve as a technical user guide for the “Honour-Roll CSV Part”, a widget created for retrieving large amounts of data from a hosted CSV file that exists within the BBIS environment. This technical guide provides a detailed summary of the part’s functionality, describes its various features, explains how it works, and gives suggestions on best practice and implementation.

# Part Overview

The Honour-Roll CSV part was initially developed to handle large amounts of information from the Spark database. Until recently, information was manually written into parts across BBIS; for example, honour-roll donor lists consisting of hundreds of names had to be entered one-by-one into an unordered list created within the HTML editor of the part. The CSV Part has optimized this process by retrieving such information from a pre-compiled file by using an HTTP GET request to the server, eliminating the need to hardcode text. The following table provides general information about the Honour-Roll CSV part.

|  |
| --- |
| Honour-Roll CSV Part General Information |
| Part Name (in BBIS Environment): Honour Roll CSV-Retrieving Part |
| Part Type: Query Results Display |
| Part Location: Site Content/X\_Test\_Parts |
| Part Location (on page): Honour Roll CSV-Retrieving Page |
| Corresponding CSV File Location: Website Administrative Documents |

Figure 1

The part combines the existing HTML layout and applied CSS classes used in the original Honour-Roll part with JavaScript that is responsible for creating an HTTP request that gets the data from the CSV file to the webpage. Originally, the part consisted of a series of unordered lists, each corresponding to a letter of the alphabet (under which the names of the donors would be alphabetized). These unordered lists existed within a scrollable <div> that lay on the webpage. The following figures demonstrate the existing layout of the original part and a screenshot of the original code.



Figure 1

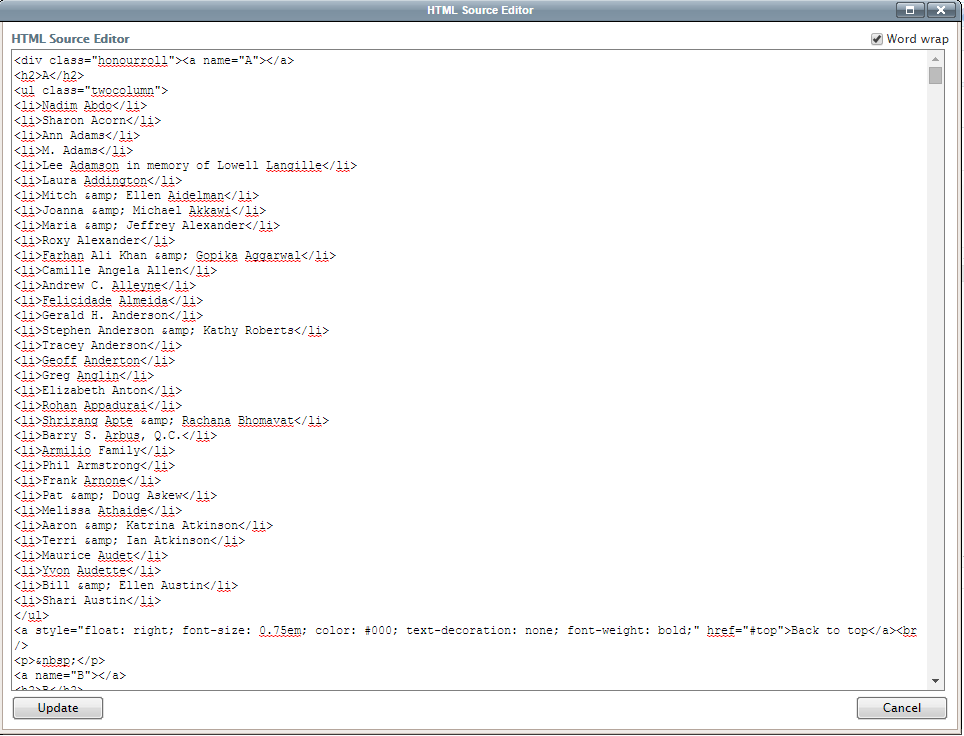


Figure 1

The preceding screenshot shows that the data is hardcoded into the part. This hardcoded list has been replaced by the following script that retrieves the above information from a file.

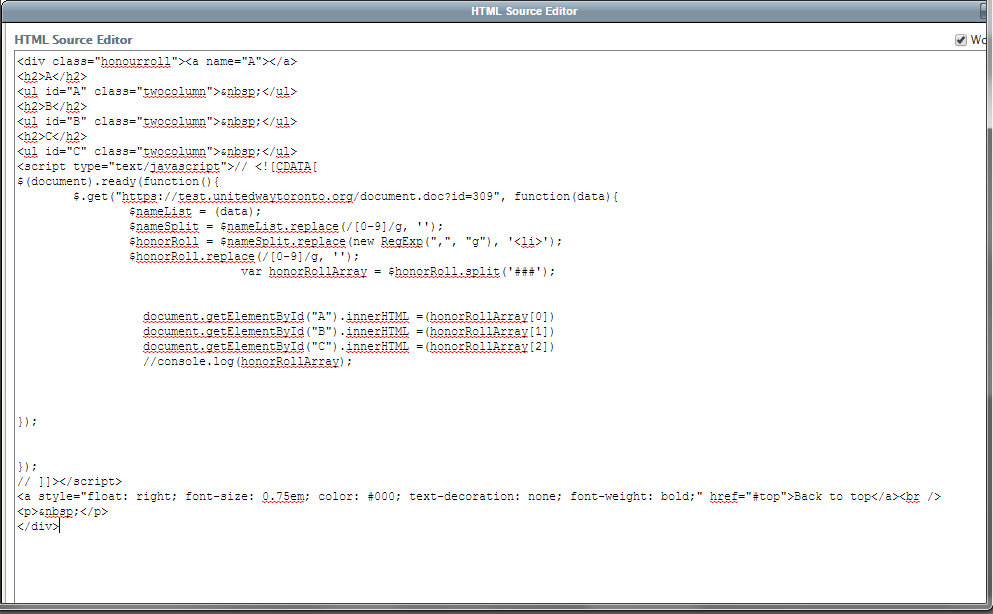


Figure 1

The next section of this document will describe the “anatomy” of the part. Using screenshots and code snippets, it will guide the user through each JavaScript line and will explain how this part can be applied to multiple pages and be used in multiple areas of the site.

# The Guide

## Summary

The Honour-Roll CSV Part consists of three major parts:

1. **CSV file** (or Excel sheet)
2. **The BBIS Widget** (the part that is placed onto the page)
3. **Alphabetized Anchor List Part** (optional)

The three parts work together to create an unordered list of virtually any information that is retrieved from the database. The CSV file provides the data or textual information for which we will be creating an HTTP request. The part will be then placed on the page and its code will be slightly modified to fit the requirements of the CSV file. The alphabetized anchor list part will be placed directly above the Honour-Roll CSV part and will be responsible for leveraging the <ul> ID’s to jump to the corresponding letter on the list. The anchor part is optional and thus will only be used when required by the user.

## CSV File/Excel Sheet

Because the CSV File will be provided by an external source, specific format instructions must be provided upon data request. The query that requests information from the database **must** include two columns: **user ID and full name**. This will generate a spreadsheet with a column of numerical ID’s and a column of corresponding names. The script in the part will look for and remove all instances of numbers in the file, and leverage the commas between the columns to separate the list (this will be discussed in more detail in the next section). Once the excel sheet is generated, the user must insert three pound signs (or hash tags) **###** between the alphabetizedlist of names along with its own ID, separating the data into alphabetical chunks (refer to figure 1 5). This will separate all the names under the “A” section from the “B” section from the “C” section, and so forth. The hash tags will essentially separate the string into alphabetized elements within an array created by the script.

The following is a screenshot of an example mock-up CSV file/Excel spreadsheet that can be used with the Honour-Roll CSV Part. The A column will contain user ID or any numerical value corresponding to each name in the list, and the B column will contain the names (or information) to be displayed on the web page itself.

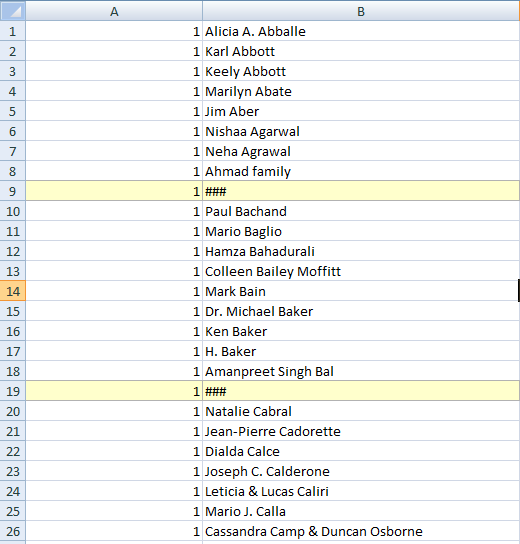


Figure 1

The sheet can be left as an Excel spreadsheet or converted into a CSV file (this can be done by saving it as CSV in Microsoft Excel). Once the file is saved, it should be uploaded to the BBIS environment **(Files > Website Administrative Documents Part).**

**Note:** When uploading the file, delete the pathname of the URL and accept the default URL link which provides a link with a unique ID. Refer to the image below:

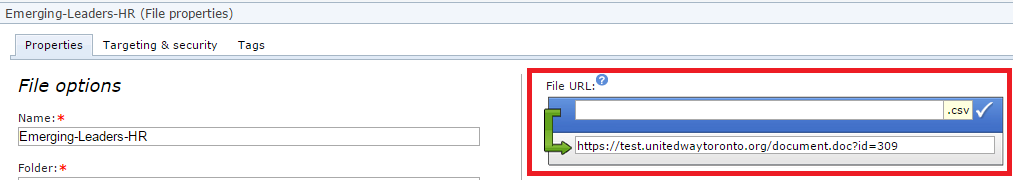


Figure 1

## BBIS Widget

The BBIS Widget relates to the actual **Honour-Roll CSV Part**. This is the “control center” of the entire part. It contains the code that controls the flow of the data from the server to the client. The part consists of HTML and JavaScript code, and is responsible for 1) getting the information from the server and 2) displaying the information to the page. This section will guide the user through each line and provide a detailed explanation for the code.

### PART I: The Source Code

The following code block provides the fundamental script used within the part.

Figure 1

#### Source Code: Line 2-8; 29

Lines 2-8;29 contain the existing HTML parts from the original Honour-Roll Part. The <div> encompasses the entire part and encases it within a scrollable “box” that contains the data. It makes a reference to the “honourroll” class in the stylesheet. Within this <div> we have the letter headings within the <h2>’s and a <ul> with a class that organizes the data into two columns, and an ID that is referenced by our JavaScript. Because each dataset corresponds to its own letter, we must create an <h2> heading and <ul> for every set of names. A heading and unordered list must be created for every letter. This HTML will be placed directly above the JavaScript for organizational purposes. Line 29 provides a closing tag for our container.

#### Source Code: Line 9-12

This section contains the initialization part of the JavaScript code that runs the part, responsible for providing functionality. As required by the BBIS HTML editor, we enclose our code within script tags before writing any JavaScript code. Within these tags, I created a function that runs once the DOM is loaded. This is a JQuery AJAX method that uses GET request (line 11) to get the information from the link of the data source (refer back to **figure 1 6** where we generated a default URL that navigates to the location of the CSV/Excel sheet).

**Note:** The **“/document.doc?id=[document number]”** URL follows the existing URL convention that exists in the file upload BBIS part.

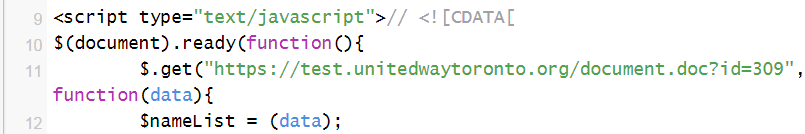


Figure 1

Please refer to the image above for reference. Within this function is another function that creates a parameter called “data” which specifies data to send to the server along with a request, which later contains the resulting data obtained by the request. I put this parameter in a variable called **$nameList**, which captures this data in a string and prepares it to be manipulated. Following line 12, the string of data we have just retrieved from the data source will go through stages of alteration and manipulation.

#### Source Code: Line 13-16

This section contains the manipulation and alteration part of the string we have just created. It will provide a line-by-line explanation of every string-manipulation stage, showing the transformation of the string.

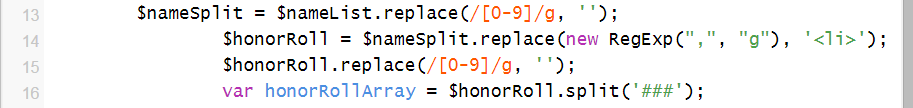


Figure 1

Please refer to the image above for reference. I created a new variable called **$nameSplit** for the purpose of removing all the associated numerical ID’s from the first column (as mentioned in the first part of this section). The .replace function searches for all instances of numbers and using the global variable (g), removes them from the string, replacing them with an empty space (thus, splitting up the data). I then took the $nameSplit variable and put it into a new variable called **$honorRoll**. Once again, we use the .replace function to replace all existing commas with an <li> tag.

**Note:** The commas are an automatically generated delimiter that manifests whenever there is more than one column in a CSV/Excel file to separate the data of the columns.

The <li> that replace the commas surround each name, positioning the data into two columns as intended by the <ul> class. On line 15, the string is altered once more, replacing the numerical ID’s with an empty space. The string is now ready to be converted into an array. Thus I created a new variable called **honorRollArray**, which splits the entire string into alphabetized chunks. I used the .split function to split the data set at every ###, which was initially placed between the alphabetized data sections. The array is now split into sections according to letter, and can be referenced by calling the specific array element.

#### Source Code: Line 18-21

This is the section that outputs the text we have just prepared.

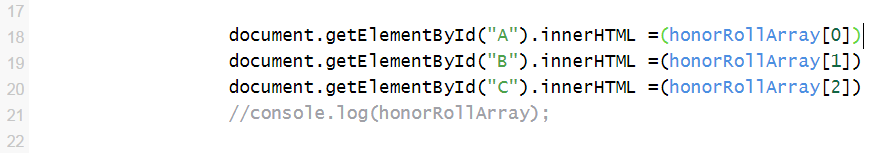


Figure 1

There are two important characteristics to each line in this section. The first part of the line gets the ID of the <ul> which it is going to populate.

**document.getElementById("A").innerHTML**

The above line would populate the <ul> that will contain all the A names. Eventually, there will be 26 lines that contain the letters A-Z. The second part of the line tells the first part with which element of the **honorRollArray** array it will populate the <ul>.

**= (honorRollArray[0])**

For example, if the data set in the CSV/Excel Sheet contains a list of names for every letter of the alphabet, there will be 25 lines, with honorRollArray[0] - honorRollArray[25]. However, this will not always be the case, since there will not always be a data set with names in every letter category. The next part of this document will discuss how to handle exceptions and different kinds of lists and data.

Line 21 is used purely for debugging purposes and, although optional, should be left in the part for future testing and debugging purposes. It outputs the array into the console log and makes sure that the script is running properly.

**Note:** It is a good practice to use console.log to test the performance of an application or to use as a breakpoint in the application to find and fix errors in JavaScript code.

#### Source Code: Line 27-28

Exiting the JavaScript code, we come back to the HTML tag that creates a small href that takes the user back to the top of the page.

### PART II: Exceptions

This section provides examples of exceptions that may occur during the implementation of this part. One example (mentioned in Part I Section III), discusses the possibility of having a dataset that does not necessarily contain data for every letter of the alphabet, which would create a mismatch between the array element number and the ID of the <ul> where the data should be placed. The following diagram defines the relationship between the array elements and the corresponding <ul>s.

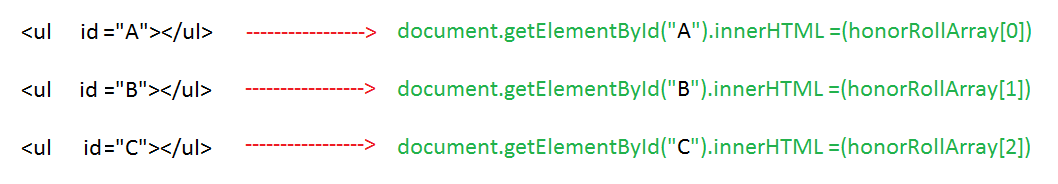


Figure 1

As we can see from the above diagram, the two elements that must always correspond to one another are the id’s of the <ul> and the element obtained by the JavaScript request. The honorRollArray then corresponds to the “address” or location of the desired data in the data set. For example, if we receive a data set for Sponsors, and there are no names that would fall under A (which means that the <ul> with id=A will not contain any data), the order will differ. The following diagram shows an example.

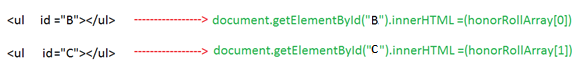


Figure 1

The second diagram shows how the relationship between the id and array element number have changed. In the first diagram, <ul id=A> corresponds to the 0’th element of the honorRollArray. Since there is no <ul id=A> in the second diagram, the 0’th element gets replaced by the first data set it finds in the data set, which in this case lists the data for all the B names. It then fills every following space in the array with the next data chunk that is separated by ###. Depending on each individual data set, the document.getElementById list will have to be adjusted accordingly to match the amount of elements within the array.

**Note:** The amount of elements in the honorRollArray will correspond to the amount of ###-separated sections in the CSV/Excel data set.

## Alphabetized Anchor List

This is the third and final part of the Honour-Roll CSV Part, designed to grab onto the <ul> ID and jump to the letter location within the honour roll list. This part was originally used with the hard-coded honour roll part and can still be applied to the new CSV part, which contains the same <ul> ID’s as the original part. This part is optional, however provides a nice feature that goes with the part.

# Future Application and Conclusion

It is certainly possible to apply the Honour Roll CSV Part to other sections of the website that requires data retrieval. As long as the same parameter concepts are being applied to the CSV/Excel sheet, the part should work. Some alterations might have to be made to fit the specific need, however the main concept (retrieval of data) has already been created, so the only changes that may have to be made are string manipulation (bringing back columns, numbers, commas, etc).