



ORLANDO

2018

Google Apps Script

Library Glue for *Create Lists*

Andy Helck
Wilkinson Public Library
Telluride Colorado
ahelck@telluridelibrary.org



Overview

- Introduce myself – Andy Helck
- Introduce the Google Apps Scripting Language
- Why you might learn it or find someone who knows it
- Present several simple projects using *Google Docs* and *Google Sheets*



Wilkinson Public Library

- We are a public library in the resort town of Telluride Colorado...
- ...part of the Marmot Consortium...
- ...using Sierra.
- We use Google Apps for email and collaboration
- I'm Andy Helck, I've been learning Sierra DNA, Sierra REST API, C# programming and most recently, Google Apps Scripting language



The Good Old Days

- I wanted a language that was already associated with a document editor of some kind. A place where I could paste text into and out of easily, and run a script to reformat the content.
- When I started doing projects like this 10 years ago, I looked to Microsoft because everybody was using *Word* and *Excel*.
- Both of these programs have a venerable scripting language called *VBA* or *Visual Basic for Applications*.
- *VBA* allows you to add buttons and menus to the user interface and do your own custom processing. But...
- ... *VBA* is antiquated and quite dreadful to use!



The Times They Are A Changin'

- To be fair, *VBA* showed the way and programmers have written many useful scripts with it. But not only is the language archaic, just finding your way around the *object model* of *Word* and *Excel* is painful.
- When Google came out with their simplified word processor and spreadsheet products, *Google Docs* and *Google Sheets* (2012), they too saw the need for a scripting language.
- Google engineers chose an existing language that was already hugely popular in the web development world. It is called *JavaScript*
- If you know a bit of *JavaScript* then you already know *Google Apps Script*. And if you start by learning *Google Apps Script (GAS)* then you are also learning *JavaScript* which is a skill you can use for a long time to come.

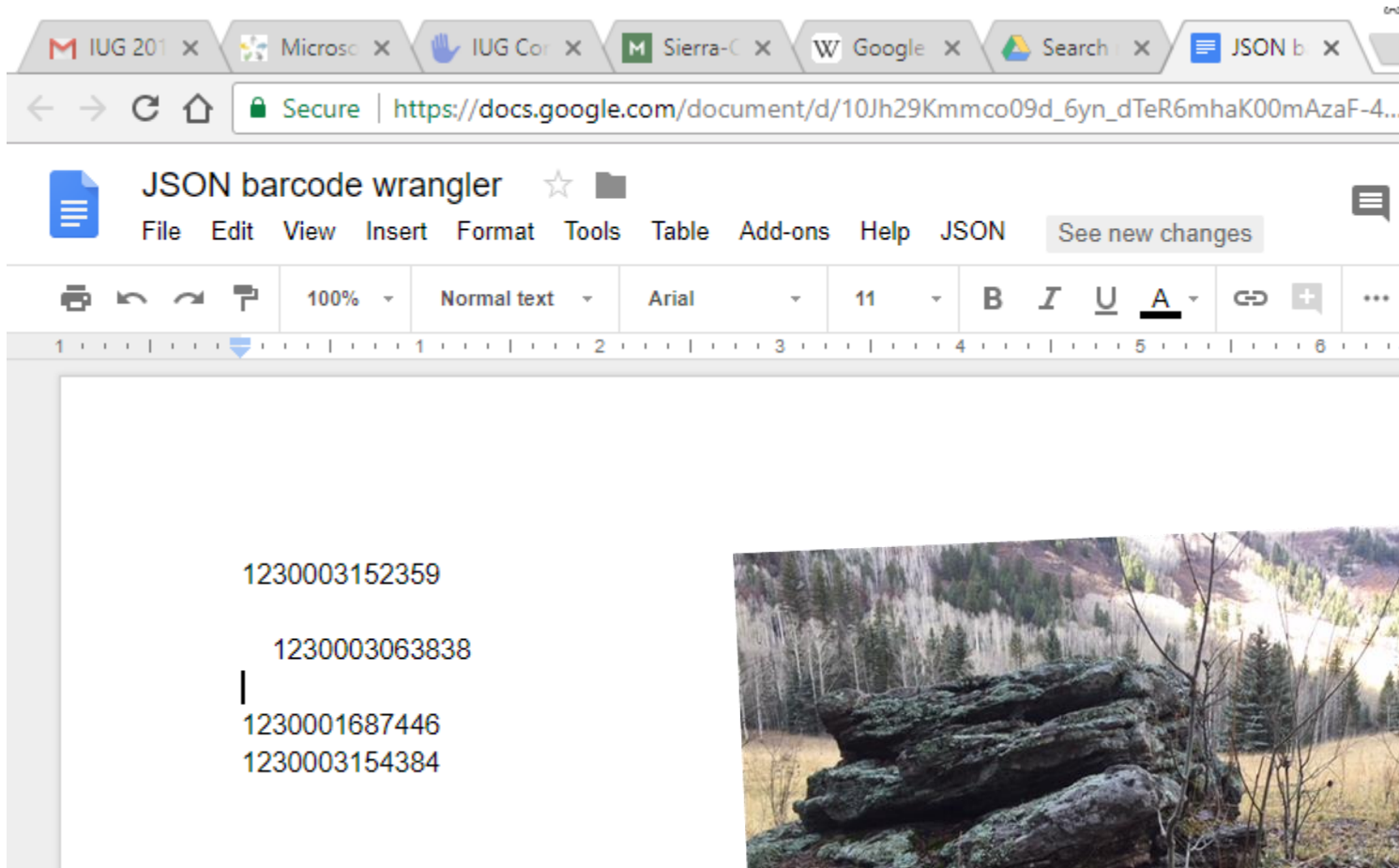


First Project – Create Lists

- The *JSON* feature in *Create Lists* is a powerful tool for pulling data from *Sierra*. Finally there is a way to enter a list of barcodes or control numbers and pull matching records!
- I want to scan item barcodes into a *Google Doc*. There should be a menu or button to allow me to ‘process’ that list.
- The contents of the document will then be re-formatted just the way *Create Lists* wants them.
- Copy and paste the newly formatted text out of *Google* and into *Sierra Create Lists* search box and proceed to make my *Review File*.



Meet *JSON Barcode Wrangler*

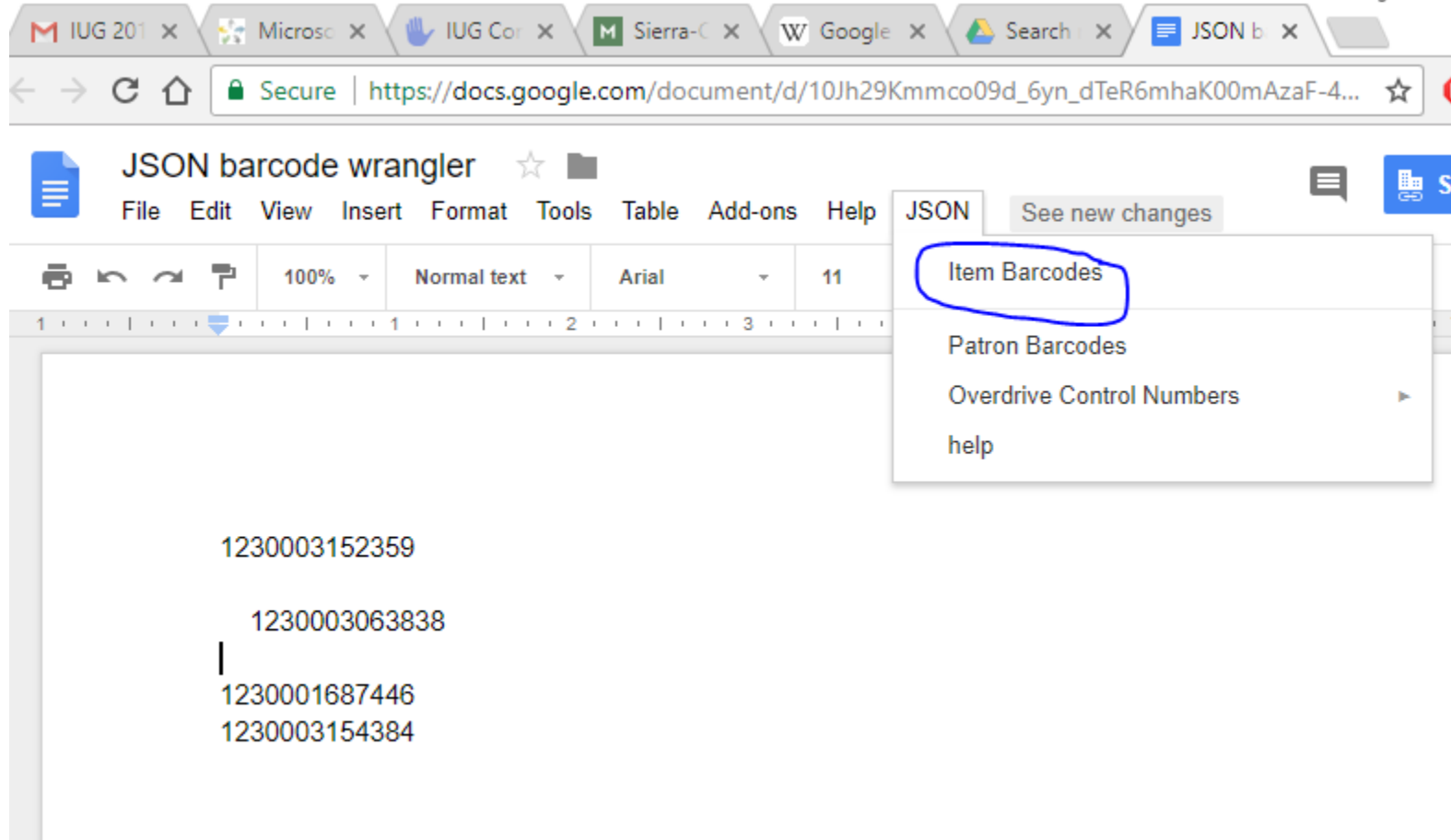


This is Chrome running on my PC. I have the *Google Document* called *JSON Barcode Wrangler* open. I have pasted in a very short list of item barcodes.

Note the main menu has an entry on the right called *JSON*. This is what my programming project added.



Ready to Wrangle!



The screenshot shows a Google Docs interface with a document titled "JSON barcode wrangler". The "JSON" menu is open, showing options: "Item Barcodes" (circled in blue), "Patron Barcodes", "Overdrive Control Numbers", and "help". The document content contains the following text:

```
1230003152359  
1230003063838  
|  
1230001687446  
1230003154384
```

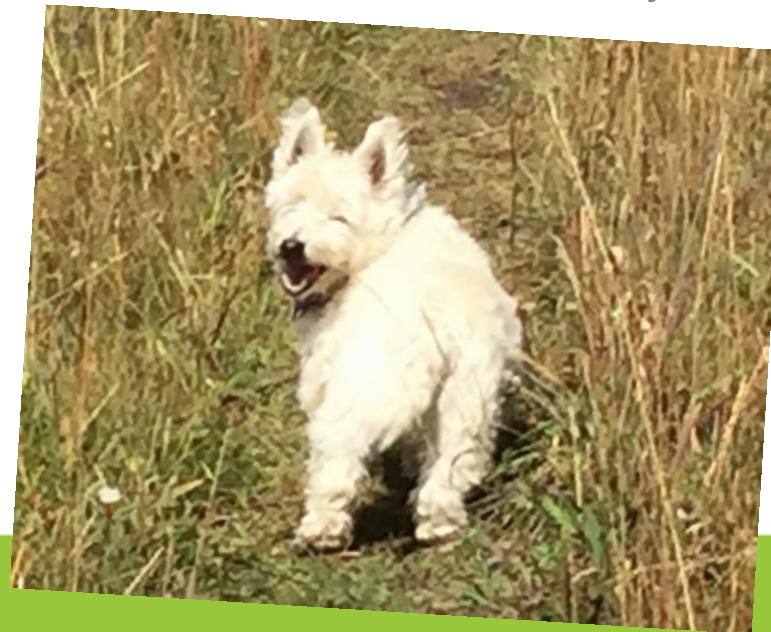
We can create nice looking menus with *Google Apps Script* that do cool stuff!



Yuck! What is this?

```
{
  "queries": [
    {
      "target": {
        "record": {
          "type": "item"
        },
        "field": {
          "tag": "b"
        }
      },
      "expr": [
        {
          "op": "in",
          "operands": [
            "1230003152359",
            "1230003063838",
            "1230001687446",
            "1230003154384"
          ]
        }
      ]
    }
  ]
}
```

- Haven't tried the JSON option in *Create Lists* yet?
- Innovative programmers added a very powerful way to specify search criteria when using *Create Lists*.
- Any criteria you enter the traditional way can be viewed in JSON format – then copied and saved into a text document for the next time you need to run the same list.

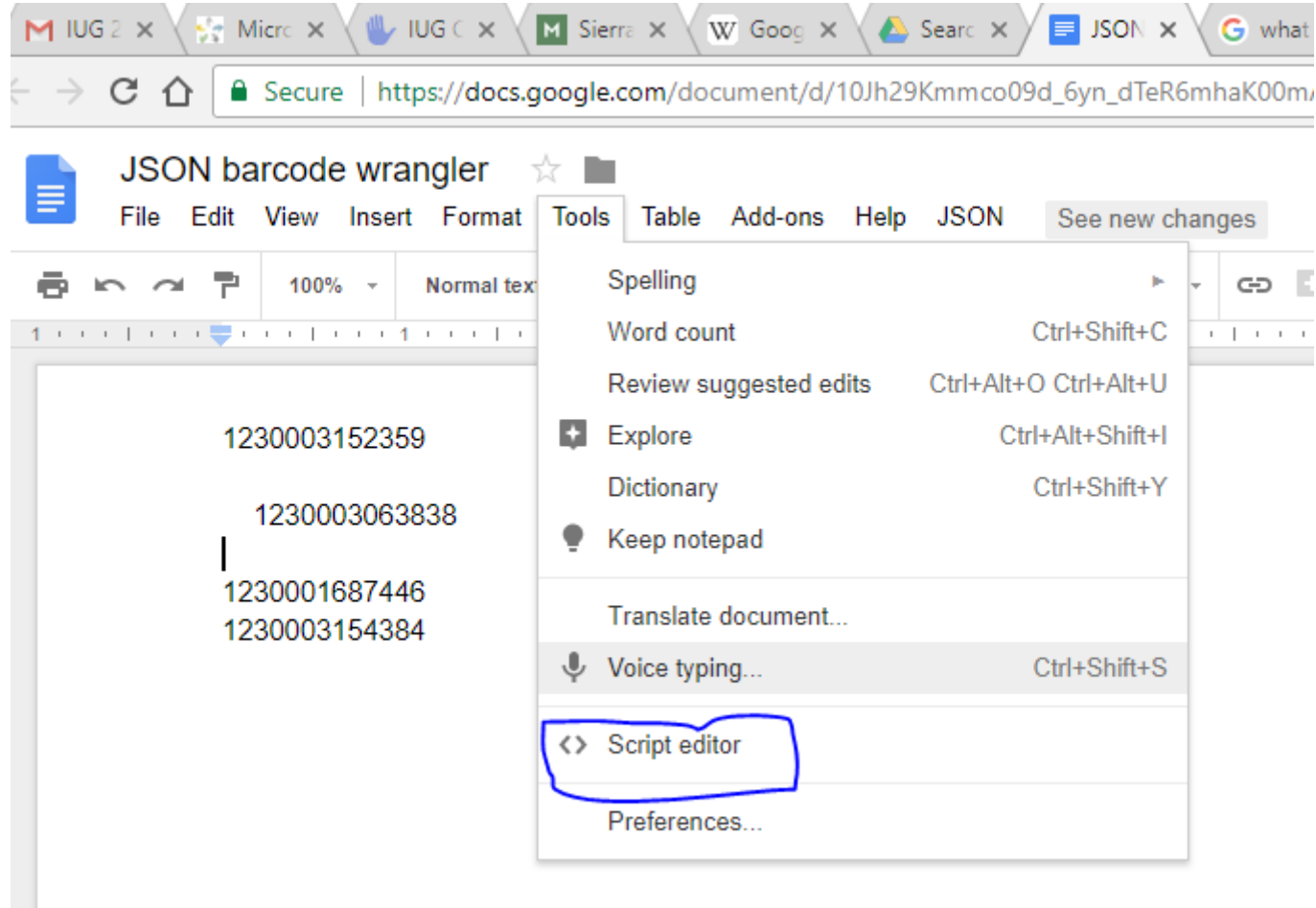


JavaScript Object Notation – Skip this section if you want!

- Programmers and web developers have been grouping and structuring data for a long time. The *JavaScript* language is particularly good at doing this. You can easily create custom data structures that exactly match your needs – cool combinations of arrays, lists and records.
- These data structures and the data they hold can be converted to plain text using lots of punctuation, namely square brackets, curly braces, commas and lots of quote signs. The indentation is a visual aid to us humans, the computer won't care if its all on one line.
- The example on the previous slide completely describes to *Sierra* the search criteria needed to find the 4 records matching the item barcodes.



Let's write some code!



- The beauty of *Google Apps Script* is that it's free and you already have it installed.



Welcome to the Script Editor

```
1  /**
2  * Sierra 2.2 Barcode Create List Helper Tool
3  *
4  * License: MIT
5  *
6  * Copyright 2016 Andy Helck
7  *
8  */
9
10
11
12 function onOpen(e) {
13   DocumentApp.getUi()
14     .createMenu('JSON')
15     .addItem('Item Barcodes', 'itemBarcodes')
16     .addSeparator()
17     .addItem('Patron Barcodes', 'patronBarcodes')
18     .addSubMenu(DocumentApp.getUi().createMenu('Overdrive Control Numbers')
19       .addItem('037|a -- IN', 'bibField037a_IN')
20       .addItem('037|a -- OR', 'bibField037a_OR'))
21     .addItem('help', 'help')
22     .addToUi();
23 }
24
25
26 function itemBarcodes() {extractBarcodes("item");}
27 function patronBarcodes() {extractBarcodes("patron");}
28
```

- The editor opens in a new tab.
- Unlike *Docs* itself, you must actually save changes before they take effect.



A Simple JavaScript function

```
wrangler.gs x json.gs x help.gs x
58
59
60 function extractBarcodes(recordType) {
61
62     // https://developers.google.com/apps-script/reference/document/text
63     var doc = DocumentApp.getActiveDocument();
64     var body = doc.getBody();
65     var text = body.editAsText();
66     var raw = text.getText();
67
68     var barcodeArray = raw.match(/[0-9]{13}/g); // all barcodes returned!
69
70     if (barcodeArray == null) {msgBox("No barcodes found"); return;}
71
72     var dedupedArray = barcodeArray.filter(function(item, pos) {
73         return barcodeArray.indexOf(item) == pos;
74     })
75
76     jsonBarcodeQuery.queries[0].target.record.type = recordType;
77     jsonBarcodeQuery.queries[0].expr[0].operands = dedupedArray;
78
79     text.setText(JSON.stringify(jsonBarcodeQuery, null, 3));
80
81     msgBox(dedupedArray.length + " " + recordType + " barcodes found and formatted for Sierra JSON search");
82 }
83
84
```

- Top 4 lines navigate the *object model*.
- A regular expression will extract all the 13 digit numbers
- A query already exists, it just needs the data inserted into it.
- Original document contents replaced with the JSON query and its embedded list of barcodes.



JavaScript vs Shakespeare

```
function msgBox(msg)
{
    var ui = DocumentApp.getUi();
    ui.alert("JSON barcode wrangler", msg, ui.ButtonSet.OK);
}
```

“**Romeo**: Courage, man; the hurt cannot be much.
Mercutio: No, 'tis not so deep as a well, nor so wide as a church-door; but 'tis enough, 'twill serve. Ask for me tomorrow, and you shall find me a grave man.”

William Shakespeare
Romeo & Juliet

words
names
punctuation



The query object in JavaScript

```
28
29 var jsonBarcodeQuery =
30   {
31     "queries": [
32       {
33         "target": {
34           "record": {
35             "type": "item" // or 'patron' will get overwritten
36           },
37           "field": {
38             "tag": "b"
39           }
40         },
41         "expr": [
42           {
43             "op": "in",
44             "operands": [
45               "0538502117517", // example only, will be overwritten
46               "0538513096387",
47               "0538513096395",
48               "0538505744416",
49               "0538512175315"
50             ]
51           }
52         ]
53       }
54     ]
55   };
56
57
```

- This is the JSON query. *JavaScript* loves these things!
- Sierra's new JSON is something I should probably learn in depth, but I was in a hurry and just treated the whole thing as boilerplate once I found where my list of barcodes was supposed to go.
- Two of the data elements need to get updated each time the script is run
- `queries[0].target.record.type`
- `queries[0].expr.operands`



Lets Try It!

Tools Add-ons Help JSON [LAST EDIT WAS 6 DAYS AGO](#)

Arial 11 B I U A

1 2 3 4 5 6

1230003152359

1230003063838

1230001687446
1230003154384

JSON barcode wrangler

4 item barcodes found and formatted for Sierra JSON search

Ok

- Run the script from the drop down menu
- It tells me how many barcodes it actually found
- I then do a count of the cart I am processing and check the exact count.



End result...

- Here it is ready to copy and paste into Sierra!

```
{
  "queries": [
    {
      "target": {
        "record": {
          "type": "item"
        },
        "field": {
          "tag": "b"
        }
      },
      "expr": [
        {
          "op": "in",
          "operands": [
            "1230003152359",
            "1230003063838",
            "1230001687446",
            "1230003154384"
          ]
        }
      ]
    }
  ]
}
```



Another Example using *Google Sheets*

- Of course you can write scripts in *Sheets* and link them to the menus...
- ...but *Sheets* has a cool feature not found in *Docs*...
- A User Defined Function is used in the spreadsheet in place of the stock functions like =SUM() or =AVERAGE(). You write your own functions and then enter them in the spreadsheet cells like a standard function.
- Particularly cool are User Defined Functions (UDFs) that return an entire block of values. These are called *Array Functions*.
- A single function call can refer to an entire grid of cells, and then produce its own grid of cells. A single function call can replace hundreds of individual calls



Lets write an Array style UDF!

- This function won't refer to existing data, it will populate a sheet with data from Sierra
- We will use III's new web APIs to fetch data using http calls
- Lets get some bib records!



Here is a *Google Sheets* for our project

The screenshot shows a Google Sheets spreadsheet titled "Orlando Bibs". The spreadsheet has a menu bar (File, Edit, View, Insert, Format, Data, Tools, Add-ons, Help) and a toolbar with various icons. The active cell is A2, containing the formula `=ANDYS_BIB_FUNCTION(isbn_numbers)`. The spreadsheet contains a table with columns for ISBN, Record Number, Title, and Author. A yellow callout box explains that the blue ISBN numbers in column A are part of a named range referenced in the formula.

	A	B	C	D	E
1	ISBN				
2	9781250165343		The blue ISBN numbers to the left are part of a <i>named range</i> that is referenced in the upper left green cell below. That cell contains a custom formula that retrieves familiar data from the Marmot Consortium using the Sierra REST API technology		
3	9780399592065				
4	9781101884003				
5	9781682681572				
6	9781101985625				
7	9781476717807				
8					
9			Record Number	Title	Author
10			5722084	The shape of water	Toro, Guillermo del, 1964- author.
11			5743949	Everything happens for a reason : and other lies I've loved	Bowler, Kate, author.
12			5460812	Fall from grace : a novel	Steel, Danielle, author.
13			5725820	Catnip : a love story	Korda, Michael, 1933- author, illustrator.
14			5737541	Closer than you know : a novel	Parks, Brad, 1974- author.

Named ranges

- + Add a range
- isbn_numbers**
'ISBN numbers'!A2:A7



And the 'code behind'

Orlando Code

ahelck@tellurideli

File Edit View Run Publish Resources Help

```
Code.gs x
18
19 function ANDYS_BIB_FUNCTION(inputCells) {
20
21   if (inputCells.constructor !== Array) throw exception("Expecting Array"); // make sure the user is passing us a range of cells
22
23   var row, nRows = inputCells.length;
24   var isbn = [];
25   var outputCells = [{"Record Number", "Title", "Author"}]; // initialize the array we are going to return
26   var apiResponse;
27
28   var token = getToken(); // authenticate to the Marmot server in Grand Junction, Colorado
29
30   for (row = 0; row < nRows; row++)
31   {
32     isbn = inputCells[row][0]; // fetch the next ISBN number from the grid of cells specified by the user
33     apiResponse = getBib(token, isbn); // query the Marmot Consortium's database for Title and Author
34     outputCells.push(apiResponse) // build up a new array with that information
35   }
36   return outputCells; // Bib data will be displayed now in the spreadsheet!
37
38 }
39
```



Here is how to use the Sierra RESTful APIs

```
40
41 function getBib(token, isbn) {
42     var marmotServerUrlBibs = "https://sierra.marmot.org:443/iii/sierra-api/v5/bibs/search?limit=1&offset=0&fields=title%2Cauthor&index=isbn&text="
43
44     var headers = {
45         "Authorization" : "Bearer " + token,
46         "Content-Type" : "application/x-www-form-urlencoded"
47     };
48
49     var params = {
50         "method" : "GET",
51         "headers" : headers,
52     };
53
54     var url = marmotServerUrlBibs + isbn;
55     var response = UrlFetchApp.fetch(url, params);
56     var headers = response.getAllHeaders();
57     var body = JSON.parse(response.getContentText());
58     var arrayResult = [];
59     try {
60         arrayResult.push(".b" + body.entries[0].bib.id + "x");
61         arrayResult.push(body.entries[0].bib.title);
62         arrayResult.push(body.entries[0].bib.author);
63     }
64     catch(x) {
65         arrayResult = ["isbn not found"];
66     }
67     return arrayResult;
68 }
```



Authenticating to the Sierra Database

```
76
77 function getToken() {
78
79     var key = "<randomn string of digits from your Administrator>";
80     var secret = "your_P&ssw0rd";
81     var marmotServerGetTokenUrl = "https://sierra.marmot.org/iii/sierra-api/v5/token";
82
83
84     var headers = {
85         "Authorization" : "Basic " + Utilities.base64Encode(key + ":" + secret),
86         "Content-Type" : "application/x-www-form-urlencoded"
87     };
88
89     var payload = {
90         "grant_type" : "client_credentials"
91     };
92
93     var params = {
94         "method" : "POST",
95         "headers" : headers,
96         "payload" : "grant_type=client_credentials",
97         "muteHttpExceptions" : true
98     };
99
100
101     var response = UrlFetchApp.fetch(marmotServerGetTokenUrl, params);
102     var token = JSON.parse(response.getContentText())["access_token"];
103     return token;
104
```



In Conclusion...

- *Google Docs* and *Sheets* can be automated using *JavaScript*
- JSON is a specification for building arbitrarily complex data structures
- **Innovative Interfaces** makes extensive use of *JSON* data to communicate complicated things like a query specification for *Create Lists*
- *Google Apps Script* adores *JSON* formatted data
- My first example showed how to put raw data into *JSON* format and paste the criteria into *Sierra Create Lists*
- The second example showed how to get data from **Innovative's** new *RESTful API* services in *JSON* format, and populate a spreadsheet with it.

