Unlocking Sierra's REST APIs

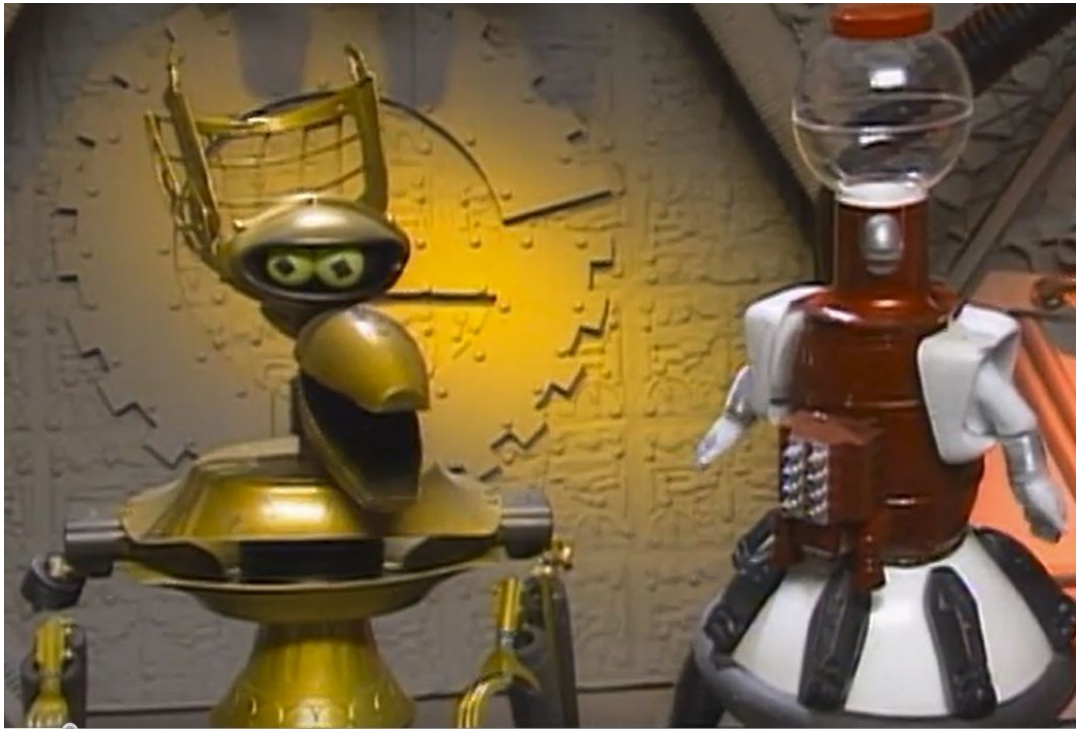
Practical Python Projects for Libraries

https://chimpy.me/iug2025

Jeremy Goldstein - Minuteman Library Network

Ray Voelker - Cincinnati & Hamilton County Public Library

Meet Jeremy and Ray



Interacting with / Consuming Sierra Data

Sierra Data Feature ⟶ **Criteria ↓**	**Sierra SQL**	**Sierra REST APIs**	**Sierra Desktop Application (SDA) / Create Lists**
Access Method	Direct database access using a PostgreSQL compatible client (typically via TCP/IP port 1032). Username and password required for authentication.	HTTP-based access through predefined API endpoints; requires no additional network configuration (typically via TCP/IP port 443 – the same port used by the WebPAC). Access Token required.	Access via the Sierra Desktop Application (SDA) interface, installed locally on the user's workstation. Account credentials needed.
Permissions	Full read-only access to all SQL views – no options to limit or prevent access to table views for clients with a valid login.	Fine-grained permissions for Create, Read, Update, and Delete (CRUD) operations. API keys can be created with limited roles (Bibs Read, Items Write, etc).	Permissions are based on user roles and workflows defined within the SDA interface.

Interacting with / Consuming Sierra Data (Cont'd)

Sierra Data Feature ⟶ **Criteria ↓**	**Sierra SQL**	**Sierra REST APIs**	**Sierra SDA / Create Lists**
Available Data	Current snapshot of all record types. Transaction data per local policy. System config and user information.	Current snapshot of all record types-some only accessible via pagination. Current checkout/hold/fine transactions	Current snapshot of all record types-indirect current transaction data when associated with a record
Query Method	SQL – Good ol' SQL	JSON (Query endpoint not available for all data points) and HTTP query parameters.	Boolean and JSON.

Interacting with / Consuming Sierra Data (Cont'd)

Sierra Data Feature ⟶ **Criteria ↓**	**Sierra SQL**	**Sierra REST APIs**	**Sierra Desktop Application (SDA) / Create Lists**
Learning Curve / Audience	Steep, but manageable learning curve for non-technical users.	Moderate learning curve for developers.	Minimal learning curve for library staff.
Flexibility / Primary Strengths	Highly flexible for reading data and creating custom reports–custom queries, complex joins, aggregations, and data manipulation tailored to specific needs and data-analysis related tasks.	Highly flexible for defined CRUD operations–provides a limited set of endpoints that are well-suited to operations involving record creation, updates and deletion (e.g. less access to data compared to SQL and the SDA)	Limited flexibility–offers no options for integration and automation tasks (with the exception of scheduler). Suited to tasks involving smaller, more manageable datasets.

Getting started: Swagger UI

sierra API

client_key | client_secret

acquisitions : The Acquisitions API — Show/Hide | List Operations | Expand Operations | Raw

agencies : The Agency API — Show/Hide | List Operations | Expand Operations | Raw

authorities : The Authority API — Show/Hide | List Operations | Expand Operations | Raw

bibs : The Bib API — Show/Hide | List Operations | Expand Operations | Raw

POST	/v6/bibs/	Create a Bib record
GET	/v6/bibs/	Get a list of bibs
DELETE	/v6/bibs/marc	Delete expired MARC data files
GET	/v6/bibs/marc	Generate a binary MARC data file of bibs
GET	/v6/bibs/metadata	Get a list of metadata
POST	/v6/bibs/query	Filter the records by a query in JSON format
GET	/v6/bibs/search	Find bib information using AWS search by author, title, or keyword
DELETE	/v6/bibs/{id}	Delete a bib by record ID
GET	/v6/bibs/{id}	Get a bib by record ID
PUT	/v6/bibs/{id}	Update a bib record
GET	/v6/bibs/{id}/marc	Get the MARC data for a single bib record

Swagger UI (Cont'd)

The **screenshot** on the right displays the Swagger UI and the **Response Body** for the **GET Request** on the `items/{id}` endpoint.

Notes on Parameters:

Path Parameters: Specify resources within the API – **part of the URL path**: `items/3000028`

Query Parameters: Provide additional details or options for the request – **part of the URL after** `?`: `items/3000028?fields=default`

`Body Parameters`: The body param isn't used for this endpoint, but it's often used in POST and PUT operations.

Parameters

Parameter	Value	Description	Parameter Type	Data Type
id	**3000028**	**the item record ID**	path	string
fields	default	a comma-delimited list of fields to retrieve	query	array

Try it out! Hide Response

Request URL

https://classic.cincinnatilibrary.org:443/iii/sierra-api/v6/items/3000028?fields=default

Response Body


```
{
  "id": "3000028",
  "updatedDate": "2019-02-19T18:11:12Z",
  "createdDate": "2012-06-30T00:20:56Z",
  "deleted": false,
  "bibIds": [
    "1000001"
  ],
  "location": {
    "code": "1lj",
    "name": "Main Children's Library Stacks"
  },
  "status": {
    "code": "-",
    "display": "CHECK SHELVES"
  },
  "volumes": [],
  "barcode": "A0000000348011",
  "callNumber": "001.944 G232, 1991"
}
```


Response Code

200

Swagger UI (Cont'd)

A **POST Request** on the `items/query` endpoint is shown on the right as it appears in the Swagger UI.

The field labeled "`json`" represents the `body param` of the POST request. This is often referred to as the **payload**, or simply **JSON**, in many HTTP clients.

Parameters

Parameter	Value	Description	Parameter Type	Data Type
offset	0	**the beginning record (zero-indexed) of the result set returned**	query	integer
limit	1000	**the maximum number of results**	query	integer
json	{ "target": { "record": { "type": "item" }, "id": 83 }, "expr": { "op": "greater_than_or_equal", "operands": ["02-26-2025", ""] } }	**a query in JSON format (see the Sierra API reference documentation for more information)**	body	string

Parameter content type: application/json

Try it out! Hide Response

Request URL

https://library.minlib.net:443/iii/sierra-api/v6/items/query?offset=0&limit=1000

Response Body

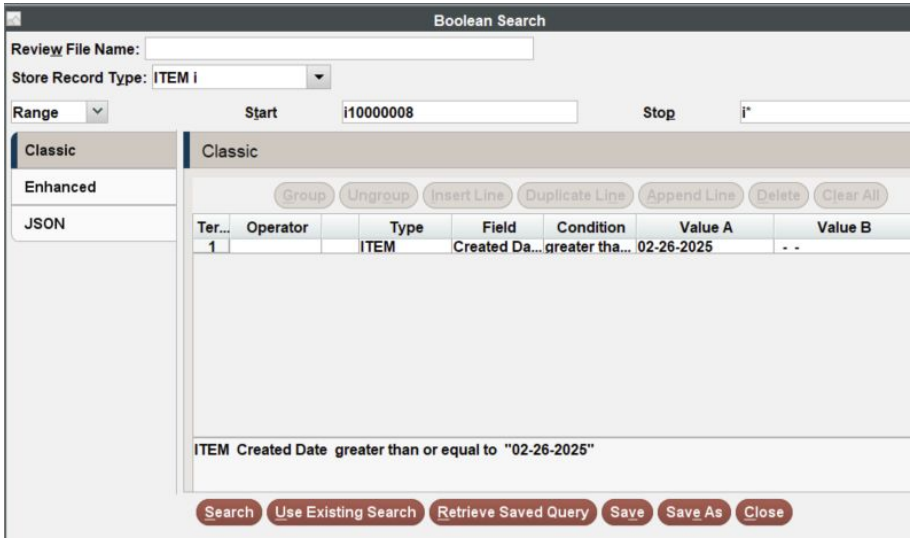

```
{
  "total": 1000,
  "start": 0,
  "entries": [
    {
      "link": "https://library.minlib.net/iii/sierra-api/v6/items/19720461"
    },
    {
      "link": "https://library.minlib.net/iii/sierra-api/v6/items/19720462"
    },
    {
      "link": "https://library.minlib.net/iii/sierra-api/v6/items/19720463"
    },
    {
      "link": "https://library.minlib.net/iii/sierra-api/v6/items/19720464"
    },
```

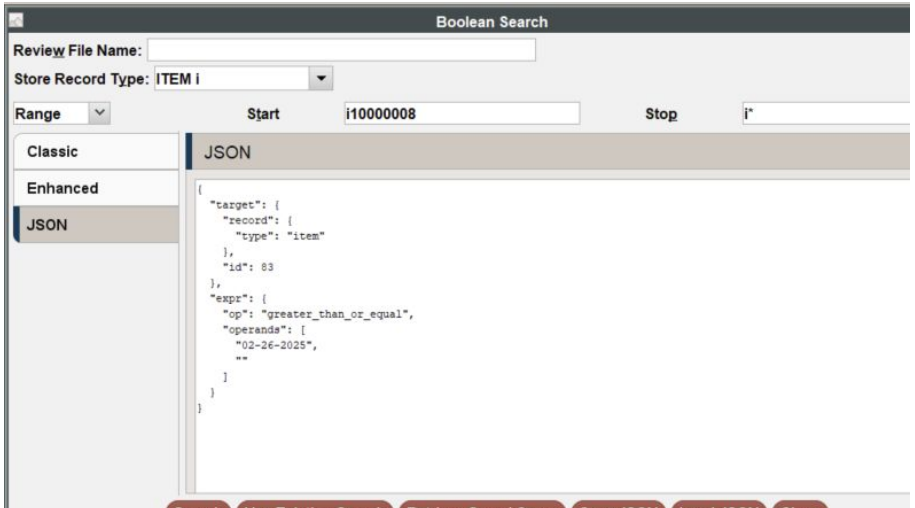
Swagger UI (Cont'd)

A **POST Request** on the `items/query` endpoint is shown on the right as it appears in the Swagger UI.

The field labeled "`json`" represents the `body param` of the POST request. This is often referred to as the **payload**, or simply **JSON**, in many HTTP clients.

For this `items/query` endpoint, the **JSON** may be retrieved from the **Sierra Create Lists Function**. By viewing the **Boolean Search** feature of a list, the **tab labeled "JSON" will provide the payload** required by this endpoint.





Swagger UI (Cont'd)

A **Delete Request** on the `items/checkouts{barcode]` endpoint is shown on the right as it appears in the Swagger UI.

Barcode is included as a required path parameter

username and **statgroup** are optional query parameters

Use delete endpoints with caution as they lack confirmations and submitting a valid request will delete the entry immediately

DELETE /v6/items/checkouts/{barcode} Check-in an item (returns to the library)

Response Class

Model | Model Schema

ErrorCode {
code (integer): the API error code,
specificCode (integer): an error code returned by an external service,
httpStatus (integer): the HTTP status code,
name (string): the error name,
description (string, *optional*): the error description
}

Response Content Type application/json

Parameters

Parameter	Value	Description	Parameter Type	Data Type
barcode	55555555555555	**the Item barcode**	path	string
username	actcirc	username to check the item in with	query	string
statgroup	111	statistic group	query	integer

Try it out! Hide Response

Request URL

https://library.minlib.net:443/iii/sierra-api/v6/items/checkouts/55555555555555?username=actcirc&statgroup=111

Swagger UI (Cont'd)

A **PUT Request** on the `patrons/holds/{holdId}` endpoint is shown on the right as it appears in the Swagger UI.

holdID is included as a required path parameter

The body represents a **patch object**, containing information that will overwrite the data in the existing record if the API is successful

In this instance the hold cannot be frozen, thus a **Response Code** of 403 is returned along with an error message in the **Response Body**

Parameters

Parameter	Value	Description	Parameter Type	Data Type
Accept-Language		the Accept-Language header	header	string
holdId	**58538307**	**the hold record ID**	path	integer
body	`{ "pickupLocation": "fp2z", "freeze": true }` Parameter content type: application/json	**the details of the hold**	body	Model | Model Schema **PatronHoldPut {** **pickupLocation** (string, *optional*): the hold's pickup location code, **freeze** (boolean, *optional*): whether the hold should be frozen (suspended) **}**

Try it out! Hide Response

Request URL

```
https://library.minlib.net:443/iii/sierra-api/v6/patrons/holds/58538307
```

Response Body

```
{
  "code": 134,
  "specificCode": 0,
  "httpStatus": 403,
  "name": "WebPAC Error",
  "description": "This hold can not be frozen."
}
```

Response Code

```
403
```

sierra-ils-utils

github.com/chimpy-me/sierra-ils-utils

A Python library designed to simplify working with the Sierra REST APIs.

Automates:

- Authentication
- Token Refreshes
- Failed Requests/Retries (esp. transient network issues)
- Logging

The screenshot to the right demonstrates installing the library, configuring and using the client to make a request in Google Colab.

*Further examples will use this library.

```
!pip install sierra-ils-utils --quiet

# Secrets can be securely stored within Colab
from google.colab import userdata

# Configure the client
from sierra_ils_utils import SierraAPI
client = SierraAPI(
    base_url=userdata.get('sierra_api_base_url'),
    client_id=userdata.get('sierra_api_key'),
    client_secret=userdata.get('sierra_api_secret'),
)

# make a request
response = client.request('GET', 'info/token')
response.raise_for_status()
```

```
<Response [200 200]>
```

Anatomy of a Sierra REST API Request (Python Example)

```python
# A sample HTTP GET request on the `bibs/` endpoint ...
response = client.request(
  'GET',   # The HTTP verb, or the operation of the request
  'bibs/', # The endpoint itself
  # params are the "query parameters" of the HTTP request ...
  params={
    'fields': 'default,marc',  # fields selected to appear in the response
    'id': '[1014568,]',        # id and dates can have an open-ended range
    'updatedDate': f"[2020-01-01T00:00:00Z,{timestamp_now}]",  # date range*
    'limit': 2000,  # 2000 is max limit offered by the API (50 is default)
  }
)
```

Anatomy of a Sierra REST API Request (Cont'd)

The **response** to the HTTP request will return an **HTTP Status Code** of **200 – OK**, which follows the **HTTP standard**.

The HTTP response body contains **JSON** text. A portion of this text (syntax-highlighted and pretty-printed) is shown on the right. It is the serialized representation of the **response objects**.

These response objects are defined as "**Resources**" in the HTTP API specification. In this case, the top-level resource is **`BibResultSet`**.

Within the `entries` key, the response includes an array of **`Bib`** resources, limited to the fields specified in the request's query parameters.


```
1  {
2      "total": 2000,
3      "start": 0,
4      "entries": [
5          {
6              "id": "1000001",
7              "updatedDate": "2021-10-07T13:52:27Z",
8              "createdDate": "2012-06-19T22:48:06Z",
9              "deleted": false,
10             "suppressed": false,
11             "isbn": "0899080871",
12             "lang": {
13                 "code": "eng",
14                 "name": "English"
15             },
16             "title": "Water monsters : opposing viewpoints"
17             "author": "Garinger, Alan, 1932-",
18             "marc": {
19                 "leader": "00000cam  2200000 a 4500",
20                 "fields": [
21                     {
22                         "100": {
23                             "subfields": [
24                                 {
25                                     "a": "Garinger, Alan,"
26                                 },
```

Anatomy of a Sierra REST API Request (Cont'd)

Notes About **Dates/Range Syntax*** in Query Params (techdocs.iii.com API Docs)

"Dates must match the date format of the property to be retrieved. In most cases, the **format is ISO 8601 combined date and time in UTC with Z (zero) offset**. Some date properties, such as catalogDate and deletedDate, are date only, with no time. Refer to the bib object and item object descriptions and examples for more information."

```
2013-12-10T20:30:00Z   # exact
```

"**Range syntax applies to dates and record IDs**…start and end values are **inclusive**."

```
[2024-01-01T00:00:00Z,2024-12-31T23:59:59Z]   # inclusive dates
[2024-01-01T00:00:00Z,]                       # open-ended date
[,1000054]                                    # open-ended id
```

*See chimpy.me/iug-2025 for info on the SierraDateTime feature

Anatomy of a Sierra REST API Request (Cont'd)

Notes About **Pagination** Query Params (techdocs.iii.com API Docs)

> "...queries that include the **offset parameter** return results much more slowly than those without the parameter. The preferred, **more efficient method of harvesting data is to use open ID ranges**"

Paginating by id:

1. In the first request, start with `id=0`
2. Extract the last record id from the result set
3. Add 1 to the last record id and set it to the next start id.
4. Loop until the system returns a page with fewer than the `limit` query parameter, or HTTP status code of `404` – record not found. E.g.:

```
limit=2000  # result set length of less than this means no more records.
```

Anatomy of a Sierra REST API Request (Cont'd)

The **Python code** on the right demonstrates the implementation of **pagination** – harvesting all the filtered records result sets from the `bibs` endpoint for the records that match the query parameters.

The **loop** continues making requests – advancing the ID to the next largest integer ID found in previous result sets – **until the length of the result set in the response is less than the limit, or ID not found** (HTTP status code 404).

```python
# Get all bib record IDs updated on or after `2025-01-21`
start_id = 0  # start at ID 0
start_date = '2025-01-21T00:00:00Z'  # range is inclusive
limit = 2000  # 2000 is max limit offered by the API (50 is default)
records = []  # store the records' metadata in a list

while True:
  try:
    response = client.request(
      'GET',
      'bibs/',
      # params are the "query parameters" of the HTTP request ...
      params={
        'fields': 'id,createdDate,updatedDate',  # filter param
        'id': f"[{start_id},]",  # open-ended id
        'updatedDate': f"[{start_date},{timestamp_now}]",  # date range
        'limit': limit,
      }
    )
    response.raise_for_status()  # throw error on status 404, record not found
    entries = response.json().get('entries', [])
    records.extend(entries)  # add the response records to the list
    if not entries or len(entries) < limit:
      break
    start_id = int(entries[-1].get('id')) + 1  # advance the start id, loop

  except Exception as e:
    print(f"{e}")
    break

len(records)  # e.g. 7077
```

Anatomy of a Sierra REST API Request (Cont'd)

Not all endpoints support pagination with an ID – e.g. `patrons/holds` **endpoint only supports the** `limit/offset` **query params**.

The **Python code** on the right demonstrates an **asynchronous** implementation of **pagination** – by batching the requests async, we can **efficiently build the entire set of** `holds`, instead of sending blocking requests one-by-one.

```python
# working with an offset can be slow, we can use async to send requests in batcl
# ... especially for endpoints that don't support
import asyncio

limit = 2000
offset = 0
holds = []  # list to store holds entries

async def get_holds(limit=2000, offset=0):
  response = await client.async_request(
    'GET',
    'patrons/holds',
    params={'limit': limit, 'offset': offset,}
  )
  response.raise_for_status()
  entries = response.json().get('entries', [])
  if entries:
    return (response.json().get('total', 0), entries)
  else:
    return (0, [])

# get the initial number of holds, and the holds themselves
total, first_entries = await get_holds(limit=limit, offset=offset)
holds.extend(first_entries)  # add the first batch of holds
offsets = range(limit, total, limit)  # generate the ranges -- start,stop,step
# create tasks for remaining batches (offset generated by range function)
tasks = [get_holds(limit=limit, offset=offset) for offset in offsets]
results = await asyncio.gather(*tasks, return_exceptions=True)
for result in results:
  if isinstance(result, Exception):
    print(f"Error: {result}")
  else:
    holds.extend(result[1])

print(len(holds), total)  # list len and tot can differ due to changes w/ holds
```

Scripts You Can Use

Interactive Tutorial

Use case

How to train future systems librarians, or database concepts, or Sierra's data structure when we can't permit direct database access to most staff

https://colab.research.google.com/drive/1IM1Dc9l0d-CqXzeJx7Otuk-6XHojJamX

API Tutorial

MLN interactive API tutorial

- Staff can learn while working with a restricted set of endpoints
- Bib and Item Read only
- Also learn some Python basics without needing a local install
- Does not use sierra-ils-utils in order to demonstrate the 0Auth process more fully

Minuteman Sierra API Tutorial.ipynb

File Edit View Insert Runtime Tools Help

Share

Table of contents

Introduction
Authentication Example
Get /items/{id} example
Get /items/{id} json practice
Post Items/Query Example
Combining API Endpoints
Retrieving BibIDs
Combine bib and item APIs
Combined bibs and items output
Export Results (Final Script)
+ Section

+ Code + Text

Connect

```python
'''
The script begins with import calls to bring in various Python packages containing functionality needed by our script
'''
import requests #package for placing HTTP requests
import json #package for working with JSON data
'''
#Use config parser and a .ini file outside of Colab
import configparser
'''
from google.colab import userdata #Google colab function for accessing secrets
from pprint import pprint #package for pretty printing structured data such as JSON
from base64 import b64encode #package for base64 encryption

'''
Defining a function that will authenticate with the Sierra API and provide us with an access token to be used on subse
In this example it is returning some additional information we will later ignore for purposed of demonstration
'''
def get_token():
    '''
    #Use outside of colab
    config = configparser.ConfigParser()
    config.read('api_info.ini')
    base_url = config['api']['base_url']
    client_key = config['api']['client_key']
    client_secret = config['api']['client_secret']
    '''
    base_url = userdata.get('api_base_url')#creating variables to store the key/secret/url from Colab's secret enviror
    client_key = userdata.get('api_key')
    client_secret = userdata.get('api_secret')
    auth_string = b64encode((client_key + ':' + client_secret).encode('ascii')).decode('utf-8')#encrypting the combine
    header = {}
    header["authorization"] = 'Basic ' + auth_string #forming header to be included with HTTP call
    url = base_url + '/token' #forming full url for the API endpoint used for authenticating the user
    response = requests.post(url, headers=header) #Making a post request using the url and header and saving results t
    json_response = json.loads(response.text) #saving the response text to a JSON object
    return json_response, response.headers, response.status_code #returns the json_response, response headers and stat

'''
Defining a function called main for running the bulk of our script
'''
def main():
```


Get /items/{id} example
Get /items/{id} json practice
Post Items/Query Example
Combining API Endpoints
Retrieving BibIDs
Combine bib and item APIs
Combined bibs and items output
Export Results (Final Script)
+ Section


```python
        #print selected fields from item and bib responses and a dividing line at the end
        print('Call #: ' + item_details['callNumber'])
        print('Title : ' + bib_details['title'])
        print('Author : ' + bib_details['author'])
        print('Barcode : ' + item_details['barcode'])
        print('-----------------------------------------------')

main()
```

```
Call #: 320.4 Favreau
Title : Democracy or else : how to save America in 10 easy steps
Author : Favreau, Jon, 1981- author.
Barcode : 31213013011657
-----------------------------------------------
Call #: 618.928583 Turban
Title : Free to be : understanding kids & gender identity
Author : Turban, Jack L., author.
Barcode : 31213013012069
-----------------------------------------------
Call #: FICTION Day
Title : One big happy family
Author : Day, Jamie, author.
Barcode : 31213013011087
-----------------------------------------------
```

Batch Update Callnumbers

Use case

Library's paperback Romance collection had call numbers with the structure

ROMANCE [First letter of last name]

Desired change to structure

ROMANCE [last name, first name]

Need to copy the 100|a from bibrecord into attached items

Batch Update Callnumbers (Cont'd)

In [1]:
```
import json
from sierra_ils_utils import SierraAPI
import csv
import configparser
```

In [2]:
```
config = configparser.ConfigParser()
config.read('Y:\\SQL Reports\\creds\\api_info.ini')

'''
.ini file contains url/key/secret for the api in the following form
[api]
base_url = https://[local domain]/iii/sierra-api/v6
client_key = [enter Sierra API key]
client_secret = [enter Sierra API secret]
'''

base_url = config['api']['base_url'] + '/'
#note sierra-ils-utils assumes base_url contains the trailing /, which the file I have been using did not contain so it
client_key = config['api']['client_key']
client_secret = config['api']['client_secret']

sierra_api = SierraAPI(base_url,client_key,client_secret)
sierra_api.request('GET','info/token')
```

Out[2]:
```
<Response [200 200]>
```

In [ ]:
```
with open('cam_romance.txt','r') as csv_file:
    csv_reader = csv.DictReader(csv_file, delimiter=',')
    for row in csv_reader:
        print(row['RECORD #(ITEM)'][1:-1] + ' ROMANCE ' + row['100|a'].rstrip(',').rstrip('.'))
        url = 'items/' + row['RECORD #(ITEM)'][1:-1]
        call_number = 'ROMANCE ' + row['100|a'].rstrip(',').rstrip('.')
        request = sierra_api.request(
            'PUT',
            url,
            json={
                'callNumbers': [call_number]
            }
        )
        request.raise_for_status()
```

https://github.com/Minuteman-Library-Network/batch-update-callnumbers/blob/main/cam%20romance.ipynb

Batch Update Callnumbers (Cont'd)

The library who requested this change gathered up their items into a review file, from which the record number, call # and 100|a fields were exported

Existing call # is not used in the script and was simply included here as a reference point

100|a requires some punctuation to be removed

Record #'s need the record_type_code prefix and check digit to be removed

batch-update-callnumbers / cam_romance.txt

jmgold init commit

Code Blame 1259 lines (1259 loc) · 58.9 KB

```
"RECORD #(ITEM)","CALL #","100|a"
"i118869747","[PB] ROMANCE M","Mallory, Margaret,"
"i122255884","[PB] ROMANCE H","Higgins, Kristan."
"i126593917","[PB] ROMANCE E","Essex, Elizabeth."
"i127675425","[PB] ROMANCE C","Campbell, Anna."
"i127712562","[PB] ROMANCE B","Burrowes, Grace."
"i12855907x","[PB] ROMANCE B","Burrowes, Grace,"
"i130296624","[PB] ROMANCE B","Boyle, Elizabeth."
"i130815986","[PB] ROMANCE C","Camp, Candace."
"i131080337","[PB] ROMANCE S","Solheim, Tracy."
"i132539652","[PB] ROMANCE A","Alers, Rochelle."
"i132799741","[PB] ROMANCE R","Ranney, Karen."
"i133073361","[PB] ROMANCE B","Burrowes, Grace."
"i133097006","[PB] ROMANCE J","Jordan, Sophie."
"i133097055","[PB] ROMANCE B","Beverley, Jo,"
"i13321543x","[PB] ROMANCE L","Lin, Jeannie."
"i13325625x","[PB] ROMANCE A","Ashe, Katharine,"
"i133386855","[PB] ROMANCE R","Roberts, Victoria."
```

Batch Update Callnumbers (Cont'd)

For each row in the open csv file:

- Extract record number to be used as path parameter
  - Use slice notion [1:-1] to indicate the desired start and endpoints of the record number
- Construct callNumber to include in path object
  - Use rstrip() function to remove unwanted ending punctuation
- Submit put request using the **items/{id}** endpoint
- Print details to screen to provide indicator the script is working

```
"RECORD #(ITEM)","CALL #","100|a"
"i118869747","[PB] ROMANCE M","Mallory, Margaret,"
"i122255884","[PB] ROMANCE H","Higgins, Kristan."
"i126593917","[PB] ROMANCE E","Essex, Elizabeth."
```

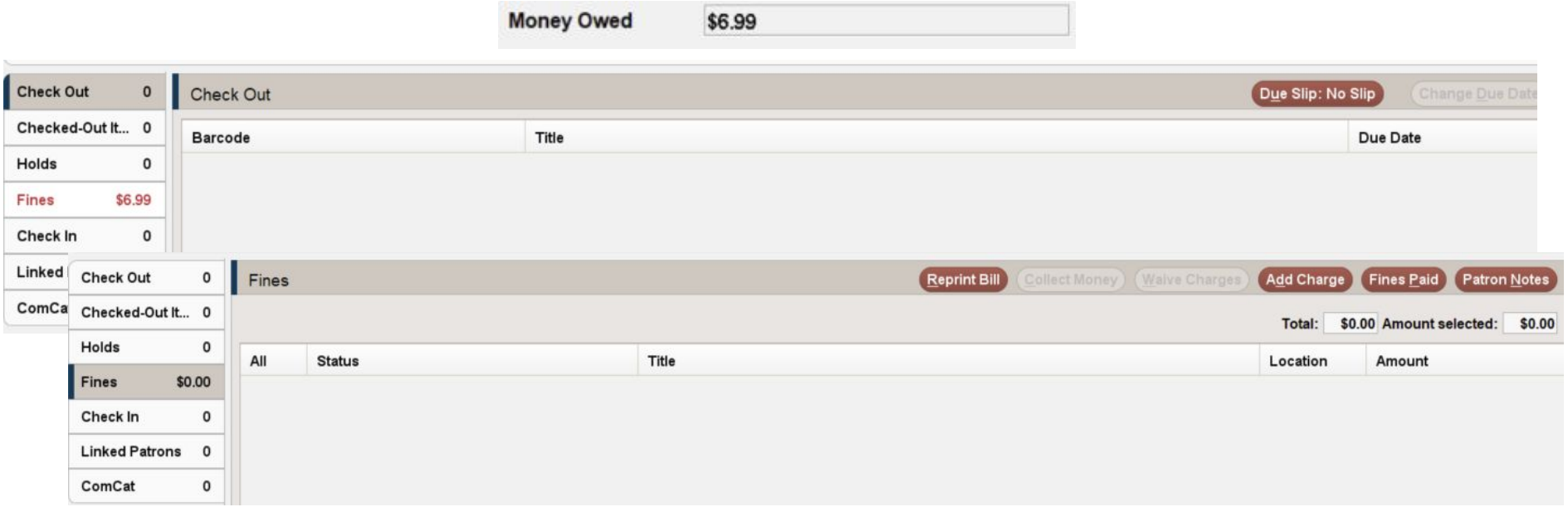
```python
with open('cam_romance.txt','r') as csv_file:
    csv_reader = csv.DictReader(csv_file, delimiter=',')
    for row in csv_reader:
        print(row['RECORD #(ITEM)'][1:-1] + ' ROMANCE ' + row['100|a'].rstrip(',').rstrip('.'))
        url = 'items/' + row['RECORD #(ITEM)'][1:-1]
        call_number = 'ROMANCE ' + row['100|a'].rstrip(',').rstrip('.')
        request = sierra_api.request(
            'PUT',
            url,
            json={
                'callNumbers': [call_number]
            }
        )
        request.raise_for_status()
```

```
11886974 ROMANCE Mallory, Margaret
12225588 ROMANCE Higgins, Kristan
12659391 ROMANCE Essex, Elizabeth
```

Correct Money Owed Discrepancies

Use Case

Roughly a dozen times a month a fine will be paid without correctly updating the amt owed field for the patron record



Solutions

Can ask III to correct via a "Repair discrepancy in Money Owed" Service commitment

Alternately you can fix this yourself by adding a manual charge in that amount and then waiving it

We wished to automate this task

Correct Money Owed Discrepancies (Cont'd)

Three functions are defined in the script

- runquery will execute a provided sql query against Sierra and return the query results
- manual_charge will create a manual charge by making a post request using the **patrons/{id}/fines/charge** endpoint
  - Requires patron_id, amount, and location fields to be provided
- clear_fine will waive a fine by making a PUT request using the **patrons/{id}/fines/payment endpoint** Requires patron_id & invoiceNumber

```python
def runquery(query):
    config = configparser.ConfigParser()
    config.read('Y:\\SQL Reports\\creds\\api_info.ini')

    try:
        conn = psycopg2.connect( config['api']['connection_string'] )
    except:
        print("unable to connect to the database")
        clear_connection()
        return

    cursor = conn.cursor()
    cursor.execute(query)
    rows = cursor.fetchall()
    return rows
```

```python
def manual_charge(patron_id,amount,location):
    url = "patrons/" + patron_id + "/fines/charge"
    params = {"amount": amount, "reason": "Residual fine","location": location}
    request = sierra_api.request('POST',url,json=params)
    request.raise_for_status()

def clear_fine(patron_id,invoiceNumber):
    url = "patrons/" + patron_id + "/fines/payment"
    params = {"payments": [{"amount": 0, "paymentType": 2, "invoiceNumber": "" + invoiceNumber + "
    request = sierra_api.request('PUT',url,json=params)
    request.raise_for_status()
```

https://github.com/Minuteman-Library-Network/SQL-Queries/blob/master/Scripted%20Reports/Correct%20Money%20Owed%20Discrepancies.ipynb

Correct Money Owed Discrepancies (Cont'd)

Two SQL queries are defined

- Error_query finds patrons where current owed_amt != SUM(fines)
- Manual_charge_query finds the manual charges created to reconcile these errors

manual_charge() is run for each result from error_query

clear_fine() is run for each result from manual_charge_query

```
error_query = """\
    SELECT
      rm.record_num,
      (p.owed_amt * 100 - (SUM(COALESCE(f.item_charge_amt*100, 0) + COALESCE(f.processing_fee_amt*100, 0) + COALESCE(f.billi
      p.home_library_code AS location
    FROM sierra_view.record_metadata rm
    JOIN sierra_view.patron_record p
      ON p.id = rm.id
    LEFT JOIN sierra_view.fine f
      ON f.patron_record_id = p.id

    GROUP BY rm.record_num, p.owed_amt,3
    HAVING p.owed_amt != SUM(COALESCE(f.item_charge_amt, 0.00) + COALESCE(f.processing_fee_amt, 0.00) + COALESCE(f.billing_f
    """

manual_charge_query = """\
    SELECT
      rm.record_num,
      f.invoice_num::varchar
    FROM sierra_view.fine f
    JOIN sierra_view.record_metadata rm
        ON f.patron_record_id = rm.id
    WHERE f.assessed_gmt::DATE = CURRENT_DATE
        AND f.charge_code = '1'
        AND f.description = 'Residual fine'
    """

#identify patrons with amt owed errors and create manual charges in the amount of those discrepancies
amt_owed_errors = runquery(error_query)
for rownum, row in enumerate(amt_owed_errors):
    manual_charge(str(row[0]),row[1],row[2])

#Find the newly created manual charges and waive them
fines_to_clear = runquery(manual_charge_query)
for rownum, row in enumerate(fines_to_clear):
    clear_fine(str(row[0]),row[1])
```

https://github.com/Minuteman-Library-Network/SQL-Queries/blob/master/Scripted%20Reports/Correct%20Money%20Owed%20Discrepancies.ipynb

Playing with Review Files

New endpoint with 6.3!

11

VOTE

Sierra 6.3 - Create a Review File endpoint

Idea Description
This idea is based in part on one submitted back in 2019 by Andy Helck. Create an endpoint that will provide the list of records included in a specified review file from Create Lists. I essentially envision this as the API version of the bool_set SQL table.

Idea Value
This would provide outside systems the means of retrieving lists of titles that staff maintain within create lists and can provide a means for services to access data without having to work out how to create Json queries.

Mike Dicus (Product Manager, Innovative) shared this idea · Jan 4, 2024 · Report...

COMPLETED · Mike Dicus (Product Manager, Innovative) responded · Dec 17, 2024
Implemented in Sierra 6.3

Show previous admin responses (2)

GET /v6/reviewFiles/ — Get a list of review files

Response Class
Model | Model Schema

ReviewFileInfo {
id (integer): the review file ID,
name (string): the name of the review file,
total (integer): the total number of entries in the review file,
recordType (string): the type of records (record IDs) stored in the review file,
username (string): the review file owner's username,
createdDate (string): the date and time the review file was created, in ISO 8601 format (yyyy-MM-dd'T'HH:mm:ssZZ)
}

Response Content Type application/json

Parameters

Parameter	Value	Description	Parameter Type	Data Type
recordType		Type of record in review files	query	string

Try it out!

GET /v6/reviewFiles/{id}/records — Get a list of record identifiers belonging to a review file

Response Class
Model | Model Schema

ReviewFileContent {
total (integer, *optional*): the total number of record identifiers in the review file,
start (integer, *optional*): the starting position of this set of record identifiers,
entries (array[string]): the record identifiers in this set
}

Response Content Type application/json

Parameters

Parameter	Value	Description	Parameter Type	Data Type
id	(required)	**the review file ID**	path	integer
limit		the maximum number of record identifiers to return	query	integer
offset		the beginning (zero-indexed) of the result set returned	query	integer

Try it out!

6.3 is only loaded on Minuteman's test server

Following slides are merely experiments to test some possibilities

Export Review File

Assign a known file number to review_file

Make GET request to **reviewFiles** endpoint containing metadata for all files containing data

Filter response to entry with id = review_file and extract the file name

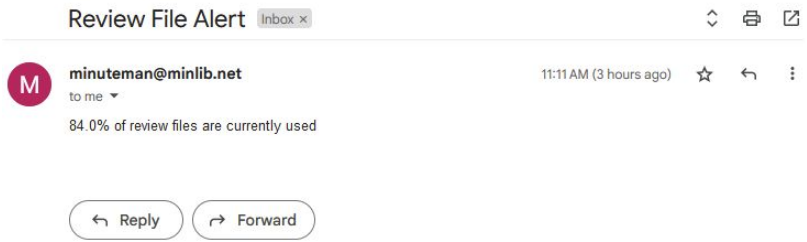
Make GET request to **reviewFiles/{id}/records** to retrieve all bibs in file.

For each record make GET request to **bibs/{id}** to pull out record fields. Removing 'b' from start of each id

```python
review_file = '647' #set review file number to export from.  Must contains bibs

review_file_metadata = client.request( #retrieves all review files containing bibs
    'GET',
    "reviewFiles",
    params={
        'recordType': 'b'
    }
)
data = review_file_metadata.json()
#reduce API response to just the entry matching review_file # entered above
review_file_metadata = [entry for entry in data if entry['id'] == int(review_file)]

print(review_file_metadata[0]['name']) #pull out the review file's name from Sierra
print('--------------------------------------------------')

review_file_list = client.request( #get list of records contained in review file
    'GET',
    "reviewFiles/"+review_file+"/records"
    )

for entry in review_file_list.json()["entries"]: #retrieve bib fields for each record in file
  title = client.request(
    'GET',
    "bibs/"+entry[1:], #must remove first character containing the record_type_code
    params={
        'fields': 'title,author,isbn,materialType'
      }
  )
  print('Title: ' + title.json().get('title','')) #display record details, accounting for null values
  print('Author : ' + title.json().get('author',''))
  print('ISBN : ' + title.json().get('isbn',''))
  print('MatType : ' + title.json().get('materialType','').get('value','UNKNOWN'))
  print('--------------------------------------------------')
```

```
Watertown Library's Featured Items
--------------------------------------------------
Title: Uprooted
Author : Novik, Naomi, author.
ISBN : 9780804179058
MatType : BOOK
--------------------------------------------------
Title: Hamster Princess : of mice and magic
Author : Vernon, Ursula, author, illustrator.
```

https://colab.research.google.com/drive/1qnt20SBwN6yLcq4Q4cPlNeeVkBqXJK8L



Set review_file_total and warning_percentage variables

Make GET request to **reviewFiles** to pull list of all files containing data

Use len() to calculate # of files in use and use to calculate review_file_percentage

If review_file_percentage >= warning_percentage, email out an alert using the smtplib library

```
[ ]
import smtplib
from email.mime.text import MIMEText

review_file_total = 652 #set total number of available review files in your system
warning_percentage = 80 #% of review files in use at which you will send an alert

review_file_list = client.request( #gather list of all review files containing data
    'GET',
    "reviewFiles"
)

all_file_metadata = review_file_list.json()
files_in_use = len(all_file_metadata) #get count of files
review_file_percentage = round(files_in_use/review_file_total,2) * 100

print(str(files_in_use) + " of " + str(review_file_total) + " files are in use")
print("Review file usage is currently " + str(review_file_percentage)+"%")

if review_file_percentage >= warning_percentage:
    sender_email = userdata.get('email')
    sender_password = userdata.get('email_password')
    receiver_email = "jgoldstein@minlib.net"

    message_text = str(review_file_percentage) + "% of review files are currently used"
    message = MIMEText(message_text)
    message["Subject"] = "Review File Alert"
    message["From"] = sender_email
    message["To"] = receiver_email

    with smtplib.SMTP_SSL("smtp.gmail.com", 465) as server:
        server.login(sender_email, sender_password)
        server.sendmail(sender_email, receiver_email, message.as_string())
    print("Email sent successfully!")
```

```
547 of 652 files are in use
Review file usage is currently 84.0%
Email sent successfully!
```

https://colab.research.google.com/drive/1qnt20SBwN6yLcq4Q4cPlNeeVkBqXJK8L

For More Information

https://chimpy.me/blog/posts/iug-2025/

Thank You

Questions?

Jeremy Goldstein - jgoldstein@minlib.net
Ray Voelker - Ray.Voelker@chpl.org