UCS Director 3rd-Party REST Integration Example

An Example UCSD REST Integration with Open Library

Ryan D. Criss
Network Consulting Engineer
Cisco Systems
rcriss@cisco.com

2/16/2016

1 Table of Contents

2 Introduction 2
3 The Right Tools for the Job 3
  3.1 A Quick Note on REST 3
  3.2 API Guide 3
  3.3 Using Postman 3
  3.4 UCS Director 4
  3.5 JavaScript Reference 5
4 Method to the Madness 5
  4.1 Find the API Calls 5
  4.2 Testing in Postman 6
  4.3 Create the UCS Director Custom Task 10
  4.4 Creating the Custom Task Code 12
    4.4.1 Retrieve Name Function 12
    4.4.2 Retrieve Details Function 13
    4.4.3 Retrieve Author Name Function 14
    4.4.4 Main Function 15
  4.5 JSON Parsing Fun 16
  4.6 Troubleshooting Your Custom Task 17
5 Dealing with SSL Certificates 17
  5.1 Potential SSL Certificate Issues 18
  5.2 Adding Certificates to the UCSD Trusted CA Keystore 18
6 Creating Your Own Custom Integrations 19

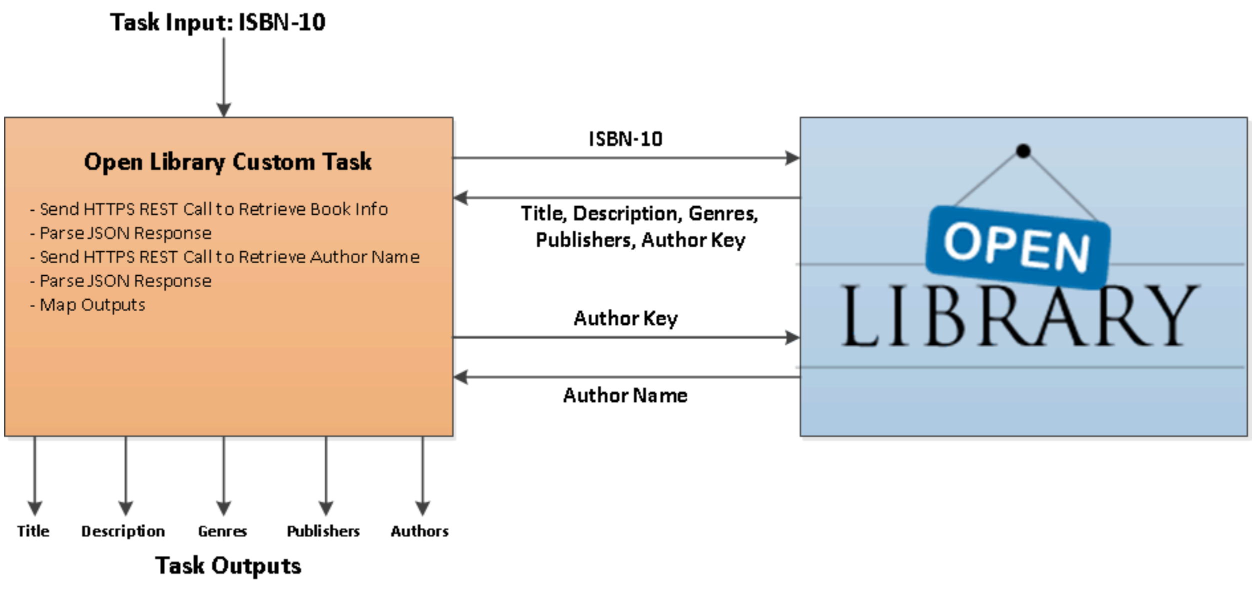
2 Introduction

While UCS Director comes with a lot of tasks and integrations out of the box, it is inevitable in your orchestration journey that you come across an integration that doesn't exist. When that happens, you may be forced to turn towards the great world of Custom Tasks. At first glance, you might be a little intimidated, especially if you haven't touched JavaScript since college. My goal for this guide is to walk you through a custom task integration using REST and show you the methodology, tools, and custom task templates to help ease your fears. When you do one or two of them, you will find it's not so bad after all!

A majority of data center integrations with UCS Director can be tackled using APIs (REST, SOAP, XML, PowerShell, etc.) or SSH. UCS Director can handle all of these, but for this guide I will just focus on RESTful integrations. In the coming weeks, I will put out some additional guides around SOAP and SSH integrations.

For this guide, we will be integrating with Open Library (http://openlibrary.org) using their RESTful API to create a custom task that retrieves various book details when you type in a book's ISBN-10. Why Open Library? Because I like books, but also because it's pretty simple and similar to a lot of other RESTful APIs out there in the data center. If you can master Open Library, you can master any other RESTful API with UCS Director!

Here is a quick peek at what we will be doing:



One more thing… A quick disclaimer. This guide was created based on my own experience working with UCS Director and creating integrations over the past several years and does not represent an officially supported Cisco integration guide. The thoughts, opinions, and methodologies represented here are my own and not those of Cisco.

3 The Right Tools for the Job

As with any project, it's important to start off with the right tools for the job. Here are several items we should gather before diving into our custom task work:

- API Guide for the system you are integrating with
- Credentials for the system you are integrating with (if necessary)
- Postman or other REST client for testing
- UCS Director with access to the system you are integrating with (Double check DNS and proxy info if necessary)
- A JavaScript reference (The folks at W3Schools do an awesome job: http://www.w3schools.com/js/)

3.1 A Quick Note on REST

I'm going to assume you have a basic understanding of what REST is in this guide, but if you are not sure what REST is, take a quick peak at the Wikipedia definition of REST: https://en.wikipedia.org/wiki/Representational_state_transfer

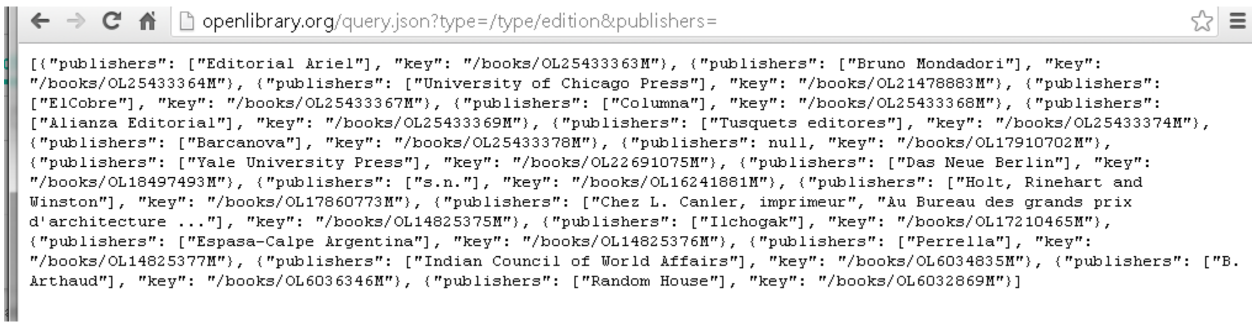
In simple terms, a RESTful API means that you send a HTTP message with some information, and the API responds with some information you want back. Pretty simple, huh?

3.2 API Guide

How do you know how to integrate with a system if you don't know what's available to integrate with? The API guide will tell us what we can actually do through the RESTful API. Before we begin, let's take a look through the Open Library API guide and get the lay of the land. You can find the Open Library API guide here: https://openlibrary.org/dev/docs/restful_api

The cool thing about REST is that you can test basic REST calls in any browser. If you navigate to one of the sample Open Library API URLs in your browser, it will return a bunch of interesting information in a very difficult to read format. This might work great for basic testing and to just get the lay of the land, but using a REST client like Postman will help make things easier to read and understand.

Here is a peek at what it would look like in your browser:



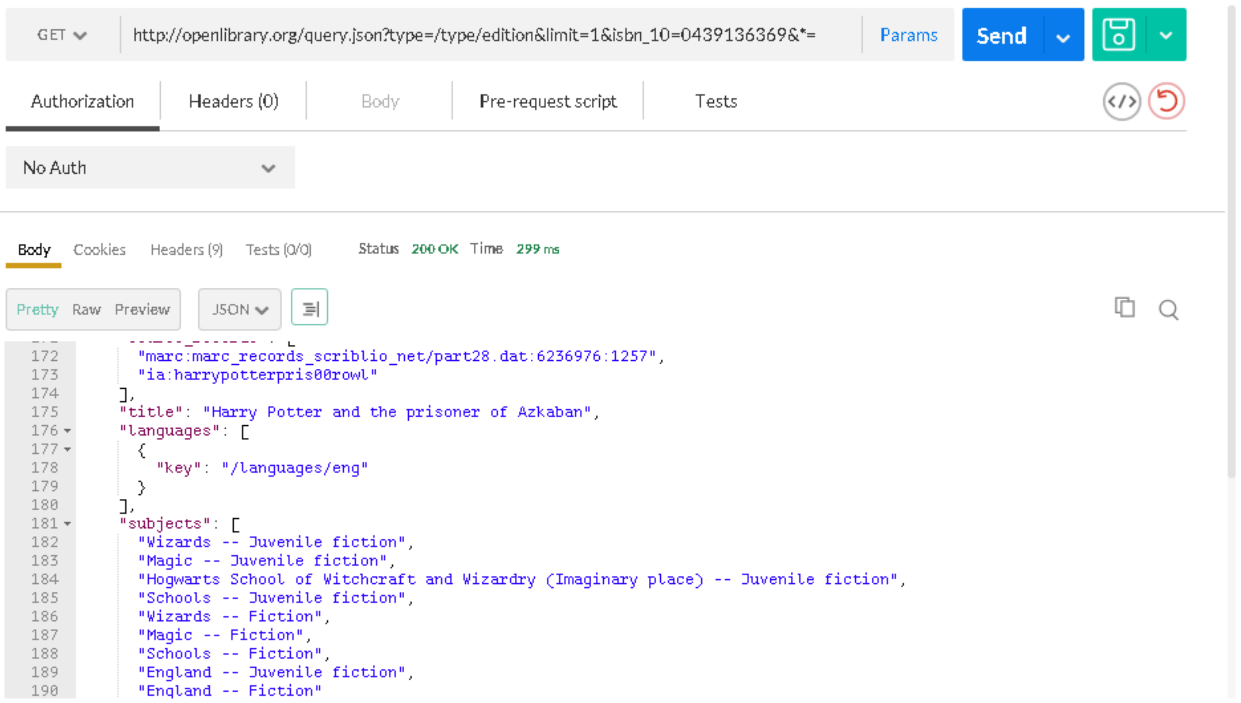
```
[{"publishers": ["Editorial Ariel"], "key": "/books/OL25433363M"}, {"publishers": ["Bruno Mondadori"], "key":
"/books/OL25433364M"}, {"publishers": ["University of Chicago Press"], "key": "/books/OL21478883M"}, {"publishers":
["ElCobre"], "key": "/books/OL25433367M"}, {"publishers": ["Columna"], "key": "/books/OL25433368M"}, {"publishers":
["Alianza Editorial"], "key": "/books/OL25433369M"}, {"publishers": ["Tusquets editores"], "key": "/books/OL25433374M"},
{"publishers": ["Barcanova"], "key": "/books/OL25433378M"}, {"publishers": null, "key": "/books/OL17910702M"},
{"publishers": ["Yale University Press"], "key": "/books/OL22691075M"}, {"publishers": ["Das Neue Berlin"], "key":
"/books/OL18497493M"}, {"publishers": ["s.n."], "key": "/books/OL16241881M"}, {"publishers": ["Holt, Rinehart and
Winston"], "key": "/books/OL17860773M"}, {"publishers": ["Chez L. Canler, imprimeur", "Au Bureau des grands prix
d'architecture ..."], "key": "/books/OL14825375M"}, {"publishers": ["Ilchogak"], "key": "/books/OL17210465M"},
{"publishers": ["Espasa-Calpe Argentina"], "key": "/books/OL14825376M"}, {"publishers": ["Perrella"], "key":
"/books/OL14825377M"}, {"publishers": ["Indian Council of World Affairs"], "key": "/books/OL6034835M"}, {"publishers": ["B.
Arthaud"], "key": "/books/OL6036346M"}, {"publishers": ["Random House"], "key": "/books/OL6032869M"}]
```

3.3 Using Postman

By now you are probably getting a headache just by looking at your REST API results in your browser. The results are formatted using a standard called JSON. We'll touch more on JSON in a bit, but the Postman tool makes it much easier to send REST API calls and see the results in a nicely formatted way.

You can download Postman here or use some other REST client/tool that you prefer: https://www.getpostman.com/

You can see below how much nicer this looks than your browser:



GET http://openlibrary.org/query.json?type=/type/edition&limit=1&isbn_10=0439136369&*=

No Auth

Status 200 OK Time 299 ms

```
    "marc:marc_records_scriblio_net/part28.dat:6236976:1257",
    "ia:harrypotterpris00rowl"
  ],
  "title": "Harry Potter and the prisoner of Azkaban",
  "languages": [
    {
      "key": "/languages/eng"
    }
  ],
  "subjects": [
    "Wizards -- Juvenile fiction",
    "Magic -- Juvenile fiction",
    "Hogwarts School of Witchcraft and Wizardry (Imaginary place) -- Juvenile fiction",
    "Schools -- Juvenile fiction",
    "Wizards -- Fiction",
    "Magic -- Fiction",
    "Schools -- Fiction",
    "England -- Juvenile fiction",
    "England -- Fiction"
```

Download it and play around with it for a bit. You'll love it!

Whenever I'm working on RESTful APIs, I always test my API calls in Postman before moving them over to UCSD. In Postman I can very quickly change my API call if I need to adjust the request and can also see the JSON results and figure out the best way to parse the results in UCSD.

3.4 UCS Director

This guide was created while using UCSD 5.4.0.0. Several things to note as you are preparing to use UCSD for RESTful integrations:

1. The UCSD orchestration node must be able to resolve the hostname that you are making the integration with. Double check your UCSD settings to ensure DNS is properly setup. You can jump onto the UCSD Orchestration node as root and ping your destination host to validate.

2. If you are connecting to an external host, make sure the proxy settings are set correctly in the custom task. Because we are making an HTTP call using a custom task, even if your UCS Director appliance is setup correctly with a proxy server, you will still need to set a proxy setting in the

custom task as shown in my task example.

3. If you are connecting to a host over HTTPS (which you should always do), you may encounter some certificate issues if you are working with a self-signed certificate or a default out-of-the-box certificate on your destination host. I have a section later on in this guide to show you how to navigate this certificate issue. Although it's troublesome sometimes to deal with certificates, you never want to have usernames, passwords, auth cookies, or auth strings sent over the network in clear text. Always make sure you are securing your connection!

3.5 JavaScript Reference

UCSD Custom Tasks are currently written in JavaScript, so knowing a little JavaScript definitely helps. However, if you have any scripting or programming background whatsoever, you should get by just fine with the help of a JavaScript reference. Don't be ashamed to bookmark a site like W3Schools or something similar (http://www.w3schools.com/js/). This site is very helpful if you are trying to figure out arrays, loops, or variables in a custom task.

4 Method to the Madness

As you begin working on your custom task, it's important to recognize there is a method to the madness and it all starts with your approach. While there are a number of ways you can approach a custom task, I typically pursue an approach like the following:

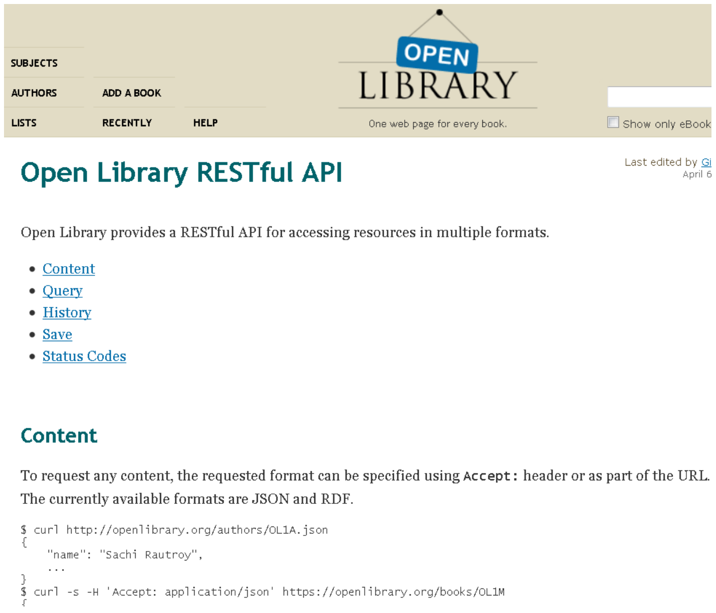
1. Find the API Calls
2. Test the API Calls in Postman
3. Create the UCSD Custom Task
4. Create the Custom Task Code
5. Parse the JSON Response
6. Troubleshooting/Testing Your Custom Task

I'll go into a little more detail for each step below.

4.1 Find the API Calls

Before we get too far, we must understand what API calls we will be making from UCSD. To do that, we must understand the RESTful API we will be integrating with. Unfortunately, that means we must also sift through some API documentation for the system we are trying to integrate with. Sometimes the API documentation is great and will lead us directly to the call we need, while other times we must stare at the documentation a bit, try some different calls out in Postman, and figure out the correct call by trial and error.

Let's go ahead and open up our Open Library API Guide again (https://openlibrary.org/dev/docs/restful_api). It should look something like this:



Open Library RESTful API

Open Library provides a RESTful API for accessing resources in multiple formats.

- Content
- Query
- History
- Save
- Status Codes

Content

To request any content, the requested format can be specified using `Accept:` header or as part of the URL. The currently available formats are JSON and RDF.

```
$ curl http://openlibrary.org/authors/OL1A.json
{
    "name": "Sachi Rautroy",
    ...
}
$ curl -s -H 'Accept: application/json' https://openlibrary.org/books/OL1M
{
```

As we go through the guide, keep in mind our goal of trying to get a bunch of book information based on the ISBN-10. After scanning through the guide, I find something that looks interesting… The Query API. They give us the following example:
http://openlibrary.org/query.json?type=/type/edition&authors=/authors/OL1A&*=

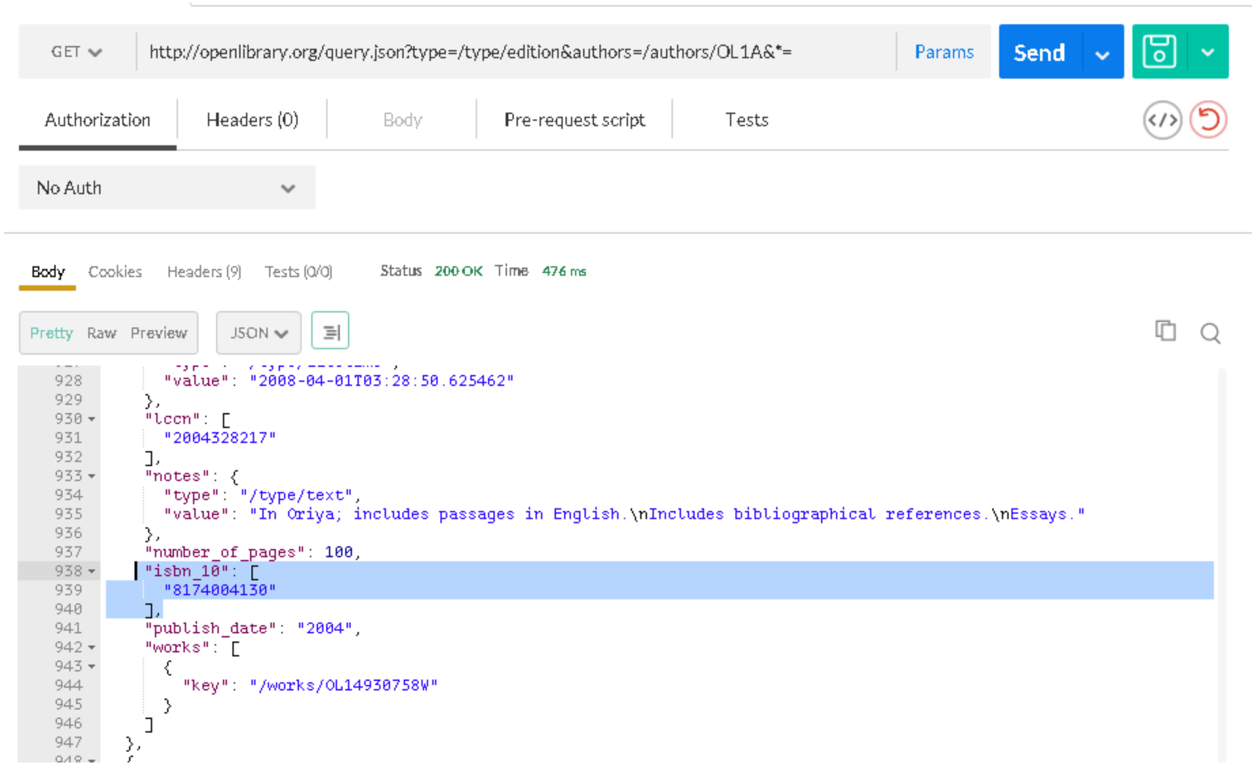
This gives us a list of all the books by a given author (The author is not the author name, but the author ID within the Open Library system). This is not exactly what we want, but if you browse through the data, you find that the ISBN-10 is included in this data, so we are on the right track. We should be able to find a way to filter our request based on the ISBN instead of the author.

This API guide does not give us any additional info on how to pull out certain information based on a criteria, so we will have to play around with it for a bit in Postman to figure out exactly the API call we want using the Query API. Lets' take this Query API and plug it into Postman to see what we can pull out.

4.2 Testing in Postman

While you can play around with the API guide and the examples in your browser, it's much easier and more efficient to use Postman to test out your API calls and look at the return data.

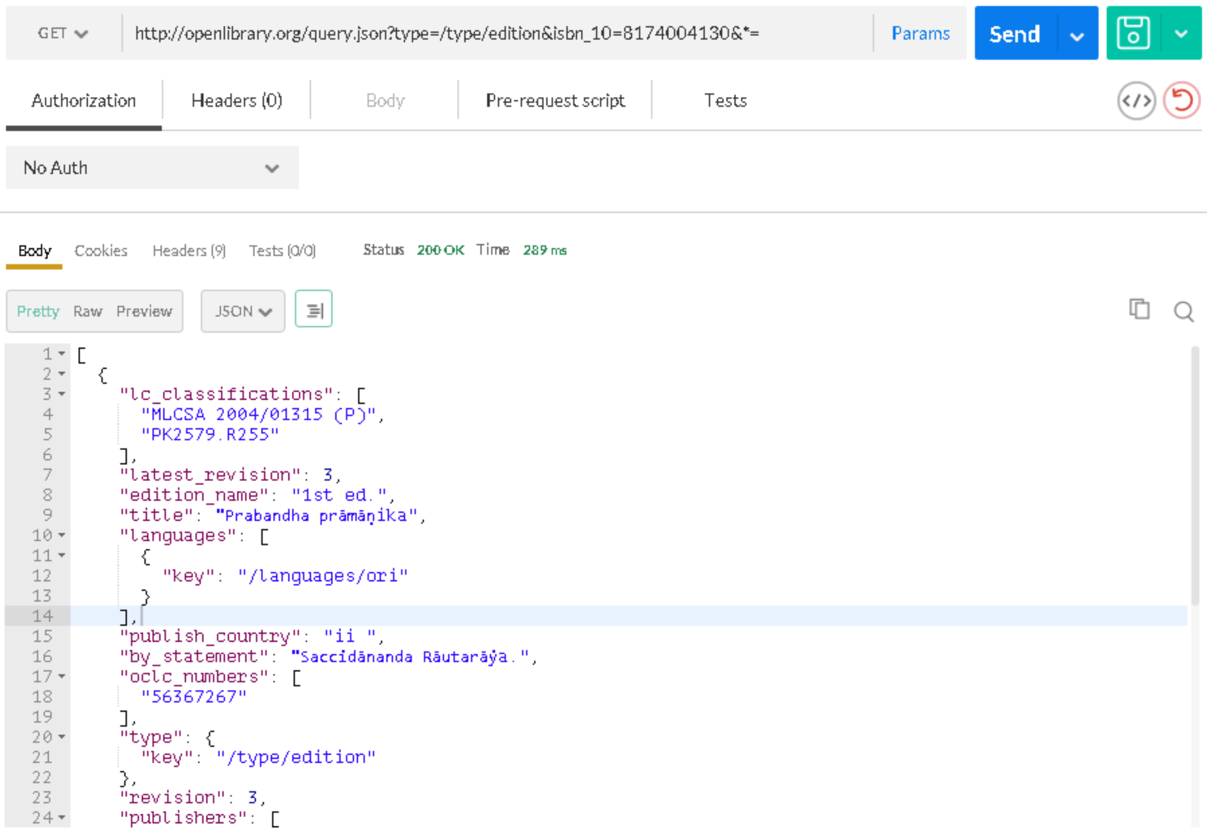
Let's plug in the example Query API example they have in the API Guide into Postman and see what we can find:



```
http://openlibrary.org/query.json?type=/type/edition&authors=/authors/OL1A&*=
```

```
      "value": "2008-04-01T03:28:50.625462"
    },
    "lccn": [
      "2004328217"
    ],
    "notes": {
      "type": "/type/text",
      "value": "In Oriya; includes passages in English.\nIncludes bibliographical references.\nEssays."
    },
    "number_of_pages": 100,
    "isbn_10": [
      "8174004130"
    ],
    "publish_date": "2004",
    "works": [
      {
        "key": "/works/OL14930758W"
      }
    ]
  },
```

After sifting through the data, we find that the isbn_10 is buried in the results. Excellent! Let’s see what we can do to change the query around so we search by ISBN-10 instead of author. I’m going to change the “authors=/authors/OL1A” section to “isbn_10=8174004130” and see what happens.



```
http://openlibrary.org/query.json?type=/type/edition&isbn_10=8174004130&*=
```

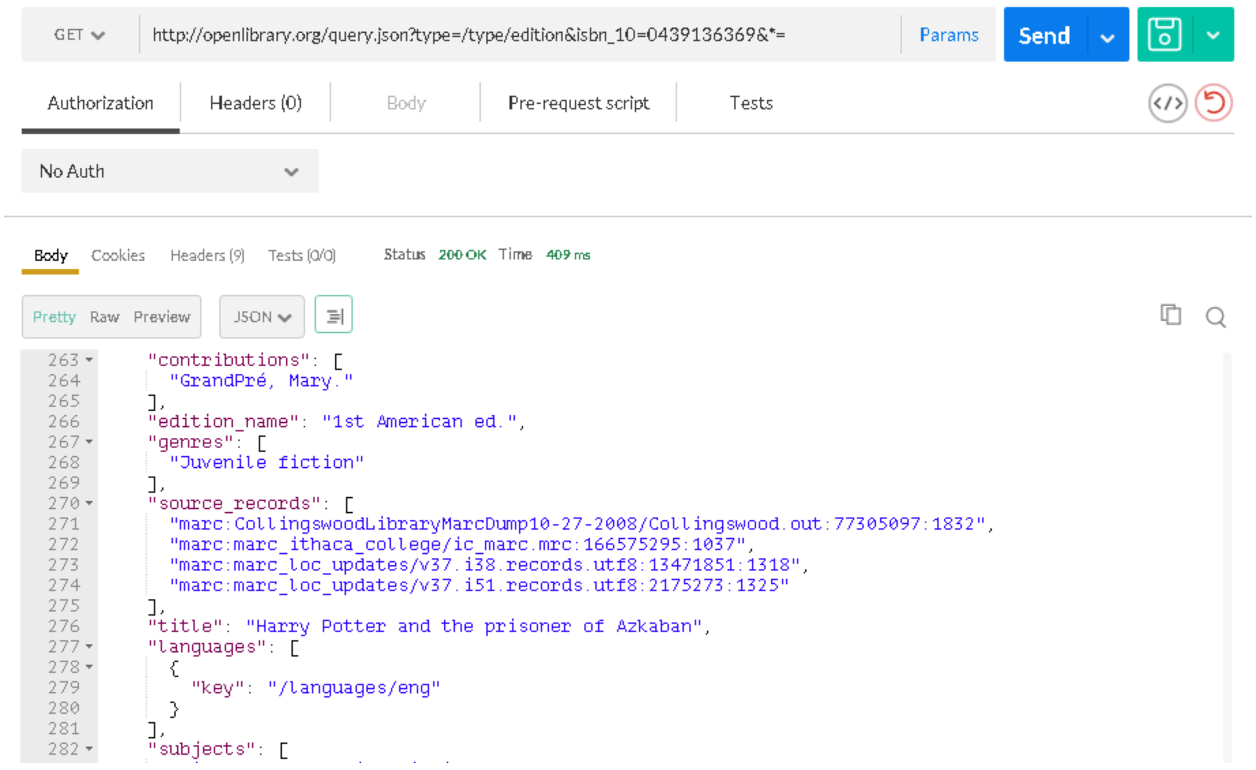
No Auth

Status 200 OK Time 289 ms

```
[
  {
    "lc_classifications": [
      "MLCSA 2004/01315 (P)",
      "PK2579.R255"
    ],
    "latest_revision": 3,
    "edition_name": "1st ed.",
    "title": "Prabandha prāmāṇika",
    "languages": [
      {
        "key": "/languages/ori"
      }
    ],
    "publish_country": "ii ",
    "by_statement": "Saccidānanda Rāutarāẏa.",
    "oclc_numbers": [
      "56367267"
    ],
    "type": {
      "key": "/type/edition"
    },
    "revision": 3,
    "publishers": [
```

Awesome! It looks like it returned a book and all its information based on the ISBN! Looks like we are on the right track. Note that the “type” parameter is required and the “*=” parameter indicates that we want to return all the information for that book. This is outlined in the API Guide.

Let’s look up the ISBN of another book and make sure it works as expected. I looked up the ISBN of a good Harry Potter book and found “0439136369”. Let’s plug that into Postman and see what happens.



http://openlibrary.org/query.json?type=/type/edition&isbn_10=0439136369&*=

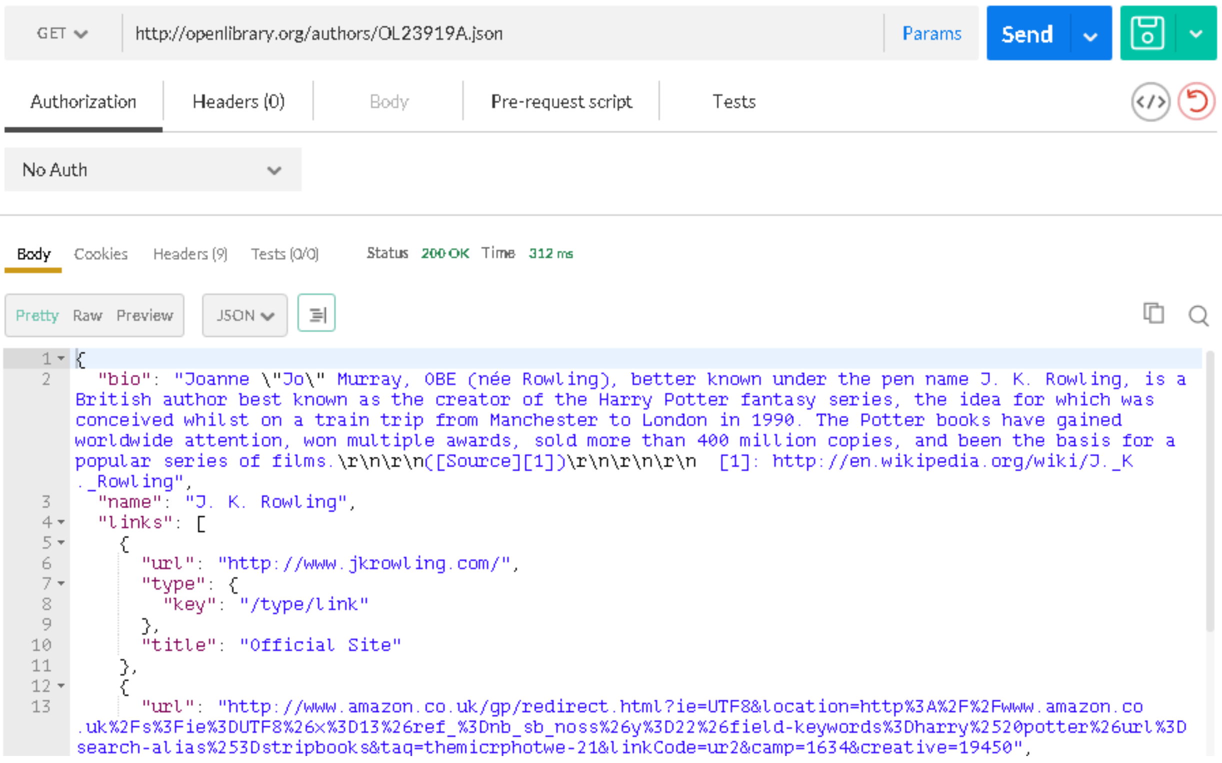
Status 200 OK Time 409 ms

```
    "contributions": [
      "GrandPré, Mary."
    ],
    "edition_name": "1st American ed.",
    "genres": [
      "Juvenile fiction"
    ],
    "source_records": [
      "marc:CollingswoodLibraryMarcDump10-27-2008/Collingswood.out:77305097:1832",
      "marc:marc_ithaca_college/ic_marc.mrc:166575295:1037",
      "marc:marc_loc_updates/v37.i38.records.utf8:13471851:1318",
      "marc:marc_loc_updates/v37.i51.records.utf8:2175273:1325"
    ],
    "title": "Harry Potter and the prisoner of Azkaban",
    "languages": [
      {
        "key": "/languages/eng"
      }
    ],
    "subjects": [
```

Excellent! This is what we want. Let's make sure we can get all the information we need. For our custom task we want to grab the title, description, genre, publisher, and author. As we go through the results, we find the title, description, genre, and publisher. However, the author information that is returned is not the author name, but the author ID. It looks like we will need to make another API call to return the author name based on the author ID.

As we go back to the Open Library API guide, we find another API call that looks up the author based on the author ID. It looks like the following API call returns the author information for the author ID of OL1A: http://openlibrary.org/authors/OL1A.json

Let's plug that into Postman and change the author ID to the Harry Potter author, which is "OL23919A".


GET http://openlibrary.org/authors/OL23919A.json

Authorization: No Auth

Status 200 OK Time 312 ms

```
{
  "bio": "Joanne \"Jo\" Murray, OBE (née Rowling), better known under the pen name J. K. Rowling, is a
British author best known as the creator of the Harry Potter fantasy series, the idea for which was
conceived whilst on a train trip from Manchester to London in 1990. The Potter books have gained
worldwide attention, won multiple awards, sold more than 400 million copies, and been the basis for a
popular series of films.\r\n\r\n([Source][1])\r\n\r\n\r\n  [1]: http://en.wikipedia.org/wiki/J._K
._Rowling",
  "name": "J. K. Rowling",
  "links": [
    {
      "url": "http://www.jkrowling.com/",
      "type": {
        "key": "/type/link"
      },
      "title": "Official Site"
    },
    {
      "url": "http://www.amazon.co.uk/gp/redirect.html?ie=UTF8&location=http%3A%2F%2Fwww.amazon.co
.uk%2Fs%3Fie%3DUTF8%26x%3D13%26ref_%3Dnb_sb_noss%26y%3D22%26field-keywords%3Dharry%2520potter%26url%3D
search-alias%253Dstripbooks&tag=themicrphotwe-21&linkCode=ur2&camp=1634&creative=19450",
```

Bingo! This API call will give us the author name based on the author ID, which is what we want.
So now we have two API calls that we will need to grab all the book information:

1. The first queries the Open Library database for a book with the given ISBN-10 and returns all the book information we need except for the author name.
   http://openlibrary.org/query.json?type=/type/edition&isbn_10=0439136369&*=

2. The second looks up the author information when given an author ID:
   http://openlibrary.org/authors/OL23919A.json

Now that we have our API calls and have tested them out in Postman, let's start building our UCS Director Custom Task.

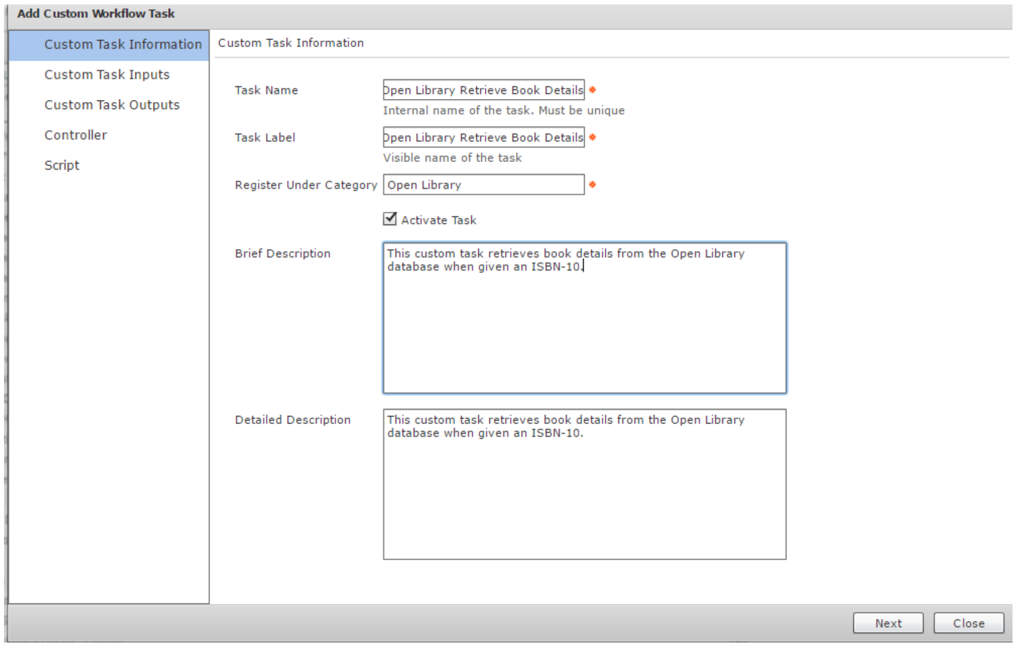
4.3 Create the UCS Director Custom Task

Now the fun begins with actually creating the task in UCS Director! First, let's think about how many tasks we want to create and what the inputs and outputs should be for each task. There are a couple different ways you can approach this:

1. Create two tasks with one API call for each task
2. Create a single task with both API calls in the same task

I prefer going with option 2 because even though there are two separate API calls being performed, we are logically just performing one task – retrieving book information from an ISBN. It is much simpler to keep everything together in one task rather than trying to make sure all your tasks are in the correct order in your workflow and making your workflow larger than it needs to be. If there were two different logical operations, then it would make sense to separate them into separate custom tasks.

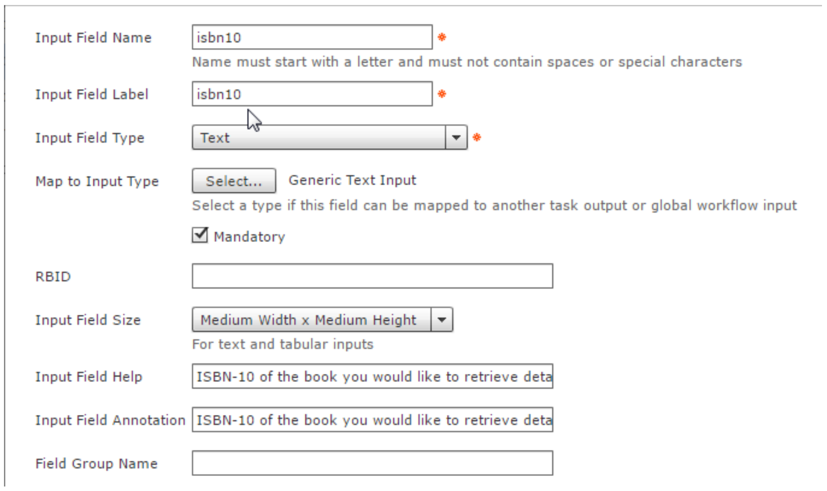
Let’s create a new task and call it “Open Library Retrieve Book Details”.



Custom Task Information

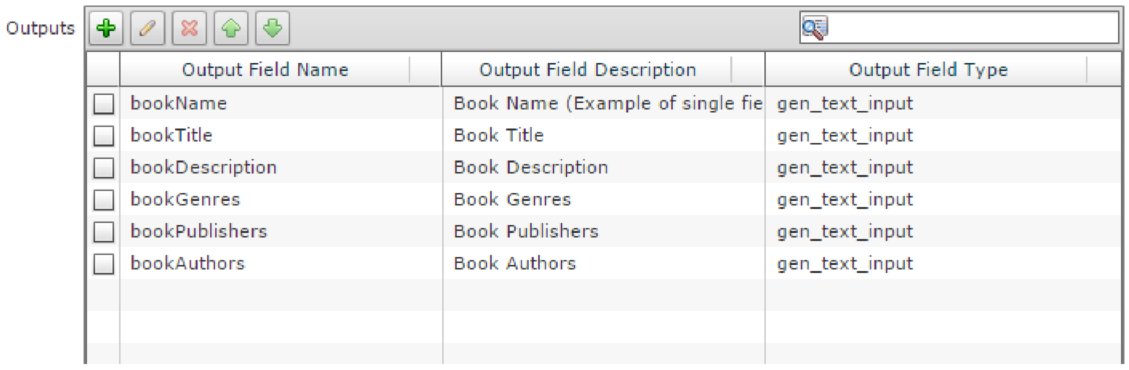
Task Name	Open Library Retrieve Book Details * Internal name of the task. Must be unique
Task Label	Open Library Retrieve Book Details * Visible name of the task
Register Under Category	Open Library *
	☑ Activate Task
Brief Description	This custom task retrieves book details from the Open Library database when given an ISBN-10.
Detailed Description	This custom task retrieves book details from the Open Library database when given an ISBN-10.

Let’s create a single input called “isbn10”:



Input Field Name	isbn10 * Name must start with a letter and must not contain spaces or special characters
Input Field Label	isbn10 *
Input Field Type	Text *
Map to Input Type	Select... Generic Text Input Select a type if this field can be mapped to another task output or global workflow input
	☑ Mandatory
RBID	
Input Field Size	Medium Width x Medium Height For text and tabular inputs
Input Field Help	ISBN-10 of the book you would like to retrieve deta
Input Field Annotation	ISBN-10 of the book you would like to retrieve deta
Field Group Name	

Now, let’s create the outputs called “bookName”, “bookTitle”, “bookDescription”, “bookGenres”, “bookPublishers”, and “bookAuthors”.


Outputs

	Output Field Name	Output Field Description	Output Field Type
☐	bookName	Book Name (Example of single fie	gen_text_input
☐	bookTitle	Book Title	gen_text_input
☐	bookDescription	Book Description	gen_text_input
☐	bookGenres	Book Genres	gen_text_input
☐	bookPublishers	Book Publishers	gen_text_input
☐	bookAuthors	Book Authors	gen_text_input

Go ahead and breeze through the next few screens as we will leave the controller and script sections blank for now.

4.4 Creating the Custom Task Code

When creating custom task code, I always prefer using a smart text editor like Notepad++ or similar since we can use some basic JavaScript code checking and color coding. I would then copy and paste the code into the custom task when I am ready.

4.4.1 Retrieve Name Function

Now let's create a very simply function that just retrieves the book title given an ISBN-10. We will start with this and expand from there.

```
function openlibraryRetrieveBookNameByIsbn10(isbn10) {
    logger.addInfo("function openlibraryRetrieveBookNameByIsbn10");

    // Setup HTTPS Connection
    var httpClient = new HttpClient();
    httpClient.getHostConfiguration().setHost("openlibrary.org", 443, "https");
    httpClient.getHostConfiguration().setProxy("proxy.test.com", 80);
    httpMethod = new GetMethod("/query.json?type=/type/edition&isbn_10="+isbn10+"&*=&limit=1");

    // Execute HTTPS and Return Results
    httpClient.executeMethod(httpMethod);
    statusCode = httpMethod.getStatusCode();
    response = httpMethod.getResponseBodyAsString();
    httpMethod.releaseConnection();

    // Check Response Code and Parse Results
    if (statusCode == 200){
        logger.addInfo("Response received: "+response);
        var jsonResponse = JSON.getJsonElement(response, null);

        // Make sure the query found a matching book.
        if (jsonResponse.size() == 0){
            logger.addError("Request Failed. The book search did not find any books with that ISBN-10.");
            ctxt.setFailed("Open Library book search failed.");
            ctxt.exit();
        }

        // Retrieve the Title from the JSON object
        var bookName = jsonResponse.get(0).get("title").getAsString();
        logger.addInfo("Book Name: " + bookName);
        return bookName;
    } else {
        // HTTP Request Failed
        logger.addError("Request Failed. HTTP response code: "+statusCode);
        logger.addError("Response = " + response);
        ctxt.setFailed("Open Library request failed.");
        ctxt.exit();
    }
} //function openlibraryRetrieveBookNameByIsbn10
```

Now let’s break this function down a little bit. This function accepts a parameter appropriately called isbn10 and returns the title of the book.

First we setup the HTTP connection, set the host, set the proxy, and set the GET method. We will want to substitute the GET query with the correct isbn10 that we feed into the function, so I added a variable there.

Now let’s execute HTTPS call and return the response. The response will return either with an exit code of 200 and the unparsed JSON results, or will return with some other code if there is an issue with the response. Let’s capture the response and close the HTTP connection.

If the response code is 200, that means the request was successful and we should see some results if the Open Library query actually returned anything. If the response code is anything other than 200, we want to fail the task. To do this, we create an if/else statement that checks the response code.

If the code is 200, we will log the response for troubleshooting purposes so we can see the unparsed JSON response. Then we will parse the JSON response looking for the title. I’ll touch on JSON parsing in a later section. Once the JSON is parsed and we pull out the title, we can assign it to a variable and return it to close out the function. If the JSON response resulted in no books, we will fail the task.

This task retrieves and returns a single item from the function, so is a pretty straightforward and easy use case for returning results. The next function will return the book title as well, but will also return all the other book information as properties in a JavaScript object.

4.4.2 Retrieve Details Function

Now let’s create a more complex function that returns all of the book details and converts all the details to properties in a JavaScript object. This may be useful in organizing a large amount of information from a REST call. This function accepts the isbn10 as a parameter and returns a single JavaScript object with properties for title, description, genres, publishers, and author keys. In addition, there might be more than one genre, publisher, or author key for each book, so the relevant properties might be an array of values.

```
function openlibraryRetrieveBookDetailsByIsbn10(isbn10) {
    logger.addInfo("function openlibraryRetrieveBookDetailsByIsbn10");

    // Setup HTTPS Connection
    var httpClient = new HttpClient();
    httpClient.getHostConfiguration().setHost("openlibrary.org", 443, "https");
    httpClient.getHostConfiguration().setProxy("proxy.test.com", 80);
    httpMethod = new GetMethod("/query.json?type=/type/edition&isbn_10="+isbn10+"&*=&limit=1");

    // Execute HTTPS and Return Results
    httpClient.executeMethod(httpMethod);
    statusCode = httpMethod.getStatusCode();
    response = httpMethod.getResponseBodyAsString();
    httpMethod.releaseConnection();

    // Check Response Code and Parse Results
    if (statusCode == 200){
        logger.addInfo("Response received: "+response);
        var jsonResponse = JSON.getJsonElement(response, null);
        // Make sure the query found a matching book.
        if (jsonResponse.size() == 0){
            logger.addError("Request Failed. The book search did not find any books with that ISBN-10.");
            ctxt.setFailed("Open Library book search failed.");
            ctxt.exit();
        }

        // Retrieve the Details from the JSON object
        var bookDetails = {};

        // Book Title
        bookDetails.title = jsonResponse.get(0).get("title").getAsString();
        logger.addInfo("Book Title: " + bookDetails.title);

        // Book Description
        bookDetails.description = jsonResponse.get(0).get("description").getAsString();
        logger.addInfo("Book Description: " + bookDetails.description);

        // Book Genres - Might be multiple genres, so will create an array
        bookDetails.genres = [];
        for (var i = 0; i < jsonResponse.get(0).get("genres").size(); i++){
            bookDetails.genres.push(jsonResponse.get(0).get("genres").get(i).getAsString());
        }
        logger.addInfo("Book Genres: " + bookDetails.genres.toString());

        // Book Publishers - Might be multiple publishers, so will create an array
        bookDetails.publishers = [];
        for (var i = 0; i < jsonResponse.get(0).get("publishers").size(); i++){
            bookDetails.publishers.push(jsonResponse.get(0).get("publishers").get(i).getAsString());
        }
        logger.addInfo("Book Publishers: " + bookDetails.publishers.toString());

        // Author Keys - Might be multiple author keys, so will create an array
        bookDetails.authorKeys = [];
        for (var i = 0; i < jsonResponse.get(0).get("authors").size(); i++){
            bookDetails.authorKeys.push(jsonResponse.get(0).get("authors").get(i).get("key").getAsString());
        }
        logger.addInfo("Author Keys: " + bookDetails.authorKeys.toString());

        // Return the bookDetails Object
        return bookDetails;

    } else {
        // HTTP Request Failed
        logger.addError("Request Failed. HTTP response code: "+statusCode);
        logger.addError("Response = " + response);
        ctxt.setFailed("Open Library request failed.");
        ctxt.exit();
    }

} //function openlibraryRetrieveBookDetailsByIsbn10
```

Notice that this function is very similar to the first function, except how we parse the JSON and save the results. We will create a “bookDetails” object with the properties title, description, genres, publishers, and authorKeys. For the title and description properties, there can only ever be one item so we don’t have to do anything special. For the genres, publishers, and authorKeys properties, there could be multiple items, so we will need to create a for loop and iterate through all the items and push the item to the array within the genres, publishers, and authorKeys properties. We then return the “bookDetails” object to close out the function.

4.4.3 Retrieve Author Name Function

Now let’s create a function that accepts the author key as a parameter and returns the author’s name. This function utilizes a different API call, but parses the response in the same way as the other functions.

```
function openlibraryRetrieveAuthorNameByAuthorKey(authorKey) {
    logger.addInfo("function openlibraryRetrieveAuthorNameByAuthorKey");

    // Setup HTTPS Connection
    var httpClient = new HttpClient();
    httpClient.getHostConfiguration().setHost("openlibrary.org", 443, "https");
    httpClient.getHostConfiguration().setProxy("proxy.test.com", 80);
    httpMethod = new GetMethod(authorKey+".json");

    // Execute HTTPS and Return Results
    httpClient.executeMethod(httpMethod);
    statusCode = httpMethod.getStatusCode();
    response = httpMethod.getResponseBodyAsString();
    httpMethod.releaseConnection();

    // Check Response Code and Parse Results
    if (statusCode == 200){
        logger.addInfo("Response received: "+response);
        var jsonResponse = JSON.getJsonElement(response, null);
        // Retrieve the author name from the JSON object
        var authorName = jsonResponse.get("name").getAsString();
        logger.addInfo("Author Name: " + authorName);
        return authorName;
    } else {
        // HTTP Request Failed
        logger.addError("Request Failed. HTTP response code: "+statusCode);
        logger.addError("Response = " + response);
        ctxt.setFailed("Open Library request failed.");
    }

} //function openlibraryRetrieveAuthorNameByAuthorKey
```

This function is pretty straightforward and closely resembles some of the other functions we already discussed.

4.4.4 Main Function

This is where all the functions get tied together. Our main function will pull in the custom task input "isbn10", execute the necessary functions, and set the custom task outputs bookName, bookTitle, bookDescription, bookGenres, bookPublishers, and bookAuthors.

```
// Task Inputs Mappings
var isbn10 = input.isbn10;

// Retrieve Book Name By ISBN-10 (Simple example of retrieving a single item)
var bookName = openlibraryRetrieveBookNameByIsbn10(isbn10);

// Retrieve Book Details By ISBN-10 (Retries the bookDetails object with the details stored in
properties)
var bookDetails = openlibraryRetrieveBookDetailsByIsbn10(isbn10);

// Retrieve Author Names by Author Keys (Because the above method only returned the author key -
not the name)
var authorNames = [];
for (var i = 0; i < bookDetails.authorKeys.length; i++){
    authorNames.push(openlibraryRetrieveAuthorNameByAuthorKey(bookDetails.authorKeys[i]));
}

// Task Output Mappings
output.bookName = bookName;
output.bookTitle = bookDetails.title;
output.bookDescription = bookDetails.description;
output.bookGenres = bookDetails.genres.toString();
output.bookPublishers = bookDetails.publishers.toString();
output.bookAuthors = authorNames.toString();
```

The book title from the “openlibraryRetrieveBookNameByIsbn10” function is saved into the bookName variable. This variable is not necessarily needed as the book title is also returned from the bookDetails object, but this variable and function just serves as an example on how to parse and handle single items.

The book details object from the “openlibraryRetrieveBookDetailsByIsbn10” function is saved into the bookDetails variable. The properties of this object are then mapped to their respective outputs.
To retrieve the author names, the author keys are retrieved from the bookDetails object and sent as a parameter to the “openlibraryRetrieveAuthorNameByAuthorKey” function. A for loop is utilized in the event there is more than one author key. The author names are then mapped to an output.

4.5 JSON Parsing Fun

The JavaScript Object Notation(JSON) is a standard way of formatting data and is used often in REST or other APIs. It’s fairly easy to use and parse in UCS Director custom tasks once you play around with it for a little bit. Here is a quick summary of how JSON is formatted:

- All JSON data is formatted in name/value pairs like the following:
```
 “author”:”John Doe”
```

- Multiple pieces of data are separated by commas like the following:
```
“title”:”The Great JSON Book”,“author”:”John Doe”,”publisher”:”John Doe Publishing”
```

- JSON objects are held within curly braces:
```
// Book JSON Object
{
       “title”:”The Great JSON Book”,
       “author”:”John Doe”,
       ”publisher”:”John Doe Publishing”
}
```

- JSON arrays are held within square brackets:
```
// Multiple Book JSON Objects
[
    {
        “title”:”The Great JSON Book”,
        “author”:”John Doe”,
        ”publishers”:”John Doe Publishing”
    },
    {
        “title”:”The Great JSON Book: Part Two”,
        “author”:”John Doe”,
        ”publishers”:[”John Doe Publishing”,"Jane Doe Publishing"]
    }
}
```

So how do you navigate a JSON object and pull out information that you want? You start with the JSON class that comes with UCS Director and parse the HTTP response as JSON:

```
response = httpMethod.getResponseBodyAsString();
var jsonResponse = JSON.getJsonElement(response, null);
```

Once the JSON is parsed as a JSON element you can navigate the JSON object using the get() method. To navigate JSON arrays, you use the get() method with an array id (0,1,2,3,etc). To pull out a value from a JSON object’s name/value pair, you use the get() method with the name of the value you would like to retrieve.

If we wanted to pull out the first publisher of the first book in the JSON array example listed above, we would use the following code:

```
var publisher1 = jsonResponse.get(0).get("publishers").get(0).getAsString();
```

If we wanted to pull out the title of the second book, we would use the following code:

```
var title2 = jsonResponse.get(1).get("title").getAsString();
```

If we wanted to retrieve all the publisher names in the second book, we can create a for loop and add the names to an array like the following:

```
var publisherNames = [];
for (var i = 0; i < jsonResponse.get(1).get("publishers").size(); i++){
        publisherNames.push(jsonResponse.get(1).get("publishers").get(i).getAsString());
}
```

So when you are working with a JSON response from an API and you need to figure out how to parse it and pull out the information you need, the best approach would be to make the API call in Postman, look at the structure of the response, and figure out how to navigate the JSON objects and arrays using the get() method.

For more information on JSON, you can look at some of the following sites:

- https://tools.ietf.org/html/rfc7159
- http://www.w3schools.com/json/
- https://en.wikipedia.org/wiki/JSON

4.6 Troubleshooting Your Custom Task

It’s very rare that you type out all your code in your favorite text editor, copy/paste it into the UCSD custom task, run it, and it works perfectly. Usually you run into trouble with a typo, an invalid HTTP request, or a JSON parsing misstep. For situations like this, I like to have two UCSD windows open – one on the workflow tab so I can execute a test workflow with my custom task, and the second window on the custom task tab so I can make any corrections or updates to my custom task without having to navigate back and forth.

Once you get your UCSD windows situated, double check your custom task code and make sure you are logging the HTTP response and status code. If the status code is not 200, you can look up the meaning of the status code and read the response in the log for clues as to what went wrong. If there is a JSON parsing error, usually your task will fail with an error about a null reference or something similar and will point you to the line of code with the issue. Double-check your parsing and get() methods to make sure you are navigating the JSON object correctly. If you are logging your HTTP response, you should be able to compare your parsing code with the actual JSON response in the logs to make sure you are navigating it correctly.

5 Dealing with SSL Certificates

5.1 Potential SSL Certificate Issues

When dealing with integrations using HTTPS (you should never use just HTTP when dealing with credentials or auth tokens), you may encounter some errors when attempting to connect. If your HTTPS task fails and you see an error in the log that mentions "SSLHandshakeException" or something about a certificate path, you are most likely running into a SSL certificate issue. This could be due to a number of reasons:

1. You are integrating with an internal system that uses a self-signed certificate and that certificate is not in the trusted certificate authority keystore in UCS Director.
2. You are integrating with an internal system that is using an invalid certificate. Some out-of-the-box certificates for systems use an invalid self-signed certificate that has a default common name instead of the server name as the common name. The common name attribute of the certificate should be the hostname.
3. You are trying to connect to an internal system using a self-signed certificate and the hostname you are connecting to is different than the certificate's common name. You should always connect to the hostname that is the same as the certificate's common name attribute or you may encounter a certificate error.
4. The system you are integrating with is not using a CA signed certificate from one of the more common Certificate Authorities. UCSD comes pre-installed with the most common Certificate Authorities. Typically, public internet facing systems would have a CA signed certificate.

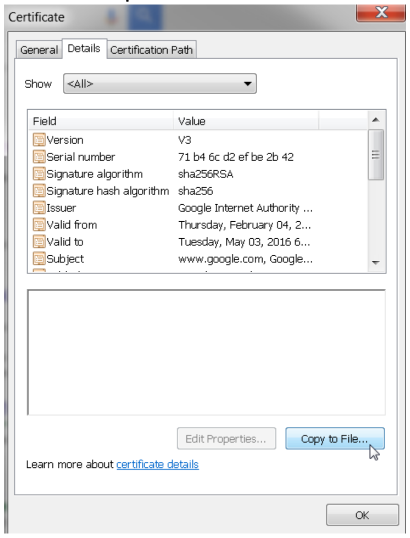
In the event of an invalid certificate, you should go to the system and regenerate a valid self-signed certificate. Once that is complete, you will need to add the certificate to the UCS Director Certificate Authority keystore.

5.2 Adding Certificates to the UCSD Trusted CA Keystore

If your certificate authority or self-signed certificate is not already in the UCSD CA keystore and you are encountering a certificate issue, you may need to add your certificate to the keystore. Here is the process on how to do that:

1. Retrieve the certificate of the system you are integrating with. For self-signed certificates, you can use your browser to save the certificate. The format should be Base-64 encoded for the

easiest import.



Field	Value
Version	V3
Serial number	71 b4 6c d2 ef be 2b 42
Signature algorithm	sha256RSA
Signature hash algorithm	sha256
Issuer	Google Internet Authority ...
Valid from	Thursday, February 04, 2...
Valid to	Tuesday, May 03, 2016 6...
Subject	www.google.com, Google...

Learn more about certificate details

2. Copy the certificate file to the UCSD primary node's /tmp directory
3. Login to the UCSD primary node as root
4. Take a backup of the cacerts file
```
# cp /opt/bin/jre/lib/security/cacerts /cacerts.bck
```
5. Use the keytool to import the certificate
```
#cd /opt/bin/jre/bin
#./keytool -import -trustcacerts -alias <hostname> -file /tmp/certfile.cer -keystore
/opt/bin/jre/lib/security/cacerts
```
6. If prompted, the default password for the cacerts keystore file is "changeit" if you have not already changed it.
7. Restart UCSD

6 Creating Your Own Custom Integrations

You have now completed your own custom RESTful integration with Open Library! Obviously, you may not find much use for an integration with Open Library in your data center environment, but hopefully this guide gave you a foundation to work from and a good template to use for your own custom RESTful integration with an actual system in your data center. As a part of this guide, you will find attached the completed Open Library custom task you can use as a starting point for your own integration. You can modify, add, or remove pieces based on your integration needs.

Stay tuned for some additional guides around more complex RESTful integrations, SOAP integrations, and advanced SSH integrations to round out your tool kit for integrating with various systems in the datacenter. If you have some additional ideas or thoughts on some UCS Director guides that may be useful, feel free to reach out and let me know!