Using an API: a hands on exercise

This Introduction to APIs was developed by Owen Stephens (owen@ostephens.com) on behalf of the British Library.

Exercise 1: Using an API for the first time

Introduction

In this exercise you are going to use a Google Spreadsheet to retrieve records from an API to Flickr, and display the results.

The API you are going to use simply allows you to submit some search terms and get a list of results in a format called RSS. You are going to use a spreadsheet to submit a search to the API, and display the results.

Understanding the API

The API you are going to use is an interface to Flickr. Flickr has a very powerful API with lots of functions, but for simplicity in this exercise you are just going to use the Flickr RSS feeds, rather than the full API which is more complex and requires you to register.

Before you can start working with the API, you need to understand how it works. To do this, we are going to look at an example URL:

https://api.flickr.com/services/feeds/photos_public.gne?tags=food&format=rss

The first part of the URL is the address of the API. Everything after the '?' are 'parameters' which form the input to the API. There are two parameters listed and they each consist of the parameter name, followed by an '=' sign, then a value.

The URL and parameters breakdown like this:

URL Part	Explanation
https://api.flickr.com/services/feeds/photos_public.gne	The address of the API
tags=food	The 'tags parameter – contains a list of tags (separated by commas) to be used to filter the list of images returned by the API. In this case just the single tag 'food' is listed.
format=rss	format=rss

Going Further

If you want to find out more about the API being used here, documentation is available at:

https://www.flickr.com/services/feeds/

There are 8 different types of feed available which are documented here. Click on each one to see what parameters it takes.

The output of the API is displayed in the browser – this is an RSS feed – it would plug into any standard RSS reader. It is also valid XML.

While the XML is not the nicest thing to look at, it should be possible to find lines that look something like:

```
Italian Crostata with cherry jam VI
```

```
https://www.flickr.com/photos/91554579@N02/13058384463/ <p><a
href="https://www.flickr.com/people/91554579@N02/">Lili B. Capaccetti</a>
posted a photo:</p>

<p><a href="https://www.flickr.com/photos/91554579@N02/13058384463/"
title="Italian Crostata with cherry jam VI"><img
src="https://farm3.staticflickr.com/2158/13058384463_b3a0416677_m.jpg"
width="160" height="240" alt="Italian Crostata with cherry jam VI"
/></a></p>
```

Each result the API returns is called an ‘item’. Each ‘item’ at minimum will have a ‘title’ and a ‘link’. In this case the link is to the image in the Flickr interface.

The key things you need to know to work with this API are:

- The address of the API
- The parameters that the API accepts as input
- The format(s) the API provides as output

Now you’ve got this information, you are ready to start using the API.

Using the API

To use the API, you are going to use a Google Spreadsheet. Go to http://drive.google.com and login to your Google account. Create a Google Spreadsheet

The first thing to do is build the API call (the query you are going to submit to the API).

First some labels:

In cell A1 enter the text ‘API Address’

In cell A2 enter the text ‘Tags’

In cell A3 enter the text ‘Format’

In cell A4 enter ‘API Call’

In cell A5 enter ‘Results’

Now, based on the information we were able to obtain by understanding the API we can fill values into column B as follows:

In cell B1 enter the address of the API

In cell B2 enter a simple, one word tag

In cell B3 enter the text ‘rss’ (omitting the inverted commas)

The first three rows of the spreadsheet should look something like (with whatever tag you’ve chased to search in B2):

	A	B
1	API Address	https://api.flickr.com/services/feeds/photos_public.gne
2	Tags	food
3	Format	rss

You now have all the parameters we need to build the API call. To do this you want to create a URL very similar to the one you looked at above. You can do this using a handy spreadsheet function/formula called ‘Concatenate’ which allows you to combine the contents of a number of spreadsheet cells with other text.

In Cell B4 type the following formula:

=concatenate(B1,“?”,“tags=”,B2,“&format=”,B3) - **Note: you may have to retype the quotations marks to avoid a parsing error.**

This joins the contents of cells B1, B2 with the text included in inverted commas in formula. Once you have entered this formula and pressed enter your spreadsheet should look like:

	A	B
1	API Address	https://api.flickr.com/services/feeds/photos_public.gne
2	Tags	food
3	Format	rss
4	API Call	https://api.flickr.com/services/feeds/photos_public.gne?tags=food&format=rss

The final step is to send this query, and retrieve and display the results. This is where the fact that the API returns results as an RSS feed comes in extremely useful. Google Spreadsheets has a special function for retrieving and displaying RSS feeds.

To use this, in Cell B5 type the following formula:

=importFeed(B4)

Because Google Spreadsheets knows what an RSS feed is, and understands it will contain one or more 'items' with a 'title' and a 'link' it will do the rest for us. Hit enter, and see the results.

Congratulations! You have built an API query, and displayed the results.

You have:

* Explored an API for Flickr

* Seen how you can 'call' the API by adding some parameters to a URL

* Understood how the API returns results in RSS format

* Used this knowledge to build a Google Spreadsheet which searches for a tag on Flickr and displays the results

Going Further

Further parameters that this API accepts are:

- id
- ids
- tagmode
- format
- lang

These are documented at https://www.flickr.com/services/feeds/docs/photos_public/. When adding parameters to a URL, you use the '&' sign between each parameter e.g.

https://api.flickr.com/services/feeds/photos_public.gne?tags=food&id=23577728@N07

This searches for all photos tagged with 'food' from a specific user (user id = 23577728@N07)

By adding a row to the spreadsheet, for this parameter, and modifying the 'concatenate' statement that builds the API Call, can you make the spreadsheet only return images with a specific tag in the British Library Flickr collection? (The Flickr ID for the British Library is '12403504@N02')

If you want to know more about the 'importFeed' function, have a look at the documentation at http://support.google.com/drive/bin/answer.py?hl=en&answer=155181

Exercise 2: Working with the BNB

Introduction

In Exercise 1, you explored a simple API displayed the results from an RSS feed. However, an RSS feed contains only minimal information (a result title, a link and a description) and may not tell you a lot about the resource. In this exercise you will see

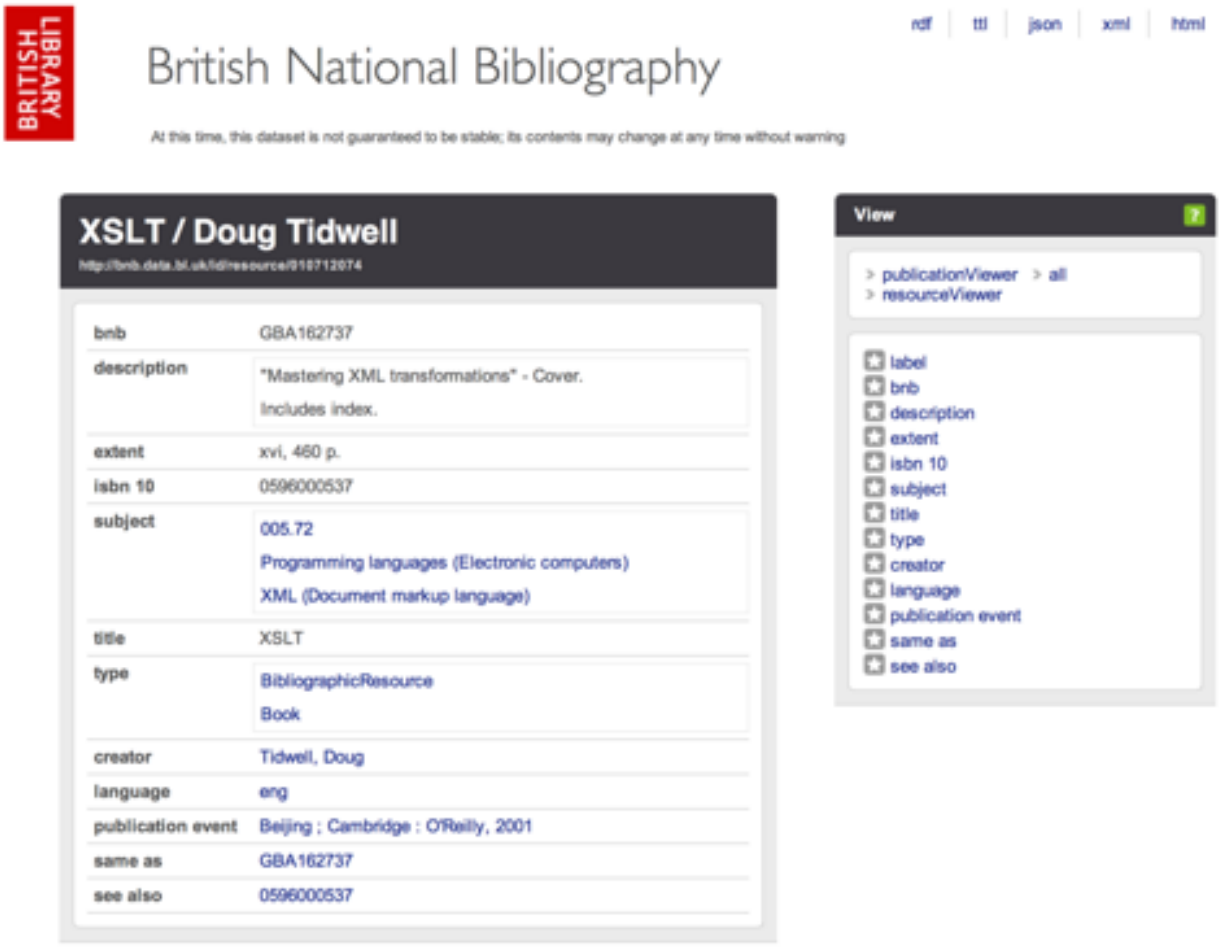
how to deal with more complex data structures by retrieving items from the BNB and extracting information from the XML data.

Exploring the full record data

For this exercise you are going to work with a ‘full record display’ for books from the BNB.

An example URL is http://bnb.data.bl.uk/id/resource/010712074

Following this URL will show a page similar to this:



This screen displays the information about this item which is available via the BNB API as an HTML page. Note that the URL of the page in the browser address bar is different to the one you clicked on. In the example given here the original URL was:

http://bnb.data.bl.uk/id/resource/010712074

while the address in the browser bar is:

http://bnb.data.bl.uk/doc/resource/010712074

You will be able to take advantage of the equivalence of these two URLs later in this exercise.

While the HTML display works well for humans, it is not always easy to automatically extract data from HTML. In this case the same information is available in a number of different formats, listed at the top righthand side of the display. The options are:

- rdf
- ttl
- json
- xml
- html

The default view in a browser is the ‘html’ version. Offering access to the data in a variety of formats gives choice to anyone working in the API. Both ‘json’ and ‘xml’ are widely used by developers, with ‘json’ often being praised for its simplicity. However, the choice of format can depend on experience, the required outcome, and external constraints such as the programming language or tool being used.

Google Spreadsheet has some built in functions for reading XML, so for this exercise the XML format is the easiest one to use.

- XML for BNB items
  To see what the XML version of the data looks like, click on the ‘xml’ link at the top right. Note the URL looks like:

http://bnb.data.bl.uk/doc/resource/010712074.xml

This is the same as the URL we saw for the HTML version above, but with the addition of '.xml'

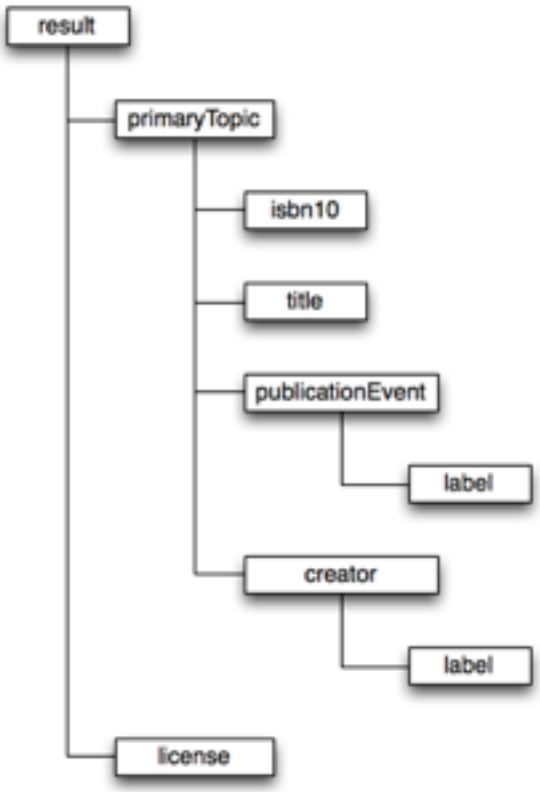
XML is a way of structuring data in a hierarchical way – one way of thinking about it is as a series of folders, each of which can contain further folders. In XML terminology, these are 'elements' and each element can contain a value, or further elements (not both). If you look at an XML file, the elements are denoted by tags – that is the element name in angle brackets – just as in HTML. Every XML document must have a single root element that contains the rest of the XML.

Going Further

To learn more about XML, how it is structured and how it can be used see this tutorial from IBM:

http://www.ibm.com/developerworks/xml/tutorials/xmlintro/

Can you guess another URL which would also get you the XML version of the BNB record? Look at the URL in the spreadsheet and compare it to the URL you actually arrive at if you follow the link.The structure of the XML returned by the BNB API has a <result> element as the root element. The diagram below partially illustrates the structure of the XML.



To extract data from the XML we have to ‘parse’ it – that is, tell a computer how to extract data from this structure. One way of doing this is using ‘XPath’. XPath is a way of writing down a route to data in an XML document.

The simplest type of XPath expression is to list all the elements that are in the ‘path’ to the data you want to extract using a ‘/’ to separate the list of elements. This is similar to how ‘paths’ to documents are listed in a file system.

In the document structure above, the XPath to the title is:

/result/primaryTopic/title

You can use a shorthand of ‘//’ at the start of an XPath expression to mean ‘any path’ and so in this case you could simply write ‘//title’ without needing to express all the container elements.

Going Further

What would the XPath be for the ISBN–10 in this example?

Why might you sometimes not want to use the shorthand '//' for 'any path' instead of writing the path out in full? Can you think of any possible undesired side effects?

Find out more about XPath in this tutorial:

http://zvon.org/comp/r/tut-XPath_1.html

Using the API

Now you know how to get structured data for a BNB item, and the structure of the XML used, given a list of URLs for books in the BNB, you can create a spreadsheet to retrieve and display information about each of the books.

Google Spreadsheets has a function called 'importXML' which can be used to import XML, and then use XPath to extract the relevant data. In order to use this you need to know the location of the XML to import, and the XPath expression you want to use.

Create a new Google Spreadsheet, and in Column A paste the following list of URLs from the BNB (with the first URL in cell A1):

http://bnb.data.bl.uk/id/resource/009406660

http://bnb.data.bl.uk/id/resource/010055357

http://bnb.data.bl.uk/id/resource/009406743

http://bnb.data.bl.uk/id/resource/010053535

http://bnb.data.bl.uk/id/resource/008418912

http://bnb.data.bl.uk/id/resource/012702152

http://bnb.data.bl.uk/id/resource/009406658

http://bnb.data.bl.uk/id/resource/009097698

http://bnb.data.bl.uk/id/resource/010975194

In order to use this list of URLs to retrieve the XML versions of the records, you'll need to add '.xml' onto the end of each of them.

The XPath expression you can use is '//isbn10'. This will find all the isbn10 elements in the XML.

With these two bits of information you are ready to use the 'importXML' function. In to Cell B1, type the formula:

=importXml(concatenate(A1,".xml"),"//ISBN10") - **Note: you may have to retype the quotations marks to avoid a parsing error.**

This creates the correct URL with the 'concatenate' function, retrieves the XML document, and uses the Xpath '//isbn10' to get the content of the element – this 10 digit ISBN.

Congratulations! You have used the BNB API to retrieve XML and extract and display information from it.

You have:

* Understood the URLs you can use to retrieve a full record from the BNB

* Understood the XML used to represent the BNB record

* Written a basic XPath expression to extract information from the BNB record