Using an API - 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	The API can return data in different formats. In this case the ‘rss’ format is selected.

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:

```
<item>
     <title>Italian Crostata with cherry jam VI</title>
     <link>https://www.flickr.com/photos/
91554579@N02/13058384463/</link>
     <description>          &lt;p&gt;&lt;a href=&quot;https://
www.flickr.com/people/91554579@N02/&quot;&gt;Lili B.
Capaccetti&lt;/a&gt; posted a photo:&lt;/p&gt;

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

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).

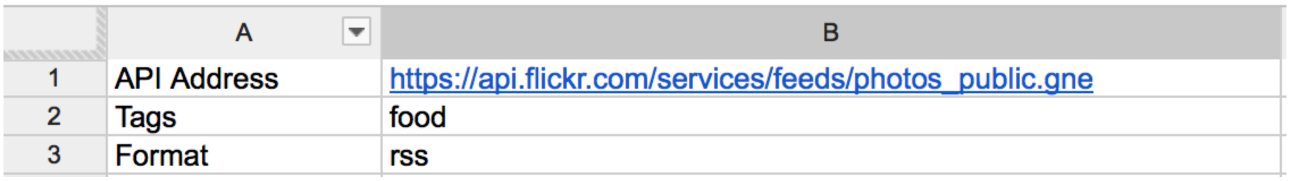
First some labels:

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):



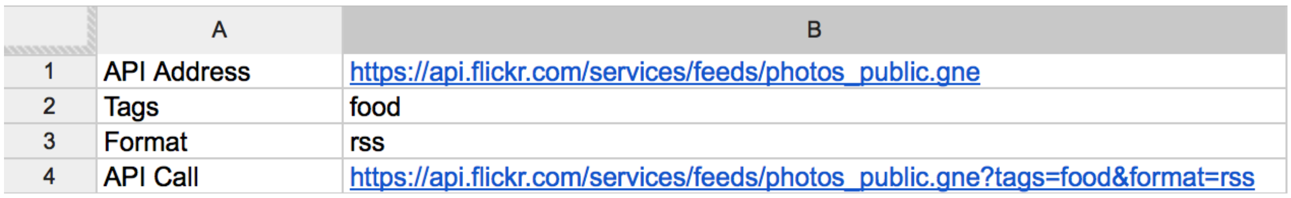
	A	B
1	API Address	https://api.flickr.com/services/feeds/photos_public.gne
2	Tags	food
3	Format	rss

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)
```

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: Searching the BNB using the web (HTML) interface

Introduction

In Exercise 1, you explored a simple, well structured, API and displayed the results from an RSS feed. This is a good example to start with as it demonstrates some of the key aspects of interacting with APIs.

However, sometimes we need to use data sources that don't have such well structured APIs, and we may have to work around the limitations of the API provided. Sometimes there won't be a formal API at all, but only a web interface from which we want to extract data.

In this exercise we will explore using a very simple web based search of the British National Bibliography (BNB) as an API. Once again this exercise will use Google Spreadsheets to submit a search to the API, and display the results.

Understanding the Search interface

In this case rather than using a formal API, you are going to use a web interface to search the BNB. In fact, a web interface is a kind of very simple API - if you send the right kind of requests to the web page and you get back an HTML document. This is usually done via a web browser - you type in a URL, or click on a link to send a request, and what you get is the HTML document displayed in your browser.

However in this exercise we are going to use Google Spreadsheets to send the request, and to extract some specific information from the HTML document we get back, rather than displaying the whole web page.

As with any API, before you can start working a web page in this way, you need to understand how it works. To do this, we are going to look at the relevant web page in a web browser:

http://bnb.data.bl.uk

If you visit this address in your browser, you'll see there is a search option. Type a search in the box and press the "Search" button and then look at the URL in the browser bar. For example if we use the search term "Tim Berners-Lee" and click 'Search', we'll see the URL:

http://bnb.data.bl.uk/search?object=tim+berners-lee

As in Exercise 1 we can see this URL is made up of a base address (everything before the '?') and then some parameters (in this case a single parameter which has the value of the search term you've typed in).

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 parameter breakdown like this:

URL Part	Explanation
http://bnb.data.bl.uk/search	The address of the API
object=tim+berners-lee	The 'object' parameter contains the search term you have entered modified to ensure that there are no invalid characters in the URL (in this example the space between 'tim' and 'berners-lee' has been replaced with a '+' character)

In more complex search interfaces we might see many more parameters, but in this case there is only a single parameter 'object' which simply contains your search term.

Going Further
Try carrying out a similar exercise using the COPAC service at http://copac.ac.uk/search.

What parameters are used here? Can you work out which parameters are used to do a search for a specific author or specific title?

Similarly look at the parameters used when doing a search from the main British Library Catalogue at http://explore.bl.uk. Can you work out what the parameters do?

The output of the search is displayed in the browser - this is just a normal web page. In the middle of the page you should see a list of the results found by your search. In this case a maximum of 10 results are displayed on the page. If there are more than 10 results available then you will see an option to view additional pages of results.

Because we want to be able to display these results in a spreadsheet, we need to understand a bit more about how the results are displayed in the page. To do this we need to look at the HTML. Most browsers have a 'view source' option which allows you to view the HTML for any page you are looking at. Find the 'view source' option for your browser and use it to view the HTML for this page.

The part of the page we are interested in is where the list of results is displayed. Find this in the HTML page - it will look something like:

```
<p class="search-result">
     <a href="http://bnb.data.bl.uk/id/resource/016891990">
     Tim Berners-Lee / Damian Harvey
     </a>
</p>
<p class="search-result">
     <a href="http://bnb.data.bl.uk/id/resource/010733921">
```

```
    Weaving the Web / Tim Berners-Lee
    </a>
</p>
<p class="search-result">
    <a href="http://bnb.data.bl.uk/id/resource/010758360">
    Weaving the Web / Tim Berners-Lee
    </a>
</p>
```

Each result in the page is inside an HTML paragraph tag (the <p>) with a 'class' of 'search-result'. In each case the result consists of a hyperlink (an <a> tag with a URL and some text).

The key things you need to know to work with a web page as an API are:

- The address of the web page
- The parameters (if any) that the web page accepts as input
- How the data you want is included in the page in terms of the structure of the relevant HTML

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

Using the web page as an API

To use the web page as an 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 search you are going to submit to the web page

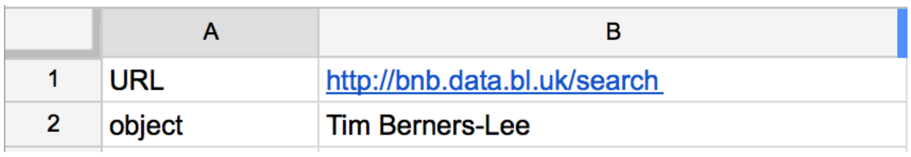
First some labels:

In cell A1 enter the text 'URL'
In cell A2 enter the text 'object'
In cell A3 enter 'BNB Search'
In cell A4 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 URL of the BNB search interface (http://bnb.data.bl.uk/search)
In cell B2 enter a search term. A good example to start with is "Tim Berners-Lee".

The first two rows of the spreadsheet should look something like:



	A	B
1	URL	http://bnb.data.bl.uk/search
2	object	Tim Berners-Lee

As with the Flickr example above, we now use these lines to build a URL which will carryout the search. You do this using the 'Concatenate' function which allows you to combine the contents of a number of spreadsheet cells with other text.

In Cell B3 type the following formula:

```
=concatenate(B1,"?","object=",substitute(B2," ","+"))
```

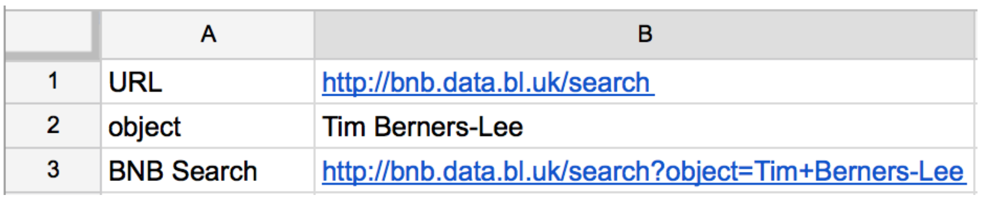
This builds the URL we saw in the browser bar when we did a search from the BNB web page:

http://bnb.data.bl.uk/search?object=Tim+Berners-Lee

The 'substitute' function in cell B3 is used to replace the space in the search term with a '+' sign, just as we saw in the web interface. Other substitutions might also be needed if you are including characters in the search term that are not allowed in a URL.

Going Further
Try using different characters in the BNB Web search and see how they are transformed in the search URL. You can use this to work out other replacements that might be required when creating a search URL in the spreadsheet.

Once you have entered this formula and pressed enter your spreadsheet should look like:



	A	B
1	URL	http://bnb.data.bl.uk/search
2	object	Tim Berners-Lee
3	BNB Search	http://bnb.data.bl.uk/search?object=Tim+Berners-Lee

So far this example is not very different to using the Flickr API in Exercise 1. However the final step to retrieve and display the result is slightly trickier in this case because our results are in relatively unstructured HTML, rather than in a nicely structured RSS feed.

There is a specific function in Google Sheets to retrieve and display data from a web page. This is the 'importHTML' function. This retrieves a web page and can extract and dispaly any information stored either in tables (<table> tag) or lists (<ul> or <ol> tags). Where the data is in tables or lists this is usually the quickest way of extracting the data you want.

Unfortunately in the case of the BNB, as we saw earlier, the search results are not in a list or a table, but rather in a series of paragraph (<p>) tags. This means we can't use the 'importHTML' function in this case - which makes extracting the results rather more complicated.

In order to extract the results from the BNB Search results page, we need to take advantage of the fact that Google Sheets has a special function for importing data from a format called 'XML'. In particular we can take advantage of the fact that well structured HTML is (generally) also valid XML. This means we can use a Google Sheets function called 'importXML' to work with HTML pages.

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.

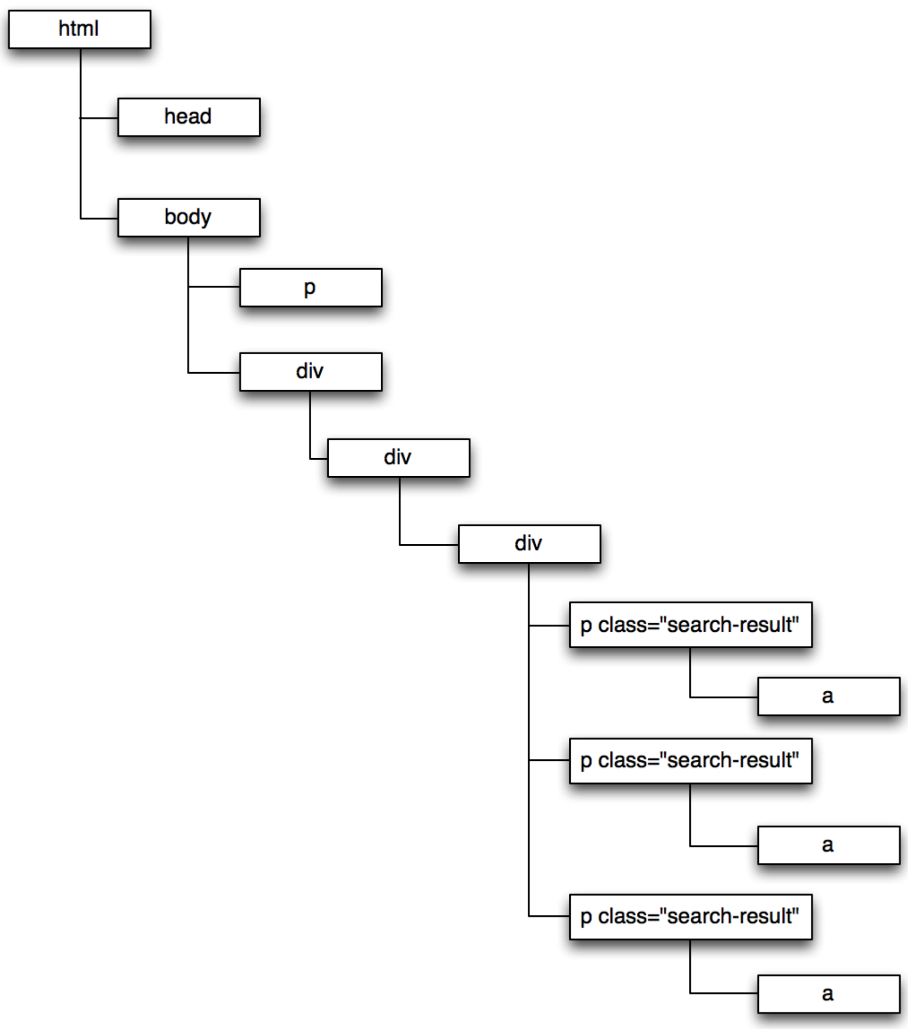
Going Further

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.

HTML documents can have quite complex structures because of the way that elements can be nested inside other elements many times. The following diagram offers a simplified view of the structure of the HTML in the BNB Search results page:



To extract data from the HTML 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 'a' elements in the structure above is:

/html/body/div/div/div/p/a

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 '//a' without needing to express the other container elements. Because '//a' means 'any path that ends with an <a> element' the XPath expression '//a' finds all <a> elements in the HTML document, where ever they are.

The XPath expression we'll be using in this case is:

```
//p[@class='search-result']/a/text()
```

This finds the text in each <a> element in each <p> element where the <p> element has a "class" of 'search-result'. If you look back at the HTML you'll see that this is where our results were in the HTML.

With these two bits of information you are ready to use the ‘importXML’ function. In to Cell B4, type the formula:

```
=importXML(B3,"//p[@class='search-result']/a/text()")
```

Once you have entered this formula, you should find that in your spreadsheet you get a list of results exactly the same as you get from the web page

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

You have:
- Used a Web Page like an API
- Seen how you can search the BNB Linked Datat by adding some parameters to a URL
- Understood how the search results are structured in the HTML
- Understood that an HTML can be treated like XML and data can be extracted using XPath expressions
- Used this knowledge to build a Google Spreadsheet which searches the BNB

Exercise 3: Retrieving full records from the BNB

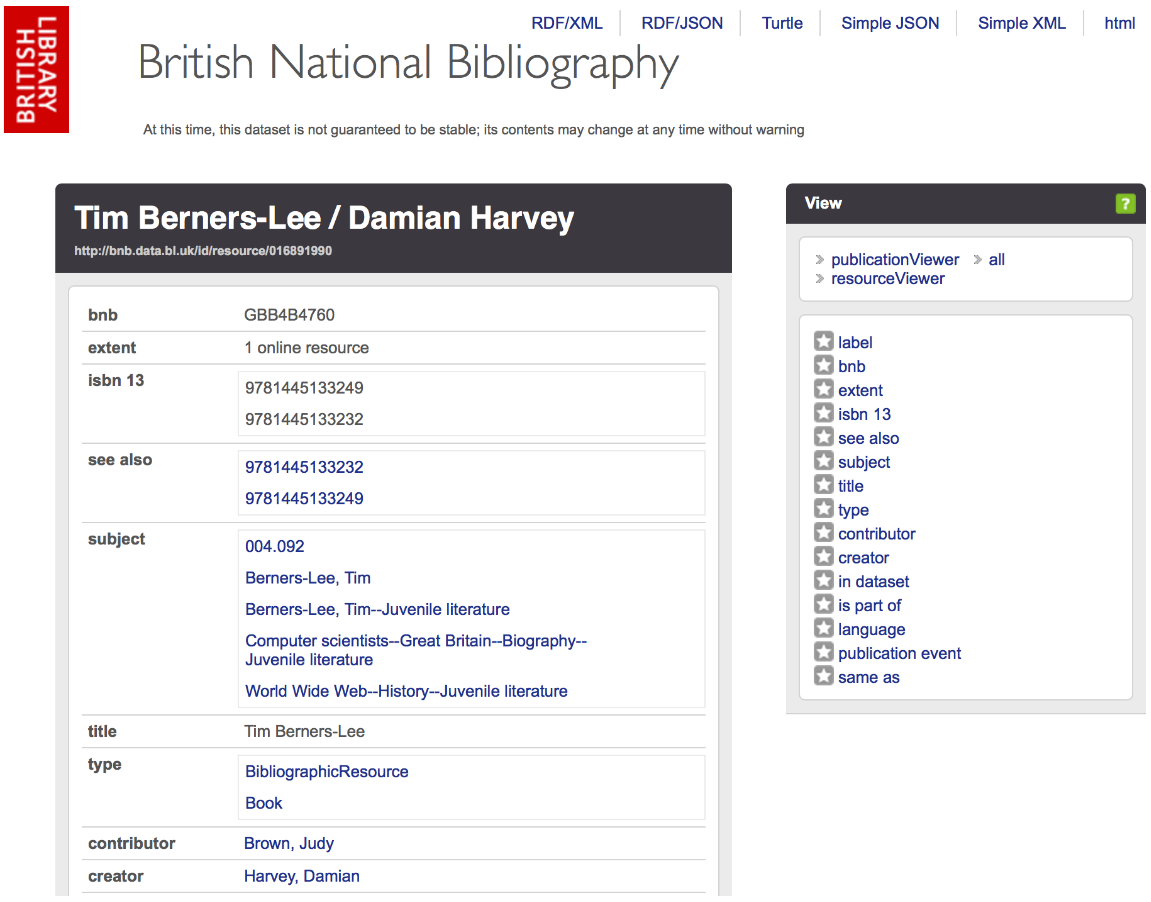
Introduction

For this exercise you are going to work with a 'full record display' for books from the BNB. This exercise builds on the work you did in Exercise 2.

Go back to the BNB Search page at http://bnb.data.bl.uk

Carry out a search - again we'll use the example of searching for "Tim Berners-Lee". You should see three results. Click on the first result "Tim Berners-Lee / Damian Harvey". This should take you to http://bnb.data.bl.uk/doc/resource/016891990

And will show a page like this:


RDF/XML | RDF/JSON | Turtle | Simple JSON | Simple XML | html

British National Bibliography

At this time, this dataset is not guaranteed to be stable; its contents may change at any time without warning

Tim Berners-Lee / Damian Harvey

http://bnb.data.bl.uk/id/resource/016891990

bnb	GBB4B4760
extent	1 online resource
isbn 13	9781445133249 9781445133232
see also	9781445133232 9781445133249
subject	004.092 Berners-Lee, Tim Berners-Lee, Tim--Juvenile literature Computer scientists--Great Britain--Biography--Juvenile literature World Wide Web--History--Juvenile literature
title	Tim Berners-Lee
type	BibliographicResource Book
contributor	Brown, Judy
creator	Harvey, Damian

View

- publicationViewer » all
- resourceViewer

- label
- bnb
- extent
- isbn 13
- see also
- subject
- title
- type
- contributor
- creator
- in dataset
- is part of
- language
- publication event
- same as

This screen displays the information about this item which is available via the BNB API as an HTML page.

As we saw in Exercise 2, while the HTML display works well for humans, it is not always easy to automatically extract data from HTML. Luckily 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 constrictions such as the programming language or tool being used.

Because Google Spreadsheet has some built in functions for reading XML, as with Exercise 2, we'll use importXML function to read the XML.

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/016891990.xml

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

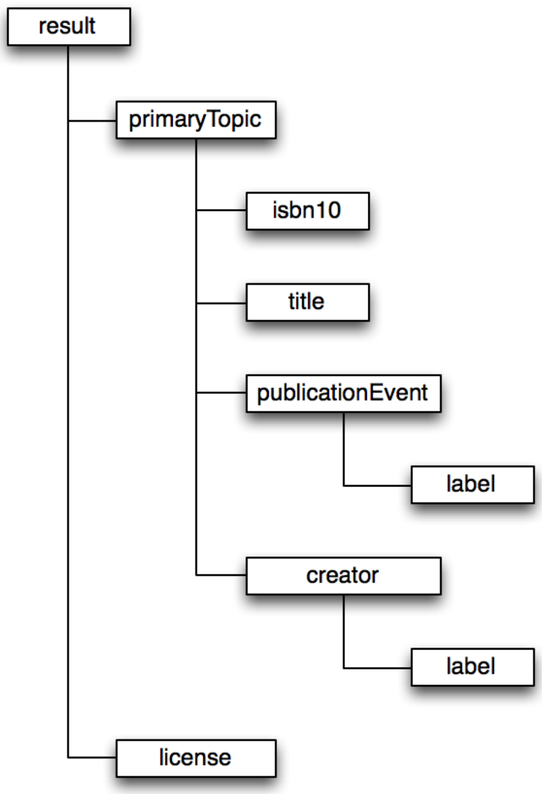
XML is a way of structuring data in a hierarchical way - one way of thinking about it is as a series of folders, each of which can contain further folders. In XML terminology, these are ‘elements’ and each element can contain a value, or further elements (not both). If you look at an XML file, the elements 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. Once again we are going to use XPath to do this. Generally writing XPath for a well structure XML document is easier than writing XPath for an HTML page - so the XPath we need this time round should be simpler to work out than that in Exercise 2.

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

/result/primaryTopic/creator

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 '//creator' without needing to express all the container elements.

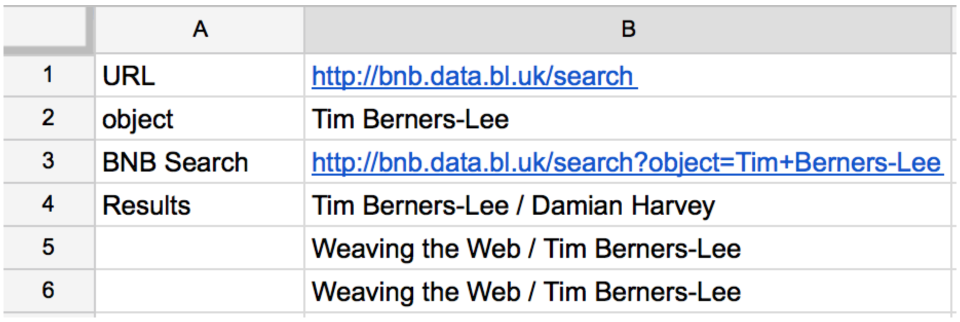
Going Further

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, we can now use this knowledge to retrieve and display information about each of the items we found using the 'Search' function in Exercise 2.

Your spreadsheet should currently look something like:



	A	B
1	URL	http://bnb.data.bl.uk/search
2	object	Tim Berners-Lee
3	BNB Search	http://bnb.data.bl.uk/search?object=Tim+Berners-Lee
4	Results	Tim Berners-Lee / Damian Harvey
5		Weaving the Web / Tim Berners-Lee
6		Weaving the Web / Tim Berners-Lee

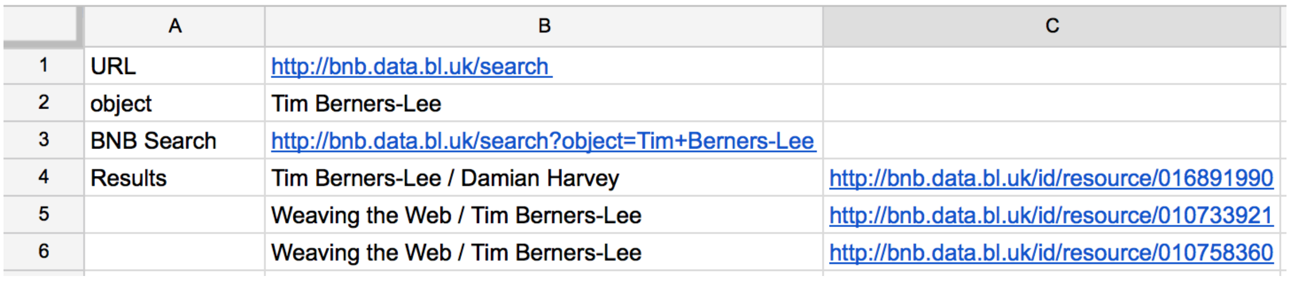
At the moment this is only displaying the text of each result from the BNB Search page. We need to add to this to find the URL for the full item.

In cell C4, you can use a very similar formula to the one you used in B4 in Exercise 2:

=importXML(B3,"//p[@class='search-result']/a/@href")

By using '@href' at the end of the Xpath expression instead of 'text()' we get the link from the <a> element in the results list, rather than the text.

The result should be like this:



	A	B	C
1	URL	http://bnb.data.bl.uk/search	
2	object	Tim Berners-Lee	
3	BNB Search	http://bnb.data.bl.uk/search?object=Tim+Berners-Lee	
4	Results	Tim Berners-Lee / Damian Harvey	http://bnb.data.bl.uk/id/resource/016891990
5		Weaving the Web / Tim Berners-Lee	http://bnb.data.bl.uk/id/resource/010733921
6		Weaving the Web / Tim Berners-Lee	http://bnb.data.bl.uk/id/resource/010758360

We now have a list of URLs for the full record for each of the items we found in the search.

Going Further
You may notice that these URLs are slightly different to the ones we see when we click on items in the results list in the web browser. There is a reason for this, but for our purposes we can treat these URLs as equivalent at the moment.

As we saw in the web interface, in order to get the XML version of the full record we need to add '.xml' onto the end of the URL.

We are going to extract the 'Creator' in this exercise. The Creator element in the XML looks like this:

```
<creator href="http://bnb.data.bl.uk/id/person/HarveyDamian">
<label>Harvey, Damian</label>
</creator>
```

This means that to get the author's name, we need to use the XPath:

//creator/label

With these two bits of information you are ready to use the 'importXML' function. In to Cell D4, type the formula:

=importXml(concatenate(C4,".xml"),"//creator/label")

This creates the correct URL with the 'concatenate' function, retrieves the XML document, and uses the Xpath '//creator/label' to get the name of the creator. Copy this formula down additional rows so that you get the creator for each item you've retrieved from the initial search of BNB.

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

Going Further
Searching the BNB using the page at http://bnb.data.bl.uk can bring back items other than books. For example, a search can retrieve a record for a person, or an organisation. In this case displaying a 'creator' won't work.

Try some alternative searches and have a look at the URLs you see in Column C. What are the options for dealing with these different types of item in the spreadsheet?

Exercise 4: Following Links for more information

Introduction

The BNB is published as Linked Data. By following links within a BNB record you can find more information.

In Exercise 3 you retrieved the name of the 'creator' from some BNB records. In this exercise you will find more information about the creator from the BNB, how many items they have created in the BNB and their Virtual International Authority File (VIAF) identifier.

Using a URL for a full item in the BNB look at the XML. The example from Exercise 3 is http://bnb.data.bl.uk/doc/resource/016891990.xml

Looking at the 'creator' element you can see that as well as the creator name there is a URL:

```
<creator href="http://bnb.data.bl.uk/id/person/HarveyDamian">
<label>Harvey, Damian</label>
</creator>
```

If you take this URL and put it into a browser, you'll see a full record for the author in the BNB. Once again you can access the XML version of this record through the "Simple XML" link. The URL is http://bnb.data.bl.uk/doc/person/HarveyDamian.xml

In the XML you should be able to see that there is a 'hasCreated' element that contains a number of 'item' elements. These are all other items in the BNB where this person is listed as the 'creator'.

```
<hasCreated>
<item href="http://bnb.data.bl.uk/id/resource/012942196">
<label>Oggy and the dinosaur / Damian Harvey</label>
</item>
<item href="http://bnb.data.bl.uk/id/resource/013800180">
<label>Boris the spider / Damian Harvey</label>
</item>
```

In the XML you can also see there is a 'sameAs' element. In Linked Data 'sameAs' is used to indicate that two equivalent identifiers. In this case the 'sameAs' element is:

```
<sameAs href="http://viaf.org/viaf/73804057"/>
```

This indicates that this person in the BNB is the same as the person with the VIAF ID http://viaf.org/viaf/73804057.

We are going to use this information from the XML to do two things:

- Extract the VIAF URL for the person
- Extract the number of items in the BNB for which the person is a creator

VIAF URL

We can extract the VIAF URL using a similar approach to the one we took in Exercise 3. In cell E4 type the formula:

=importXML(concatenate(C4,".xml"),"//creator/@href")

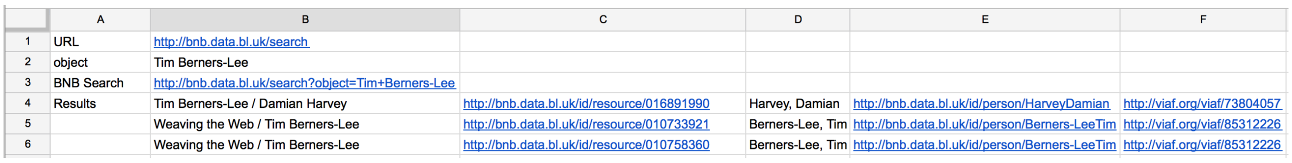
This will retrieve the BNB URL for the creator from the XML for the BNB item. Copy this down any additonal rows in your spreadsheet.

We can now use the XML version of this record to

We can now use the XML version of this record to extract the VIAF ID for the person. In Cell F4 type the formula:

=importXML(concatenate(E4,".xml"),"//sameAs/@href")

Once again, copy this down as many rows as you have in your spreadsheet. You should end up with a sheet that looks like this:



	A	B	C	D	E	F
1	URL	http://bnb.data.bl.uk/search				
2	object	Tim Berners-Lee				
3	BNB Search	http://bnb.data.bl.uk/search?object=Tim+Berners-Lee				
4	Results	Tim Berners-Lee / Damian Harvey	http://bnb.data.bl.uk/id/resource/016891990	Harvey, Damian	http://bnb.data.bl.uk/id/person/HarveyDamian	http://viaf.org/viaf/73804057
5		Weaving the Web / Tim Berners-Lee	http://bnb.data.bl.uk/id/resource/010733921	Berners-Lee, Tim	http://bnb.data.bl.uk/id/person/Berners-LeeTim	http://viaf.org/viaf/85312226
6		Weaving the Web / Tim Berners-Lee	http://bnb.data.bl.uk/id/resource/010758360	Berners-Lee, Tim	http://bnb.data.bl.uk/id/person/Berners-LeeTim	http://viaf.org/viaf/85312226

As VIAF also provides XML data and Linked Data, in theory it would be possible to retrieve further information from the VIAF service. Unfortunately the XML used by VIAF cannot be easily processed using Google Sheets, so this is as far as we can take this exercise in this case.

Counting items
To count the number of items in the BNB which a person has created, we can combine the 'importXML' function with another function called 'COUNTA' which counts the number of values in a list or range in Google Sheets.

First we need the Xpath to the items this person created listed in the XML. This is "//hasCreated/item". We then combine this with COUNTA to get a count of the values, rather than listing them all in the spreadsheet.

In cell G4 type the formula:

```
=COUNTA(importxml(concatenate(E4,".xml"),"//hasCreated/item"))
```

Once again you need to copy this formula to any additional rows in the spreadsheet.

Going Further

If you want to further explore the capabilities of Google Sheets to work with HTML and XML you can read more about this at https://mashe.hawksey.info/2012/10/feeding-google-spreadsheets-exercises-in-import/

Explore the different possibilities of extracting data from the BNB. You can try extracting information on subjects, or following links to other items within the BNB