chem-bla-ics

Programming in the Life Sciences #4: communication from within HTML

Egon Willighagen

Published October 9, 2013

Citation
Willighagen, E. (2013). Programming in the Life Sciences #4: communication from within HTML. In *chem-bla-ics*. chem-bla-ics. https://doi.org/10.59350/7ex09-4x603

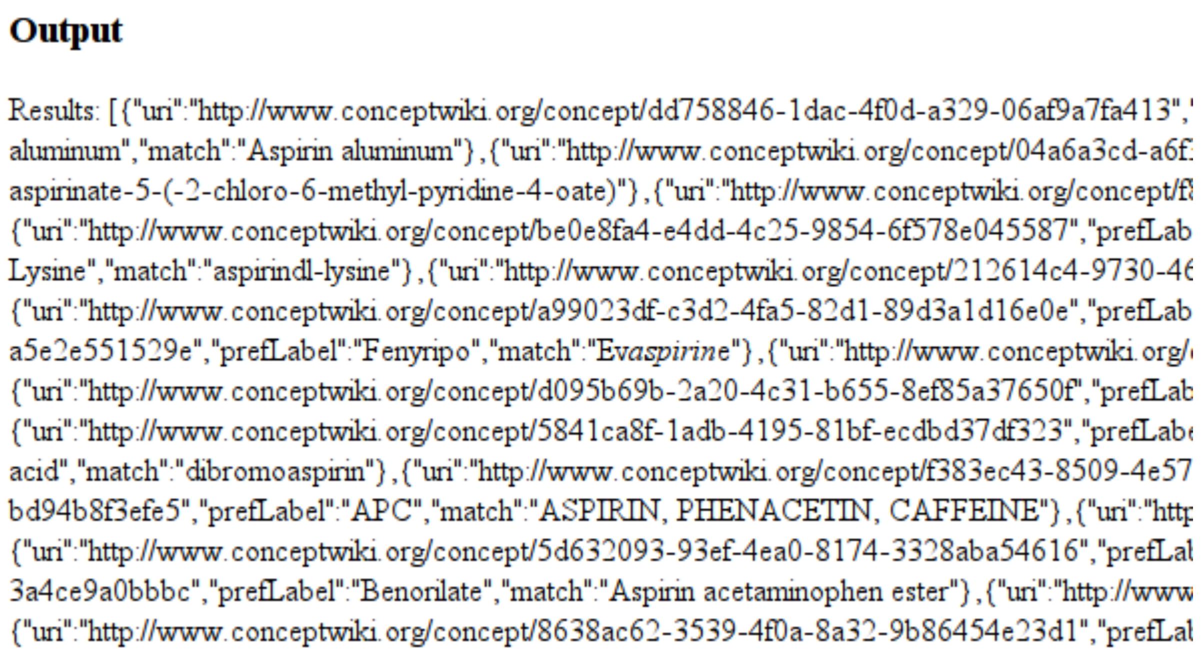
Keywords
Pra3006, Javascript, Html, Openphacts

Abstract
The purpose of a web service is that you give it a question or task, and that it returns an answer. For example, we can ask the Open PHACTS platform what compounds it knows with aspirin in the name. We pass the question (with the API key) and get a list of matching compounds. Now, this communication is complex: it happens at many levels, which are spelled out in the Internet Model.

The purpose of a web service is that you give it a question or task, and that it returns an answer. For example, we can ask the Open PHACTS platform what compounds it knows with aspirin in the name. We pass the question (with the API key) and get a list of matching compounds. Now, this communication is complex: it happens at many levels, which are spelled out in the Internet Model. There are various variants of the stack of communication layers, but we are interested mostly in the top layers, at the *application layer*. In fact, for this course this model only serves as supporting information for those who want to learn more.

Practically, what matters here is how to ask the question and how to understand the answer.

We are supported in these practicalities with JavaScript libraries, in particular the ops.js library and general JSON functionality provided by most browsers (unless the student decided to use a *different* programming language, in which there are different libraries). Personally, I have only very limited JavaScript experience, and this mostly goes back to the good old Userscript and Greasemonkey days (wow! the paper is actually the 4th highest scoring BMC Bioinformatics article!). But because my JavaScript knowledge is limited and rusty, I spent a good part of today, to get a basic example running. Very basic, and barely exceeding the communication details. That is, this is the output in the browser:



Output

```
Results: [{"uri":"http://www.conceptwiki.org/concept/dd758846-1dac-4f0d-a329-06af9a7fa413","
aluminum","match":"Aspirin aluminum"},{"uri":"http://www.conceptwiki.org/concept/04a6a3cd-a6f
aspirinate-5-(-2-chloro-6-methyl-pyridine-4-oate)"},{"uri":"http://www.conceptwiki.org/concept/f
{"uri":"http://www.conceptwiki.org/concept/be0e8fa4-e4dd-4c25-9854-6f578e045587","prefLab
Lysine","match":"aspirindl-lysine"},{"uri":"http://www.conceptwiki.org/concept/212614c4-9730-4
{"uri":"http://www.conceptwiki.org/concept/a99023df-c3d2-4fa5-82d1-89d3a1d16e0e","prefLab
a5e2e551529e","prefLabel":"Fenyripo","match":"Evaspirine"},{"uri":"http://www.conceptwiki.org/
{"uri":"http://www.conceptwiki.org/concept/d095b69b-2a20-4c31-b655-8ef85a37650f","prefLab
{"uri":"http://www.conceptwiki.org/concept/5841ca8f-1adb-4195-81bf-ecdbd37df323","prefLab
acid","match":"dibromoaspirin"},{"uri":"http://www.conceptwiki.org/concept/f383ec43-8509-4e57
bd94b8f3efe5","prefLabel":"APC","match":"ASPIRIN, PHENACETIN, CAFFEINE"},{"uri":"http
{"uri":"http://www.conceptwiki.org/concept/5d632093-93ef-4ea0-8174-3328aba54616","prefLa
3a4ce9a0bbbc","prefLabel":"Benorilate","match":"Aspirin acetaminophen ester"},{"uri":"http://www
{"uri":"http://www.conceptwiki.org/concept/8638ac62-3539-4f0a-8a32-9b86454e23d1","prefLa
```

So, what does the question look like? The question is actually hardcoded in the HTML source, but the page does take two parameters: the `app_key` and `app_id` that come with your Open PHACTS account.

The ops.js library helps us, and wraps the Open PHACTS LDA methods in JavaScript methods. Thus, rather can crafting special HTTP calls, we use two JavaScript calls:

```
var searcher = new Openphacts.ConceptWikiSearch(
  "https://beta.openphacts.org",
  params["app_id"], params["app_key"]
);
searcher.byTag(
  'Aspirin', '20', '4', '07a84994-e464-4bbf-812a-a4b96fa3d197',
  callback
);
```

The first statement creates an LDA method object, while the second makes an actual question. I have not defined the callback variable, which actually is a JavaScript function that looks like:

```
var callback = function(success, status, response){
  var result = searcher.parseResponse(response);
  document.getElementById("output").innerHTML =
    "Results: " + JSON.stringify(result);
};
```

When the LDA web service returns data, this method gets called, providing asynchronous functionality to keep the web page responsive. But when called, it first parses the returned data, and then puts the JSON output as text in the HTML. The output that is given in the earlier screenshot.

So, hurdle taken. From here on it’s easier. Regular looping over the results, creating some HTML output, etc. The full source code if this example is available as Gist.