# Next generation asynchronous webservices #2

**Egon Willighagen**

Published November 4, 2008

**Citation**
Willighagen, E. (2008). Next generation asynchronous webservices #2. In *chem-bla-ics*. chem-bla-ics. https://doi.org/10.59350/esdth-ef628

**Keywords**
Xmpp, Java, Bioclipse, Web

**Abstract**
Getting back to some webservice stuff (see part #1 of this series )... actually, I'll use cloud service from now on, since web service is reserved for SOAP/WSDL (see my EMBRACE presentation ). Let me present this bit of JavaScript I just ran in Bioclipse2:

## chem-bla-ics

Getting back to some webservice stuff (see part #1 of this series )... actually, I'll use *cloud service* from now on, since *web service* is reserved for SOAP/WSDL (see my EMBRACE presentation ). Let me present this bit of JavaScript I just ran in Bioclipse2:

```
xws.connect();
service = xws.getService("cdk.ws1.bmc.uu.se");
service.discoverSync(9000);
service.getFunctions();
f = service.getFunction("calculateMass");
ios = f.getIoSchemataSync(9000);
iof = xws.getIoFactory(ios);
smiDoc = iof.createSmilesDocument();
smiDoc.setSmiles("CCC");
result = f.invokeSync(smiDoc.toString(), 9000);
obj = iof.getOutputObject(result);
print("Mass: " +  obj.getStringValue())
```

At first, it might look a bit verbose to just calculate the mass of a molecule, and it is, and it is not even written in XML. Hahahaha

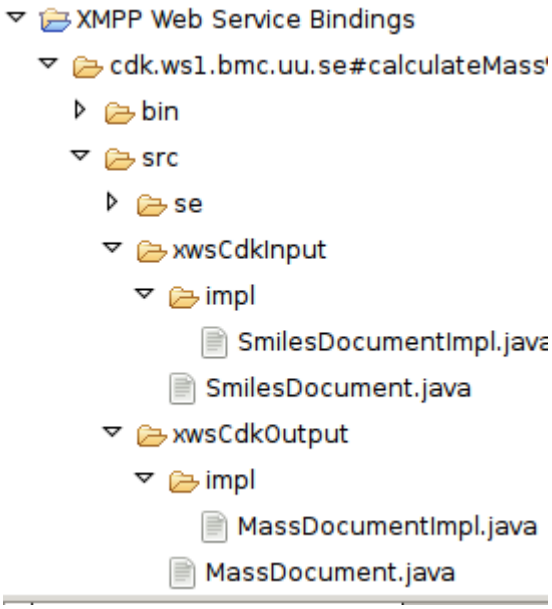
Anyway, the code rocks, thanx to Johannes' great work on his xws4j library! I'll explain the script. The first line gets Bioclipse online using a *Jabber* account, which you set via Bioclipse' preferences pages. The next few lines allows you to connect to a cloud service, this one running on `ws1.bmc.uu.se` and called `cdk`. With the `getFunctions()` method we query which functions are available, called ports in WDSL if not mistaken, from which we pick the `calculateMass` one.

And then the action joins in. One nice feature of the IO-DATA proposal is that the function itself defines the XML Schema it uses for input and output, and does not rely on WDSL to do that (or maybe recent SOAP specs allows that too). So, we query the function for its schemata, and the xws4j library then something funky happens: we order the library to create a data model on the fly for this service! From this we get a Java data model for the service. This allows us to use `createSmilesDocument()` and `setSmiles()`. That's function-specific stuff!

Of course, we do not have to do that. For example, the second function I wrote (`generate3Dcoordinates`) eats and spits CML, and I'd rather rely on CMLDOM or CDK as data model then. But more on that later...

The Bioclipse xws4j plugin actually puts the data model in my workspace, so that I can easily introspect the API:

### chem-bla-ics




The last three lines invoke the function (synchronously, as it's cheap), and get the mass from the function output. BTW, I should stress that a function does not require any specific implementation regarding synchronous or asynchronous calls. You write **one** function, and can call it in either way you like. The library hides all IO-DATA details around that.