Using Web Services from Local Scripts

EMBRACE Workshop on Client Side Scripting for Web Services

Francisco Simões Roque, February 6, 2008
Honolulu University?

I want to use tool x,y,z from Honolulu University within one of my scripts but I don’t want to care about installing, maintaining, licenses, etc...
But wait! They have implemented it as a Web Service!

- They have a well documented WSDL
- We know the service it provides
- We know what functions it makes available
- We know what to expect in the output
Remote tools become local functions!

\[ y = f(x_1, x_2, x_3, \ldots) \]

function

result

parameters
Remote tools become local functions!

\[ y = \sin(x) \]
Remote tools become local functions!

$$y = \text{Blast}(\text{sequence}, \text{PDB})$$
Remote tools become local functions!
Remote tools become local functions!
Local functions can be used in local scripts

```python
sequence = getSequence(P53_HUMAN)

blastresult = Blast(sequence, PDB)

IF blastresult.score > 0.8 THEN
  Display blastresult
```
Choose your language of choice

- Perl using SOAP::Lite
- Python using SOAPpy
- C++ and gSOAP
- Java and Apache SOAP
Ubiquity across platforms and languages

• Access a UNIX server running a C program from Windows and Perl

• If the server changes to Solaris and Java, everything transitions without a glitch
Arguments and return values may be hard to understand

- Documentation is important
- Resort to other tools like SoapUI
#!/usr/bin/python

from SOAPpy import WSDL


EUtilsProxy = WSDL.Proxy(wsdlFile)

eFetchIn = { 'db' : 'protein',
            'id' : 'P53_HUMAN' } 

eFetchOut = EUtilsProxy.run_eFetch(eFetchIn)

print '>'+eFetchOut['GBSeq']['GBSeq_primary-accession']
print eFetchOut['GBSeq']['GBSeq_sequence'].upper()
A Simple Example

```python
#!/usr/bin/python
from SOAPpy import WSDL


EUtilsProxy = WSDL.Proxy(wsdlFile)

eFetchIn = { 'db' : 'protein',
             'id' : 'P53_HUMAN' };

eFetchOut = EUtilsProxy.run_eFetch(eFetchIn)

print '>'+ eFetchOut['GBSeq']['GBSeq_primary-accession']
print eFetchOut['GBSeq']['GBSeq_sequence'].upper()
```
A Simple Example

```python
#!/usr/bin/python
from SOAPpy import WSDL

EUtilsProxy = WSDL.Proxy(wsdlFile)

eFetchIn = {'db': 'protein',
            'id': 'P53_HUMAN'}

eFetchOut = EUtilsProxy.run_eFetch(eFetchIn)

print '>' + eFetchOut['GBSeq']['GBSeq_primary-accession']
print eFetchOut['GBSeq']['GBSeq_sequence'].upper()
```
A Simple Example

```python
#!/usr/bin/python

from SOAPpy import WSDL


EUtilsProxy = WSDL.Proxy(wsdlFile)

eFetchIn = {'db': 'protein',
            'id': 'P53_HUMAN'}

eFetchOut = EUtilsProxy.run_eFetch(eFetchIn)

print '>%s' % eFetchOut['GBSeq']['GBSeq_primary-accession']
print eFetchOut['GBSeq']['GBSeq_sequence'].upper()
```
A Simple Example

```
#!/usr/bin/python
from SOAPpy import WSDL
EUtilsProxy = WSDL.Proxy(wsdlFile)
eFetchIn = {'db': 'protein', 'id': 'P53_HUMAN'}
eFetchOut = EUtilsProxy.run_eFetch(eFetchIn)
print '>' + eFetchOut['GBSeq']['GBSeq_primary-accession']
print eFetchOut['GBSeq']['GBSeq_sequence'].upper()
```
A Simple Example

```python
#!/usr/bin/python

from SOAPpy import WSDL


EUtilsProxy = WSDL.Proxy(wsdlFile)

eFetchIn = { 'db': 'protein',
            'id': 'P53_HUMAN' }

eFetchOut = EUtilsProxy.run_eFetch(eFetchIn)

print '>' + eFetchOut['GBSeq']['GBSeq_primary-accession']
print eFetchOut['GBSeq']['GBSeq_sequence'].upper()
```
import the python module

```
#!/usr/bin/python
from SOAPpy import WSDL

EUtilsProxy = WSDL.Proxy(wsdlFile)

eFetchIn = { 'db' : 'protein',
            'id' : 'P53_HUMAN' }

eFetchOut = EUtilsProxy.run_eFetch(eFetchIn)

print '>'+eFetchOut['GBSeq']['GBSeq_primary-accession']
print eFetchOut['GBSeq']['GBSeq_sequence'].upper()
```

load the WSDL file

connect to the Web Service provider

build the input for the Service

run the Web Service

print the results
EFetch Overview
Last updated: July 27, 2005

**EFetch:** Retrieves records in the requested format from a list of one or more UIs or from user's environment. Click on a database below to display database specific documentation.

- [URL Parameters](#)
  - **Database**
  - **Web Environment Query_key Tool E-mail Address**
- [Literature Database](#) (PubMed, Journals, PubMed Central, OMIM)
- [Sequence and other Molecular Biology Databases](#) (Nucleotide, Protein, Gene, etc.)
- Taxonomy
- User System Requirements
- Entrez DTDs
- Demonstration Program
- Announcement Mailing List
- Help Desk

**Base URL:** http://eutils.ncbi.nlm.nih.gov/entrez/eutils/efetch.fcgi?

**URL parameters:** (NOTE: Utility parameters may be case sensitive. Use lower case characters in all parameters except for WebEnv.)

Database (Click the appropriate section in the Table of Contents for database specific URL parameters.)

- `db=`database name

Web Environment: Value previously returned in XML results from ESearch and used with EFetch in place of primary ID result list.

- `WebEnv=WgHmICDG`, etc.

Query_key: The value used for a history search number or previously returned in XML results from ESearch or EPost.

- `query_key=6`

**Note:** WebEnv is similar to the cookie that is set on a user's computers when accessing PubMed on the web. If the parameter `usehistory=` is included in an ESearch URL both a WebEnv (cookie string) and query_key (history number) values will be returned in the results. Rather then using the retrieved PMIDs in an EFetch URL you may simply use the
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:efet="http://www.ncbi.nlm.nih.gov/entrez/eutils/soap/soap_adapter_1_5.cgi">
  <soapenv:Body>
    <efet:eFetchRequest>
      <!-- Optional -->
      <efet:eFetchResponse>
        <!-- Optional -->
      </efet:eFetchResponse>
    </efet:eFetchRequest>
  </soapenv:Body>
</soapenv:Envelope>
  <soapenv:Header/>
  <soapenv:Body>
    <efetch_request_request>
      <efetch_request_request_response>
        <efetch_request_request_result>
          <!--Optional:-->
          <efetch_request_request_result_response>
            <efetch_request_request_result_response_result>
              <!--Optional:-->            </efetch_request_request_result_response_result>
          </efetch_request_request_result_response>
          <!--Optional:-->        </efetch_request_request_result>
        </efetch_request_request_response>
        <!--Optional:-->
      </efetch_request_request_response>
      <efetch_request_request_body>
        <efetch_request_request_request>
        </efetch_request_request_request>
      </efetch_request_request_body>
    </efetch_request_request>
    <efetch_request_response>
      <efetch_request_response_response>
        <efetch_request_response_response_result>
        </efetch_request_response_response_response_result>
      </efetch_request_response_response_response>
      <efetch_request_response_response_result>
      </efetch_request_response_response_response>
    </efetch_request_response_response>
    <efetch_request_response_response>
      <efetch_request_response_response_result>
      </efetch_request_response_response_response>
    </efetch_request_response_response>
  </soapenv:Body>
</soapenv:Envelope>
```python
#!/usr/bin/python
from SOAPpy import WSDL
protid = 'P53_HUMAN'
EUtilsProxy = WSDL.Proxy(wsdlFile)
eFetchIn = { 'db' : 'protein',
            'id' : protid}
eFetchOut = EUtilsProxy.run_eFetch(eFetchIn)
print '>' + eFetchOut['GBSeq']['GBSeq_primary-accession']
print eFetchOut['GBSeq']['GBSeq_sequence'].upper()
```

```perl
#!/usr/bin/perl
use SOAP::Lite;
my $protid = 'P53_HUMAN';
my $efetch = SOAP::Lite
my $result = $efetch -> run_eFetch(
    SOAP::Data->name(db => "protein"),
    SOAP::Data->name(id => $protid));
print ">";
print $result->{GBSeq}->{'GBSeq_primary-accession'};
print "\n";
print uc($result->{GBSeq}->{GBSeq_sequence});
print "\n";
```
And the result:
And now for something slightly more complex

- Get a full bacterial genome by providing a GenBank accession number
- Predict the presence of signal peptide cleavage sites
- Print the predicted protein sequences with cleavage sites, their names, and respective scores

We will use and interconnect two Web Services

**GenomeAtlas** and **SignalP**
CBS Web Services

The services listed below are SOAP based Web Services: they are strictly defined in WSDL files and available to client side applications running on the users' own computers. Each service comes with extensive documentation intended to facilitate usage.

The access to the services is free and unlimited for all academic users. For other users the access is also free but limited. Please see here for the details of the access policy.

The work on SOAP based Web Services at CBS is in progress; please see the documentation for detailed description of the implementation decisions made so far, the status of the effort and the future plans.

Please click here for the complete list of CBS services (also not SOAP based).

NUCLEOTIDE SEQUENCES

Whole genome visualization and analysis
   GenomeAtlas
   DNA structural atlases for complete microbial Genomes.
   SIDDbase
   Calculation of Stress-induced DNA duplex destabilization.

Gene finding and splice sites
   EasyGene
   Genes in prokaryotes.
   RNAmmer
   Ribosomal RNA sub units.

AMINO ACID SEQUENCES

Protein sorting
   SignalP
   Signal peptide and cleavage sites in gram+, gram- and eukaryotic amino acid sequences.

Post-translational modifications of proteins
   NetGlycate
   Glycation of ε amino groups of lysines in mammalian proteins.
   NetNGlyc
   N-linked glycosylation sites in human proteins.
   NetOGlyc
   O-GalNAc (mucin type) glycosylation sites in mammalian proteins.
   NetPhos
   Generic phosphorylation sites in eukaryotic proteins.

Immunological features
   NetChop
   Proteosomal cleavages (MHC ligands).
   NetCTL
   Integrated class I antigen presentation.

Protein function and structure
   TMHMM
   Transmembrane helices in proteins.
GenomeAtlas 3.1

DNA structural atlases for complete Microbial Genomes

WSDL

Schema definitions

We recommend that the first time users should load the WSDL file above to SoapUI and investigate the Web Services operations in that environment. SoapUI is a desktop application for inspecting, invoking, developing and functional/load/compliance testing of Web Services over HTTP. It can be downloaded free of charge from http://www.soapui.org/.

Other versions

<table>
<thead>
<tr>
<th>Ver.</th>
<th>Description</th>
<th>Last updated</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1</td>
<td>DNA structural atlases for complete Microbial Genomes</td>
<td>2007-01-26 (current, most recent)</td>
</tr>
<tr>
<td>3.0</td>
<td>DNA structural atlases for complete</td>
<td>2007-01-11</td>
</tr>
</tbody>
</table>

Examples of client side scripts using the service

<table>
<thead>
<tr>
<th>Filename</th>
<th>Type</th>
<th>Compatibility</th>
<th>Author</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>getProt_simple.pl</td>
<td>Perl</td>
<td>3.0, 3.1</td>
<td>Peter Fischer Hallin</td>
<td>Extract protein sequences of two genome sequences using the getProt operation: Output is Fasta on stdout.</td>
</tr>
</tbody>
</table>

Documentation

This Web Service accesses the database records of the GenomeAtlas database v.3

1. getSeq
   Set the genomic sequence by specifying the GenBank accession number
#!/usr/bin/python

import sys
from SOAPpy import WSDL

accession = sys.argv[1]

GenomeAtlasWSDL = 'http://www.cbs.dtu.dk/ws/GenomeAtlas/GenomeAtlas_3_1.wsdl'

AtlasProxy = WSDL.Proxy(GenomeAtlasWSDL)

getProtIn = {'accession': accession}

getProtOut = AtlasProxy.getProt(getProtIn)
#!/usr/bin/python

import sys
from SOAPpy import WSDL

accession = sys.argv[1]

GenomeAtlasWSDL = 'http://www.cbs.dtu.dk/ws/GenomeAtlas/GenomeAtlas_3_1.wsdl'

AtlasProxy = WSDL.Proxy(GenomeAtlasWSDL)

getProtIn = {'accession': accession}

getProtOut = AtlasProxy.getProt(getProtIn)

The retrieved proteic sequences are stored in this variable
SignalP 3.0 Server

SignalP 3.0 server predicts the presence and location of signal peptide cleavage sites in amino acid sequences from different organisms: Gram-positive prokaryotes, Gram-negative prokaryotes, and eukaryotes. The method incorporates a prediction of cleavage sites and a signal peptide/non-signal peptide prediction based on a combination of several artificial neural networks and hidden Markov models.

View the version history of this server. All the previous versions are available online, for comparison and reference.

**SUBMISSION**

Paste a single sequence or several sequences in FASTA format into the field below:

Submit a file in FASTA format directly from your local disk:
`/cbs/home/people/macman/SOAP/Uppsala/AL123456.fasta`

**Organism group**
- Eukaryotes
- Gram-negative bacteria
- Gram-positive bacteria

**Method**
- Neural networks
- Hidden Markov models
- Both

**Graphics**
- No graphics
- GIF (inline)
- GIF (inline) and EPS (as links)

**Output format**
- Standard
- Full

**Truncation**
Truncate each sequence to max. $\sqrt{70}$ residues.
Your job is being processed

Wait here to watch the progress of your job, or fill in the form below to get an email message upon completion.

To get notified by email: Notify me via email

To cancel the job: Cancel job

This page will update itself automatically.
SignalP 3.0 Server - prediction results

Technical University of Denmark

Using neural networks (NN) trained on Gram-positive bacteria

>AL123456_CDS_1-1524

SignalP-NN result:

SignalP-NN prediction (gram+ networks): AL123456 CDS 1-1524
SignalP-NN prediction (gram+ networks): AL123456 CDS 1-1524

# data

>AL123456_CDS_1-1524  length = 70
# Measure  Position  Value  Cutoff signal peptide?
max. C  35  0.167  0.52  NO
max. Y  30  0.116  0.32  NO
max. S  17  0.474  0.97  NO
mean S  1-29  0.108  0.51  NO
D  1-29  0.112  0.45  NO
SignalP 3.1

Prediction of signal peptide and cleavage sites in gram+, gram- and eukaryotic amino acid sequences

WSDL
SignalP/SignalP_3_1.wsdl
../common/ws_common_1_0.xsd
ws_signalP_3_1.xsd

We recommend that the first time users should load the WSDL file above to SoapUI and investigate the Web Services operations in that environment. SoapUI is a desktop application for inspecting, invoking, developing and functional/load/compliance testing of Web Services over HTTP. It can be downloaded free of charge from http://www.soapui.org/.

Examples of client side scripts using the service

<table>
<thead>
<tr>
<th>Filename</th>
<th>Type</th>
<th>Compatibility</th>
<th>Author</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>signalP.py</td>
<td>Python</td>
<td>GenomeAtlas 3.1</td>
<td>Francisco Roque</td>
<td>Extract the protein sequences from a given GenBank entry containing a prokaryote genome using the GenomeAtlas 3.1 WSDL getProt operation, and submit them to SignalP 3.1 predicting the presence of secretory signal peptides.</td>
</tr>
</tbody>
</table>

Documentation

INTRODUCTION

This Web Service implements SignalP v. 3.1. It predicts the presence and location of signal peptide cleavage sites in amino acid sequences from different organisms: Gram-positive prokaryotes, Gram-negative prokaryotes, and eukaryotes. The method incorporates a prediction of cleavage sites and a signal peptide/non-signal peptide prediction based on a combination of several artificial neural networks and hidden Markov models. The method is described in detail in the following article:
2nd part: SignalP

Client side

SOAP Client

Submit DNA sequence, sequence identifier and program parameters

Retrieve prediction results

Server side

WSDL file

www.cbs.dtu.dk/ws

RunSimple()

Input data and parameters

Client side

WSDL file

www.cbs.dtu.dk/ws

SOAP Client

Submit DNA sequence, sequence identifier and program parameters

Input data and parameters

Server side

runRNAmer()

Job identifier

pollQueue()
2nd part: SignalP

```python
SignalPWSDL = 'http://www.cbs.dtu.dk/ws/SignalP/SignalP_3_1.wsdl'

SignalPIn = {'organism': 'gram+', 'sequences': getProtOut.sequences}

SignalPPProxy = WSDL.Proxy(SignalPWSDL)

jobid = SignalPPProxy.runService(SignalPIn)

job = SignalPPProxy.pollQueue(jobid)

while job.status != 'FINISHED':
    job = SignalPPProxy.pollQueue(jobid)
    time.sleep(5)

SignalPOut = SignalPPProxy.fetchResult(jobid)
```
2nd part: SignalP

SignalPWSDL = 'http://www.cbs.dtu.dk/ws/SignalP/SignalP_3_1.wsdl'

SignalPIn = {'organism': 'gram+',
             'sequences': getProtOut.sequences}

SignalPPProxy = WSDL.Proxy(SignalPWSDL)

jobid = SignalPPProxy.runService(SignalPIn)

job = SignalPPProxy.pollQueue(jobid)

while job.status != 'FINISHED':
    job = SignalPPProxy.pollQueue(jobid)
    time.sleep(5)

SignalPOut = SignalPPProxy.fetchResult(jobid)
2nd part: SignalP

SignalPWSDL = 'http://www.cbs.dtu.dk/ws/SignalP/SignalP_3_1.wsdl'

SignalPIn = {'organism': 'gram+',
             'sequences': getProtOut.sequences}

SignalPPProxy = WSDL.Proxy(SignalPWSDL)

jobid = SignalPPProxy.runService(SignalPIn)

job = SignalPPProxy.pollQueue(jobid)

while job.status != 'FINISHED' :
    job = SignalPPProxy.pollQueue(jobid)
    time.sleep(5)

SignalPOut = SignalPPProxy.fetchResult(jobid)

---

from the 1st part

asynchronous job workaround
2nd part: SignalP

SignalPWSDL = 'http://www.cbs.dtu.dk/ws/SignalP/SignalP_3_1.wsdl'

SignalPIn = {'organism': 'gram+',
             'sequences': getProtOut.sequences}

SignalPProxy = WSDL.Proxy(SignalPWSDL)

jobid = SignalPProxy.runService(SignalPIn)

job = SignalPProxy.pollQueue(jobid)

while job.status != 'FINISHED':
    job = SignalPProxy.pollQueue(jobid)
    time.sleep(5)

SignalPOut = SignalPProxy.fetchResult(jobid)

Results are stored in this variable
#!/usr/bin/python

import sys, time
from SOAPpy import WSDL

accession = sys.argv[1]

GenomeAtlasWSDL = 'http://www.cbs.dtu.dk/ws/GenomeAtlas/GenomeAtlas_3_1.wsdl'
SignalPWSDL = 'http://www.cbs.dtu.dk/ws/SignalP/SignalP_3_1.wsdl'

AtlasProxy = WSDL.Proxy(GenomeAtlasWSDL)

getProtIn = {'accession' : accession}

getProtOut = AtlasProxy.getProt(getProtIn)

dictseq = dict([(entry.ident[:20], (entry.ident, entry.seq)) for entry in getProtOut.sequences.entry])

SignalPln = {'organism' : 'gram+',
             'sequences' : getProtOut.sequences}

SignalPProxy = WSDL.Proxy(SignalPWSDL)

jobid = SignalPProxy.runService(SignalPln)

job = SignalPProxy.pollQueue(jobid)

while job.status != 'FINISHED':
    job = SignalPProxy.pollQueue(jobid)
    time.sleep(5)

SignalPOut = SignalPProxy.fetchResult(jobid)

for record in SignalPOut.prediction.gff._record:
    if record.comment == 'Y':
        print "seqname: %s" % dictseq[record.seqname][0]
        print "score: %s" % record.score
        print "seq: %s" % dictseq[record.seqname][1]
#!/usr/bin/python
import sys, time
from SOAPpy import WSDL

accession = sys.argv[1]

GenomeAtlasWSDL = 'http://www.cbs.dtu.dk/ws/GenomeAtlas/GenomeAtlas_3_1.wsdl'
SignalPWSDL = 'http://www.cbs.dtu.dk/ws/SignalP/SignalP_3_1.wsdl'

AtlasProxy = WSDL.Proxy(GenomeAtlasWSDL)
getProtIn = {'accession': accession}
getProtOut = AtlasProxy.getProt(getProtIn)
dictseq = dict([(entry.ident[:20], (entry.ident, entry.seq)) for entry in getProtOut.sequences.entry])

SignalPln = {'organism': 'gram+',
             'sequences': getProtOut.sequences}

SignalPProxy = WSDL.Proxy(SignalPWSDL)
jobid = SignalPProxy.runService(SignalPln)
job = SignalPProxy.pollQueue(jobid)

while job.status != 'FINISHED':
    job = SignalPProxy.pollQueue(jobid)
    time.sleep(5)

SignalPOut = SignalPProxy.fetchResult(jobid)

for record in SignalPOut.prediction.gff_record:
    if record.comment == 'Y':
        print "seqname: %s" % dictseq[record.seqname][0]
        print "score: %s" % record.score
        print "seq: %s" % dictseq[record.seqname][1]

results can be printed in any format, written to files, integrated with other services, etc.
Results

```
chico Web Services % python signalP.py AL123456
Fetched 3991 sequences
SignalP service launched with jobid: 86cb21437d356825e7a737176e5affa6
The job is ACTIVE
The job is ACTIVE
The job is ACTIVE
The job is ACTIVE
The job is ACTIVE
The job is ACTIVE
The job is ACTIVE
The job is ACTIVE
The job is ACTIVE
The job is ACTIVE
The job is ACTIVE
The job is FINISHED
The job is FINISHED, fetching results.
seqname: AL123456_CDS_20234-18759
score: 0.635
seq: MNASLRRISVTMALIVLLLLNATMTQVTADGLRADDPRQVRLLDEYRSRQRGQTAGGQLLAYSVATDGRFRFLRVYNNPEVYAPVTGFYSRLRSSTALERAEDPILNGSDRLFGRRLADFTGRPGDPRGNDTTINPIQAGWAMQGQGPGCQGGVVALEPSGFLKILALVSSPSYDPNLMLASHNPEVQAQAWQRLGDNPASPLTNRAISETYPGSTFKVTTAAALAAAGATEQELTAAPITPLPGSTQALENYGAPCPGDEPSTVRLEAFKYKSCTAFVQGILRGADDALRMARAFGDSLPPRP
TPLQVAESTVPISALATLMSITQPIXQKQTMDALPPLNAEIAATIANGITMPYPYVLGSLKGDPDLANISSTTVQYQORAVSPQVAAKLTELVMGAEEKVQGGAPGVQIASKTGTAEHGTDPRHPTPHANYIAFAPAQAPKVAVAVLVENTRAGDLSATGGLAAAPGRAVIEAALQGEP
seqname: AL123456_CDS_21640-20231
score: 0.527
seq: MTTRLQAPVATTPPLTRRANELLLLLCFAAVITFAALLVQANQGGVPWDLTSYLALFLTLFGSAHLAIRRFAPYTDPLPLLPPVALLNGLGLVWIHRLDLVDNEIGEHRHPSANQGMLWTLVGVAFALVTFKHDRQLARYGYICGLGLVFLAVPALLPAALSENQGAKIWRILPGFSIQPAEFSKILLLIFFSAVVLVAKRGFLTSAHHCLNLGTMPLLPPRDLAPLALLAVNISGVMFKEKDGLASSLYTSLFVYVLATQFRSWVQVLTIFAAATLGALYAYFIEHVRLRVQTLWLPADPDGTGYQIVQSLFSFATGIGFTGLNGQDPDVTNPAASTDFIIAAAGFEELGLVGLTAILMLYTIVIIRGLRTIAATRSDFGKLLAAGLSLTAIQFLIVVGGVRPLTGLTTPW
```
Take home messages

• Web Services can be easily converted to local tools, accessible from anywhere in the world
• Implementations in C++, Java, Perl, Python, ...
• Easy integration with common Bioinformatics pipelines
• Interconnection between Services is possible (straightforward when they have common datatypes)
The exercise

• Remember to check the method’s input and output format in SoapUI

• Test everything with print statements

• Most of the things are done for you, but try to understand what’s happening.
Acknowledgments

Kristoffer Rapacki
Peter Hallin