Using Lisp as a powerful SOAP / Web Service Testing Tool

Written by Sebastian K. Glas, 01/2016

It doesn't matter if one likes the concept of web services or not. Working as a business analyst I have found them in many enterprise environments, especially in large and distributed system environments. Using web services might be a global, IT-strategic architecture and design decision. In order to be able to work as an analyst, I have to be able to

- understand existing services (especially the functional side)
- design and describe new system integration scenarios based on SOAP / web services
- (at times) test web service implementations from a functional point of view.

When testing web services as a system or business analyst (and testing and playing around with them is probably the best way to get to know them), I have found Lisp to be extremely useful and powerful.

This article describes how to create your own web service testing toolkit with all advantages of Lisp, such as syntactic abstraction, interactive use, and iterative and incremental improvement. It is also an excellent choice for regression testing.

I will assume that you have a Lisp development environment ready (in my case I use Emacs + Slime + Clozure Common Lisp, see my Post here) and that you are familiar with a tool like SoapUI.

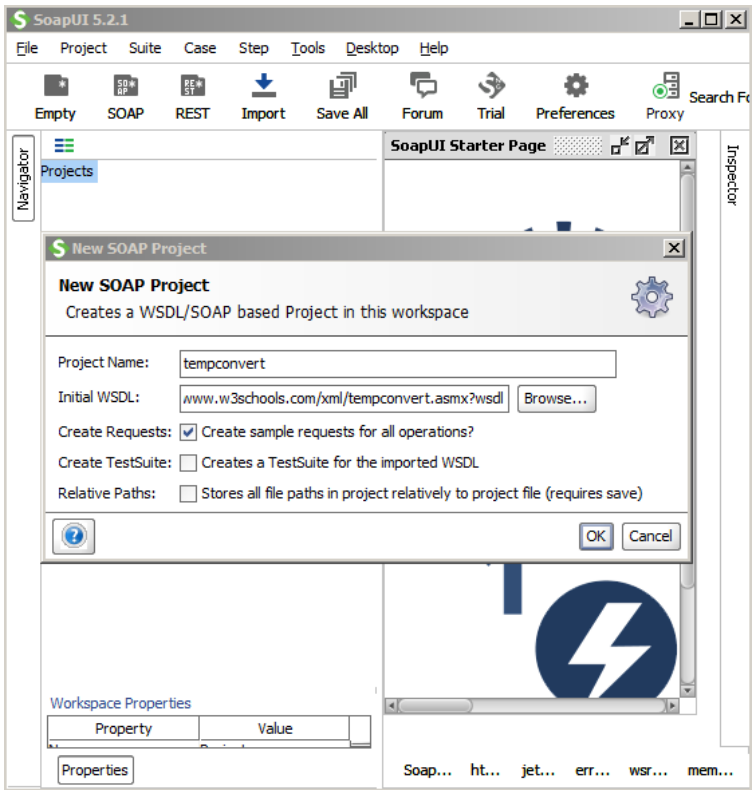
In this tutorial, we will, step-by-step, transform a SoapUI web service call to a Lisp web service call and then add abstraction. Some basic understanding of SOAP / web services and Lisp is required (this scenario only describes SOAP over HTTP).

1.1 The Basics

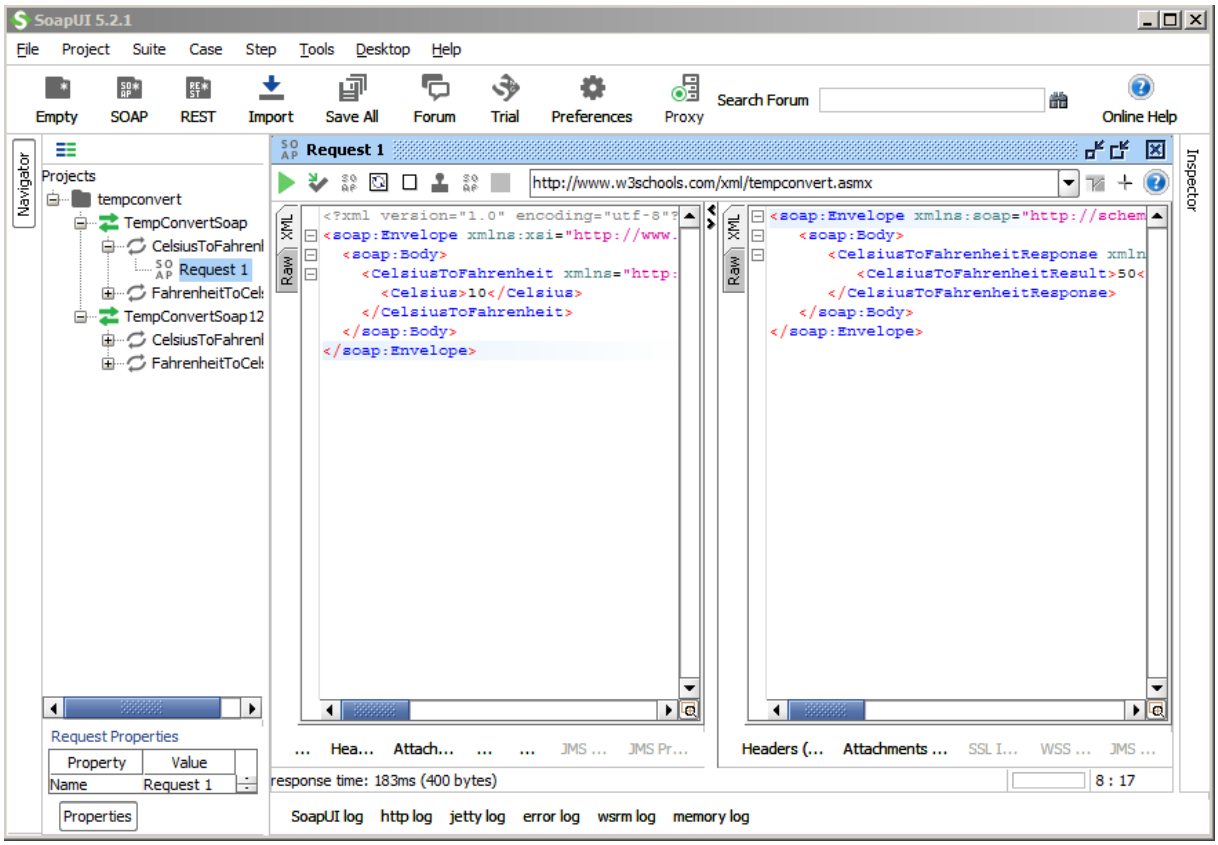
First, you should be able to make a web service call via SoapUI. Download and install SoapUI, then ask your development colleagues to send you the required information, an example call, or a SoapUI project file. You can then either create a SoapUI example project from the WSDL, or you can create one manually.

In this article we will use the trivial "Temperature Conversion Tool" from W3Cschools as an example. It can be found at:

http://www.w3schools.com/xml/tempconvert.asmx?wsdl

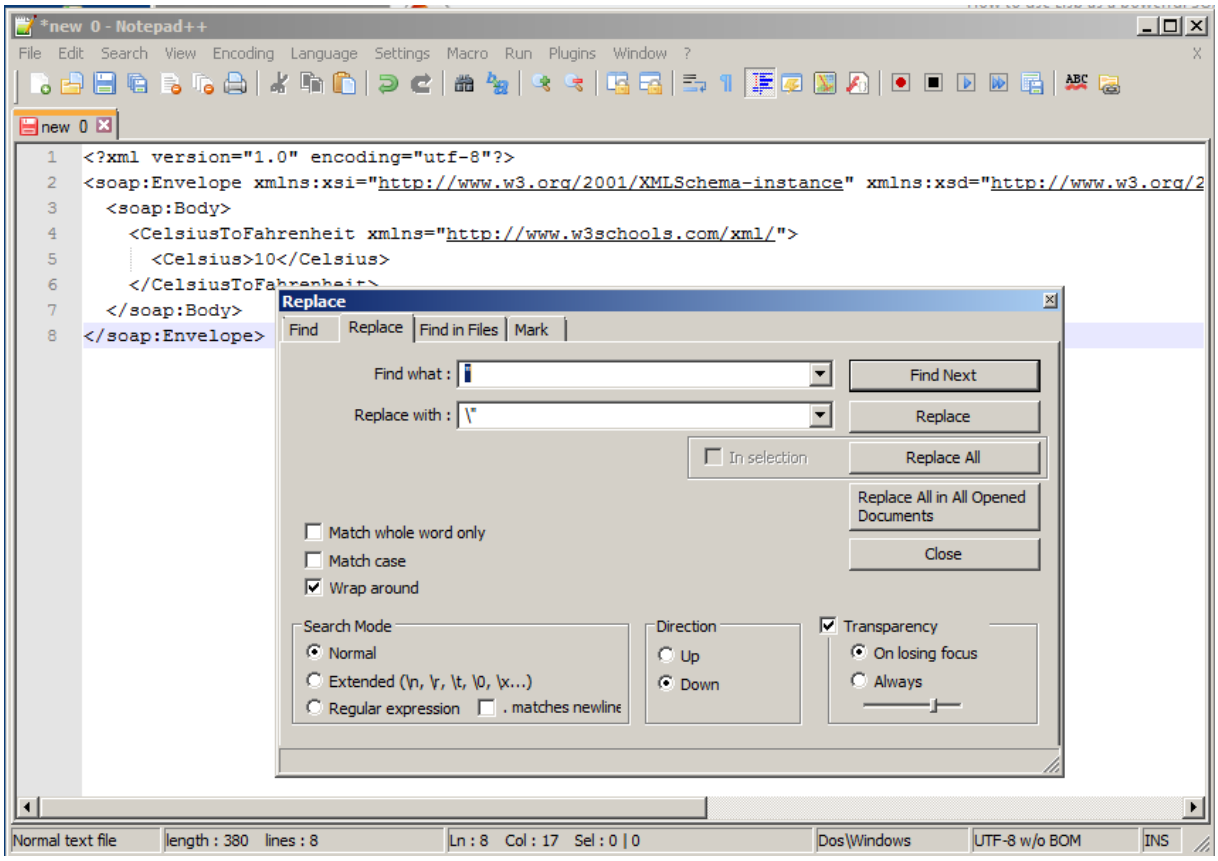


(The generated empty example request in SoapUI was wrong, which is why I replaced it with a copy of the XML example request from the W3S Page http://www.w3schools.com/xml/tempconvert.asmx?op=CelsiusToFahrenheit):



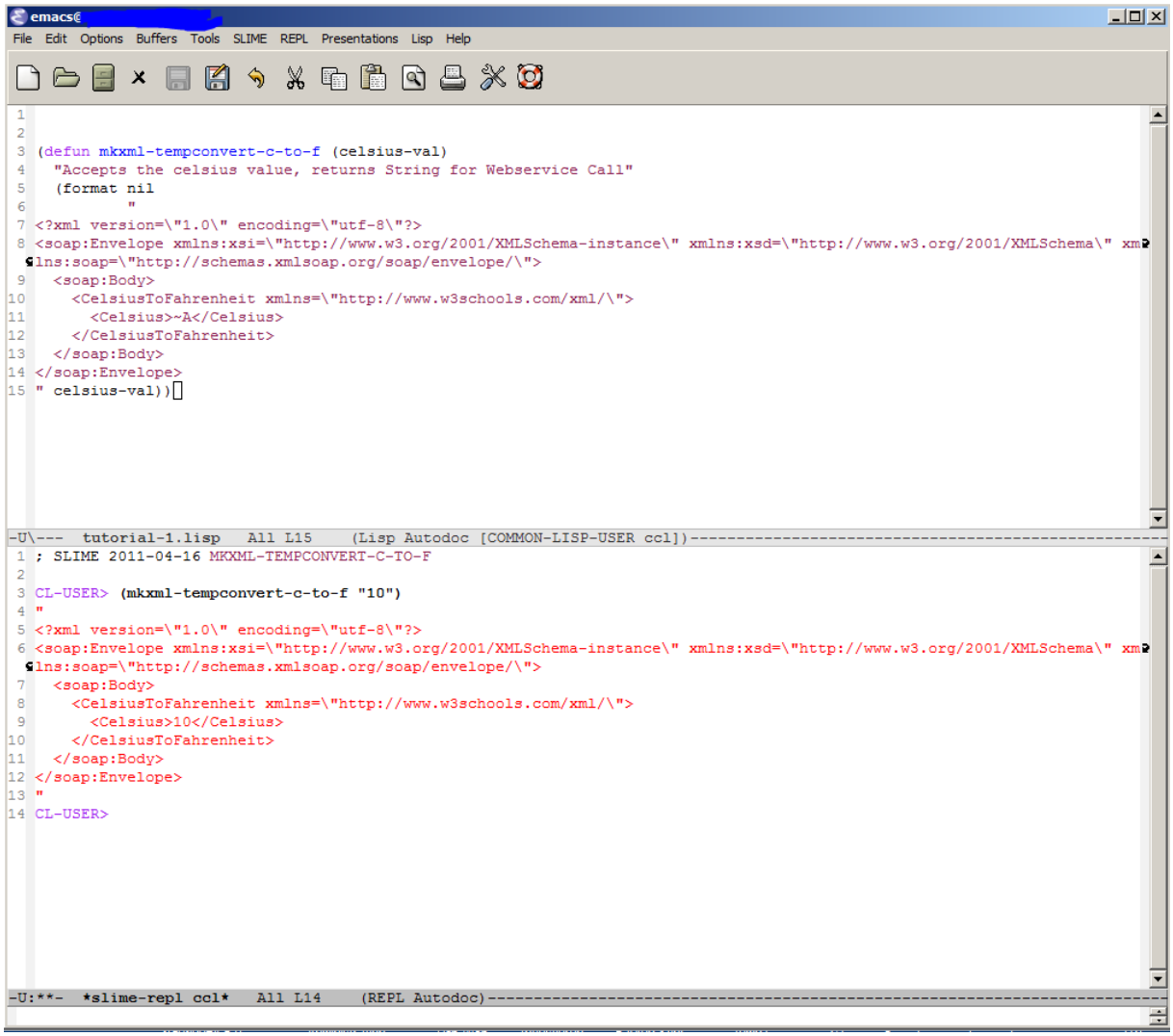
In the above example, I only had to set one variable (the Celsius degree value) and was then able to execute the call. The server returned the response (as displayed on the right.) The "10" was converted into "50".

We will use this example call as input for our next step. We need to replace all `"` with the escaped version for Lisp, which is `\"`. I use Notepad++ to do this:

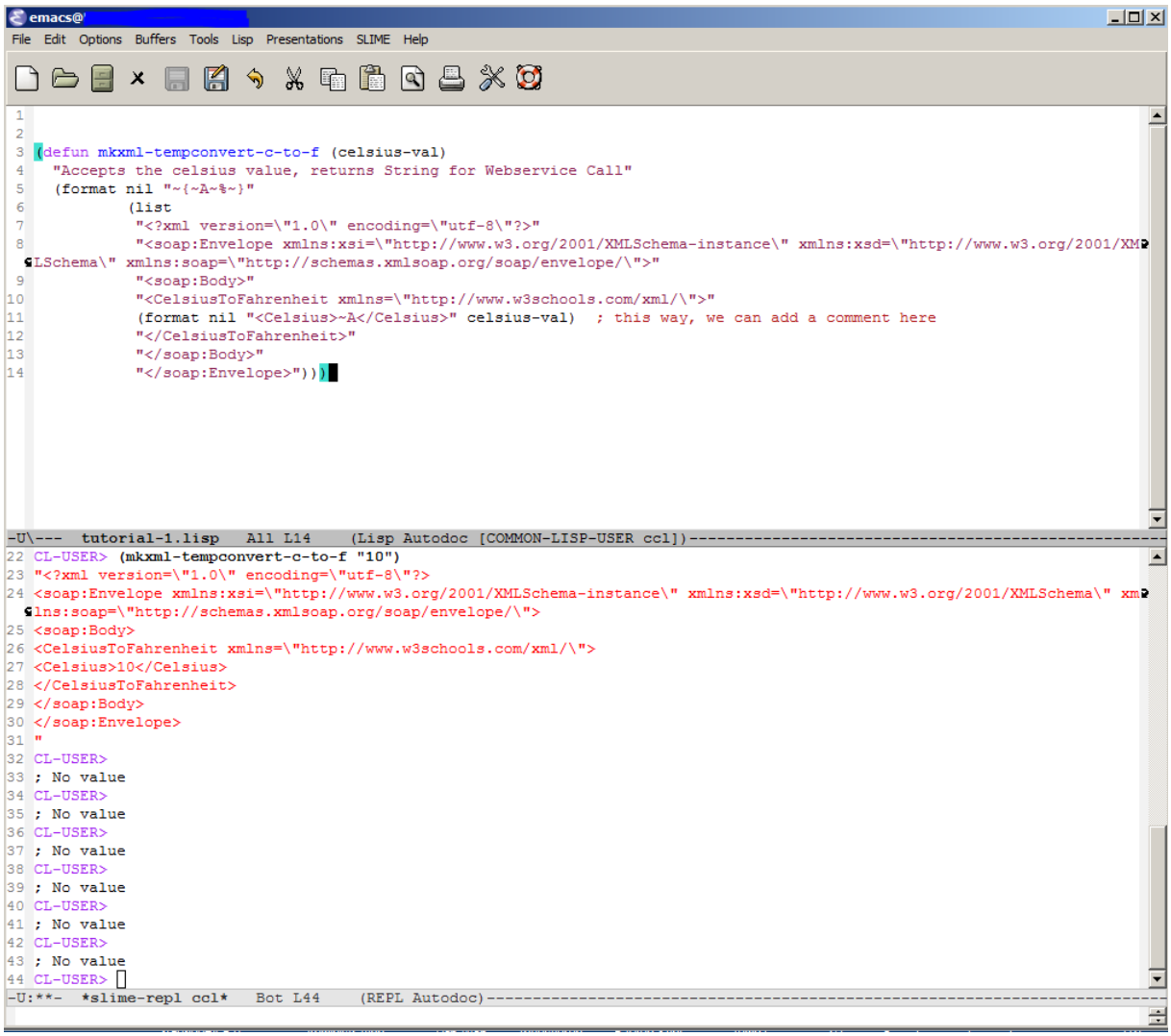


In this simplified approach, instead of using an XML library to model the complete, possibly complex XML structure of requests, we manually construct/modify only the string object to set the required variables for our test. This may not be sufficient for all use-cases; however, I have found it to serve the purpose at most simple test cases.

Now we have the material to build a Lisp function which accepts our variable and returns a string which represents the request and can be sent to the server. We call it mk-xml- …, short for "make-xml". I now simply copy-and-paste from the Notepad++ and replace the value with the ~A format directive:



I used a simple "format" function call. If you have to replace more values in the call, you can run out of format arguments. In that case, I would recommend to use "format" to accept a list of strings; for those lines with variables, you replace the value with another format function. Finally, you re-construct the complete string object like this:



We can now create the web service call based on the functional input values.

Of course, you could use a DB Library (e.g. CLSQL) to fetch example values from your database or do other queries to construct your call, or you could interactively request input from your REPL.

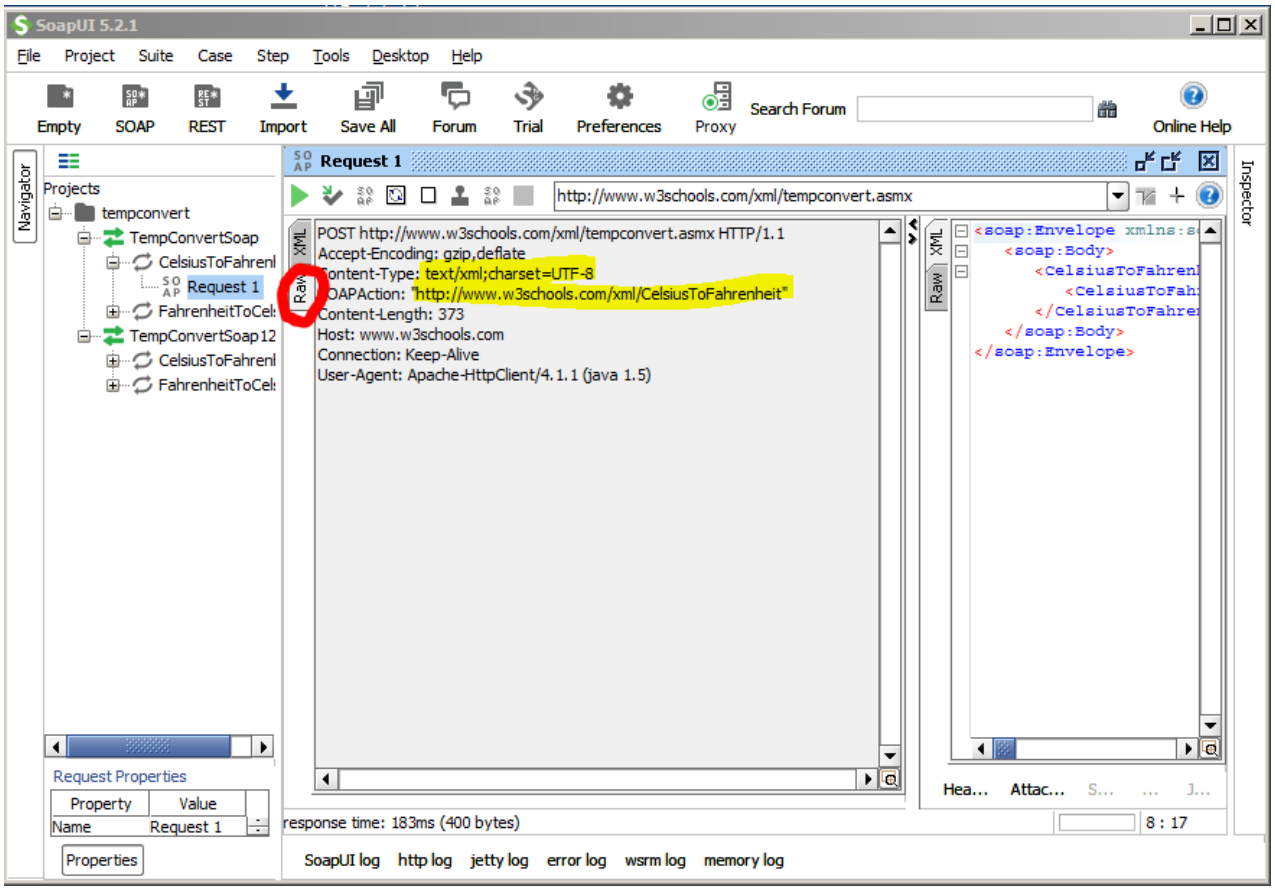
1.2 Creating the Web Service Call

As a next step, we need to send the call as a web service post via HTTP to the web service endpoint.
To do so, we use Edi Weitz´s HTTP client Drakma:

```
CL-USER> (quicklisp:quickload "DRAKMA")
```

We will have to send POST data to the web service endpoint, so we need a function which takes the URI and a string object as input. Optionally, we might have to provide a web service method which is an additional header attribute.

Now take a look at the "raw" request in SaopUI. We need to construct something in DRAKMA which looks like the raw request:

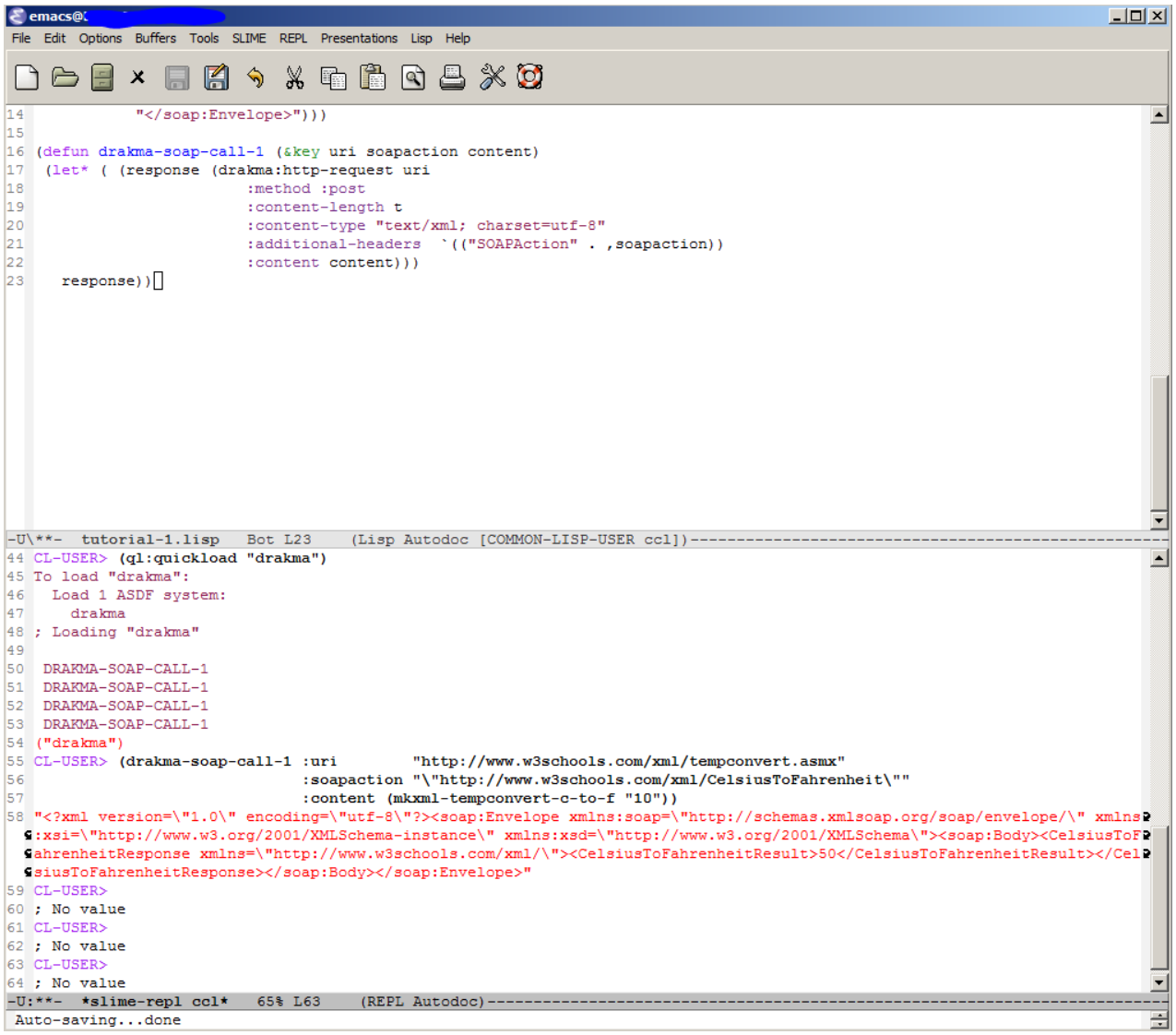


This is the first version of the function I wrote:

```
(defun drakma-soap-call-1 (&key uri soapaction content)
 (let* ( (response (drakma:http-request uri
                    :method :post
                    :content-length t
                    :content-type "text/xml; charset=utf-8"
                    :additional-headers  `(("SOAPAction" . ,soapaction))
                    :content content)))
   response))
```

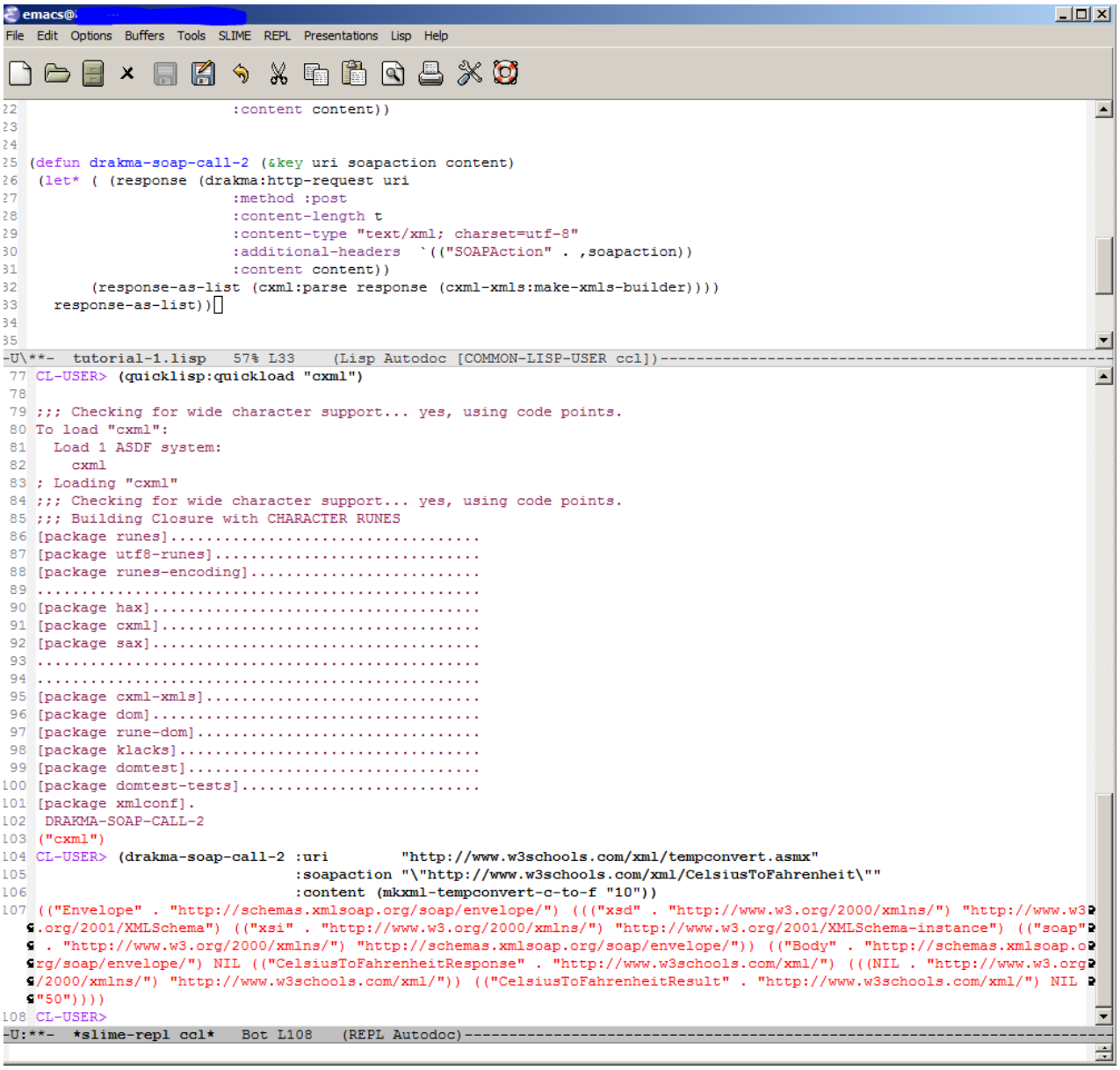
As you can see, the function accepts three parameters: the web service endpoint, the action parameter, and the content. For the content, we wrote our XML creation function. Drakma returns the server response simply as a string.

The following is an example call including the server response. You will find all required information in your successful SoapUI example call:



Drakma returns more than just the server response; please refer to the documentation if you like to check e.g. the response code or meta-information.

If you would like to get a list as a response, you could convert the server response from an XML structure to a list structure with cxml (this is version 2 of our Drakma Soap Call function):

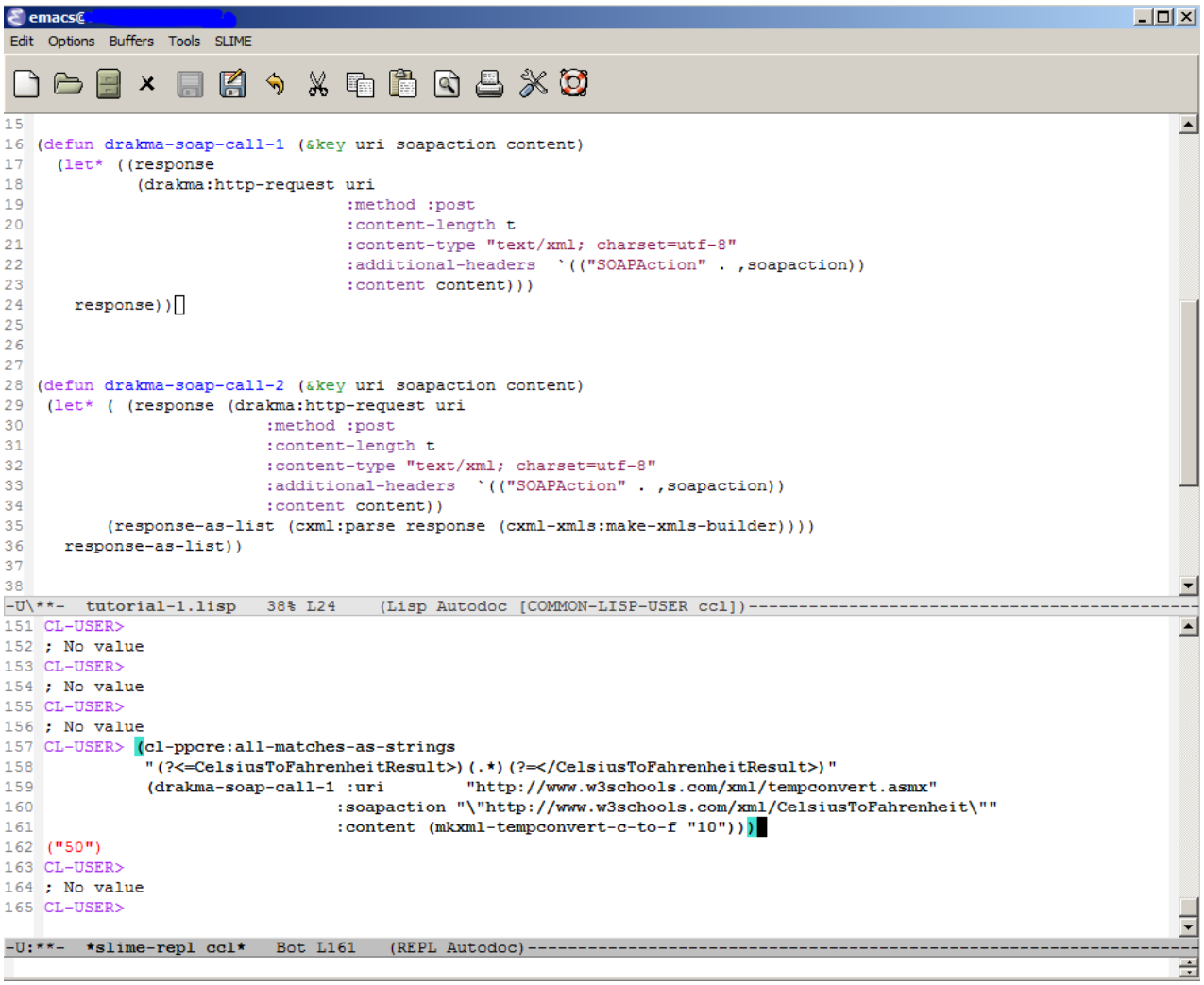


Or, if you only do some testing and a "quick hack" is sufficient, you might want to use a regular expression on the server response string object to extract the interesting value or values. Edi Weitz´s cl-ppcre allows that. In our case, we want to extract the temperature value of the response. In order to do so we build a regex with a left and a right delimiter (the XML tags) and grab the contents in between. Here, I used all-matches-as-strings:

```
(cl-ppcre:all-matches-as-strings
          "(?<=CelsiusToFahrenheitResult>)(.*)(?=</CelsiusToFahrenheitResult>)"
          (drakma-soap-call-1 :uri         "http://www.w3schools.com/xml/tempconvert.asmx"
                              :soapaction "\"http://www.w3schools.com/xml/CelsiusToFahrenheit\""
                              :content (mkxml-tempconvert-c-to-f "10")))
```

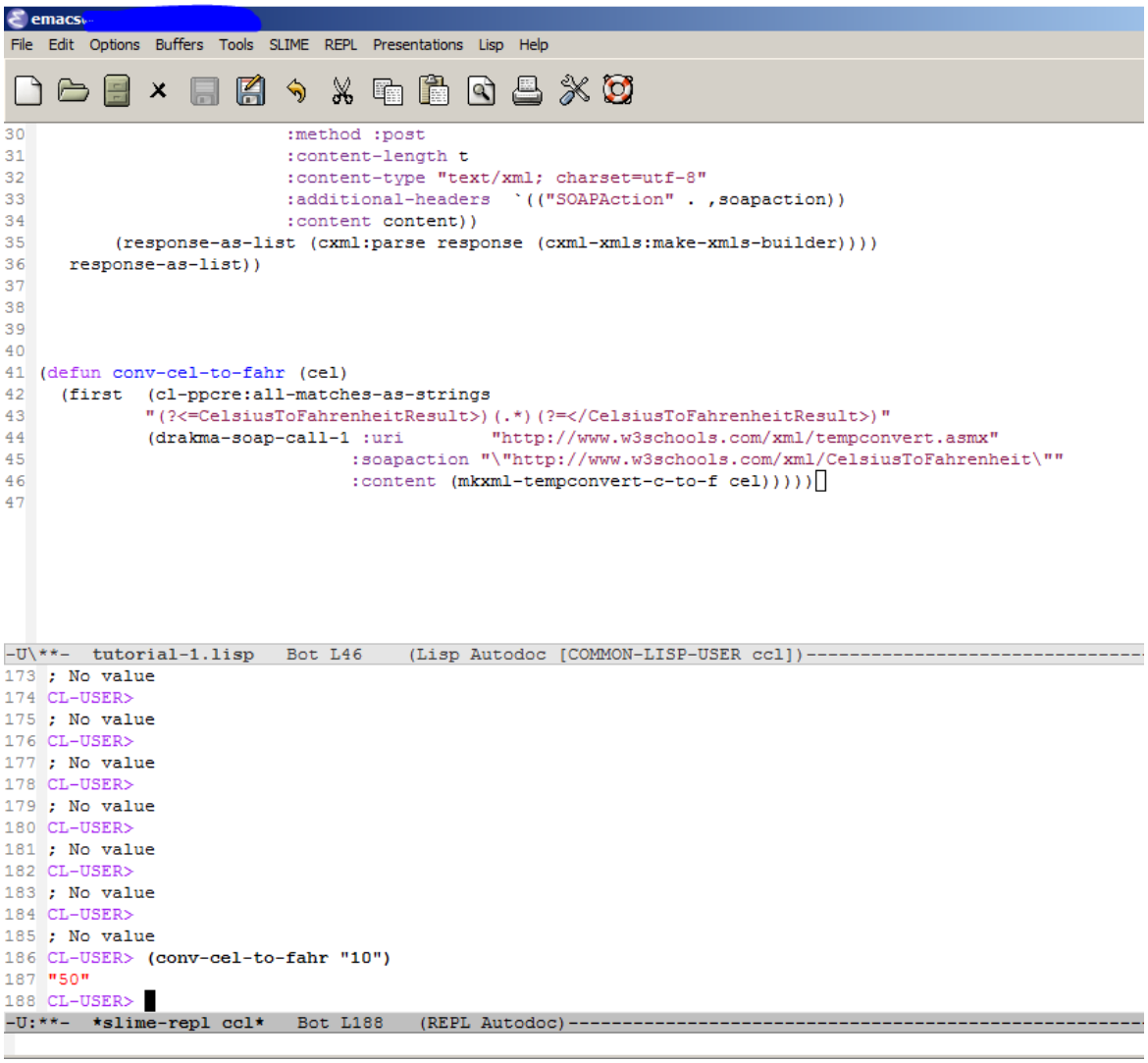
Of course, the analyst has to be aware of the assumptions made when extracting information like this.

So when using our first version with the regex it looks as follows:



1.3 Adding Abstraction

We can now build an abstract version of this web service call which hides all details:



Now we can test multiple values with just two lines like these:

```
CL-USER> (dolist (cel (list "5" "10" "15" "20"))
           (format t "~A~%" (conv-cel-to-fahr cel)))
41
50
59
68
NIL
CL-USER>
```

Of course, this temperature service is a very trivial example, but it should show the idea.

1.4 More

- You can build variations of the drakma-soap-call function, if your content-type is different; if your server response is binary, you may convert the response back to a string object via (flexi-streams:octets-to-string …) (see Drakma documentation.)
- If you query multiple backend systems, you might want to handle a timeout situation, e.g. with the Drakma built-in connection time-out.
- Debugging: First try to get the web service call through with SoapUI; if this doesn´t work, ask the developers. As a next step, check the raw request and make sure Drakma is building an identical request.

I hope you liked this short tutorial. Abstraction is the key to more powerful, productive testing. With the tools described above you can build a complete set of automated test web service calls. You can combine calls, or add DB-checking or values from databases. And Lisp is very practical for this purpose.

Thanks

Over the last ten years, I´ve spent time learning Lisp. This would not have been possible without the work of some people. Of all the many sources I've used and people I´ve learned from, I´d like to particularly thank (in alphabetical order):
Zach Beane (Quicklisp), Paul Graham ("ANSI Common Lisp", "On Lisp"), Rainer Joswig (many articles, especially his explanations on StackOverflow), David B. Lamkins ("Successful Lisp"), Peter Seibel ("Practical Common Lisp") and Edi Weitz (his libraries and his starter-kit.)

Thanks to Jessie Dietz for the linguistic revision.

Source

```
; SOAP / Webservices with Lisp
; Sebastian K. Glas 2016

(quicklisp:quickload :drakma)
(quicklisp:quickload :cl-ppcre)
(quicklisp:quickload :cxml)


(defun mkxml-tempconvert-c-to-f (celsius-val)
  "Accepts the celsius value, returns String for Webservice Call"
  (format nil "~{~A~%~}"
          (list
           "<?xml version=\"1.0\" encoding=\"utf-8\"?>"
           "<soap:Envelope xmlns:xsi=\"http://www.w3.org/2001/XMLSchema-instance\"
xmlns:xsd=\"http://www.w3.org/2001/XMLSchema\"
xmlns:soap=\"http://schemas.xmlsoap.org/soap/envelope/\">"
           "<soap:Body>"
           "<CelsiusToFahrenheit xmlns=\"http://www.w3schools.com/xml/\">"
           (format nil "<Celsius>~A</Celsius>" celsius-val)  ; this way, we can add a
comment here
           "</CelsiusToFahrenheit>"
           "</soap:Body>"
           "</soap:Envelope>")))

(defun drakma-soap-call-1 (&key uri soapaction content)
  (let* ((response
```

```
        (drakma:http-request uri
                             :method :post
                             :content-length t
                             :content-type "text/xml; charset=utf-8"
                             :additional-headers  `(("SOAPAction" . ,soapaction))
                             :content content)))
    response))



(defun drakma-soap-call-2 (&key uri soapaction content)
 (let* ( (response (drakma:http-request uri
                     :method :post
                     :content-length t
                     :content-type "text/xml; charset=utf-8"
                     :additional-headers  `(("SOAPAction" . ,soapaction))
                     :content content))
       (response-as-list (cxml:parse response (cxml-xmls:make-xmls-builder))))
   response-as-list))



(defun conv-cel-to-fahr (cel)
  (first  (cl-ppcre:all-matches-as-strings
        "(?<=CelsiusToFahrenheitResult>)(.*)(?=</CelsiusToFahrenheitResult>)"
        (drakma-soap-call-1 :uri        "http://www.w3schools.com/xml/tempconvert.asmx"
                            :soapaction
"\"http://www.w3schools.com/xml/CelsiusToFahrenheit\""
                            :content (mkxml-tempconvert-c-to-f cel)))))
```