Interoperable WSDL/SOAP web services introduction: Python ZSI, Excel XP, gSOAP C/C++ & Applix SS

Holger Joukl
LBBW Financial Markets Technologies

22nd July 2005

Abstract

Despite the hype & buzzword-storm, building web services servers and clients is still not as easy as promised. This is partly due to the relative newness of technology. For the most part, though, this stems from the actual complexness of the protocols/specs, the toolkit magic behind which this complexness is hidden, and the documentation gaps that exist for the toolkits. This document is intended to be a step-by-step tutorial/practice report, to ease the use for web services newcomers (like me).
It features

- the Python ZSI module that is used to build the server side machinery and
- several clients that access the exposed services from
  - Python (ZSI)
  - MS Excel XP (XP Web Services Toolkit 2.0, VB)
  - C/C++ (gSOAP)
  - Applix spreadsheets (gSOAP+ELF)

Contents

1 Introduction **3**

2 Interlude: ZSI 1.6.1 code modifications **4**

3 Simple datatypes: The rpc/literal SquareService **4**

3.1 The SquareService WSDL . . . 4

3.2 A Python ZSI server for the SquareService . . . 5

3.2.1 Generating stubs from WSDL . . . 5

3.2.2 Writing the SquareService web server . . . 6

3.2.3 Hooking-in the service implementation . . . 7

3.3 A Python ZSI client for the SquareService . . . 9

3.4 An Excel XP Visual Basic client for the SquareService . . . 11

3.4.1 VB stub generation . . . 11

3.4.2 VB client code & spreadsheet access . . . 11

3.5 A gSOAP C++ client for the SquareService . . . 11

3.5.1 Generation from WSDL . . . 12

3.5.2 Client implementation . . . 13

3.5.3 gSOAP client compilation . . . 14

4 Strucured datatypes: The rpc/literal DateService **14**

4.1 The DateService WSDL . . . 15

4.2 A Python ZSI DateService server . . . 16

4.2.1 Code generation from WSDL . . . 16

4.2.2 The DateService web server . . . 17

4.2.3 The DateService implementation . . . 17

4.3 A Python ZSI client for the DateService . . . 19

4.4 An Excel XP Visual Basic client for the DateService . . . 24

4.4.1 VB stub generation . . . 24

4.4.2 VB client implementation & spreadsheet access . . . 24

4.4.3 Web Service References Toolkit pitfalls . . . 28

4.5 A gSOAP C++ client for the DateService . . . 29

4.5.1 Code generation from WSDL . . . 29

4.5.2 Client implementation . . . 30

4.5.3 gSOAP Client compilation . . . 31

4.6 Applix spreadsheets as DateService clients . . . 31

4.6.1 Code generation from WSDL . . . 31

4.6.2 Applix client implementation . . . 31

5 A document/literal service: The FinancialService 43

5.1 The FinancialService WSDL . . . 43

5.2 A Python ZSI FinancialService server . . . 45

5.2.1 Code generation from WSDL . . . 45

5.2.2 The DateService web server . . . 45

5.2.3 The FinancialService implementation . . . 48

5.3 A Python ZSI client for the FinancialService . . . 49

5.4 An Excel XP VB client for the FinancialService . . . 52

5.4.1 VB FinancialService stub generation . . . 52

5.4.2 VB FinancialService client implementation & spreadsheet access . . . 52

5.5 A gSOAP C++ client for the FinancialService . . . 53

5.5.1 Code generation from WSDL . . . 53

5.5.2 Client implementation . . . 54

5.5.3 gSOAP Client compilation . . . 54

6 Aftertoughts 55

A ZSI 1.6.1 patches 55

1 Introduction

We assume the reader is familiar with Python, C/C++ and/or MS Excel/VisualBasic and/or Applix to a certain degree. The web service server components are implemented in Python, but any of the client side technologies can be skipped if not of interest.

While some basic concepts regarding WSDL, SOAP, HTTP servers are presented here implicitly, this document is not a tutorial on these. If you want to know more there´s plenty of stuff on the web.

The code examples have been developed with the primary goal to “make it work at all”, in a learning-by-doing manner. Thus there is lots of room for enhancements, e.g. getting rid of hardcoded path names etc.

Throughout this document, certain host names (“dev-b.handel-dev.local”) or ports (“8080”) are used in the examples- you will have to substitute those with the appropriate setup for your site, of course. Naturally, this also affects all URLs defined in the example WSDLs and used to retrieve these WSDLs.

These are the toolkit versions discussed here:

- Python 2.3.4
- PyXML 0.8.3
- ZSI 1.6.1
- gcc 2.95.2, gcc 3.4.3
- gSOAP 2.7.1
- MS Office XP Web Services Toolkit 2.0

Conceptually, we use a WSDL-centric approach: The starting point for all example service and client implementations will be the WSDL description. Note that this might differ from certain toolkits that start out with the service implementation in the host language and generate the WSDL for you to expose the implemented service. We regard the latter to have a tendency to not promote interoperability and to tie in implementation language details, which is certainly not what we want.[1]

Striving for interoperability, only the WS-I-compliant rpc/literal and document/literal WSDL styles are presented here.

The DateService WSDL (sect. 4.1) and worker code (sect. 4.2.3) and the Applix/ELF macro code (sect. 4.6.2) are courtesy of Rainer Kluger (LBBW Financial Markets Technologies).

2 Interlude: ZSI 1.6.1 code modifications

The Python server and client side implementations are based on the ZSI 1.6.1 module.[2] Some modifications had to be made in order to get everything to work. This might not be necessary for newer versions of ZSI, so it´s well worth trying to create or access your services *without* modifying the ZSI code base.

If needed, the necessary patches to ZSI 1.6.1 for the following example services can be found in addendum A.

3 Simple datatypes: The rpc/literal SquareService

This first example will implement an overly simple service that exposes a function which takes a `double` argument and returns the square of it (x^2) as a `double`. I.e. this examples uses simple scalar datatypes, one single argument and one single return value.

3.1 The SquareService WSDL

This is the WSDL file that determines the contract for the SquareService, called `SquareService.wsdl`:

```
<?xml version="1.0"?>
<definitions name="SquareService"
 targetNamespace="http://dev-b.handel-dev.local:8080/SquareService"
 xmlns:tns="http://dev-b.handel-dev.local:8080/SquareService"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
 xmlns="http://schemas.xmlsoap.org/wsdl/">
    <message name="getSquareRequest">
        <part name="x" type="xsd:double"/>
    </message>
    <message name="getSquareResponse">
        <part name="return" type="xsd:double"/>
    </message>
    <portType name="SquarePortType">
        <operation name="getSquare">
            <documentation> the square method </documentation>
            <input message="tns:getSquareRequest"/>
```

[1]This came up partly due to the fact that the chosen server implementation (Python ZSI) offers no such tool and partly as a gut feeling. Since then, this opinion has grown stronger and has also been backed up by several practitioners´ readings at a recent conference (Stuttgarter Softwaretechnik Forum 2005, Stuttgart-Vaihingen, Germany).

[2]The attempt has been made to move everything on to ZSI 1.7, but unfortunately the first try on the FinancialService resulted in a `wsdl2py` error (SF Tracker # 1241503), so this has been put off for the moment.

```
            <output message="tns:getSquareResponse"/>
        </operation>
    </portType>
    <binding name="SquareBinding" type="tns:SquarePortType">
        <soap:binding style="rpc" trans-
port="http://schemas.xmlsoap.org/soap/http"/>
        <operation name="getSquare">
            <soap:operation
             soapAction="http://dev-b.handel-
dev.local:8080/SquareService/getSquare"/>
            <input>
                <soap:body use="literal"
                 namespace="http://dev-b.handel-dev.local:8080/SquareService"/>
            </input>
            <output>
                <soap:body use="literal"
                 namespace="http://dev-b.handel-dev.local:8080/SquareService"/>
            </output>
        </operation>
    </binding>
    <service name="SquareService">
        <documenta-
tion>Returns x^2 (x**2, square(x)) for a given float x</documentation>
        <port name="SquarePort" binding="tns:SquareBinding">
            <soap:address location="http://dev-b.handel-
dev.local:8080/SquareService"/>
        </port>
    </service>
</definitions>
```

Comments:

- The `style` "rpc" and the `use` "literal" are used, to be WS-I-compliant. WS-I only supports rpc/literal and document/literal.

3.2 A Python ZSI server for the SquareService

The Python ZSI package [1] is one of two pywebsvcs packages implementing web services for Python, namely SOAP messaging and WSDL capabilities. It is powerful and very easy to get started with, but lacks some documentation enhancements when it comes to WSDL-driven service generation. While the tools to do that are already there, documentation is sparse and examples are hard to find. We hope to close the gap a bit in the next section.

All examples here are based on ZSI 1.6.1.

3.2.1 Generating stubs from WSDL

ZSI comes with two python scripts to generate code from a WSDL file:

- `wsdl2py` is used to generate python bindings for the service.
- `wsdl2dispatch` generates a server frame for service dispatch where the actual worker functions will be hooked into.

If you have installed ZSI on top of your python installation you can invoke the scripts like this (change your installation base path according to your setup):[3]

1. `wsdl2py`:

   ```
   /apps/pydev/bin/wsdl2py -f SquareService.wsdl
   ```

   This will generate the file `SquareService_services.py`.

2. `wsdl2dispatch:`

   ```
   /apps/pydev/bin/wsdl2dispatch -f SquareService.wsdl
   ```

   This will generate the file `SquareService_services_server.py`.

What do we have now? We have bindings to work with the services in python and a skeleton for dispatching to the actual worker methods. What we

still need is

- the main program that runs a (HTTP-) server with a request handler for the services and
- the hooks to invoke the worker methods.

Luckily, ZSI includes the ZSI.ServiceContainer module which implements the server for us.

3.2.2 Writing the SquareService web server

This is our main program `mySquareServer.py`. It basically provides a request handler addition, puts our service into the ServiceContainer and starts the HTTP server on a given port:

```
#! /apps/pydev/bin/python2.3
from ZSI.ServiceContainer import ServiceContainer, SOAPRequestHandler
from SquareService_services_server import SquareService
import os
class MySOAPRequestHandler(SOAPRequestHandler):
    """Add a do_GET method to return the WSDL on HTTP GET requests.
    Please note that the path to the wsdl file is derived from what
    the HTTP invocation delivers (which is put into the self.path
    attribute), so you might want to change this addressing scheme.
    """

    def do_GET(self):
        """Return the WSDL file. We expect to get the location from the
        invocation URL ("path").
        """
        wsdlfile = os.path.join('.', self.path.replace('/', "", 1) + ".wsdl")
        print ">>>>> using wsdlfile", wsdlfile
        wsdl = open(wsdlfile).read()
        self.send_xml(wsdl)
# Copied from ZSI.ServiceContainer, extended to instantiate with a custom
# request handler
def AsServer(port=80, services=(), RequestHandlerClass=SOAPRequestHandler):
    ■'port --
```

[3]The installation base path for all examples here is /apps/pydev/.

```
        services -- list of service instances
    ■'
    address = (■, port)
    sc = ServiceContainer(address, RequestHandlerClass=RequestHandlerClass)
    for service in services:
        path = service.getPost()
        sc.setNode(service, path)
    sc.serve_forever()
AsServer(port=8080, services=[SquareService()], RequestHandler-
Class=MySOAPRequestHandler)
```

We wouldn´t have needed to write the custom request handler `MySOAPRequestHandler` if not for the do_GET method. But both Python ZSI clients using the `ServiceProxy` class and MS VisualBasic SOAP clients expect to receive the WSDL when issueing HTTP GET, which is actually common behaviour to get the service description (apart from UDDI).

Similarly, the `AsServer(...)` function had to be extended to make use of our custom request handler.

3.2.3 Hooking-in the service implementation

The only thing left now is to hook the implementation of the service into the generated server skeleton. We need to

- dispatch to the correct service method,
- feed it the arguments received via a SOAP request and
- set the return values for the SOAP response.

This is the implementation for the SquareService getSquare method (or operation, in WSDL terms):

```
from SquareService_services import *
from ZSI.ServiceContainer import ServiceSOAPBinding
class SquareService(ServiceSOAPBinding):

    # This dictionary is used to dispatch to the appropriate method.
    # Not that the dict key(s) are identical to the soapAction attributes of the
    # <soap:operation ...> field(s) in the WSDL:
    # ...
    # <soap:operation
    # soapAction="http://dev-b.handel-dev.local:8080/SquareService/getSquare"/>
    # ...
    # The value(s) for the key(s) are the generated soap_<...> method names.
    soapAction = {
        'http://dev-b.handel-
dev.local:8080/SquareService/getSquare': 'soap_getSquare',
        }

    def __init__(self, post='/SquareService', **kw):
        ServiceSOAPBinding.__init__(self, post)

    def soap_getSquare(self, ps):
        # input vals in request object
        # MANUALLY CORRECTED:
        # args = ps.Parse( getSquareRequestWrapper() )
        # Use the class instead of an instance of the class.
        # Note: The erroneous code generation happens for rpc/literal, but not
        # for rpc/encoded, where using an instance works (?).

        args = ps.Parse( getSquareRequestWrapper )

        # assign return values to response object
        response = getSquareResponseWrapper()

        # >>> ADDED MANUALLY
        # Here we hook in the actual worker method
        response._return = self.getSquare(args._x)
        # <<<

        return response

    # the (handwritten) worker code
    def getSquare(self, x):
        """Return square(x).
        """
        return x**2
```

Note that ZSI does almost all the work for us, again. The only additions we had to make are:

- Implementing the `getSquare(...)` worker method. We could also have invoked a function, used a lambda, put the worker code into `soap_getSquare`, etc.
- Hooking `getSquare` in. This is done in the

  ```
  ...
  # >>> ADDED MANUALLY
  # Here we hook in the actual worker method
  response._return = self.getSquare(args._x)
  # <<<
  ...
  ```

  bits where

  – the `x` input argument is taken from the incoming SOAP request and handed to the `getSquare` method

  – the `return` field of the SOAP response message to be sent out is set with the `getSquare` result

  As you can see in the WSDL above the "getSquareRequest" message has a "part" with the name "x"; ZSI exposes this as attribute "_x" of the incoming parsed SOAP request message "args" instance. The same applies to the "return" part of the response message. ZSI exposes this to python as attribute "_return" of the `getSquareResponseWrapper` instance.

- Correcting the line

  ```
  # args = ps.Parse( getSquareRequestWrapper() )
  ```

  to

  ```
  args = ps.Parse( getSquareRequestWrapper )
  ```

  This seems to be a bug in the code generation.[4]

[4] When experimenting with rpc/encoded-style first, the generated code line worked "as is".

Comments:

- The dispatch to the appropriate service operation is handled in the `soapAction` dictionary. This dictionary maps the action that is requested to the method that is invoked. The ZSI standard request handler (that we inherit from) takes the HTTP header field soapAction and propagates its value to this dispatch mechanism. Thus, everything works out-of-the-box if you use the "soapAction" operation-attribute in your WSDL file (and if your service client actually provides this header field with its request). If you want to use different ways to dispatch (e.g. by using the toplevel node of the incoming SOAP body) you will have to make further modifications to the ZSI standard request handler.

3.3 A Python ZSI client for the SquareService

We implement a client that calls getSquare from the SquareService in `myServiceProxyClient.py` as follows:

```
#!/apps/pydev/bin/python2.3
import sys
import getopt
from ZSI import ServiceProxy
#------------------------------------------------------------------------------
# default configuration
#------------------------------------------------------------------------------
port = 8080
host = 'dev-b'
#------------------------------------------------------------------------------
# command line parsing
#------------------------------------------------------------------------------
def usage(rcode=1):
    print "usage: myServiceProxyClient.py [--host=<hostname> --port=,-c<port> --
help, -h]"
    sys.exit(rcode)
try:
    optlist, args = getopt.getopt(sys.argv[1:], "hp:", ['help', 'port='])
except getopt.GetoptError:
    usage()
for opt, arg in optlist:
    print opt, arg
    if opt in ["-h", "--help"]:
        usage(0)
    elif opt in ["--host"]:
        host = arg
        continue
    elif opt in ["-p", "--port"]:
        port = int(arg)
        continue
#------------------------------------------------------------------------------
# Service client code
#------------------------------------------------------------------------------

url = 'http://' + host + ':' + str(port) + '/SquareService'
# Hmm, if we want to send to the correct service location we
# must set use_wsdl.
service = ServiceProxy(url, use_wsdl=True, tracefile=sys.stdout)
print 'service is', service
print service.__dict__
print '\nAccessing service getSquare...'
while 1:
    # Must use keyword arguments if use_wsdl was set
    x = float(raw_input("Enter number: "))
    result = service.getSquare(x=x)
    print 'result:', result
```

This is pretty straightforward. Most of the code handles the command line stuff which has nothing to do with web services in the first place. The only thing we actually have to do is to create a `ServiceProxy` instance with the service URL and tell it to use the WSDL (which it gets from a HTTP GET request). We can then simply access the getSquare method; only make sure you give its argument as a keyword argument.

Note that this client does not even use the generated stub code: Everything is accessed through the `ServiceProxy` object that gets all necessary information from the WSDL.

This is the output of an example client run:

```
$ ./myServiceProxyClient.py
service is <ZSI.ServiceProxy.ServiceProxy instance at 0x18dbe8>
{'_wsdl': <ZSI.wstools.WSDLTools.WSDL instance at 0x192350>,
'getSquare': <ZSI.ServiceProxy.MethodProxy instance at 0x48d698>,
'_typesmodule': None, '_soapAction': None, '_use_wsdl': True,
'_ns': None,
'_tracefile': <open file '<stdout>', mode 'w' at 0x128060>,
'_op_ns': None, '_name': u'SquareService', '_nsdict': {},
'_port': <ZSI.wstools.WSDLTools.Port instance at 0x48d5d0>,
'_service': <ZSI.wstools.WSDLTools.Service instance at 0x48d580>,
'__doc__': u'Returns x^2 (x**2, square(x)) for a given float x'}
Accessing service getSquare...
Enter number: 4
_________________________________ Tue Jul  5 13:30:42 2005 REQUEST:
<?xml version="1.0" encoding="utf-8"?>
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:ZSI="http://www.zolera.com/schemas/ZSI/"
  SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" >
<SOAP-ENV:Body>
<getSquare>
<x xsi:type="xsd:double">4.000000</x>
</getSquare>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
_________________________________ Tue Jul  5 13:30:42 2005 RESPONSE:
Server: ZSI/1.1 BaseHTTP/0.3 Python/2.3.4
Date: Tue, 05 Jul 2005 11:30:42 GMT
Content-type: text/xml; charset="utf-8"
Content-Length: 560
<?xml version="1.0" encoding="utf-8"?>
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:ZSI="http://www.zolera.com/schemas/ZSI/"
  SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" >
<SOAP-ENV:Body>
<getSquareResponse  xmlns="">
<return xsi:type="xsd:double">16.000000
</return>
</getSquareResponse >
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
result: {u'return': 16.0}
Enter number:
```

3.4 An Excel XP Visual Basic client for the SquareService

3.4.1 VB stub generation

To generate the stub code you use the Web Service References Tool:

1. Open the VB Editor (Alt-F11).
2. Read the WSDL:
   (a) Tools–>Web Services References...
   (b) Click "Web Service URL" and enter the web service´s URL. In our example this is "http://dev-b.handel-dev.local:8080/SquareService" (same as the <soap:address location="..."/> in the WSDL).[5]Click "Search" (the server must be running, of course). You should see the SquareService with its single exposed method in the Search Results field now.
   (c) Activate the SquareService´s checkbox and press "Add". The stub code is being generated.

3.4.2 VB client code & spreadsheet access

To access the web service, we need to manually write VB functions/sub procedures.[6] For the SquareService with its trivial datatypes this is quite simple:

1. Insert–>Module
2. Add the module code. We implement a VB function for this purpose:

```
Public Function getSquare(ByRef x As Double) As Double
    Dim service As New clsws_SquareService
    getSquare = service.wsm_getSquare(x)
End Function
```

Excel spreadsheet invocation of the SquareService You can now use the `getSquare` function wherever a builtin function could be used, without further ado.

3.5 A gSOAP C++ client for the SquareService

gSOAP is an open source web services development toolkit for C/C++. It seems to be very mature and complete and has an impressive record of being used in real-world applications by major companies. It also features good documentation.

[5]Note that the server name and port is not sufficient for our server implementation as we use the RequestHandler path attribute to find the appropriate WSDL file to serve.

[6]Not quite sure about the terminology here - this has been the first exposure to VB. Consult your local VB guru for all VB-related questions; all code presented here has been developed by trial-and-error.

3.5.1 Generation from WSDL

With our SquareService server running, we can generate the client stubs:

1. First, gSOAP needs a header file for the service that is create with the `wsdl2h` generator:

   ```
   /apps/pydev/bin/wsdl2h -o squareService.h
   http://dev-b.handel-dev.local:8080/SquareService
   ```

   Just specify the name for the header file and the URL where the WSDL can be received with a HTTP GET request. The `squareService.h` header will be created:

   ```
   ** The gSOAP WSDL parser for C and C++ 1.2.0
   ** Copyright (C) 2000-2005 Robert van Engelen, Genivia Inc.
   ** All Rights Reserved. This product is provided "as is", without any war-
   ranty.
   ** The gSOAP WSDL parser is released under one of the following two li-
   censes:
   ** GPL or the commercial license by Genivia Inc. Use option -
   l for more info.
   Saving squareService.h
   Connecting to 'http://dev-b.handel-
   dev.local:8080/SquareService' to retrieve WSDL... done
   Cannot open file 'typemap.dat'
   Problem reading type map file typemap.dat.
   Using internal type definitions for C++ instead.
   Warning: part 'x' uses literal style and must refer to an ele-
   ment rather than a type
   Warning: part 'x' uses literal style and must refer to an ele-
   ment rather than a type
   To complete the process, compile with:
   soapcpp2 squareService.h
   $ ls -l
   total 6
   -rw-r--r--   1 hjoukl   intern       5411 Jul  5 10:54 squareService.h
   ```

   Note that gSOAP tells us about what it thinks might cause problems in our WSDL; we ignore that for this example.

2. Next we let gSOAP create the stub code for us, using the newly-created header:

   ```
   $ /apps/pydev/bin/soapcpp2 -I /data/pydev/DOWNLOADS/gsoap-
   2.7/soapcpp2  squareService.h
   **  The gSOAP Stub and Skeleton Compiler for C and C++ 2.7.1
   **  Copyright (C) 2000-2005, Robert van Engelen, Genivia Inc.
   **  All Rights Reserved. This product is provided "as is", without any war-
   ranty.
   **  The gSOAP compiler is released under one of the following three li-
   censes:
   **  GPL, the gSOAP public license, or the commercial license by Genivia Inc.
   Saving soapStub.h
   Saving soapH.h
   Saving soapC.cpp
   Saving soapClient.cpp
   Saving soapServer.cpp
   Saving soapClientLib.cpp
   ```

```
    Saving soapServerLib.cpp
    Using ns1 service name: SquareBinding
    Using ns1 service style: document
    Using ns1 service encoding: literal
    Using ns1 service location: http://dev-b.handel-dev.local:8080/SquareService
    Using ns1 schema namespace: http://dev-b.handel-dev.local:8080/SquareService
    Saving soapSquareBindingProxy.h client proxy
    Saving soapSquareBindingObject.h server object
    Saving SquareBinding.getSquare.req.xml sample SOAP/XML request
    Saving SquareBinding.getSquare.res.xml sample SOAP/XML response
    Saving SquareBinding.nsmap namespace mapping table
    Compilation successful
```

We must explicitly give the `gSOAP/soapcpp2` directory as include directory. This is not being installed with the gSOAP installation (in the "make install" step) but resides in the path where you extracted the gSOAP tarball. It contains some special header files gSOAP does not install on your system.

You might have noticed that `soapcpp2` says it is using service style "document" as opposed to what´s defined in the WSDL ("rpc"); this seems to be a cosmetic issue only and does not affect the usability of the generated code.

The above command produces the following client stubs (server skeleton code also by the way, but we will not use it here):

```
$ ls -l
    total 407
    -rw-r--r--   1 hjoukl   intern        449 Jul  5 13:01 SquareBind-
    ing.getSquare.req.xml
    -rw-r--r--   1 hjoukl   intern        475 Jul  5 13:01 SquareBind-
    ing.getSquare.res.xml
    -rw-r--r--
    1 hjoukl   intern        561 Jul  5 13:01 SquareBinding.nsmap
    -rw-r--r--   1 hjoukl   intern      40483 Jul  5 13:01 soapC.cpp
    -rw-r--r--   1 hjoukl   intern       2450 Jul  5 13:01 soapClient.cpp
    -rw-r--r--
    1 hjoukl   intern        464 Jul  5 13:01 soapClientLib.cpp
    -rw-r--r--   1 hjoukl   intern      10935 Jul  5 13:01 soapH.h
    -rw-r--r--   1 hjoukl   intern       2846 Jul  5 13:01 soapServer.cpp
    -rw-r--r--
    1 hjoukl   intern        464 Jul  5 13:01 soapServerLib.cpp
    -rw-r--r--
    1 hjoukl   intern       1258 Jul  5 13:01 soapSquareBindingObject.h
    -rw-r--r--
    1 hjoukl   intern       1507 Jul  5 13:01 soapSquareBindingProxy.h
    -rw-r--r--   1 hjoukl   intern       5206 Jul  5 13:01 soapStub.h
    -rw-r--r--   1 hjoukl   intern       5411 Jul  5 10:54 squareService.h
```

What´s left now is to implement the client program and make use of the generated code.

3.5.2 Client implementation

This is a sample client to access the SquareService:

```
$ cat myCSquareClient.cpp
#include "soapH.h"
```

```
#include "SquareBinding.nsmap"
#include <iostream>
int main(void)
{
    struct soap soap;
    double x = 0;
    double square = 0;
    soap_init(&soap);
    while (1) {
        std::cout << "Enter x value: ";
        std::cin >> x;
        if (soap_call_ns1__getSquare(&soap, NULL, NULL, x, square) == SOAP_OK) {
            std::cout << "Result: " << square << std::endl;
        } else {
            soap_print_fault(&soap, stderr);
        }
    }
    soap_destroy(&soap);
    soap_end(&soap);
    soap_done(&soap);
    return 0;
}
```

There are also other ways to invoke this service with the gSOAP mechanisms, namely the `SquareBinding` class defined in `SquareServiceBindingProxy.h`, which would take care of the initialization & destruction activities needed in the above client code. We will use this (better) approach in the next examples.

3.5.3 gSOAP client compilation

gcc 2.95.2

```
$ g++ -o myCSquareClient -R/apps/prod/lib -I/apps/pydev/include
-L/apps/pydev/lib soapC.cpp soapClient.cpp myCSquareClient.cpp -lgsoap++ -
lsocket
```

gcc 3.4.3 Note: Compiling with gcc 3.4.3, the nsl library had to be added to the linked libraries:

```
/apps/local/gcc/3.4.3/bin/g++ -o myCSquareClient -R/apps/pydev/gcc/3.4.3/lib
-I/apps/pydev/gcc/3.4.3/include -L/apps/pydev/gcc/3.4.3/lib soapC.cpp
soapClient.cpp myCSquareClient.cpp -lgsoap++ -lsocket -lnsl
```

4 Strucured datatypes: The rpc/literal DateService

Let´s move on to a more elaborate service, elaborate in the sense of using structured datatypes now (not that the service example itself was particularly ingenious). Anyway, we will now implement the DateService service that exposes two methods:

- <date structure> getCurrentDate(<string>) takes a string argument and returns the current date as a datetime structure
- <date structure> getDate(<int>, <date structure>) takes an integer offset and a date structure as arguments and returns the given date plus the offset (in days) as a date structure

4.1 The DateService WSDL

The DateService is described in `DateService.wsdl`:

```
<?xml version="1.0"?>
<definitions name="DateService"
  targetNamespace="http://dev-b.handel-dev.local:8080/DateService.wsdl"
  xmlns:tns="http://dev-b.handel-dev.local:8080/DateService.wsdl"
  xmlns="http://schemas.xmlsoap.org/wsdl/"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:myType="DateType_NS"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">
  <types>
    <schema xmlns="http://www.w3.org/2001/XMLSchema"
      targetNamespace="DateType_NS" >
      <complexType name="Date">
            <sequence>
              <element name="year" nillable="true" type="xsd:integer"/>
              <element name="month" nillable="true" type="xsd:integer"/>
              <element name="day" nillable="true" type="xsd:integer"/>
              <element name="hour" nillable="true" type="xsd:integer"/>
              <element name="minute" nillable="true" type="xsd:integer"/>
              <element name="second" nillable="true" type="xsd:integer"/>
              <element name="weekday" nillable="true" type="xsd:integer"/>
              <element name="dayOfYear" nillable="true" type="xsd:integer"/>
              <element name="dst" nillable="true" type="xsd:integer"/>
            </sequence>
      </complexType>
    </schema>
  </types>
  <message name="getCurrentDateRequest">
    <part name="input" type="xsd:string"/>
  </message>
  <message name="getCurrentDateResponse">
    <part name="today" type="myType:Date"/>
  </message>
  <message name="getDateRequest">
    <part name="offset" type="xsd:integer"/>
    <part name="someday" type="myType:Date"/>
  </message>
  <message name="getDateResponse">
    <part name="day" type="myType:Date"/>
  </message>
  <portType name="DateService_PortType">
    <operation name="getCurrentDate">
      <input message="tns:getCurrentDateRequest"/>
      <output message="tns:getCurrentDateResponse"/>
    </operation>
    <operation name="getDate">
      <input message="tns:getDateRequest"/>
      <output message="tns:getDateResponse"/>
    </operation>
  </portType>
  <binding name="DateService_Binding" type="tns:DateService_PortType">
```

```
    <soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
    <operation name="getCurrentDate">
      <soap:operation soapAction="urn:DateService.wsdl#getCurrentDate"/>
      <input>
        <soap:body use="literal" namespace="urn:DateService.wsdl"/>
      </input>
      <output>
        <soap:body use="literal" namespace="urn:DateService.wsdl"/>
      </output>
    </operation>
    <operation name="getDate">
      <soap:operation soapAction="urn:DateService.wsdl#getDate"/>
      <input>
        <soap:body parts="offset someday" use="literal" names-
pace="urn:DateService.wsdl"/>
      </input>
      <output>
        <soap:body use="literal" namespace="urn:DateService.wsdl"/>
      </output>
    </operation>
  </binding>
  <service name="simple Date Service">
    <documentation>Date Web Service</documentation>
    <port name="DateService_Port" binding="tns:DateService_Binding">
      <soap:address location="http://dev-b.handel-dev.local:8080/DateService"/>
    </port>
  </service>
</definitions>
```

Comments:

- Again, rpc/literal has been chosen.
- A ComplexType “Date” is defined in the <types> section. This type is being used as a return type (getCurrentDate, getDate) and as a method argument type (getDate).

4.2 A Python ZSI DateService server

The tasks at hand are the same as for the SquareService example.

4.2.1 Code generation from WSDL

1. `wsdl2py`:

```
$ /apps/pydev/bin/wsdl2py -f DateService.wsdl
==> DateService_services.py
==> DateService_services_types.py
```

2. `wsdl2dispatch`:

```
/apps/pydev/bin/wsdl2dispatch -f DateService.wsdl
==> DateService_services_server.py
```

4.2.2 The DateService web server

The server implementation is exactly the same as in section 3.2.2, with the only difference of putting a DateService instance into the ServiceContainer now:

```
#! /apps/pydev/bin/python2.3
from ZSI.ServiceContainer import ServiceContainer, SOAPRequestHandler
from DateService_services_server import DateService
import os
class MySOAPRequestHandler(SOAPRequestHandler):
    """Add a do_GET method to return the WSDL on HTTP GET requests.
    Please note that the path to the wsdl file is derived from what
    the HTTP invocation delivers (which is put into the self.path
    attribute), so you might want to change this addressing scheme.
    """

    def do_GET(self):
        """Return the WSDL file. We expect to get the location from the
        invocation URL ("path").
        """
        wsdlfile = os.path.join('.', self.path.replace('/', "", 1) + ".wsdl")
        print ">>>>> using wsdlfile", wsdlfile
        wsdl = open(wsdlfile).read()
        self.send_xml(wsdl)
# Copied from ZSI.ServiceContainer, extended to instantiate with a custom
# request handler
def AsServer(port=80, services=(), RequestHandlerClass=SOAPRequestHandler):
    ■'port --
       services -- list of service instances
    ■'
    address = (■, port)
    sc = ServiceContainer(address, RequestHandlerClass=RequestHandlerClass)
    for service in services:
        path = service.getPost()
        sc.setNode(service, path)
    sc.serve_forever()
AsServer(port=8080, services=[DateService()], RequestHandler-
Class=MySOAPRequestHandler)
```

4.2.3 The DateService implementation

As in the previous example, the actual implementation must be hooked into the server skeleton. This is done in the `DateService_services_server.py` file:

```
from DateService_services import *
from ZSI.ServiceContainer import ServiceSOAPBinding
# needed for worker code
import time
class DateService(ServiceSOAPBinding):
    soapAction = {
        'urn:DateService.wsdl#getCurrentDate': 'soap_getCurrentDate',
        'urn:DateService.wsdl#getDate': 'soap_getDate',
        }
    def __init__(self, post='/DateService', **kw):
        ServiceSOAPBinding.__init__(self, post)

    def soap_getCurrentDate(self, ps):
        # input vals in request object
        # MANUALLY CORRECTED:
        # args = ps.Parse( getCurrentDateRequestWrapper() )
        # Use the class instead of an instance of the class.
        # Note: The erroneous code generation happens for rpc/literal, but not
        # for rpc/encoded, where using an instance works (?).
        args = ps.Parse( getCurrentDateRequestWrapper )

        # assign return values to response object
        response = getCurrentDateResponseWrapper()

        ## ADDED
        response._today = self.getCurrentDate(args._input)
        ## END ADDED

        return response
    def soap_getDate(self, ps):
        # input vals in request object
        # MANUALLY CORRECTED:
        # args = ps.Parse( getDateRequestWrapper() )
        # Use the class instead of an instance of the class.
        # Note: The erroneous code generation happens for rpc/literal, but not
        # for rpc/encoded, where using an instance works (?).
        args = ps.Parse( getDateRequestWrapper )
        # assign return values to response object
        response = getDateResponseWrapper()
        ## ADDED
        re-
sponse._day = self.getDate(offset=args._offset, date=args._someday)
        ## END ADDED
        return response
## ADDED WORKER CODE
    def getCurrentDate(self, input=None):
        dt = time.localtime(time.time())
        class today:
            _year = dt[0]
            _month = dt[1]
            _day = dt[2]
            _hour = dt[3]
            _minute = dt[4]
            _second = dt[5]
            _weekday = dt[6]
            _dayOfYear = dt[7]
            _dst = dt[8]

        return today
   def getDate(self, offset=None, date=None):
        sec = 3600 * 24  ## seconds/hour * 24h
        providedDate_tuple = (date._year, date._month, date._day,
                              date._hour, date._minute, date._second,
                              date._weekday, date._dayOfYear, date._dst)
        providedDate_sec = time.mktime(providedDate_tuple)
        offset_sec = sec * offset
        newDate_sec = providedDate_sec + offset_sec
        newDate_tuple = time.localtime(newDate_sec)
        if not offset:
            offset = 0
        if not date:
            sys.exit()
        someDay = getDateResponse()
        someDay._year = newDate_tuple[0]
        someDay._month = newDate_tuple[1]
        someDay._day = newDate_tuple[2]
        someDay._hour = newDate_tuple[3]
        someDay._minute = newDate_tuple[4]
        someDay._second = newDate_tuple[5]
        someDay._weekday = newDate_tuple[6]
        someDay._dayOfYear = newDate_tuple[7]
        someDay._dst = newDate_tuple[8]
        return someDay
## END ADDED
```

If you take a closer look at the two method implementations, you will notice that in `getCurrentDate(...)` the returned date is just a (nested) python class. ZSI handles the serialization of that class into the actual SOAP message for us.[7]In `getDate(...)`, though, we directly use the `getDateResponse` instance imported from the generated `DateService_services.py`. which already is a fully fledged ZSI-typecoded representation of the getDate response.

4.3 A Python ZSI client for the DateService

Using generated type mapping Instead of using the ServiceProxy class which dynamically fetches the WSDL description and gets the necessary type-serialization information from it, we can use the generated code to write a client:

```
#! /apps/pydev/bin/python2.3
import sys, time
from DateService_services import *
def main():
    loc = simple_Date_ServiceLocator()
    portType = loc.getDateService_PortType(tracefile=sys.stdout)
    while 1:
        offset = raw_input("Enter offset as int [0]: ")
        try:
            offset = int(offset)
        except ValueError:
            offset = 0
        x = getCurrentDateRequestWrapper()
        x._input = 'Test'
        myToday = portType.getCurrentDate(x)
        dateRequest = getDateRequestWrapper()

        # We use the current date as input to getDate
        dateRequest._someday = myToday._today
        dateRequest._offset = offset
```

[7]The python object must be structurally equivalent to the XML datatype that is defined in the WSDL and that constitutes ZSI´s typecodes.

```
        date = portType.getDate(dateRequest)

        print '\n\nRESULT'
        print '%10s = %s' % ('today', make_asctime(myToday._today))
        print '%6s + %d = %s' % ('to-
day', dateRequest._offset, make_asctime(date._day))
# just a helper
def make_asctime(date_object):
    timeTuple = (date_object._year, date_object._month, date_object._day,
                 date_object._hour, date_object._minute, date_object._second,
                 date_object._weekday, date_object._dayOfYear, date_object._dst
                 )
    return time.asctime(timeTuple)
if __name__ == '__main__':
    main()
```

As you can see, ZSI provides us with the getCurrentDateRequestWrapper and getDateRequestWrapper classes. These handle the serialization transparently and we use them to set the argument values. Then, we hand them into the corresponding methods of the portType objects the <service name>_ServiceLocator returned.

Alternatively, we could use a ServiceProxy again:

```
#!/apps/pydev/bin/python2.3
import sys
import getopt
from ZSI import ServiceProxy
import ZSI.wstools.WSDLTools
#------------------------------------------------------------------------------
# default configuration
#------------------------------------------------------------------------------
port = 8080
host = 'dev-b'
#------------------------------------------------------------------------------
# command line parsing
#------------------------------------------------------------------------------
def usage(rcode=1):
    print "usage: myServiceProxyClient.py [--host=<hostname> --port=,-c<port> --
help, -h]"
    sys.exit(rcode)
try:
    optlist, args = getopt.getopt(sys.argv[1:], "hp:", ['help', 'port='])
except getopt.GetoptError:
    usage()
for opt, arg in optlist:
    print opt, arg
    if opt in ["-h", "--help"]:
        usage(0)
    elif opt in ["--host"]:
        host = arg
        continue
    elif opt in ["-p", "--port"]:
        port = int(arg)
        continue
```

```
url = 'http://' + host + ':' + str(port) + '/DateService'
service = ServiceProxy(url, use_wsdl=True, tracefile=sys.stdout,
                       ns='http://dev-b.handel-dev.local:8080/DateService')
print '\nAccessing service DateService, method getCurrentDate...'
while 1:
    # Must use keyword arguments if use_wsdl was set
    input = raw_input("Enter something: ")
    result = service.getCurrentDate(input=input)
    print 'getCurrentDate: result =', result
    offset = int(raw_input("Enter offset: "))
## Hmm, this does not work. Why?
##    class someday:
##        _year = result['today']['year']
##        _month = result['today']['month']
##        _day = result['today']['day']
##        _hour = result['today']['hour']
##        _minute = result['today']['minute']
##        _second = result['today']['second']
##        _weekday =result['today']['weekday']
##        _dayOfYear = result['today']['dayOfYear']
##        _dst = result['today']['dst']
   someday = {
        'year': result['today']['year'],
        'month': result['today']['month'],
        'day': result['today']['day'],
        'hour': result['today']['hour'],
        'minute': result['today']['minute'],
        'second': result['today']['second'],
        'weekday': result['today']['weekday'],
        'dayOfYear': result['today']['dayOfYear'],
        'dst': result['today']['dst'],
        }
   print 'getDate: result = ', service.getDate(offset=offset, someday=soVmeday)
```

This is the output of a sample client sesssion of the ServiceProxy solution:

```
$ ./myServiceProxyClient.py
service is <ZSI.ServiceProxy.ServiceProxy instance at 0x18f418>
{'_wsdl': <ZSI.wstools.WSDLTools.WSDL instance at 0x18f648>,
'_typesmodule': None,
'_soapAction': None, '_use_wsdl': True,
'_ns': 'http://dev-b.handel-dev.local:8080/DateService',
'_tracefile': <open file '<stdout>', mode 'w' at 0x128060>,
'_op_ns': None,
'_name': u'simple Date Service', '_nsdict': {},
'getCurrentDate': <ZSI.ServiceProxy.MethodProxy instance at 0x495f58>,
'_port': <ZSI.wstools.WSDLTools.Port instance at 0x495eb8>,
'_service': <ZSI.wstools.WSDLTools.Service instance at 0x495e40>,
'__doc__': u'Date Web Service',
'getDate': <ZSI.ServiceProxy.MethodProxy instance at 0x49e058>}
Accessing service DateService, method getCurrentDate...
Enter something:
_________________________________ Tue Jul 19 09:08:51 2005 REQUEST:
<?xml version="1.0" encoding="utf-8"?>
<SOAP-ENV:Envelope
```

```
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:ZSI="http://www.zolera.com/schemas/ZSI/"
  xmlns="http://dev-b.handel-dev.local:8080/DateService"
  SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" >
<SOAP-ENV:Body>
<getCurrentDate>
<input id="124038" xsi:type="xsd:string"></input>
</getCurrentDate>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
_________________________________ Tue Jul 19 09:08:53 2005 RESPONSE:
Server: ZSI/1.1 BaseHTTP/0.3 Python/2.3.4
Date: Tue, 19 Jul 2005 07:08:53 GMT
Content-type: text/xml; charset="utf-8"
Content-Length: 955
<?xml version="1.0" encoding="utf-8"?>
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:ZSI="http://www.zolera.com/schemas/ZSI/"
  SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" >
<SOAP-ENV:Body>
<getCurrentDateResponse xmlns="urn:DateService.wsdl">
<today xmlns="urn:DateService.wsdl">
<year xsi:type="xsd:integer">2005</year>
<month xsi:type="xsd:integer">7</month>
<day xsi:type="xsd:integer">19</day>
<hour xsi:type="xsd:integer">9</hour>
<minute xsi:type="xsd:integer">8</minute>
<second xsi:type="xsd:integer">53</second>
<weekday xsi:type="xsd:integer">1</weekday>
<dayOfYear xsi:type="xsd:integer">200</dayOfYear>
<dst xsi:type="xsd:integer">1</dst>
</today>
</getCurrentDateResponse>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
getCurrentDate: re-
sult = {u'today': {u'hour': 9, u'dst': 1, u'month': 7, u'second': 53,
u'dayOfYear': 200, u'weekday': 1, u'year': 2005, u'day': 19, u'minute': 8}}
Enter offset: 9
getDate: re-
sult =  _________________________________ Tue Jul 19 09:11:51 2005 REQUEST:
<?xml version="1.0" encoding="utf-8"?>
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:ZSI="http://www.zolera.com/schemas/ZSI/"
  xmlns="http://dev-b.handel-dev.local:8080/DateService"
  SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" >
<SOAP-ENV:Body>
<getDate>
<offset xsi:type="xsd:integer">9</offset>
<someday>
<year xsi:type="xsd:integer">2005</year>
<month xsi:type="xsd:integer">7</month>
<day xsi:type="xsd:integer">19</day>
<hour xsi:type="xsd:integer">9</hour>
<minute xsi:type="xsd:integer">8</minute>
<second xsi:type="xsd:integer">53</second>
<weekday xsi:type="xsd:integer">1</weekday>
<dayOfYear xsi:type="xsd:integer">200</dayOfYear>
<dst xsi:type="xsd:integer">1</dst>
</someday>
</getDate>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
_________________________________ Tue Jul 19 09:11:52 2005 RESPONSE:
Server: ZSI/1.1 BaseHTTP/0.3 Python/2.3.4
Date: Tue, 19 Jul 2005 07:11:52 GMT
Content-type: text/xml; charset="utf-8"
Content-Length: 937
<?xml version="1.0" encoding="utf-8"?>
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:ZSI="http://www.zolera.com/schemas/ZSI/"
  SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" >
<SOAP-ENV:Body>
<getDateResponse xmlns="urn:DateService.wsdl">
<day xmlns="urn:DateService.wsdl">
<year xsi:type="xsd:integer">2005</year>
<month xsi:type="xsd:integer">7</month>
<day xsi:type="xsd:integer">28</day>
<hour xsi:type="xsd:integer">9</hour>
<minute xsi:type="xsd:integer">8</minute>
<second xsi:type="xsd:integer">53</second>
<weekday xsi:type="xsd:integer">3</weekday>
<dayOfYear xsi:type="xsd:integer">209</dayOfYear>
<dst xsi:type="xsd:integer">1</dst>
</day>
</getDateResponse>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
{u'day': {u'hour': 9, u'dst': 1, u'month': 7, u'second': 53, u'dayOfYear': 209,
u'weekday': 3, u'year': 2005, u'day': 28, u'minute': 8}}
```

4.4 An Excel XP Visual Basic client for the DateService

4.4.1 VB stub generation

Refer to 3.4.1 for details. The URL to receive the WSDL file is "http://dev-b.handel-dev.local:8080/DateService" in our example.

The Web Services References toolkit creates the class `clsws_simpleDateService` for us, with these methods:

```
Public Function wsm_getCurrentDate(ByVal str_input As String) As struct_Date
...
Public Function wsm_getDate(ByVal dcml_offset As Double,
ByVal obj_someday As struct_Date) As struct_Date
...
```

Also, the class struct_Date is generated to represent our structured datatype:

```
'*****************************************************************
'This class was created by the Web Service References Tool 2.0.
'
'Created: 6/29/2005 12:54:18 AM
'
'Description:
'This class is a Visual Basic for Applications class representation of the user-
defined
'type as defined by http://dev-b.handel-dev.local:8080Plus/DateService.
'
'This class only contains the Date,
'as defined in the WSDL.
'
'Changes to the code in this class may result in incorrect behavior.
'
'*****************************************************************
Public year As Long
Public month As Long
Public day As Long
Public hour As Long
Public minute As Long
Public second As Long
Public weekday As Long
Public dayOfYear As Long
Public dst As Long
```

4.4.2 VB client implementation & spreadsheet access

There are 2 ways to make the web service available in a spreadsheet: Using a sub procedure (also called macro) or using a function. Note that

- a macro can not be called directly from a worksheet cell, so you need a command button to call it
- a function called directly or indirectly from a worksheet cell can not change another cell´s value

Sub procedures (macros) While macros *can* accept passed arguments, you can´t pass such arguments from within a spreadsheet using command buttons. This leaves us with the possibility to hard-wire the cell adresses we want to use to store our result values or to use "defined names" - we opt for the latter. To add the following client code, insert a command button into the spreadsheet and double-click it while still in design mode. This will add sub procedure stubs to the sheet objects of the VBA project:[8]

```
Private Sub getCurrentDateButton2_Click()
    Dim inputRange As Range
    Dim outputRange As Range
    Set inputRange = Range("input_3")
    Set outputRange = Range("today_2")
    Dim dateObj As struct_Date
    Dim service As New clsws_simpleDateService

    Set dateObj = service.wsm_getCurrentDate(str_input)

    fillRangeFromDateStructure outputRange, dateObj

End Sub
Private Sub getDateButton2_Click()
    Dim inputDateRange As Range
    Dim inputOffsetRange As Range
    Dim outputRange As Range

    Set inputOffsetRange = Range("offset_3")
    Set inputDateRange = Range("someday_3")
    Set outputRange = Range("day_2")

    Dim dateRes As struct_Date
    Dim dateInput As New struct_Date

    fillDateStructureFromRange dateInput, inputDateRange
    Dim service As New clsws_simpleDateService
    Set dateRes = service.wsm_getDate(inputOffsetRange.Value, dateInput)

    fillRangeFromDateStructure outputRange, dateRes

End Sub
Private Sub fillDateStructureFromRange(ByRef dateRes As struct_Date,
ByRef inputDateRange As Range)

    dateRes.year = inputDateRange.Item(1, 1)
    dateRes.month = inputDateRange.Item(2, 1)
    dateRes.day = inputDateRange.Item(3, 1)
    dateRes.hour = inputDateRange.Item(4, 1)
    dateRes.minute = inputDateRange.Item(5, 1)
    dateRes.second = inputDateRange.Item(6, 1)
    dateRes.weekday = inputDateRange.Item(7, 1)
    dateRes.dayOfYear = inputDateRange.Item(8, 1)
    dateRes.dst = inputDateRange.Item(9, 1)
End Sub
Private Sub fillRangeFromDateStructure(ByRef outputRange As Range,
ByRef inputDate As struct_Date)
```

[8]By default, the stubs are called like the command buttons (e.g. "CommandButton_Click()"). Note that we changed the command button names here to reflect their purpose, which results in this example´s macro names.

```
    outputRange.Item(1, 1) = inputDate.year
    outputRange.Item(2, 1) = inputDate.month
    outputRange.Item(3, 1) = inputDate.day
    outputRange.Item(4, 1) = inputDate.hour
    outputRange.Item(5, 1) = inputDate.minute
    outputRange.Item(6, 1) = inputDate.second
    outputRange.Item(7, 1) = inputDate.weekday
    outputRange.Item(8, 1) = inputDate.dayOfYear
    outputRange.Item(9, 1) = inputDate.dst
End Sub
```

This code defines two macros (and two helpers) to access the DateService methods. They can be invoked from a spreadsheet by the corresponding command buttons. Note that both macros rely on hardcoded defined names. To use the macros, you have to define these names in the spreadsheet. Moreover, the names must refer to appropriate cell ranges: The range "input_3" used in the `getCurrentDateButton2_Click()` macro refers to a single-cell range, whereas the "someday_3" and "day_2" ranges are expected to contain 8 elements (cells).

Functions Of course it would be much nicer if no hardcoded names were needed. And indeed, there are solutions to do this. A simple approach is to return a single string value that is a concatenation of a date´s fields (insert those functions with Insert->Module...):

```
Public Function getCurrentDate(ByVal str_input As String) As String
    '*****************************************************************
    'Excel Sheet entry point to call wsm_getCurrentDate Proxy function,
    'manually created.
    '*****************************************************************
    Dim dateObj As struct_Date
    Dim service As New clsws_simpleDateService

    Set dateObj = service.wsm_getCurrentDate(str_input)
    getCurrentDate = Str(dateObj.year) + "/" + Str(dateObj.month) + "/" +
Str(dateObj.day) + " " + Str(dateObj.hour) + ":" + Str(dateObj.minute) + ":" +
Str(dateObj.second)

End Function
Public Function getDate(ByVal i_offset As Integer, ByRef inputDat-
eRange As Range)
As String
    '*****************************************************************
    'Excel Sheet entry point to call wsm_getCurrentDate Proxy function,
    'manually created.
    '*****************************************************************


    Dim dateRes As struct_Date
    Dim dateInput As New struct_Date


    dateInput.year = inputDateRange.Item(1, 1)
    dateInput.month = inputDateRange.Item(2, 1)
    dateInput.day = inputDateRange.Item(3, 1)
    dateInput.hour = inputDateRange.Item(4, 1)
    dateInput.minute = inputDateRange.Item(5, 1)
    dateInput.second = inputDateRange.Item(6, 1)
    dateInput.weekday = inputDateRange.Item(7, 1)
    dateInput.dayOfYear = inputDateRange.Item(8, 1)
    dateInput.dst = inputDateRange.Item(9, 1)

    Dim service As New clsws_simpleDateService
    Set dateRes = service.wsm_getDate(i_offset, dateInput)

    getDate = Str(dateRes.year) + "/" + Str(dateRes.month) + "/" +
Str(dateRes.day) + " " + Str(dateRes.hour) + ":" + Str(dateRes.minute) + ":" +
Str(dateRes.second)

End Function
```

Unfortunately, the structure of a date is lost with this approach, as well as the actual fields´ datatypes (imagine a struct consisting of different datatypes). There is a better possibility:

```
Public Function getCurrentDateAsArray(ByRef inputR As Range) As Variant
    Dim resArr(8, 0) As Variant


    Dim dateObj As struct_Date
    Dim service As New clsws_simpleDateService

    Set dateObj = service.wsm_getCurrentDate(str_input)
    resArr(0, 0) = dateObj.year
    resArr(1, 0) = dateObj.month
    resArr(2, 0) = dateObj.day
    resArr(3, 0) = dateObj.hour
    resArr(4, 0) = dateObj.minute
    resArr(5, 0) = dateObj.second
    resArr(6, 0) = dateObj.weekday
    resArr(7, 0) = dateObj.dayOfYear
    resArr(8, 0) = dateObj.dst
    getCurrentDateAsArray = resArr

End Function
Public Function getDateAsArray(ByVal i_offset As Integer, ByRef inputDat-
eRange As Range)
As Variant

    Dim resArr(8, 0) As Variant
    Dim dateRes As struct_Date
    Dim dateInput As New struct_Date


    dateInput.year = inputDateRange.Item(1, 1)
    dateInput.month = inputDateRange.Item(2, 1)
    dateInput.day = inputDateRange.Item(3, 1)
    dateInput.hour = inputDateRange.Item(4, 1)
    dateInput.minute = inputDateRange.Item(5, 1)
    dateInput.second = inputDateRange.Item(6, 1)
    dateInput.weekday = inputDateRange.Item(7, 1)
    dateInput.dayOfYear = inputDateRange.Item(8, 1)
    dateInput.dst = inputDateRange.Item(9, 1)
    Dim service As New clsws_simpleDateService
    Set dateRes = service.wsm_getDate(i_offset, dateInput)

    resArr(0, 0) = dateRes.year
    resArr(1, 0) = dateRes.month
    resArr(2, 0) = dateRes.day
    resArr(3, 0) = dateRes.hour
    resArr(4, 0) = dateRes.minute
    resArr(5, 0) = dateRes.second
    resArr(6, 0) = dateRes.weekday
    resArr(7, 0) = dateRes.dayOfYear
    resArr(8, 0) = dateRes.dst
    getDateAsArray = resArr

End Function
```

To make use of these functions, you have to insert them into the spreadsheet as so-called “array formulas”. This means you must enter the formula like this:

1. Select the desired result cells (which would normally mean a range of cells in this case).
2. Enter the function call, e.g.:

   ```
   =getCurrentDateAsArray(A1)
   ```

3. Press CTRL+SHIFT+ENTER.

You´ll now see the entered formula in brackets if you select one of the range´s cells, e.g.:

```
{=getCurrentDateAsArray(input_4)}
```

4.4.3 Web Service References Toolkit pitfalls

An early version of the DateService WSDL brought up a problem with the WS References Toolkit. This version featured an input parameter with the same name as the return value for one of the service methods:

```
...
<message name="getDateRequest">
  <part name="offset" type="xsd:integer"/>
  <part name="someday" type="myType:Date"/>
</message>
<message name="getDateResponse">
  <part name="someday" type="myType:Date"/>
</message>
...
```

The generated VB stubs for this WSDL did not work. They seemed to implicate modifying the input object (handed in `ByRef`) instead of returning a new return value:

```
Public Sub wsm_getDate(ByVal dcml_offset As Dou-
ble, ByRef obj_someday As struct_Date)
...
```

While the code looked ok, it just didn´t work, for whatever reason.

4.5 A gSOAP C++ client for the DateService

4.5.1 Code generation from WSDL

1. Header:

```
$ /apps/pydev/bin/wsdl2h -o dateService.h http://dev-b.handel-
dev.local:8080/DateService
** The gSOAP WSDL parser for C and C++ 1.2.0
** Copyright (C) 2000-2005 Robert van Engelen, Genivia Inc.
** All Rights Reserved. This product is provided "as is", without any war-
ranty.
** The gSOAP WSDL parser is released under one of the following two li-
censes:
** GPL or the commercial license by Genivia Inc. Use option -
l for more info.
Saving dateService.h
Connecting to 'http://dev-b.handel-
dev.local:8080/DateService' to retrieve WSDL... done
Cannot open file 'typemap.dat'
Problem reading type map file typemap.dat.
Using internal type definitions for C++ instead.
Warning: part 'today' uses literal style and must refer to an ele-
ment rather than a type
Warning: part 'input' uses literal style and must refer to an ele-
ment rather than a type
Warning: part 'input' uses literal style and must refer to an ele-
ment rather than a type
Warning: part 'day' uses literal style and must refer to an ele-
ment rather than a type
Warning: part 'offset' uses literal style and must refer to an ele-
ment rather than a type
Warning: part 'someday' uses literal style and must refer to an ele-
ment rather than a type
Warning: part 'offset' uses literal style and must refer to an ele-
ment rather than a type
Warning: part 'someday' uses literal style and must refer to an ele-
ment rather than a type
To complete the process, compile with:
soapcpp2 dateService.h
```

2. Stub code:

```
0 hjoukl@dev-b .../C++ $ /apps/pydev/bin/soapcpp2
-I /data/pydev/DOWNLOADS/gsoap-2.7/soapcpp2 dateService.h
** The gSOAP Stub and Skeleton Compiler for C and C++ 2.7.1
** Copyright (C) 2000-2005, Robert van Engelen, Genivia Inc.
** All Rights Reserved. This product is provided "as is", without any war-
ranty.
** The gSOAP compiler is released under one of the following three licenses:
** GPL, the gSOAP public license, or the commercial license by Genivia Inc.
Saving soapStub.h
Saving soapH.h
Saving soapC.cpp
Saving soapClient.cpp
Saving soapServer.cpp
Saving soapClientLib.cpp
Saving soapServerLib.cpp
Using ns3 service name: DateService_USCOREBinding
Using ns3 service style: document
Using ns3 service encoding: literal
Using ns3 service location: http://dev-b.handel-dev.local:8080/DateService
Using ns3 schema namespace: urn:DateService.wsdl
Saving soapDateService_USCOREBindingProxy.h client proxy
Saving soapDateService_USCOREBindingObject.h server object
Saving DateService_USCOREBinding.getCurrentDate.req.xml sample SOAP/XML re-
quest
Saving DateService_USCOREBinding.getCurrentDate.res.xml sample SOAP/XML re-
sponse
Saving DateService_USCOREBinding.getDate.req.xml sample SOAP/XML request
Saving DateService_USCOREBinding.getDate.res.xml sample SOAP/XML response
Saving DateService_USCOREBinding.nsmap namespace mapping table
Compilation successful
0 hjoukl@dev-b .../C++ $
```

4.5.2 Client implementation

As already announced in section 3.5.2 we´ll now access the service through the DateService proxy class, which can be found in `soapDateService_USCOREBindingProxy.h`. We use it to call the two WS-methods in our sample client:

```
// Contents of file "myclient_use_proxy.cpp"
#include "soapDateService_USCOREBindingProxy.h"
#include "DateService_USCOREBinding.nsmap"
int main()
{
    DateService ds;
    ns2__Date *today, *someday;
    ns3__getCurrentDateResponse today_response;
    ns3__getDateResponse someday_response;
    std::string text, input;
    text="TEST";
    std::cout << "(1) Calling 'getCurrentDate()'-
Web Service method:" << std::endl;
    if(ds.ns3__getCurrentDate(text, today_response) == SOAP_OK)
        {
            today = today_response.today;
            std::cout << "\nCurrent date:" << std::endl;
            std::cout << "\tyear: "  << *today->year  << std::endl;
            std::cout << "\tmonth: " << *today->month << std::endl;
            std::cout << "\tday: "   << *today->day   << std::endl;
            std::cout << "\thour: "   << *today->hour   << std::endl;
            std::cout << "\tminute: "   << *today->minute   << std::endl;
            std::cout << "\tsecond: "   << *today->second   << std::endl;
        }
    else
        soap_print_fault(ds.soap, stderr);
    std::cout << "\n(2) Calling 'getDate()'- Web Service method:" << std::endl;
    std::cout << "\n(2)Please enter an integer for the 'offset'"<< std::endl;
    std::cout << "\toffset = ";
    std::cin >> input;
    someday = today;
    if(ds.ns3__getDate(input, someday, someday_response) == SOAP_OK)
        {
            someday = someday_response.day;
            std::cout << "\nSome Day:" << std::endl;
            std::cout << "\tyear: " << *someday->year << std::endl;
            std::cout << "\tmonth: "<< *someday->month << std::endl;
            std::cout << "\tday: " << *someday->day << std::endl;
            std::cout << "\thour: "   << *today->hour   << std::endl;
            std::cout << "\tminute: "   << *today->minute   << std::endl;
            std::cout << "\tsecond: "   << *today->second   << std::endl;
        }
    else
    return 0;
}
```

You can find the classes/structs that represent the WSDL input and output datatypes in the generated file `soapStub.h`. This file is really pretty much self-explaining, so the usage in client implementations should be straightforward.

4.5.3 gSOAP Client compilation

gcc 2.95.2

```
 $ g++ -o myClient_use_proxy -I/apps/pydev/include -L/apps/pydev/lib
-R /apps/prod/lib myClient_use_proxy.cpp soapC.cpp soapClient.cpp
-lsocket -lgsoap++
```

gcc 3.4.3

```
$ /apps/pydev/gcc/3.4.3/bin/g++ -o myClient_use_proxy -I/apps/pydev/include
-L/apps/pydev/gcc/3.4.3/lib -R /apps/pydev/gcc/3.4.3/lib
myClient_use_proxy.cpp soapC.cpp soapClient.cpp
-lsocket -lgsoap++ -lnsl
```

4.6 Applix spreadsheets as DateService clients

While Applix does not come with built-in web services support it is extensible with C shared libraries. We can use this feature to wrap gSOAP client code, making web services accessible from within Applix spreadsheets.

4.6.1 Code generation from WSDL

The C/C++ client stubs must be generated as described in section4.5.1.

4.6.2 Applix client implementation

ELF C extension To make the DateService callable from Applix we need to write an ELF shared library C extension. The `ax_DateService.cpp` extension implements the gSOAP web services access:

```
#include "elfapi.h"
#include "soapDateService_USCOREBindingProxy.h"
#include "DateService_USCOREBinding.nsmap"
#define TRUE 1
extern "C" elfData getCurrentDate();
extern "C" elfData getCurrentDate_2();
extern "C" elfData getDate();
extern "C" elfData getDate_2();
/*
*       Define the function table
*/
AxCallInfo_t    funcTable[]={
        { "Web Services", /* func type ... "financial", "math" ... */
          getCurrentDate,  /* The C routine to call  */
          "getCurrentDate", /* name to use in Applixware (usually identi-
cal to the function
                               name)   */
          "string getCurrentDate(string)",  /* shows the func-
tion name and its arguments */
          TRUE  /* An integer that governs the treatment of ERROR and NA values,
                   and whether the function is displayed in the Spread-
sheets Functions
                   dialog box. */
        },
        { "Web Services", /* func type ... "financial", "math" ... */
          getCurrentDate_2,    /* The C routine to call  */
          "getCurrentDate_2",  /* name to use in Applixware (usually identi-
cal to the
                                  function name)   */
          "date getCurrentDate_2(string)",  /* shows the func-
tion name and its arguments */
          TRUE           /* An integer that governs the treatment of ER-
ROR and NA values,
                             and whether the function is dis-
played in the Spreadsheets
                             Functions dialog box. */
        },
        { "Web Services", /* func type ... "financial", "math" ... */
          getDate,        /* The C routine to call  */
          "getDate",      /* name to use in Applixware (usually identi-
cal to the function
                             name)   */
          "date_string getDate(int, date)", /* shows the func-
tion name and its arguments */
          TRUE   /* An integer that governs the treatment of ERROR and NA val-
ues,
                    and whether the function is displayed in the Spread-
sheets Functions
                    dialog box. */
        },
        { "Web Services", /* func type ... "financial", "math" ... */
          getDate_2,      /* The C routine to call  */
          "getDate_2",    /* name to use in Applixware (usually identi-
cal to the function
                             name)   */
          "date getDate_2(int, date)",   /* shows the function name and its ar-
guments */
          TRUE  /* An integer that governs the treatment of ERROR and NA values,
                   and whether the function is displayed in the Spread-
sheets Functions
                   dialog box. */
        },
        {  NULL,
           NULL,
           NULL,
           NULL,
           NULL
        }
};
/*
*       Function AxGetCallInfo returns the function table.
*       This function is called by Applixware when you run the macro
*       RPC_CONNECT@ or INSTALL_C_LIBRARY@. This function must exist
*       in the RPC program or Shared Library.
*/
DLL_EXPORT AxCallInfo_t *AxGetCallInfo()
{
        return(funcTable);
}
/*=========================================*/
/* func                                    */
/*=========================================*/
extern "C" elfData getCurrentDate(elfData args)
/* args is an array of arguments passed from ELF */
{
    elfData arrayElem;
    elfData retValue;
    char *val;
    /* Check argument number & types */
    if (AxArraySize(args) != 1) {
        AxError(1, "function takes one single argument", "getCurrentDate");
    }
    arrayElem = AxArrayElement(args, 0);
    if (!AxIsString(arrayElem)) AxError(1, "argument must be a string", "getCur-
rentDate");
    DateService ds;
    ns2__Date *today;
    ns3__getCurrentDateResponse today_response;
    val = AxStrFromDataPtr(arrayElem);
    std::string text(val);
    std::string soapError = "SOAP Error";
    std::string result = "";
    if(ds.ns3__getCurrentDate(text, today_response) == SOAP_OK)
        {
            today = today_response.today;
            result = *today->year
                + "/" + *today->month
                + "/" + *today->day
                + " " + *today->hour
                + ":" + *today->minute
                + ":" + *today->second;
          }
    else
        {
            //soap_print_fault(ds.soap, stderr);  // gSOAP error
            soapError = "SOAP error";
            AxError(1, soapError.c_str(), "getCurrentDate");
            //AxError(1, "SOAP error", "getCurrentDate");
        }
     retValue = AxMakeStrData(result.length(), result.c_str());
     return retValue;
}
extern "C" elfData getCurrentDate_2(elfData args)
/* args is an array of arguments passed from ELF */
{
    elfData arrayElem;
    elfData retArray;
    char *val;
    /* Check argument number & types */
    if (AxArraySize(args) != 1) {
        AxError(1, "function takes one single argument", "getCurrentDate");
    }
    arrayElem = AxArrayElement(args, 0);
    if (!AxIsString(arrayElem)) AxError(1, "argument must be a string", "getCur-
rentDate");
    DateService ds;
    ns2__Date *today;
    ns3__getCurrentDateResponse today_response;
    val = AxStrFromDataPtr(arrayElem);
    std::string text(val);
    std::string soapError = "";
    if(ds.ns3__getCurrentDate(text, today_response) == SOAP_OK)
        {
            today = today_response.today;
            retArray = AxMakeArray(0);
            retArray = AxAddIntToArray(retArray, 0, atoi((*today-
>year).c_str()));
            retArray = AxAddIntToArray(retArray, 1, atoi((*today-
>month).c_str()));
            retArray = AxAddIntToArray(retArray, 2, atoi((*today-
>day).c_str()));
            retArray = AxAddIntToArray(retArray, 3, atoi((*today-
>hour).c_str()));
            retArray = AxAddIntToArray(retArray, 4, atoi((*today-
>minute).c_str()));
            retArray = AxAddIntToArray(retArray, 5, atoi((*today-
>second).c_str()));
            retArray = AxAddIntToArray(retArray, 6, atoi((*today-
>weekday).c_str()));
            retArray = AxAddIntToArray(retArray, 7, atoi((*today-
>dayOfYear).c_str()));
            retArray = AxAddIntToArray(retArray, 8, atoi((*today-
>dst).c_str()));
          }
    else
```

```
        {
            //soap_print_fault(ds.soap, stderr);  // gSOAP error
            soapError = "SOAP error";
            AxError(1, soapError.c_str(), "getCurrentDate");
            //AxError(1, "SOAP error", "getCurrentDate");
        }
    return retArray;
}
extern "C" elfData getDate(elfData args)
/* args is an array of arguments passed from ELF */
{
    elfData elf_offset, elf_date, retValue;
    /* Check argument number & types */
    if (AxArraySize(args) != 2) {
        AxError(1, "function takes 2 arguments", "getCurrentDate");
    }
    elf_offset = AxArrayElement(args, 0);
    if (!AxIsInt(elf_offset)) AxError(1, "argument must be an integer", "get-
Date");
    elf_date = AxArrayElement(args, 1);
    if (!AxIsArray(elf_date)) AxError(1, "argument must be an array", "get-
Date");
    DateService ds;
    //ns2__Date someday, date;
    ns2__Date *someday, date;
    ns3__getDateResponse someday_response;
    std::string offset(AxStrFromDataPtr(elf_offset));
    std::string year(AxStrFromDataPtr(AxArrayElement(AxArrayElement(elf_date, 0), 0)));
    std::string month(AxStrFromDataPtr(AxArrayElement(AxArrayElement(elf_date, 1), 0)));
    std::string day(AxStrFromDataPtr(AxArrayElement(AxArrayElement(elf_date, 2), 0)));
    std::string hour(AxStrFromDataPtr(AxArrayElement(AxArrayElement(elf_date, 3), 0)));
    std::string minute(AxStrFromDataPtr(AxArrayElement(AxArrayElement(elf_date, 4), 0)));
    std::string sec-
ond(AxStrFromDataPtr(AxArrayElement(AxArrayElement(elf_date, 5), 0)));
    std::string week-
day(AxStrFromDataPtr(AxArrayElement(AxArrayElement(elf_date, 6), 0)));
    std::string day-
OfYear(AxStrFromDataPtr(AxArrayElement(AxArrayElement(elf_date, 7), 0)));
    std::string dst(AxStrFromDataPtr(AxArrayElement(AxArrayElement(elf_date, 8), 0)));
    date.year = &year;
    date.month = &month;
    date.day = &day;
    date.hour = &hour;
    date.minute = &minute;
    date.second = &second;
    date.weekday = &weekday;
    date.dayOfYear = &dayOfYear;
    date.dst = &dst;
    std::string result = "";
    //retValue = AxMakeStrData(offset.length(), offset.c_str());
    //return retValue;
    //std::cout << "(1) Calling 'getCurrentDate()'-
Web Service method:" << std::endl;
    if(ds.ns3__getDate(offset, &date, someday_response) == SOAP_OK)
        {
            someday = someday_response.day;
            result = *someday->year
                + "/" + *someday->month
                + "/" + *someday->day
                + " " + *someday->hour
                + ":" + *someday->minute
                + ":" + *someday->second;
          }
    else
        {
            //soap_print_fault(ds.soap, stderr);  // gSOAP error
            AxError(1, "SOAP error", "getCurrentDate");
        }
    retValue = AxMakeStrData(result.length(), result.c_str());
    return retValue;
}
extern "C" elfData getDate_2(elfData args)
/* args is an array of arguments passed from ELF */
{
    elfData elf_offset, elf_date, retArray;
    /* Check argument number & types */
    if (AxArraySize(args) != 2) {
        AxError(1, "function takes 2 arguments", "getCurrentDate");
    }
    elf_offset = AxArrayElement(args, 0);
    if (!AxIsInt(elf_offset)) AxError(1, "argument must be an integer", "get-
Date");
    elf_date = AxArrayElement(args, 1);
    if (!AxIsArray(elf_date)) AxError(1, "argument must be an array", "get-
Date");
    DateService ds;
    //ns2__Date someday, date;
    ns2__Date *someday, date;
    ns3__getDateResponse someday_response;
    std::string offset(AxStrFromDataPtr(elf_offset));
    std::string year(AxStrFromDataPtr(AxArrayElement(AxArrayElement(elf_date, 0), 0)));
    std::string month(AxStrFromDataPtr(AxArrayElement(AxArrayElement(elf_date, 1), 0)));
    std::string day(AxStrFromDataPtr(AxArrayElement(AxArrayElement(elf_date, 2), 0)));
    std::string hour(AxStrFromDataPtr(AxArrayElement(AxArrayElement(elf_date, 3), 0)));
    std::string minute(AxStrFromDataPtr(AxArrayElement(AxArrayElement(elf_date, 4), 0)));
    std::string sec-
ond(AxStrFromDataPtr(AxArrayElement(AxArrayElement(elf_date, 5), 0)));
    std::string week-
day(AxStrFromDataPtr(AxArrayElement(AxArrayElement(elf_date, 6), 0)));
    std::string day-
OfYear(AxStrFromDataPtr(AxArrayElement(AxArrayElement(elf_date, 7), 0)));
    std::string dst(AxStrFromDataPtr(AxArrayElement(AxArrayElement(elf_date, 8), 0)));
    date.year = &year;
    date.month = &month;
    date.day = &day;
    date.hour = &hour;
    date.minute = &minute;
    date.second = &second;
    date.weekday = &weekday;
    date.dayOfYear = &dayOfYear;
    date.dst = &dst;
    std::string result = "";
    int return_integer = TRUE;
    //std::cout << "(1) Calling 'getCurrentDate()'-
Web Service method:" << std::endl;
    if(ds.ns3__getDate(offset, &date, someday_response) == SOAP_OK)
        {
            someday = someday_response.day;
            retArray = AxMakeArray(0);
            if (return_integer == TRUE)
                {
                    retArray = AxAddIntToArray(retArray, 0, atoi((*someday-
>year).c_str()));
                    retArray = AxAddIntToArray(retArray, 1, atoi((*someday-
>month).c_str()));
                    retArray = AxAddIntToArray(retArray, 2, atoi((*someday-
>day).c_str()));
                    retArray = AxAddIntToArray(retArray, 3, atoi((*someday-
>hour).c_str()));
                    retArray = AxAddIntToArray(retArray, 4, atoi((*someday-
>minute).c_str()));
                    retArray = AxAddIntToArray(retArray, 5, atoi((*someday-
>second).c_str()));
                    retArray = AxAddIntToArray(retArray, 6, atoi((*someday-
>weekday).c_str()));
                    retArray = AxAddIntToArray(retArray, 7, atoi((*someday-
>dayOfYear).c_str()));
                    retArray = AxAddIntToArray(retArray, 8, atoi((*someday-
>dst).c_str()));
                }
            else
                {
                    retArray = AxAddStrToArray(retArray, 0, (*someday-
>year).c_str());
                    retArray = AxAddStrToArray(retArray, 1, (*someday-
>month).c_str());
                    retArray = AxAddStrToArray(retArray, 2, (*someday-
>day).c_str());
                    retArray = AxAddStrToArray(retArray, 3, (*someday-
>hour).c_str());
                    retArray = AxAddStrToArray(retArray, 4, (*someday-
>minute).c_str());
                    retArray = AxAddStrToArray(retArray, 5, (*someday-
>second).c_str());
                    retArray = AxAddStrToArray(retArray, 6, (*someday-
>weekday).c_str());
                    retArray = AxAddStrToArray(retArray, 7, (*someday-
>dayOfYear).c_str());
                    retArray = AxAddStrToArray(retArray, 8, (*someday-
>dst).c_str());
                }
          }
    else
        {
            AxError(1, "SOAP error", "getCurrentDate");
        }
    return retArray;
}
```

The code above implements the following functions:

- getCurrentDate: Return a string containing the current date. Takes a string argument.
- getCurrentDate_2: Return the current date as an array. Takes a string argument.
- getDate: Return the input date + offset as a string. Takes an integer offset and a 9-element date array as arguments.
- getDate_2: Return input date + offset as an array. Takes an integer offset and a 9-element date array as arguments.

To compile the code, you have to add the Applix ELF headers to the include path:

```
g++ -shared -o ax_DateService.so -R/apps/prod/lib
-I/apps/prod/applix/applix/versions/4.43.1021.544.343/axdata/elf
-I/apps/pydev/include -L/apps/pydev/lib soapC.cpp soap ax_DateService.cpp
-lgsoap++ -lsocket
```

The resulting shared library `ax_DateService.so` must be loaded into Applix before it can be used. This is done with the `install_c_library@(<extension lib>)` macro:

```
#define path "/data/pydev/WebServicesPresentation/DateService_rpc_literal/Applix/"
macro load()
        install_c_library@(path++"ax_DateService.so")
endmacro
macro unload()
        unbind_c_library@(path++"ax_DateService.so")
endmacro
macro reload()
        unbind_c_library@(path++"ax_DateService.so")
      install_c_library@(path++"ax_DateService.so")
endmacro
```

After the shared library has been dynamically loaded, the functions can be used like built-in function inside the spreadsheet. To make sensible use of the `getDate_2` function which returns the date struct as an array, you have to invoke it as an array function:[9]

1. Select an array with the appropriate number of cells (9 cells, one-dimensional in our case).
2. Enter the function call, e.g.:

   ```
   =getDate_2(offset_5, someday_5)
   ```

3. Press CTRL-SHIFT-ENTER to insert the formula as array function.

[9]Compare to section 4.4.2.

The actual extension function calls might also be wrapped into additional macro code. This makes it possible to invoke the functions from command buttons or to populate predefined, hardcoded result ranges with the results of a service call. Compared to Excel, an additional possibility for Applix is the definition of an extra argument for the output range *name* (as a string). Given the name, this range can then be filled with the result value structure, removing the need to hardcode range names in the macro code.[10]

The following macro code show some of the possibilities. Refer to the code comments for explanations:

```
#include "spsheet_.am"
/*
* Macros for Web Service getCurrentDate()
*/
/* <<<<  getCurrentDate_Macro >>>>> */
'             -->  Returning ASCII string
'             -->  called from within a cell
'             -->  providing an input parameter
Macro getCurrentDate_Macro(text)
var retStr
/* Web Service call */
retStr = getCurrentDate(text)
return(retStr)
endmacro
/* <<<<<<<<<<<<< END >>>>>>>>>>>>>*/
/* <<<<  getCurrentDate_2_Macro >>>>> */
'              --> no return value: returning ASCII string with population
'               -->  called by Buttons
'               -->  no input parameter possible
'               -->  hard-coded input-parameter'
'               -->  hard coded named range for result population
Macro getCurrentDate_2_Macro()
var format ss_cell_ cellInfo
var celladdr, cell
var retStr, text
cellAddr = ss_extract_range_info@("input_3")
cellInfo = ss_get_cell@(cellAddr[0], cellAddr[1], 0)
text=cellInfo.display_str
/* Web Service call */
retStr = getCurrentDate(text)
/* Result Population */
cellAddr = ss_extract_range_info@("today_1")
 ss_put_cell@(ss_coordinate@(cellAddr[0], cellAddr[1], 0), retStr)
'Alternativ
'cell=ss_coordinate@(cellAddr[0], cellAddr[1], 0)
'new_task@("ss_put_cell@", cell, retStr)
endmacro
/* <<<<<<<<<<<<< END >>>>>>>>>>>>>*/
/* <<<<  getCurrentDate_3_Macro >>>>> */
'              --> no return value: returning date array  with population
'               -->  called by cell
'               -->  input parameter for result population
Macro getCurrentDate_3_Macro(text, today)
'for parameter to-
day the name (string) of a named range has to be passed, not the named
```

[10]In Applix, macros can be called from spreadsheet cells but places restrictions on modifying spreadsheet cells from within a macro.

```
' ranged object by itself --> this would pass an array to the macro, where the
'content of the named range is provided. But we need the coordi-
nates of the cells of the
'named range
var format ss_cell_ cellInfo
var celladdr, cell
var retStr, date, rowStart, rowEnd, col
var k, i
/* Web Service call */
date = getCurrentDate_2(text)
/* Result Population */
'cellAddr = ss_extract_range_info@("today")
' --> hardcoded if no parameter is allowed
cellAddr = ss_extract_range_info@(today)
' --> not possible to get named range as parameter for result because
' the content of the named range is used by APPLIX NOT THE COORDINATES
rowStart=cellAddr[1]
rowEnd=cellAddr[3]
col=cellAddr[2]
k=0
for i=rowStart to rowEnd
        cellInfo= ss_get_cell@(col, i, 0)
        'date[k]= { cellInfo.display_str+0 }
        'ss_put_cell@(ss_coordinate@(cellAddr[col], cellAddr[i], 0), date[k])
        'Alternativ
        cell=ss_coordinate@(col, i, 0)
        new_task@("ss_put_cell@", cell, date[k])
        'ss_put_cell@(col,i,date[k])
        k=k+1
next i
endmacro
/* <<<<<<<<<<<<< END >>>>>>>>>>>>>*/
/*
* Macros for Web Service getDate()
*/
/* <<<<  getDate_Macro >>>>> */
'            -->  Returning ASCII string
'            -->  called from within a cell
'            -->  providing  input parameter
Macro getDate_macro(offset, someday)
var retStr, date
date=someday
/* Web Service call */
retStr = getDate(offset, date)
return(retStr)
endmacro
/* <<<<<<<<<<<<< END >>>>>>>>>>>>>*/
/* <<<<  getDate_1_Macro >>>>> */
'               --> Not used
'             --> no return value: returning ASCII string with population
'              -->  called by Buttons  or from within a cell
'              -->  no input parameter possible
'              -->  hard-coded input-parameter'
'              -->  hard coded named range for result population
Macro getDate_1_macro()
```

```
var format ss_cell_ cellInfo
var retStr, offset, date, rowStart, rowEnd, col
var cellAddr, k, i, cell
/* Preparing Web Service input parameter */
cellAddr = ss_extract_range_info@("offset")
cellInfo = ss_get_cell@(cellAddr[0], cellAddr[1], 0)
offset=cellInfo.display_str+0
cellAddr = ss_extract_range_info@("someday")
rowStart=cellAddr[1]
rowEnd=cellAddr[3]
col=cellAddr[2]
k=0
for i=rowStart to rowEnd
        cellInfo= ss_get_cell@(col, i, 0)
        date[k]= { cellInfo.display_str+0 }
        k=k+1
next i
/* Web Service call */
retStr = getDate(offset, date)
/* Result Population */
cellAddr = ss_extract_range_info@("result_macro1")
/* Gilt fuer Start des Makros ueber Button  */
' ss_put_cell@(ss_coordinate@(cellAddr[0], cellAddr[1], 0), retStr)
/* Gilt fuer Start des Makros in einer Zelle  */
cell=ss_coordinate@(cellAddr[0], cellAddr[1], 0)
'cell="A:E12"
new_task@("ss_put_cell@", cell, retStr)
return(retStr)
endmacro
/* <<<<<<<<<<<<< END >>>>>>>>>>>>>*/
/* <<<<  getDate_2_Macro >>>>> */
'              --> no return value: returning date array with population
'               -->  called by Buttons
'               -->  no input parameter possible
'               -->  hard-coded input-parameter'
'               -->  hard coded named range for result population
Macro getDate_2_macro()
var format ss_cell_ cellInfo
var celladdr, cell
var date, rowStart, rowEnd, col
var k, i
var  offset, someday
/* Perparing Web Service input parameter */
cellAddr = ss_extract_range_info@("offset_3")
cellInfo = ss_get_cell@(cellAddr[0], cellAddr[1], 0)
'offset=cellInfo.display_str + 0
offset=cellInfo.value
cellAddr = ss_extract_range_info@("someday_3")
rowStart=cellAddr[1]
rowEnd=cellAddr[3]
col=cellAddr[2]
k=0
for i=rowStart to rowEnd
        cellInfo = ss_get_cell@(col, i, 0)
        someday[k] = {cellInfo.value}
        'someday[k]=ss_coordinate@(col, i, 0)
        'new_task@("ss_put_cell@", cell, date[k])
        'ss_put_cell@(col,i,date[k])  --> not working
        k=k+1
next i
'DUMP_ARRAY@(someday)
/* Web Serviuce call */
date = getDate_2(offset, someday)
/* Result Population */
cellAddr = ss_extract_range_info@("day_1")
' --> not possible to get named range as parameter for result because
' the content of the named range is used by APPLIX NOT THE COORDINATES
rowStart=cellAddr[1]
rowEnd=cellAddr[3]
col=cellAddr[2]
k=0
for i=rowStart to rowEnd
        cell=ss_coordinate@(col, i, 0)
        new_task@("ss_put_cell@", cell, date[k])
        'ss_put_cell@(col,i,date[k])  --> not working
        k=k+1
next i
/*
* DUMP_ARRAY@(retStruct)
*/
endmacro
/* <<<<<<<<<<<<< END >>>>>>>>>>>>>*/
/* <<<<  getDate_3_Macro >>>>> */
'             --> no return value: returning date array with population
'              -->  called by cells
'              -->  input parameter for WebService call
'              --> named range input-parameter for result population
'
Macro getDate_3_macro(offset, someday, day)
var format ss_cell_ cellInfo
var celladdr, cell
var date, rowStart, rowEnd, col
var k, i
/* Preparation Web Service input parameter not necessary --> provided by call */
/* Web Service call */
date = getDate_2(offset, someday)
/* Result Population */
'cellAddr = ss_extract_range_info@("day")
' --> hardcoded if no parameter is allowed
cellAddr = ss_extract_range_info@(day)
' --> not possible to get named range as parameter for result because
' the content of the named range is used by APPLIX NOT THE COORDINATES
rowStart=cellAddr[1]
rowEnd=cellAddr[3]
col=cellAddr[2]
k=0
for i=rowStart to rowEnd
        cell=ss_coordinate@(col, i, 0)
        new_task@("ss_put_cell@", cell, date[k])
        'ss_put_cell@(col,i,date[k])  --> not working
        k=k+1
next i
/*
* DUMP_ARRAY@(retStruct)
*/
endmacro
/* <<<<<<<<<<<<< END >>>>>>>>>>>>>>*/
```

5 A document/literal service: The FinancialService

The previous example services had in common that they used rpc/literal binding. While this is WS-I-compliant, there is a tendency to propagate the usage of the document/literal fashion. With rpc/literal, you might define types or elements to be used in the request and response messages, in the `<types>` section. We did that for the DateService, see section 4.1. You use these elements or types as arguments or receive them as return value of the service call. They are "packed" into the request or the response tags in the actual SOAP message, as can be seen in the sample client output in 4.3.

The difference with document/literal binding is that the *complete* message inside `<SOAP-ENV:Body>` is described in the `<types>` section, as an XML schema - not only the call parameters/return values. The big advantage is obvious: Such a message can be XML Schema-validated.

As it is quite possible with document/literal to leave the service method name out of the request/response message at all, depending on your `<types>` definitions, there can also be a downside to this. Because if you do so, it might be difficult for the server implementation to dispatch correctly.[11] To avoid this, we explicitly model the method name into our request, as a toplevel element. This is called "document-wrapped" style.

Most of the steps from WSDL to implementation should be familiar by now; we will thus concentrate on the doc/literal differences and mainly present the (commented) code.

5.1 The FinancialService WSDL

The FinancialService is described in `FinancialService.wsdl`:

```
<?xml version="1.0"?>
<definitions name="FinancialService"
  targetNamespace="http://dev-b.handel-dev.local:8080/FinancialService.wsdl"
  xmlns:tns="http://dev-b.handel-dev.local:8080/FinancialService.wsdl"
  xmlns="http://schemas.xmlsoap.org/wsdl/"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:finType="http://dev-b.handel-dev.local:8080/FinancialService_NS"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">
  <types>
    <schema xmlns="http://www.w3.org/2001/XMLSchema"
      targetNamespace="http://dev-b.handel-dev.local:8080/FinancialService_NS" >
      <!-- the getPV operation request message content, in doc-wrapped manner
        (with toplevel element that explicitly names the operation) -->
      <element name="getPV">
        <complexType>
          <sequence>
            <element name="irate" type="xsd:float"/>
            <element name="CFSequence">
              <complexType>
```

[11] At least if dispatching does not rely on the SOAPAction header field.

```
                <sequence>
                  <element name="CF" minOccurs="0" maxOc-
curs="unbounded" type="xsd:float"/>
                </sequence>
              </complexType>
            </element>
          </sequence>
        </complexType>
      </element>
      <!-- the answer to the getPV operation request -->
      <element name="PV" type="xsd:float"/>
    </schema>
  </types>
  <message name="getPVRequest">
    <part name="msg" element="finType:getPV"/>
  </message>
  <message name="getPVResponse">
    <part name="msg" element="finType:PV"/>
  </message>
  <portType name="FinancialService_PortType">
    <operation name="getPV">
      <input message="tns:getPVRequest"/>
      <output message="tns:getPVResponse"/>
    </operation>
  </portType>
  <binding name="FinancialService_Binding" type="tns:FinancialService_PortType">
    <soap:binding style="document" trans-
port="http://schemas.xmlsoap.org/soap/http"/>
    <operation name="getPV">
      <soap:operation soapAction=""/>
      <input>
        <soap:body use="literal"/>
      </input>
      <output>
        <soap:body use="literal"/>
      </output>
    </operation>
  </binding>
  <service name="FinancialService">
    <documentation>Financial Web Service. Methods: -getPV(irate, CFSequence):
Return present value for given interest rate and Cash Flows.</documentation>
    <port name="FinancialService_Port" binding="tns:FinancialService_Binding">
      <soap:address location="http://dev-b.handel-
dev.local:8080/FinancialService"/>
    </port>
  </service>
</definitions>
```

Notes:

- The FinancialService´s getPV operation returns the net present value for a given interest rate r and a series of cash flows. It uses document/literal binding, so the full ingoing and outgoing message structure is defined in the WSDL `<types>` section.
- The `soapAction` attribute is empty (`<soap:operation soapAction=""/>`). This means the server will have to use a different way to dispatch to the method implementation.

5.2 A Python ZSI FinancialService server

5.2.1 Code generation from WSDL

1. wsdl2py:

```
$ /apps/pydev/bin/wsdl2py -f FinancialService.wsdl
==> FinancialService_services.py
==> FinancialService_services_types.py
```

2. wsdl2dispatch:

```
/apps/pydev/bin/wsdl2dispatch -f FinancialService.wsdl
==> FinancialService_services_server.py
```

5.2.2 The DateService web server

As mentioned above, the server implementation will need to use a different dispatch mechanism due to the lacking `soapAction` specification. Therefore, the do_POST method of the standard ZSI.ServiceContainer has been modified:

```
#!/apps/pydev/bin/python2.3
__doc__ = """
   This module started out from the ZSI.ServiceContainer implementation, but
   uses a different approach to address the service methods (which is actually
   more like in the ZSI.dispatch module). In the ZSI.ServiceContainer module
   service method ('action'), look-up is based on the SOAPAction SOAP header
   field. As this field is not set by most clients (not even by standard ZSI
   clients used in the ServiceProxy class and of course not by Excel XP
   clients), this had to be changed.
   Update: If your WSDL file features the <soap:operation soapAction=.../> tags
   in its bindings sections, then the ZSI.ServiceContainer approach actually
   does work. In this case, a client using ZSI.ServiceProxy will set the
   SOAPAction header field. Hmm, do not know yet about Excel...
   Workflow is as follows:
   - the server (ZSI.ServiceContainer) dispatches the incoming HTTP connections
     to the request handler (MySOAPRequestHandler)
   -
the request handler implements methods for GET (return the service WSDL) and
   POST (dispatch the appproprate service)
   - the _Dispatch function delegates service invocation to the ServiceContainer
   -
ServiceContainer looks up the service and invokes the service method (=action)
   -
the service which inherits from ServiceSOAPBinding returns the answer message
   that is then sent back by the _Dispatch function
   Personally I think the workflow is a bit strange (why does the server invoke
   the service, not the service request handler, why separate the dis-
patch into the
   _Dispach function), but that is the way it is done in the ZSI example.
"""
import getopt
import os
import sys
import cStringIO as StringIO
```

```
from ZSI import ParseException, FaultFromException
from ZSI import _copyright, _seqtypes, resolvers
from ZSI.parse import ParsedSoap
from ZSI.writer import SoapWriter
from ZSI.dispatch import SOAPRequestHandler as BaseSOAPRequestHandler
from ZSI.ServiceContainer import ServiceContainer
from ZSI.ServiceContainer import PostNotSpecified
from ZSI.wstools.WSDLTools import WSDLReader
from FinancialService_services_server import FinancialService
#-------------------------------------------------------------------------------
# default configuration
#-------------------------------------------------------------------------------
port = 8080
host = 'http://dev-b.handel-dev.local'
#-------------------------------------------------------------------------------
# command line parsing
#-------------------------------------------------------------------------------
def usage(rcode=1):
    print "usage: ./myDateServer.py --port=,-c<port> [--help, -h]"
    sys.exit(rcode)
try:
    optlist, args = getopt.getopt(sys.argv[1:], "hp:", ['help', 'port='])
except getopt.GetoptError:
    usage()
for opt, arg in optlist:
    if opt in ["-h", "--help"]:
        usage(0)
    elif opt in ["-p", "--port"]:
        port = int(arg)
        continue
#-------------------------------------------------------------------------------
# module code
#-------------------------------------------------------------------------------
class MySOAPRequestHandler(BaseSOAPRequestHandler):
    """SOAP handler.
    Dispatching is delegated to the _Dispatch function which wants a post
    and an action attribute to specify the service object and a service method.
    Post is determined using the self.path attribute inherited from
    BaseHTTPRequestHandler, containing the incoming request path. Ac-
tion is taken
    from body root element of the parsed SOAP message, similar to what is done
    in the ZSI.dispatch module.
    """
    def do_POST(self):
        ■'The POST command.
        ■'
##        print ">>>>> %s.do_POST" % self
##        print ">>>>> %s.path: %s" % (self, self.path)
##        print "%s.headers:" % self
##        print self.headers
        # SOAPAction is a SOAP header field. Should read the SOAP specs...
        # We strip off the quotation marks. Why ZSI puts them in anyway
        # is not clear to me (gSOAP does not, same as VB WSR toolkit).
        soapAction = self.headers.getheader('SOAPAction').strip('\'"')
        action = soapAction
        post = self.path
        if not post:
            raise PostNotSpecified, 'HTTP POST not specified in request'
        post = post.strip('\'"')
        try:
            ct = self.headers['content-type']
            if ct.startswith('multipart/'):
                cid = resolvers.MIMEResolver(ct, self.rfile)
                xml = cid.GetSOAPPart()
                ps = ParsedSoap(xml, resolver=cid.Resolve)
            else:
                length = int(self.headers['content-length'])
                payload = self.rfile.read(length)
                print ">>>> Request message:"
                print payload
                ps = ParsedSoap(payload)
        except ParseException, e:
            self.send_fault(FaultFromZSIException(e))
        except Exception, e:
            # Faulted while processing; assume it's in the header.
            self.send_fault(FaultFromException(e, 1, sys.exc_info()[2]))
        else:
            # Take the action from the parsed SOAP body root element
##            print ">>>> post =", post
##            print ">>>> action =", ps.body_root.localName
##            print ">>>> namespaceURI =", ps.body_root.namespaceURI
##            print ">>>> ps.body_root =", ps.body_root.__dict__
##            print ">>>> ps.GetElementNSdict:", ps.GetElementNSdict(ps.body_root)
            # This time, we try to dispatch the actual worker meth-
ods with a fully
            # namespace-qualified name...
            # ... but only if SOAPAction was not given...
            # Trying to leave soapAction attribute out of the wsdl com-
pletely lead
            # ZSI to put a SOAPAction of "None" into the header, as a string.
            # Should probably better be left empty in this case.
            if not action or action == "None":
                if ps.body_root.namespaceURI:
                    ac-
tion = ps.body_root.namespaceURI + ':' + ps.body_root.localName
                else:
                    action = ps.body_root.localName
            _Dispatch(ps, self.server, self.send_xml, self.send_fault,
                post=post, action=action)
    def do_GET(self):
        """Return the WSDL file. We expect one WSDL file per service,
        and the file path is determined by the incoming HTTP GET request
        path. Thus, if you want to use the service from Excel XP, you
        have to add the service name to the service address, e.g.:
        http://dev-b.handel-dev.local:8080/FinancialService
        """
##        print ">>>>> %s.do_GET" % self
##        print ">>>>> %s.path: %s" % (self, self.path)
        wsdlfile = os.path.join('.', self.path.replace('/', "", 1) + ".wsdl")
##        print ">>>>> using wsdlfile", wsdlfile
```

```
        wsdl = open(wsdlfile).read()
        self.send_xml(wsdl)
def _Dispatch(ps, server, SendResponse, SendFault, post, action, ns-
dict={}, **kw):
    """Send ParsedSoap instance to ServiceContainer, which dispatches to
    appropriate service via post, and method via action.  Response is a
    self-describing pyobj, which is passed to a SoapWriter.
    Call SendResponse or SendFault to send the reply back, appropriately.
        server -- ServiceContainer instance
    """
    try:
        result = server(ps, post, action)
    except Exception, e:
        return SendFault(FaultFromException(e, 0, sys.exc_info()[2]), **kw)
    if result == None:
        return
    reply = StringIO.StringIO()
    try:
        SoapWriter(reply, nsdict=nsdict).serialize(result)
        return SendResponse(reply.getvalue(), **kw)
    except Exception, e:
        return SendFault(FaultFromException(e, 0, sys.exc_info()[2]), **kw)
def AsServer(port=80, services=()):
    ■'port --
       services -- list of service instances
    ■'
    address = (■, port)
    sc = ServiceContainer(address, RequestHandlerClass=MySOAPRequestHandler)
    for service in services:
        path = service.getPost()
        sc.setNode(service, path)
    sc.serve_forever()
# Put the service classes you want to expose in here.
# We expect one WSDL file for each service, and only one port defined
# for the service.
expose_services = [
    FinancialService,
    ]
service_instances = []
for serviceclass in expose_services:
    service_instances.append(serviceclass())
AsServer(port, service_instances)
```

This code should be pretty self-explanatory. The modified do_POST method will now use the namespace-qualified toplevel element as dispatch action, if no SOAPAction HTTP header field was given (which won´t be the case due to our WSDL file). Apart from that, it is very similar to the server implementation in section 3.2.2.

5.2.3 The FinancialService implementation

Once again, the implentation is hooked into the `<...>_services_server.py` skeleton. This time, the generated `soapAction` dictionary is empty. We add the `'<ns>:<operation>'` key here that maps to the `soap_getPV` method:

```
from FinancialService_services import *
from ZSI.ServiceContainer import ServiceSOAPBinding
class FinancialService(ServiceSOAPBinding):
    soapAction = {
        # This would be used if soapAction was specified in WSDL
        'urn:FinancialService/getPV': 'soap_getPV',
        # If no soapAction is in WSDL, we dispatch using the root element name
        # (namespace-qualified)
        'http://dev-b.handel-
dev.local:8080/FinancialService_NS:getPV': 'soap_getPV',
        }
    def __init__(self, post='/FinancialService', **kw):
        ServiceSOAPBinding.__init__(self, post)
    def soap_getPV(self, ps):
        # input vals in request object
        args = ps.Parse( getPVRequestWrapper() )
        # Worker code: Actual present value calculation
        t = 0
        PV = 0.0
        for CF in args._CFSequence._CF:
            PV += (CF or 0.0) * ((args._irate / 100.0 + 1) ** (-t))
            t += 1
        print "Present value is: ", PV
        # assign return values to response object
        class SimpleTypeWrapper(float): typecode = getPVResponseWrapper()
        # WARNING specify value eg. SimpleTypeWrapper(1)
        response = SimpleTypeWrapper(PV)
        return response
```

The request message fields are accessed through `args`, which is a `FinancialService_services.getPVRequestWrapper` instance, in the same "_<name>"-syntax as before.

5.3 A Python ZSI client for the FinancialService

We use the generated stub code in the client implementation `myClient_FinancialService.py`:

```
#!/apps/pydev/bin/python2.3
import sys
import getopt
import ZSI.wstools.WSDLTools
from FinancialService_services import *
#------------------------------------------------------------------------------
# default configuration
#------------------------------------------------------------------------------
port = 8080
host = 'dev-b'
#------------------------------------------------------------------------------
# command line parsing
#------------------------------------------------------------------------------
def usage(rcode=1):
    print "usage: myClient_FinancialService.py [--host=<hostname> --port=,-
c<port> --help, -h]"
    sys.exit(rcode)
```

```
try:
    optlist, args = getopt.getopt(sys.argv[1:], "hp:", ['help', 'port='])
except getopt.GetoptError:
    usage()
for opt, arg in optlist:
    print opt, arg
    if opt in ["-h", "--help"]:
        usage(0)
    elif opt in ["--host"]:
        host = arg
        continue
    elif opt in ["-p", "--port"]:
        port = int(arg)
        continue

url = 'http://' + host + ':' + str(port) + '/FinancialService'
service = FinancialServiceLoca-
tor().getFinancialService_PortType(tracefile=sys.stdout)
print '\nAccessing service FinancialService, method getPV...'
while 1:
    # Must use keyword arguments if use_wsdl was set
    irate = None
    while not (0.0 <= irate <= 100.0):
        try:
            irate = float(raw_input("Enter interest rate in percent: "))
        except ValueError, e:
            print e
    CFSequence = []
    period = 0
    while 1:
        try:
            CFSequence.append(float(raw_input("Enter CF(t=%s) [Ctrl-
C to end]: " % (period,))))
            period += 1
        except ValueError, e:
            print e
        except KeyboardInterrupt:
            print "...done."
            break
    print "CF sequence is:", CFSequence
    print "Calculation interest rate is:", irate
    getPV = getPVRequestWrapper()
    #print "getPV", getPV
    #print "getPV.__dict__", getPV.__dict__
    getPV._irate = irate
    class CFSequence_class:
        _CF = CFSequence
    getPV._CFSequence = CFSequence_class

    #print "getPV", getPV
    #print "getPV.__dict__", getPV.__dict__
    result = service.getPV(getPV)
    print 'result:', result
```

After the necessary data has been read from the user, the appropriate attributes of the `getPVRequestWrapper` instance are set. To model the sequence of cash flow elements, the local class `CFSequence_class` is used. This class is “structurally equivalent” to the XML Schema `CFSequence` complexType as defined in the WSDL. ZSI then handles all the type mapping and serialization issues for us.

Sample client session output:

```
Accessing service FinancialService, method getPV...
Enter interest rate in percent: 4
Enter CF (t=0): -100
Enter CF (t=1): 5
Enter CF (t=2): 5
Enter CF (t=3): 105
Enter CF (t=4): ^C...done.
CF sequence is: [-100.0, 5.0, 5.0, 105.0]
Calculation interest rate is: 4.0
_________________________________ Wed Jul 20 15:39:14 2005 REQUEST:
<?xml version="1.0" encoding="utf-8"?>
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:ZSI="http://www.zolera.com/schemas/ZSI/"
  SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" >
<SOAP-ENV:Body>
<getPV xmlns="http://dev-b.handel-dev.local:8080/FinancialService_NS">
<irate xsi:type="xsd:float">4.000000</irate>
<CFSequence xmlns="http://dev-b.handel-dev.local:8080/FinancialService_NS">
<CF xsi:type="xsd:float">-100.000000</CF>
<CF xsi:type="xsd:float">5.000000</CF>
<CF xsi:type="xsd:float">5.000000</CF>
<CF xsi:type="xsd:float">105.000000</CF>
</CFSequence>
</getPV>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
_________________________________ Wed Jul 20 15:39:15 2005 RESPONSE:
Server: ZSI/1.1 BaseHTTP/0.3 Python/2.3.4
Date: Wed, 20 Jul 2005 13:39:14 GMT
Content-type: text/xml; charset="utf-8"
Content-Length: 560
<?xml version="1.0" encoding="utf-8"?>
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:ZSI="http://www.zolera.com/schemas/ZSI/"
  SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" >
<SOAP-ENV:Body>
<PV xmlns="http://dev-b.handel-dev.local:8080/FinancialService_NS"
xsi:type="xsd:float">2.775091</PV>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
result: 2.775091
```

5.4 An Excel XP VB client for the FinancialService

5.4.1 VB FinancialService stub generation

Refer to 3.4.1 for details. The URL to receive the WSDL description is "http://dev-b.handel-dev.local:8080/FinancialServic
for this example.

5.4.2 VB FinancialService client implementation & spreadsheet access

Insert the following code (Insert->Module...):

```
Public Function getPV(ByRef irate As Double, ByRef CFs As Range) As Single

    Dim xml_getPV As MSXML2.IXMLDOMNodeList

    Set xml_document = CreateObject("Microsoft.XMLDOM")
    Dim xml_root As MSXML2.IXMLDOMNode
    Set xml_root = xml_document.createElement("getPV")
    xml_document.appendChild xml_root

    Dim irateNode As IXMLDOMNode
    Set irateNode = xml_document.createElement("irate")
    Dim irateVar As Variant
    irateVar = irate
    irateNode.nodeTypedValue = irate
    xml_root.appendChild irateNode

    Dim CFSeqNode As IXMLDOMNode
    Set CFSeqNode = xml_document.createElement("CFSequence")
    xml_root.appendChild CFSeqNode

    Dim CFNode As IXMLDOMNode
    For Each cf In CFs
      Set CFNode = xml_document.createElement("CF")
      CFNode.Text = cf.Value
      CFSeqNode.appendChild CFNode
    Next cf

    'Set xml_getPV = xml_document.getElementsByTagName("getPV")
    Set xml_getPV = xml_document.getElementsByTagName("getPV/*")

    Dim myFinancialService As New clsws_FinancialService
    getPV = myFinancialService.wsm_getPV(xml_getPV)


End Function
```

This implements the function `getPV` that can be invoked from within a spreadsheet. It expects the interest rate and a range of cells (containing the cash flows) as input parameters. As opposed to the previous rpc/literal examples, the generated VB client stubs expect us to build up the DOM structure for the request message on our own this time. The function does so and calls the `wsm_getPV` method of the `clsws_FinancialService` object - the "service proxy" instance of the actual web service. The resulting return value of the web service - the net present value of the cash flows - is then given back to the caller.

5.5 A gSOAP C++ client for the FinancialService

5.5.1 Code generation from WSDL

1. Header:

```
$ /apps/pydev/bin/wsdl2h -o financialService.h
http://dev-b.handel-dev.local:8080/FinancialService
**  The gSOAP WSDL parser for C and C++ 1.2.0
**  Copyright (C) 2000-2005 Robert van Engelen, Genivia Inc.
**  All Rights Reserved. This product is provided "as is", without any war-
ranty.
**  The gSOAP WSDL parser is released under one of the following two li-
censes:
**  GPL or the commercial license by Genivia Inc. Use option -
l for more info.
Saving financialService.h
Connecting to 'http://dev-b.handel-dev.local:8080/FinancialService'
to retrieve WSDL... done
Cannot open file 'typemap.dat'
Problem reading type map file typemap.dat.
Using internal type definitions for C++ instead.
To complete the process, compile with:
soapcpp2 financialService.h
```

2. Stub code:

```
$ /apps/pydev/bin/soapcpp2 -I /data/pydev/DOWNLOADS/gsoap-2.7/soapcpp2/
financialService.h
**  The gSOAP Stub and Skeleton Compiler for C and C++ 2.7.1
**  Copyright (C) 2000-2005, Robert van Engelen, Genivia Inc.
**  All Rights Reserved. This product is provided "as is", without any war-
ranty.
**  The gSOAP compiler is released under one of the following three li-
censes:
**  GPL, the gSOAP public license, or the commercial license by Genivia Inc.
**WARNING**: anonymous struct will be named '_Struct_1' (de-
tected at line 73 in
financialService.h)
Saving soapStub.h
Saving soapH.h
Saving soapC.cpp
Saving soapClient.cpp
Saving soapServer.cpp
Saving soapClientLib.cpp
Saving soapServerLib.cpp
Using ns1 service name: FinancialService_USCOREBinding
Using ns1 service style: document
Using ns1 service encoding: literal
Using ns1 service location: http://dev-b.handel-
dev.local:8080/FinancialService
Using ns1 schema namespace: http://dev-b.handel-
dev.local:8080/FinancialService.wsdl
Saving soapFinancialService_USCOREBindingProxy.h client proxy
Saving soapFinancialService_USCOREBindingObject.h server object
Saving FinancialService_USCOREBinding.getPV.req.xml sample SOAP/XML request
```

```
    Saving FinancialService_USCOREBinding.getPV.res.xml sample SOAP/XML response
    Saving FinancialService_USCOREBinding.nsmap namespace mapping table
    Compilation successful (1 warning)
```

5.5.2 Client implementation

```
#include "soapFinancialService_USCOREBindingProxy.h"
#include "FinancialService_USCOREBinding.nsmap"
int main()
{
    FinancialService fs;
    _ns2__getPV getPV;
    float PV;
    float irate;
    float CFval;
    std::string input;
    while (1) {
        std::cout << "Enter interest rate in %: ";
        std::cin >> irate;
        std::vector<float> CF;
        for (int t=0;; t++) {
            std::cout << "Enter CF(t=" << t << ")[e to exit]: ";
            std::cin >> input;
            if (input != "e") {
                CFval = atof(input.c_str());
                CF.push_back(CFval);
            } else {
                break;
            }
        }
        getPV.irate = irate;
        getPV.CFSequence.CF = CF;
        if (fs.__ns1__getPV(&getPV, PV) == SOAP_OK) {
            std::cout << "PV=" << PV << endl;
        } else
            soap_print_fault(fs.soap, stderr);
    }
}
```

5.5.3 gSOAP Client compilation

gcc 2.95.2

```
g++ -o myClient -R/apps/prod/lib  -I/apps/pydev/include -L/apps/pydev/lib
soapC.cpp soapClient.cpp myClient_use_proxy.cpp -lgsoap++ -lsocket
```

gcc 3.4.3

```
/apps/pydev/gcc/3.4.3/bin/g++ -o myClient_use_proxy -I/apps/pydev/include
-L/apps/pydev/gcc/3.4.3/lib -R /apps/pydev/gcc/3.4.3/lib
myClient_use_proxy.cpp soapC.cpp soapClient.cpp -lsocket -lgsoap++ -lnsl
```

6 Aftertoughts

[3] states that one of the advantages of “literal” binding is a reduced SOAP message payload, as unnecessary type information can be left out. This type information is not needed any more, as it is specified in the WSDL and thus known to both service producer and consumer. A closer look at the messages that ZSI produces, however, shows that the `xsi:type=■...■` attributes are packed into the SOAP message anyway (e.g. sample session output in section 3.3). While you can set an attribute `typed=0` for the typecode information, it remains unclear if you have to do that for every single message element.

Example: A modified SquareService implementation:

```
...
# >>> ADDED MANUALLY
# Here we hook in the actual worker method
response._return = self.getSquare(args._x)
# <<<
response.typecode.typed = 0
response.typecode.ofwhat[0].typed = 0
return response
....
```

A ZSI 1.6.1 patches

The following patches to ZSI 1.6.1 made all the examples “just work” for me. One would have to read up on/know the specs to decide if they are actually correct; I guess this is up to the ZSI specialists.

wsdl2python.py.patch (based on revision: # $Id: wsdl2python.py,v 1.69 2004/11/30 00:28:38 boverhof Exp $)

- Fixes the problem that namespaces are not put into the generated code for rpc/literal and document/literal. Note that ZSI clients seem not to care too much about these issues, but a gSOAP client will not accept the answer of a ZSI-based server if the namespace is not correct.
- For doc/literal, adds xmlns=”..” to top level simple elements

Code:

```
775,776c775,777
<                       namespace = ■
<                 self.typecode += '\n%s%s.typecode = Struct(%s,[%s], pname=name, aname="_%%s" %% name, oname="%%s  xmlns=\\"%s\\"" %% name )'\
---
>                       if namespace:
>                           namespace = "'" + namespace + "'"
>                 self.typecode += '\n%s%s.typecode = Struct(%s,[%s], pname=name, aname="_%%s" %% name, oname="%%s xmlns=\\"%%s\\"" %% (name, ns) )'\
778c779
<                                  tcs,namespace)
---
>                                  tcs)
783a785,786
>                   if namespace:
>                       namespace = "'" + namespace + "'"
795,796c798,799
<               self.typecode += '%stypecode = %s( name=%s, ns=None ).typecode'\
<                                % (ID1, message.getName(), name )
---
>               self.typecode += '%stypecode = %s( name=%s, ns=%s ).typecode'\
>                                % (ID1, message.getName(), name, namespace )
799,800c802,803
<               self.typecode += '\n%s%s.__init__( self, name=%s, ns=None )' \
<                                % (ID2, message.getName(), name )
---
>               self.typecode += '\n%s%s.__init__( self, name=%s, ns=%s )' \
>                                % (ID2, message.getName(), name, namespace)
1233c1236,1238
<               self.basector.set('\n\n%s%s.__init__(self,pname=name, aname="_%%s" %% name,  **kw)' % (ID3,tpc))
---
>               # If the element has a non-empty ns attribute, add xmlns to the tag oname
>               self.basector.set('\n\n%s%s.__init__(self, pname=name, aname="_%%s" %% name, oname="%%s" %% (name + (ns and \'
xmlns="%%s"\' %% ns)), **kw)' % (ID3,tpc))

>
```

ServiceProxy.py.patch (no revision information in file)

- Fixes a problem with rash conversion of toplevel element maxOccurs attribute value to int. This attribute might be None, so we need to check if conversion is possible.

Code:

```
224c224,232
< minOccurs = int(element.getAttribute('minOccurs'))
---
>
> # toplevel elements (=ElementDeclaration) must not have minOccurs and
> # maxOccurs attributes, as opposed to local elements. So while we can
> # query these, direct conversion to int might prove fatal.
> # Maybe this should be handled in the XMLSchema module? Hmm, while the
> # attributes themselves are prohibited in the schema spec, what is the
> # intended value for toplevel components?
>
> minOccurs = element.getAttribute('minOccurs')
228c236
< typeObj.repeatable = (maxOccurs == 'unbounded') or (int(maxOccurs) > 1)
---

> typeObj.repeatable = (maxOccurs == 'unbounded') or (maxOccurs and int(maxOccurs) > 1)
```

TC.py.patch (based on revision: # $Header: /cvsroot/pywebsvcs/zsi/ZSI/TC.py,v 1.35 2004/11/04 19:44:17 boverhof Exp $)

- When using doc/literal and returning a message that just consists of a simple element, a possible xmlns='...' attribute in the name must not occur in the closing tag, so we remove it. This patch fixes the serialize method, where necessary.
  Note: Did not change the serialize method for class Any, as I am not sure about the different naming options that seem to happen there (e.g. if "rpc" is used instead of "name", can it also contain 'xmlns=...'? If so, is 'xmlns=...' also guaranteed to be the last attr?)

Code:

```
28a29,38
> def _without_trailing_xmlns_attr(name):
>     """Return name with xmlns='...' attribute removed, for use in closing
>     tags. xmlns must be the last attr in name.
>     """
>     i = name.find('xmlns')
>     if i > 0:
>         return name[:i - 1]
>     else:
>         return name
>
276d285
<
451,456c460
<         i = n.find('xmlns')
<         if i > 0:
<             ctag = '</%s>' % n[:i - 1]
<         else:
<             ctag = '</%s>' % n
<
---
>         ctag = '</%s>' % _without_trailing_xmlns_attr(name)
627c631
<               (n, attrtext, tstr, pyobj, n)
---
>               (n, attrtext, tstr, pyobj, _without_trailing_xmlns_attr(n))
751a756
>         ctagn = _without_trailing_xmlns_attr(n)
757c762
<             print >>sw, ('<%s%s%s>INF</%s>') % (n, attrtext, tstr, n)
---
>             print >>sw, ('<%s%s%s>INF</%s>') % (n, attrtext, tstr, ctagn)
759c764
<             print >>sw, ('<%s%s%s>-INF</%s>') % (n, attrtext, tstr, n)
---
>             print >>sw, ('<%s%s%s>-INF</%s>') % (n, attrtext, tstr, ctagn)
761c766
<             print >>sw, ('<%s%s%s>NaN</%s>') % (n, attrtext, tstr, n)
---
>             print >>sw, ('<%s%s%s>NaN</%s>') % (n, attrtext, tstr, ctagn)
764c769
<               (n, attrtext, tstr, pyobj, n)
---
>               (n, attrtext, tstr, pyobj, ctagn)
798c803
<         print >>sw, '<%s%s%s>%d</%s>' % (n, attrtext, tstr, pyobj, n)
---
>         print >>sw, '<%s%s%s>%d</%s>' % (n, attrtext, tstr, pyobj, _without_trailing_xmlns_attr(n))
867c872
<         print >>sw, '</%s>' % n
---

>         print >>sw, '</%s>' % _without_trailing_xmlns_attr(n)
```

References

[1] Salz, Rich; Blunck, Christopher: ZSI: The Zolera Soap Infrastructure. <http://pywebsvcs.sourceforge.net/zsi.html>, Release 1.6.1, December 08, 2004.

[2] van Engelen, Robert: gSOAP 2.7.3 User Guide. <http://www.cs.fsu.edu/˜engelen/soap.html>, Jun 27, 2005.

[3] Butek, Russell: Which style of WSDL should I use? <http://www-106.ibm.com/developerworks/webservices/library/ws-whichwsdl>, May 24, 2005.