gSOAP for OpenVMS Integrity

March 2017

The gSOAP Toolkit for SOAP Web Services and XML-Based Applications is a cross-platform open source C and C++ software development toolkit. It generates C/C++ RPC code, XML data bindings, and efficient schema-specific parsers for SOAP Web services and other applications that benefit from an XML interface.

1. Introduction

Thank you for your interest in this port of gSOAP to OpenVMS. The current release of the OpenVMS gSOAP port is based on the 2.8.32 gSOAP distribution.

The initial motivation for porting gSOAP to OpenVMS was a question from a customer asking how they could call a remote Web service from their OpenVMS COBOL/ACMS application. A number of OpenVMS users have crafted novel solutions to address this type of problem; however, gSOAP potentially provides a more complete, proven, and flexible solution. Since legacy applications can easily call C libraries, gSOAP is well positioned to act as a Web service client for legacy applications written in any OpenVMS 3GL.

In addition, gSOAP may be used to implement Web services on the OpenVMS platform. The other popular technology for implementing Web services on OpenVMS is the Java implementation of Apache AXIS2 used in conjunction with Apache Tomcat. These offerings provide OpenVMS users with web service technology based on their choice of programming language and environment.

The following notes briefly describe how to install the gSOAP kit and how to get started with the simple example applications. If you have any trouble with the kit, have suggestions for how it might be improved, or would like a distribution for another version of the operating system, please let us know, and we will do our best to oblige.

Finally, be sure to read the gSOAP documentation (see http://www.cs.fsu.edu/~engelen/ for details), specifically the gSOAP User Guide. Aside from the handful of OpenVMS-specific functions that have been added as part of the porting exercise, there should be little or no variance in terms of usage from what is described in the User Guide (and if there is, we would like to hear about it).

1.1. What's new in this release?

- This release is based on gSOAP 2.8.32, which is a significant jump from the previous 2.8.22-based release. Please refer to documentation at http://www.cs.fsu.edu/~engelen/ for a description of these enhancements.

1.2. Outstanding issues

- When using gSOAP on systems that employ On Disk Structure Level 2 (ODS-2), the `soapcpp2` utility will fail if generating sample XML message files and there are Web Service methods with names longer than 35 characters. The workaround to this problem is to specify the `-x` command line option, which will suppress the generation of sample XML message files. This issue may be resolved in future releases; however installation on an ODS-5 file system is recommended.
- When using gSOAP servers in CGI mode with either Apache or WASD, problems can be encountered with the output of the CGI program being truncated. These problems are caused by

inappropriate interpretation of carriage-control characters. To avoid these problems, include the following code before calling `soap_serve`():

```
stdout = freopen("SYS$OUTPUT", "w", stdout, "ctx=bin", "ctx=xplct");
```

2. Requirements

The kit you are receiving has been compiled and built using the operating system and compiler versions listed below. While it is highly likely that you will have no problems installing and using the kit on systems running higher versions of the products listed, we cannot say for sure that you will be so lucky if your system is running older versions. Note that the UnZip utility is required to unpack the supplied ZIP kit.

- OpenVMS 8.4-2L1
- HP TCP/IP Services V5.7 or higher (it has been reported that the kit also works with the MultiNet TCP/IP stack)
- C compiler - HP C V7.3-018 or higher
- UnZip 5.42 (or similar) for OpenVMS (required to unpack the supplied ZIP kit)
- Optional components:
    - C++ compiler - HP C++ V7.3 or higher (note that the C++ compiler is only required if you wish to do development in C++ as opposed to C).
    - VSI SSL1 V1.0-2KA (only required if you wish to use SSL). Please note that this release will not work correctly with earlier versions of OpenSSL for OpenVMS.
    - VSI CSWS V2.4-3E (or higher) is required if you wish to use gSOAP in conjunction with MOD_GSOAP and the Apache web server. Please note that the `MOD_GSOAP.EXE` module provided with this kit gSOAP is not compatible with earlier versions of CSWS.

In addition to the above requirements, it is assumed that the reader has a good knowledge of OpenVMS and of software development in the OpenVMS environment.

2.1. Optional products

While gSOAP was implemented to facilitate the development of Web services applications using C and C++, language integration issues aside, under OpenVMS in particular there is no good reason why it cannot be used to develop Web services applications using practically any 3GL you can think of, such as COBOL, BASIC, Pascal, or FORTRAN. In addition, we have developed a simple module that can be used to implement a gSOAP-based Web services layer on top of an existing ACMS application. Optional products might therefore include your language compiler(s) of choice and ACMS.

It should probably be commented at this time that regardless of your language preference, it will invariably be necessary to write some C code; however, this is typically going to be little more than a thin veneer that takes care of language incompatibilities, such as C expecting null-terminated strings, and so on. You are therefore probably going to need to have a competent C programmer handy to assist with such matters. We will be more than happy to assist with any questions regarding this matter.

This release of gSOAP for OpenVMS includes support for FastCGI. See Section 5.5 for details regarding FastCGI for OpenVMS and how to obtain a kit.

3. Recommended reading

Before getting too carried away, do be sure to read the very comprehensive gSOAP User Guide (http://www.cs.fsu.edu/~engelen/soapdoc2.pdf).

4. Contents of the kit

The kit is currently provided as a ZIP file, which allows individual users to easily install the software under their own accounts, as opposed to having to negotiate with the system administrator for some sort of system-wide installation.

The "`unzip -l`" command may be used to obtain a complete listing of the contents of the ZIP file. The listing is not included here as it is of considerable in length.

5. Installing the kit

Unpacking and installing the ZIP file kit is straightforward. After copying the supplied ZIP file (`GSOAP-VMS-I64-15.ZIP`) to a suitable location (preferably on an ODS-5 disk), unpack the contents of the relevant ZIP file using the `unzip` command.

For example:

```
$ unzip GSOAP-VMS-I64-15.ZIP

       Archive:  DISK$ODS5DISK:[BIGGLES]gSOAP-VMS-I64-15.ZIP;1
          creating: [.gsoap.bin]
          creating: [.gsoap.include]
          creating: [.gsoap.lib]
              ...
              ...
```

After unpacking the kit you will have a `[.gsoap]` directory in your current directory that contains the extracted files.

5.1. Post-installation steps

To complete the installation (regardless of the installation method), it is now necessary to define the logical name `gsoap$root` to point to your top-level `[.gsoap]` directory. If you are performing a system-wide installation then this logical name should be defined using `/system`, otherwise a process-level logical can be defined. For example, something similar to one of the following commands might be used to define the `gsoap$root` logical name for a system-wide installation:

```
$ define/sys/exe/trans=conc gsoap$root NODDY$DKA0:[gsoap.]
```

or:

```
$ define/sys/exe/tran=(conc,term) gsoap$root -
"''f$getdvi("DISK$IVMS82","DEVNAM")'[sys0.syscommon.gsoap.]"
```

Before trying one of the examples, and after reading the gSOAP User Guide, you might like to have a quick look around the `[.gsoap]` directory tree.

<u>Note</u>: If you are upgrading from a previous release and use `mod_gsoap`, be sure to replace the current version of `mod_gsoap.exe` in `apache$common:[modules]` with the new version supplied with this release (`gsoap$root:[lib]mod_gsoap.exe`).

5.2. Privileges and quotas

A moderate level of privilege is required in order to run applications developed using the current release of the OpenVMS gSOAP port.

To ensure optimum performance, after opening send and receive sockets, the gSOAP code performs calls to the `setsockopt()` TCP/IP system call to increase the maximum buffer length for socket packets. In order to perform this socket operation, the account in question requires a system `UIC` or `SYSPRV`, `BYPASS`, or `OPER` privilege. Subsequent releases may provide facilities to make these calls to `setsockopt()` optional (perhaps based on the value or existence of some logical definition), or by the provision of a privileged image.

Generally speaking, there are no special gSOAP-imposed quota requirements for applications developed using the gSOAP port, although a reasonably high `BYTLM` is recommended. The authors typically operate with quota settings similar to the following (on I64 or Alpha), which should be more than adequate for most purposes:

```
Maxjobs:          0  Fillm:       4096  Bytlm:       2000000
Maxacctjobs:      0  Shrfillm:       0  Pbytlm:            0
Maxdetach:        0  BIOlm:        900  JTquota:        8192
Prclm:            0  DIOlm:        900  WSdef:          4096
Prio:             4  ASTlm:        900  WSquo:         16384
Queprio:          0  TQElm:        900  WSextent:      32767
CPU:         (none)  Enqlm:       8192  Pgflquo:     2000000
```

5.3. Debug library

To assist with debugging of applications, the OpenVMS gSOAP port includes the object library `gsoapdbg.olb`. This library is essentially the same as `gsoap.olb`; however the contents of the library have been compiled with the `SOAP_DEBUG` macro defined, which causes a considerable body of debug code to be included. Applications linked with the debug version of the object library with create three log files (`recv.log`, `sent.log`, and `test.log`) that will contain a considerable volume of potentially useful information about how the application is operating. Note that any of your application files that include `stdsoap2.h` should also be compiled with the macro `SOAP_DEBUG` defined. Please refer to the gSOAP documentation for more information regarding debugging.

5.4. SSL support

This release includes the object library `gsoapssl.olb`, which may be used to build gSOAP applications that use OpenSSL to provide secure communication. In order to make use of this capability, the following points should be observed:

- VSI SSL1 V1.0-2KA (or higher) must be installed and correctly configured on all relevant machines.
- All gSOAP-generated code or code that includes gSOAP header files must be compiled with the `WITH_OPENSSL` macro defined (`/define=WITH_OPENSSL`).
- Programs must be linked with the `gsoapssl.olb` object library and with the OpenSSL shareable images `SSL1$LIBSSL_SHR32.EXE` and `SSL1$LIBCRYPTO_SHR32.EXE`, which reside in `SYS$LIBRARY`.
- Refer to the relevant sections of the gSOAP User Guide for details of how to modify your code to deal with SSL and certificates.

The sample build procedure `build_with_ssl.com` in `gsoap$root:[samples.calc]` illustrates how to compile and link gSOAP applications to use OpenSSL in accordance with the notes presented above.

5.5. FastCGI support

In addition to the libraries described above, this release includes the object libraries `gsoapfcgi.olb` (C) and `gsoapxxfcgi.olb` (C++), which may be used to build gSOAP applications that use FastCGI (see http://www.fastcgi.org for additional information). In order to make use of this capability, the following points should be noted:

- FastCGI for OpenVMS must be installed and configured.
- All gSOAP C/C++ code or code that includes gSOAP header files must be compiled with the `WITH_FASTCGI` macro defined (`/define=WITH_FASTCGI`).

The sample code in `gsoap$root:[samples.fcgi]` illustrates how to compile and link gSOAP applications to use FastCGI. The reader should refer to the release notes provided with the FastCGI for OpenVMS software for additional information regarding use of the software and the configuration of WASD or Apache to act as a FastCGI server.

5.6. Possible issues for those not running ACMS

As noted previously, one of the OpenVMS-specific extensions that have been included with this kit is a module that can be used to implement a gSOAP-based Web services layer on top of an ACMS application. If you are not running ACMS on the OpenVMS system where you intend to use the gSOAP kit, you might want to remove the ACMS agent module from the pertinent object library to avoid linker warnings and other potential problems in the future:

```
$ lib/delete="agent"/log gsoap$root:[lib]gsoap.olb
```

Note the use of the double quotes around the module name in the above command.

This raises an interesting point that should probably be discussed at this time. Coming from UNIX-land, the gSOAP source code contains a good number of function names that exceed 31 characters in length along with a good number of function names that are in mixed case. This mixed case must be preserved when doing the OpenVMS port.

As a consequence, one needs to be very careful with regard to the case of function names in any developed code, and it may be necessary to use the `case_sensitive=yes|no` directive in linker options files. It may also be necessary to bracket some C header files with `#pragma` directives to control how the names of functions and variables specified in those header files are treated by the compiler. For example, the `#pragma` directives in the example below will ensure that the compiler does not upper-case function and variable names specified in the two `#include` statements, but will preserve case (refer to the C compiler documentation for more information):

```
#pragma names save
#pragma names as_is
#include "soapH.h"
#include "add.nsmap"
#pragma names restore
```

The examples and their associated build procedures will serve to illustrate this and several other matters.

6. Sample applications

The `gsoap$root:[samples]` directory contains several very simple examples that illustrate various aspects of the gSOAP toolkit and several specific features of the OpenVMS port. The following text briefly discusses each example. As a general point of note, each of the example directories contains a command procedure named `build.com` that can be run to build the example in question. These build procedures contain the minimum code in order to compile and link the samples.

Note: Amendments to them will be gladly received and incorporated with a future version.

Note that the examples below are built using samples directories installed with the kit. It is advisable to make copies of the files into a different directory in order to preserve the original kit contents.

1. `gsoap$root:[samples.calc]`

   This sample application is taken pretty much verbatim from one of the standard gSOAP samples. It has been extended to include a very simple COBOL client that calls one of the Web service methods.

Before running the build procedure, have a look through the code and refer to the gSOAP User Guide to help you understand what is going on.

The reason to glance through the code before running the build procedure is that quite a number of files – 23 including object and executables – are generated by gSOAP during the build which can make it a little hard to see what you actually started with.

Before running the build procedure, edit `calcclient.c` and `cobclient.cob`, and change the value of the URL variable. For example, in the COBOL client, change the URL in the following statement to something appropriate for your environment:

```
move "http://16.41.224.180:8181/calcserver.exe" to url.
```

Be sure to remember the port number, 8181 above, you have specified. If you now run the build procedure (`@build`), you should eventually end up with three executable programs: `calcclient.exe`, `calcserver.exe`, and `cobclient.exe` (obviously you need a COBOL compiler for this example).

Next, define a foreign command for `calcserver`:

```
$ calcserver :== $gsoap$root:[samples.calc]calcserver.exe
```

You should now be able to run `calcserver`, specifying the port number you used in your URL (see above) as the single command line parameter. For example, if decided to stick with port 8181 from the original sample code, you would type:

```
$ calcserver 8181
```

Assuming that all is well, `calcserver` will now be listening on port 8181 for incoming Web service requests. Try running `calcclient.exe` and `cobclient.exe` to verify that things are working correctly.

When looking through the code, you will have noticed that the COBOL client (`cobclient.cob`) calls several functions whose names begin with `GSOAP$`. You might also have noticed that the COBOL client does not call the generated gSOAP client stub directly, but instead calls a simple wrapper function written in C. The `GSOAP$` functions are the OpenVMS-specific extensions to gSOAP that were alluded to previously. A brief description of each of these functions is provided later in this document.

The reason that the COBOL client in this instance does not call the generated client stub directly is because COBOL treats all function names in uppercase[1], regardless of how you specify them in your COBOL code, and there is no particularly nice way around this. One way around this would have been to specify the details the Web service methods in uppercase; however, you are often going to require a wrapper anyway to deal with other 3GL language integration issues such as data type conversions, and so on.

Finally, in the `[.calc]` directory you will also see a DCL command procedure called `build_with_ssl.com`. As noted in Section 5.4, this command procedure illustrates how to compile and link gSOAP applications to use OpenSSL.

2. `[.quote]`

   This is another very simple example adapted from the gSOAP distribution. The example illustrates how to implement a simple client application to invoke the Web service at http://services.xmethods.net/soap to get the current share price for the specified company code. I will not go into too much detail around this example; however, one of the things to note is that you can specify proxy servers to allow you to call Web services through a firewall. Note that a `GSOAP$` function (`GSOAP$SET_PROXY`) is provided to allow you to specify the proxy server details from non-C languages such as COBOL.

[1] Note that the latest version of the COBOL compiler (V2.9) does in fact support mixed case function and variable names.

Note: the above example has been moved on the Xmethods Web site and will, as a result, not work. We will correct the documentation to reflect the change in the near future.

3. `[.math]`

   This example is designed to implement the same functionality as the math example provided with the HP Web Services Integration Toolkit (WSIT), as the authors intend to compare the performance of WSIT with gSOAP.[2]

   There is no client application provided with this example. soapUI was used as a generic SOAP client to test the Web service.

   If you wish to try using soapUI with this sample application, you will need to copy the WSDL file (`demo.wsdl`) generated by gSOAP for the application to your PC. If you require specific instructions around how to set up soapUI, please let us know and we will try to oblige.

4. `[.agent]`

   This example is only of relevance to those using ACMS. As noted previously, one of the OpenVMS extensions to the gSOAP toolkit is a module that can be used to implement a gSOAP-based Web services layer on top of an ACMS application.

   The contents of the `[.agent]` directory illustrate the usage of this module to call the "`add`" task implemented by the ACMS ADD example in `ACMSDI$EXAMPLES`. Again you will note that no client application is provided with this example. Once again, soapUI was used to test and exercise the application.

   Note: you will need to adapt the user name passed into HP ACMS to match that of your environment. Remember also that the username has to be defined in the HP ACMS User Definition File with the "`/agent`" privilege.

5. `[.COBOL]`

   This is a very simple example that illustrates how to create a Web Service that uses an existing COBOL application. The example does very little, but hopefully serves to illustrate some of the factors that need to be considered when developing such services. The document `readme.pdf` in this directory steps through and explains various aspects of the process. Note that the document (`readme.pdf`) also describes building a service that can run either standalone or under Apache via `mod_gsoap`. The notes describing how to set up and configure `mod_gsoap` may be useful to those considering this approach for the deployment of their services.

6. `[.FORTRAN]`

   This example provides a simple illustration of how to use gSOAP to create a Web Service that uses existing FORTRAN code, and how to create a simple client program to call this service from existing FORTRAN code. A description of the example is provided in the document `readme.pdf` in the example directory.

7. `[.fcgi]`

   This example illustrates how to use gSOAP with FastCGI. After building the application using the supplied command procedure (build.com), the reader should refer to the FastCGI for OpenVMS release notes for details of how to configure and run the application. See Section 5.5 for details for details on FastCGI for OpenVMS and how to obtain a kit.

8. `[.wsse]`

   This directory contains an example program that illustrates the use of the WSSE plug-in that is now included with the distribution. This fairly complicated example also uses the gSOAP DOM implementation and SSL. It should be noted that it will be necessary to obtain appropriate certificates in order to use the SSL functions.

[2] Note that there are some articles on gSOAP performance relative to products such as Apache Axis to be found on the gSOAP Web site (specifically, see http://www.cs.fsu.edu/~engelen/soapperformance.html).

In addition to the above examples, the authors have implemented several considerably more complex pieces of code using gSOAP without any problems. As noted early in this document, while there are still various bits and pieces to be ported, the main components of gSOAP are available and are fully functional.

7. Configuring and using Apache mod_gsoap

The COBOL example shipped with the kit can be built both for use with the gSOAP Standalone Server as well as with the Apache `mod_gsoap` module. This applies to any program, of course; these steps refer to the COBOL example, but may be used for any other sample routines.

The gSOAP Standalone Server is very useful for development and testing. However, the standalone server code provided with gSOAP is not as scalable or fault tolerant as the Apache networking code, and the server functionality provided by the standalone server is also somewhat limited. Therefore, for production environments it is better to run services under Apache using the `mod_gsoap` module.

Deploying services under Apache and `mod_gsoap` is straightforward, although a reasonably high level of privilege is required to perform the installation. Assuming that you have built the Web service to work with Apache (`@build apache`), the following steps need to be performed:

- Copy `gsoap$root:[lib]mod_gsoap.exe` to `apache$root:[modules]` Ensure that the ownership and permissions on the copied file are the same as the other modules in the directory

- Modify the Apache configuration file (`apache$root:[conf]httpd.conf`) to include the `mod_gsoap` module and a definition for your new service. For example, you might add the following statements at the bottom of the file:

  ```
  LoadModule gsoap_module modules/mod_gsoap.exe
  <Location /add>
     SetHandler gsoap-handler
     SOAPLibrary add_server
  </Location>
  ```

  The `LoadModule` statement instructs Apache to load the `mod_gsoap` module. The `<Location>` tag specifies an alias for and the location of our Web service. Assuming that Apache is running on node bugs.bunny.com on port 81, the `<Location /add>` tag means that we will access our service using a URL like http://bugs.bunny.com:81/add. The statement "`SOAPLibrary add_server`" instructs mod_gsoap that the shareable image implementing our service is `add_server`. In this case, `add_server` is a logical name. (See next step for the definition). You could alternatively specify the full path to the service (`add.exe`), of course

- Define a system-wide logical name for `add_server` that points to the service, that is, the sharable image called `add.exe` that we built with the `@build apache` command. If you felt so inclined, you could also install `add.exe`; however this is not necessary

- Restart Apache to pick up the new service

You should now be able to access the "add" service running under Apache using a URL similar to that specified above.

8. So what's missing?

As noted previously, the bulk of the gSOAP functionality is present, and it should be possible to do most of what is described in the gSOAP User Guide. If there are additional components that you would like to see added to the distribution, please let us know, and we will endeavour to factor your request into a subsequent release.