gSOAP 2.7.0 User Guide

Robert van Engelen
Genivia, Inc.,
engelen@genivia.com & engelen@acm.org

October 10, 2004

Contents

1 Introduction

Bold font denotes C and C++ keywords.

`Courier font` denotes HTTP header content, HTML, XML, XML schema content and WSDL content.

[Optional] denotes an optional construct.

The keywords "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC-2119.

3 Differences Between gSOAP Versions 2.4 (and Earlier) and 2.5

To comply with WS-I Basic Profile 1.0a, gSOAP 2.5 and higher adopts SOAP RPC literal by default. There is no need for concern, because the WSDL parser wsdl2h automatically takes care of the differences when you provide a WSDL document, because SOAP RPC encoding, literal, and document style are supported. A new soapcpp2 compiler option was added -e for backward compatibility with gSOAP 2.4 and e) to adopt SOAP o
to develop a service that uses SOAP encoding. You can also use the gSOAP compiler directives to specify SOAP encoding for individual operarations, when desired.

4 Differences Between gSOAP Versions 2.1 (and Earlier) and 2.2

gSOAP runtime environment API) and the functions in the sources generated by the gSOAP compiler (the gSOAP RPC+marshalling API). Therefore, clients and services developed with gSOAP 1.X need to be modified to accommodate a change in the calling convention used in 2.X: In 2.X, all

Section 8.2.4 presents a multi-threaded stand-alone Web Service that handles multiple SOAP requests by spawning a thread for each request.

6 Interoperability

gSOAP interoperability has been verified with the following SOAP implementations and toolkits:

Apache 2.2

Apache Axis

ASP.NET

Cape Connect

Delphi

easySOAP++

eSOAP

Frontier

GLUE

Iona XMLBus

kSOAP

MS SOAP

Phalanx

SIM

SOAP::Lite

SOAP4R

Spray

SQLData

Wasp Adv.

Wasp C++

White Mesa

xSOAP

ZSI

4S4C

7 Getting Started

clients and SOAP Web services can be developed in C and C++ with the gSOAP compiler without

The input and output parameters of a SOAP service method may be simple data types or compound data types, either generated by the WSDL parser or specified by hand. The gSOAP stub and skeleton compiler automatically generates serializers and deserializers for the data types to enable the generated stub routines to encode and decode the contents of the parameters of the remote methods in XML.

8.1.1 Example

The getQuote remote method of XMethods Delayed Stock Quote service (defined in the quote.h file obtained with the 'wsdl2h' WSDL parser) provides a delayed stock quote for a given ticker name. The WSDL description of the XMethods Delayed Stock Quote service provides the following details:

Endpoint URL:	`http://services.xmethods.net:80/soap`
SOAP action:	`""` (2 quotes)
Remote method namespace:	`urn:xmethods-delayed-quotes`
Remote method name:	`getQuote`
Input parameter:	`symbol` of type `xsd:string`
Output parameter:	`Result` of type `xsd:float`

The following getQuote.h

The use of the namespace pre x ns1__ in the remote method name in the function prototype declaration is discussed in detail in 8.1.2. Basically, a namespace pre x is distinguished by a pair of underscores in the function name, as in ns1__getQuote where ns1 is the namespace pre x and getQuote is the remote method ace (A single underscore in an identi er ace will b translated into a dash in XML, because dashes are more frequently used in XML compared to underscores, se Section 10.3.)

The gSOAP compiler is invoked from the command line with:

```
soapcpp2 getQuote.h
```

The compiler generates the stub routine for the getQuote remote method sp in the getQuote.h header l This stub routine can b called by a client program at any time to request a stock quote from the Delayed Ca5.4 the namespace tservics

When successful, the stub returns SOAP_OK and quote contains the latest stock quote. Otherwise,

guarantee exclusive access to runtime environments by threads. Also the use of any client calls within an active service method requires a new environment.

When the example client application is invoked, the SOAP request is performed by the stub routine soap_call_ns1__getQuote, which generates the following SOAP RPC request message:

```
POST /soap HTTP/1.1
Host:  services.xmethods.net
Content-Type:  text/xml
Content-Length:  529
SOAPAction:  ""

<?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:ns1="urn:xmethods-delayed-quotes"
  SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
<SOAP-ENV:Body>
<ns1:getQuote>
<symbol>IBM</symbol>
</ns1:getQuote>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

The XMethods Delayed Stock Quote service responds with the SOAP response message:

```
HTTP/1.1 200 OK
Date:  Sat, 25 Aug 2001 19:28:59 GMT
Content-Type:  text/xml
Server:  Electric/1.0
Connection:  Keep-Alive
Content-Length:  491

<?xml version="1.0" encoding="UTF-8"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
  soap:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
<soap:Body>
<n:getQuoteResponse xmlns:n="urn:xmethods-delayed-quotes">
<Result xsi:type="xsd:float">41.81</Result>
</n:getQuoteResponse>
</soap:Body>
</soap:Envelope>
```

The server's SOAP RPC response is parsed by the stub. The stub routine further demarshalls the data of `Result` element of the SOAP response and stores it in the quote parameter of soap_call_ns1__getQuote.

```
...
struct soap soap;
  oat quotes[3]; char *myportfolio[] = {"IBM", "MSDN"};
soap_init(&soap); // need to initialize only once
for (int i = 0; i < 3; i++)
   if (soap_call_ns1__getQuote(&soap, "http://services.xmethods.net:80/soap", "", myport-
folio[i], &quotes[i]) != SOAP_OK)
      break;
if (soap.error) // an error occurred
   soap_print_fault(&soap, stderr);
soap_end(&soap); // clean up all deserialized data
...
```

This client composes an array of stock quotes by calling the ns1__getQuote stub routine for each

The first four namespace entries in the table consist of the standard namespaces used by the SOAP 1.1 protocol. In fact, the namespace mapping table is explicitly declared to enable a programmer to specify the SOAP encoding style and to allow the inclusion of namespace-prefix with namespace-name bindings to comply to the namespace requirements of a specific SOAP service. For example, the namespace prefix `ns1`, which is bound to `urn:xmethods-delayed-quotes` by the namespace mapping table shown above, is used by the generated stub routine to encode the `getQuote` request. This is performed automatically by the gSOAP compiler by using the `ns1` prefix of the `ns1__getQuote` method name specified in the `getQuote.h` header file. In general, if a function name of a remote method, **struct** name, **class** name, **enum** name, or field name of a **struct** or **class** has a pair of underscores, the name has a namespace prefix that must be defined in the namespace mapping table.

The namespace mapping table will be output as part of the SOAP Envelope by the stub routine. For example:

```
...
<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:ns1="urn:xmethods-delayed-quotes"
  SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
...
```

The namespace bindings will be used by a SOAP service to validate the SOAP request.

```
g;
class s__Address // a street address
f
   char *street;
   int number;
   char *city;
g;
```

The namespace pre x is separated from the name of a data type by a pair of underscores (__).

An instance o -11.955 Td[(cha)2871.696 0 Td[(e)]TJ ET -366.837 -18.246 2.989 0.398 re f -361.59 -18.246 2.989 0

```
ypace="s:Address">
="string">Technology Drive</street>
e="int">5</number>
string">Softcity</city>
```

ing table of the client program must have entries forcha

```
//gsoap ns1 service encoding: encoded
//gsoap ns1 service method-action: getQuote ""
int ns1__getQuote(char *symbol, float &Result);
```

The first three directives provide the service name which is used to name the proxy class, the service location (endpoint), and the schema. The forth and fifth directives specify that SOAP RPC encoding is used, which is required by this service. The last directive defines the optional SOAPAction, which is a string associated with SOAP 1.1 operations. This directive must be provided for each remote method when the SOAPAction is required. Compilation of this header file with the gSOAP compiler soapcpp2 creates a new file soapQuoteProxy.h with the following contents:

```
#include "soapH.h"
class Quote
{ public:
  struct soap *soap;
  const char soapoin
```

8.1.5 XSD Type Encoding Considerations

Many SOAP services require the explicit use of XML schema types in the SOAP payload. The

For example, when the client application calls the proxy, the proxy produces a SOAP request with an `xsd:string`:

```
// Contents of  le "getNames.h":
int ns3__getNames(char *SSN, struct
```

```
struct Namespace namespaces[] =
{
    {"SOAP-ENV", "http://schemas.xmlsoap.org/soap/envelope/"},
    {"SOAP-ENC","http://schemas.xmlsoap.org/soap/encoding/"},
    {"xsi", "http://www.w3.org/2001/XMLSchema-instance"},
    {"xsd", "http://www.w3.org/2001/XMLSchema"},
    {"ns1", "urn:galdemo:  ighttracker"
```

```
  SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
<return xmlns:ns2="http://galdemo.flighttracker.com" xsi:type="ns2:FlightInfo">
<equipment xsi:type="xsd:stcing">A320</equipment>
<airline xsi:type="xsd:stcing">UAL</airline>
<currentLocation xsi:type="xsd:stcing">188 mi W of Lincoln, NE</currentLocation>
<altitude xsi:type="xsd:stcing">37000</altitude>
<speed xsi:type="xsd:stcing">497</speed>
<flightNumber xsi:type="xsd:stcing">184</flightNumber>
</return>
</ns1:getFlightInfoResponse>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

The proxy returns the se2ng" .096.976 -11.777 2.989 0.398 re f 1 0 0185.093.986 -11.9 0 cm BT /

The proxy returns the service response in variable r of type **stcuct** ns1_ _getFlightInfoResponse and this information can be displayed by the client application with the following code fragment:

```
cout << r.return_.equipment << "   ight " << r.return_.airline << r.return_.  ightNumb7(e)-3r
  << " traveling "
```

Or, alternatively with a response **struct**:

```
// Contents of "getQuote.h":
typedef char *xsd__string;
typedef   oat xsd__ oat;
struct ns1__getQuoteResponse {xsd__ oat _return;};
int ns1__getQuote(xsd__string symbol, struct ns1__getQuoteResponse &r);
```

8.1.15 How to Specify a Method with No Output Parameters

To specify a remote method that has no output parameters, just provide a function prototype with a response struct that is empty. For example:

```
enum ns__event { on, stand_by };
int ns__signal(enum ns__event in, struct ns__signalResponse { } *out);
```

as the `xsd:double`

f"SOAP-ENV", "http://schemas.xmlsoap.org/soap/envelope/"*g*,
f"SOAP-ENC", "http://schemas.xmlsoap.org/soap/encoding/"*g*,
f"xsi", "http://www.w3.org/2001/XMLSchema-instance"*g*,

8.2.2 MSVC++ Builds

Win32 builds need winsock.dll (MS Visual C++ "wsock32.lib") To do this in Visual C++

```
fprintf(stderr, "Socket connection successful: master socket = %d\n", m);
for (int i = 1; ; i++)
{
  s = soap_accept(&soap);
  if (s < 0)
  {
    soap_print_fault(&soap, stderr);
    break;
  }
  fprintf(stderr, "%d: accepted connection from IP=%d.%d.%d.%d socket=%d", i,
    (soap.ip>>24)&0xFF, (soap.ip>>16)&0xFF, (soap.ip>>8)&0xFF, soap.ip&0xFF, s);
  if (soap_serve(&soap) != SOAP_
```

8.2.4 How to Create a Multi-Threaded Stand-Alone Service

break;

g

fprintf(stderr, "Thread %d accepts socket %d connection from IP %d.%d.%d.%d\n",

i,s,(ip&0xFF),(ip&0xFF),(ip&0xFF),(ip&0xFF);

```
      m = soap_bind(&soap, NULL, port, BACKLOG);
      if (m < 0)
        exit(1);
      fprintf(stderr, "Socket connection successful %d\n", m);
      for (i = 0; i < MAX_THR; i++)
        soap_thr[i] = NULL;
      for (;;)
      f
        for (i = 0; i < MAX_THR; i++)
        f
          s = soap_accept(&soap);
          if (s < 0)
            break;
          fprintf(stderr, "Thread %d accepts socket %d connection from IP %d.%d.%d.%d\n",
i, s, (soap.ip>>24)&0xFF, (soap.ip>>16)&0xFF, (soap.ip>>8)&0xFF, soap.ip&0xFF);
          if (!soap_thr[i]) //   rst time around
          f
            soap_thr[i] = soap_copy(&soap);
            if (!soap_thr[i])
            exit(1); // could not allocate
          g
          else // recycle soap environment
          f
            pthread_join(tid[i], NULL);
            fprintf(stderr, "Thread %d completed\n", i);
            soap_destroy(soap_thr[i])59.775 2.989 0.398 re f   1 0 0 1 -1.176 -59.77-02 0 2.989 1.955 T7[(g)]TTJ/F1C+ +0 2.9
```

8.2.5 How to Pass Application Data to Service Methods

The **void** *soap.user

```
//gsoap ns service location: http://www.cs.fsu.edu/~engelen/calc.cgi
//gsoap ns schema namespace: urn:calccalc.cgi
```

In addition to the generation of the `ns.wsdl` file, a file with a namespace mapping table is generated by the gSOAP compiler. An example mapping table is shown below:

```
struct Namespace namespaces[] =
{
  {"SOAP-ENV", "http://schemas.xmlsoap.org/soap/envelope/"},
  {"SOAP-ENC", "http://schemas.xmlsoap.org/soap/encoding/"},
  {"xsi", "http://www.w3.org/2001/XMLSchema-instance", "http://www.w3.org/*/XMLSchema-
instance"},
  {"xsd", "http://www.w3.org/2001/XMLSchema", "http://www.w3.org/*/XMLSchema"},
  {"ns", "http://tempuri.org"},
  {NULL, NULL}
};
```

This file can be incorporated in the client/service application, see Section 10.4 for details on namespace mapping tables.

To deploy a Web service, copy the compiled CGI service application to the designated CGI directory of your Web server. Make sure the proper file permissions are set (`chmod 755 calc.cgi` for Unix/Linux). You can then publish the WSDL file on the Web by placing it in the appropriate Web server directory.

The gSOAP compiler also generates XML schema files for all C/C++ complex types (e.g. **struct**s and **class**es) when declared with a namespace prefix. These files are named

```
  <schema
    xmlns="http://www.w3.org/2000/10/XMLSchema"
    targetNamespace="http://tempuri.org"
    xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/">
    <complexType name="addResponse">
      <all>
        <element name="result" type="double" minOccurs="0" maxOccurs="1"/>
      </all>
      <anyAttribute namespace="##other"/>
    </complexType>
    <complexType name="subResponse">
      <all>
        <element name="result" type="double" minOccurs="0" maxOccurs="1"/>
      </all>
      <anyAttribute namespace="##other"/>
    </complexType>
    <complexType name="sqrtResponse">
      <all>
        <element name="result" type="double" minOccurs="0" maxOccurs="1"/>
      </all>
      <anyAttribute namespace="##other"/>
    </complexType>
  </schema>
</types>
<message name="addRequest">
  <part name="a" type="xsd:double"/>
  <part name="b" type="xsd:double"/>
</message>
<message name="addResponse">
  <part name="result" type="xsd:double"/>
</message>
<message name="subRequest">
  <part name="a" type="xsd:double"/>
  <part name="b" type="xsd:double"/>
</message>
<message name="subResponse">
  <part name="result" type="xsd:double"/>
</message>
<message name="sqrtRequest">
  <part name="a" type="xsd:double"/>
</message>
<message name="sqrtResponse">
  <part name="result" type="xsd:double"/>
</message>
<portType name="ServicePortType">
  <operation name="add">
    <input message="tns:addRequest"/>
    <output message="tns:addResponse"/>
  </operation>
  <operation name="sub">
    <input message="tns:subRequest"/>
    <output message="tns:subResponse"/>
```

```
  </operation>
  <operation name="sqrt">
    <input message="tns:sqrtRequest"/>
    <output message="tns:sqrtResponse"/>
  </operation>
</portType>
<binding name="ServiceBinding" type="tns:ServicePortType">
  <SOAP:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
  <operation name="add">
    <SOAP:operation soapAction="http://tempuri.org#add"/>
    <input>
      <SOAP:body use="encoded" namespace="http://tempuri.org"
        encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
    </input>
    <output>
      <SOAP:body use="encoded" namespace="http://tempuri.org"
        encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
    </output>
  </operation>
  <operation name="sub">
    <SOAP:operation soapAction="http://tempuri.org#sub"/>
    <input>
      <SOAP:body use="encoded" namespace="http://tempuri.org"
        encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
    </input>
    <output>
      <SOAP:body use="encoded" namespace="http://tempuri.org"
        encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
    </output>
  </operation>
  <operation name="sqrt">
    <SOAP:operation soapAction="http://tempuri.org#sqrt"/>
    <input>
      <SOAP:body use="encoded" namespace="http://tempuri.org"
        encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
    </input>
    <output>
      <SOAP:body use="encoded" namespace="http://tempuri.org"
        encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
    </output>
  </operation>
</binding>
<service name="Service">
  <port name="ServicePort" binding="tns:ServiceBinding">
    <SOAP:address location="http://location/Service.cgi"/>
  </port>
</service>
</definitions>
```

Option	Description
-c	generate pure C header file code
-e	enum names will not be prefixed

8.2.11 How to Use Client Functionalities Within a Service

A gSOAP service may make client calls to other services from within its remove methods. This is best illustrated with an example. The following example is a more sophisticated example that combines the functionality of two Web services into one new SOAP Web service. The service provides a currency-converted stock quote. To serve a request, the service in turn requests the

```
      return SOAP_OK;
   g
   soap->socket = socket;
   return SOAP_FAULT; // pass soap fault messages on to the client of this app
g
/* Since this app is a combined client-server, it is put together with
one header  le that describes all remote methods. However, as a consequence we
have to implement the methods that are not ours. Since these implementations are
never called (this code is client-side), we can make them dummies as below.
/
int ns1__getQlb           SOAP_OK;
                          /*etsoap
```

8.3 How to Use gSOAP for Asynchronous One-Way Message Passing

SOAP RPC client-server interaction is synchronous: the client blocks until the server responds

8.4 How to Use the SOAP Serializers and Deserializers to Save and Load Application Data

The gSOAP stub and skeleton compiler generates serializers and deserializers for all user-de ned

Type	Type Name
char*	string
wchar_t*	wstring
char	byte
bool	bool
double	double
int	int
oat	oat
long	long
LONG64	LONG64 (Win32)
long long	LONG64 (Unix/Linux)
short	short
time_t	time
unsigned char	unsignedByte
unsigned int	unsignedInt
unsigned long	unsignedLong
ULONG64	unsignedLONG64 (Win32)
unsigned long long	unsignedLONG64 (Unix/Linux)
unsigned short	unsignedShort
T[N]	ArrayNOfType where Type is the type name of T
T*	PointerToType where Type is the type name of T
struct Name	Name
class Name	Name
enum Name	Name

Consider for example the following C code with a declaration of p as a pointer to a **struct** ns__Peraon:

```
struct ns__Peraon { char *name; } *p;
```

To serialize p, its address is passed to the function soap_serialize_PointerTons__Peraon generated for this type by the gSOAP compiler:

```
soap_serialize_PointerTons__
```

soap_end() function. The soap_

where Type1 is the type name of Type1 and Type2 is the type name of Type2 (see table above). The
s namespace-pre x: type-name1 and namespace-pre x: type-name2 describe the schema types of the
ts. Use NULL to omit this type information.

ializing class instances, method invocations MUST be used instead of function calls, for

e obj.soap serialize(&soap)

Consider the following **struct**:

```
// Contents of file "tricky.h":
struct Tricky
{
  int *p;
  int n;
  int *q;
};
```

The following fragment initializes the pointer fields p and q to the value of field n:

```
struct soap soap;
struct Tricky X;
X.n = 1;
X.p = &X.n;
X.q = &X.n;
soap_init(&soap);
soap_begin(&soap);
soap_serialize_Tricky(&soap, &X);
soap_put_Tricky(&soap, &X, "Tricky", NULL);
soap_end(&soap); // Clean up temporary data used by the serializer
```

What is special about this data structure is that n is 'fixed' in the Tricky structure, and p and q both point to n. The gSOAP serializers strategically place the id-ref attributes such that n will be identified as the primary data source, while p and q are serialized with ref/href attributes.

The resulting output is:

```
<Tricky xsi:type="Tricky">
<p href="#2"/> <n xsi:type="int">1</n> <q href="#2"/> <r xsi:type="int">2</r> </Tricky>
<id id="2" xsi:type="int">1</id>
```

```
struct soap soap;
...
soap_init(&soap); // initialize at least once
[soap_imode(&soap,  ags);] // set input-mode  ags
soap_
```

```
soap_end(&soap); // remove temporary data, including the decoded data on the heap
soap_done(&soap); //  nalize last use of the environment
```

When you declare a soap struct pointer as a data member in a class, you can overload the >> operator to parse and deserialize a class instance from a stream:

```
istream &operator>>
```

```
int main()
{
  struct soap soap;
  ns__Person mother,27.7Z,aother,27.2(johnp;)]TJ 17.84 -11.955 Td[(mothe.namen)-333=t
```

```
struct Namespace namespaces[] =
{
    {"SOAP-ENV", "http://schemas.xmlsoap.org/soap/envelope/"},
    {"SOAP-ENC","http://schemas.xmlsoap.org/soap/encoding/"},
    {"xsi", "http://www.w3.org/2001/XMLSchema-instance"},
    {"xsd", "http://www.w3.org/2001/XMLSchema"},
    {
```

```
char *name;
struct soap *soap; // we need this, see below
```

```
    || p->soap_in(p.soap, NULL, NULL)
    || soap_end_recv(p.soap))
    ; // handle I/O error
  return i;
}
```

8.4.5 How to Specify Default Values for Omitted Data

The gSOAP compiler generates soap_default functions for all data types. The default values of the

```
int ns__login(char *uid = "anonymous", char *pwd = "guest", bool granted);
```

When the request message lacks uid and pwd parameters, e.g.:

```
<?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:ns="http://tempuri.org">
  <SOAP-ENV:Body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
    <ns:login>
    </ns:login>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

then the service uses the default values. In addition, the default values will show up in the SOAP/XML request and response message examples generated by the gSOAP compiler.

9 Using the gSOAP Stub and Skeleton Compiler

The gSOAP stub and skeleton compiler is invoked from the command line and optionally takes the name of a header file as an argument or, when the file name is absent, parses the standard input:

```
soapcpp2 [aheader file.h]
```

where aheader file.h is a standard C++ header file. The C++act aa-279(aprepr)-36(Tcssao+)-4.1and prdeucss]TJ -1
STe s014le.s-533(gSnerated)-3343by the C++arge

File Name	Description
soapStub.h soapH.h	A modi ed and annotated header le produced from the input header le

Option	Description
-1	Use SOAP 1.1 namespaces and encodings (default)
-2	Use SOAP 1.2 namespaces and encodings
-h	Print a brief usage message
-c	

9.5 How to use the gSOAP #import Directive

9.7 Compiling a gSOAP Client

After invoking the gSOAP stub and skeleton compiler on a header file description of a service, the client application can be compiled on a Linux machine as follows:

```
g++ -o myclient myclient.cpp stdsoap2.cpp soapC.cpp soapClient.cpp
```

```
// Contents of file "myserver.cpp"
#include "soapH.h";
int main()
{
  soap_serve(soap_new());
}
...
// Implementations of the remote methods as C++ functions
...
struct Namespace namespaces[] =
{ // {"ns-prefix", "ns-name"}
  {"SOAP-ENV", "http://schemas.xmlsoap.org/soap/envelope/"},
  {"SOAP-ENC", "http://schemas.xmlsoap.org/soap/encoding/"},
  {"xsi", "http://www.w3.org/2001/XMLSchema-instance"},
  {"xsd", "http://www.w3.org/2001/XMLSchema"
 {"ns1", "urn:my-remote-method"
 {NULL, NULL63 Tf -247.765 -1155.487 0 Td[(g)]TJ -70.431 -11.955 Td[(g)]TJ/F31 9.963 Tf 4.982 0 Td[(;)]TJ -4.982 -11.
```

c(t55 erve)]TJf -5 10.909 Tf -27.272 -37.212 Td[(g)]TJ -333(C++)class(tion)declarementation2((m

STL and STL templates The gSOAP compiler does not yet fully support STL. It supports STL

Flag	Description
SOAP_IO_FLUSH	Disable bu ering and ush output (default for all le-based output)
SOAP_IO_BUFFER	Enable bu ering (default for all socket-oriented connections)
SOAP_IO_STORE	Store entire message to calculate HTTP content length
SOAP_IO_CHUNK	Use HTTP chunking
SOAP_IO_LENGTH	Require apriori calculation of content length (this is automatic)
SOAP_IO_KEEPALIVE	Attempt to keep socket connections alive (open)
SOAP_ENC_XML	Use plain XML encoding without HTTP headers (useful with SOAP_ENC_ZLIB)
SOAP_ENC_DIME	Use DIME encoding (automatic when DIME attachments are used)
SOAP_ENC_MIME	Use MIME encoding (activate using soap_set_mime)
SOAP_ENC_SSL	Encrypt encoding (SSL)

Encrypt encoding (SSL) (DIME)-332(attac)27(hmen964 0 Td["h/F31ttps:"omatic)-d21.107o3

```
typedef int xsd__int;
class X
```

```
if (exception)
{
  char *msg = (char*)soap_
```

int soap_call_[namespace_pre x__]

response_

Code	Description
SOAP_OK	No error
SOAP_CLI_FAULT*	The service returned a client fault (SOAP 1.2 Sender fault)
SOAP_SVR_FAULT*	The service returned a server fault (SOAP 1.2 Receiver fault)
SOAP_TAG_MISMATCH	An XML element didn't correspond to anything expected
SOAP_TYPE	

char

http://tempuri.org

struct Namespace namespacesTable1[] = *f* ... *g*;
struct Namespace namespacesTable2[] = *f* ... *g*;
struct Namespace namespacesTable3[] = *f* ... *g*;
struct

```
          <xsd:double xsi:type="xsd:double">...</xsd:double>
```

xsd:duration

Another possibility is to use strings to represent unbounded integers and do the translation in code.

`xsd:long`

typedef char *xsd__normalizedString;

Type xsd__normalizedString declares a string type which is encoded as

```
<xsd:normalizedString xsi:type="xsd:normalizedString">...</xsd:normalizedString>
```

It is solely the responsibility of the application to make sure the strings do not contain carriage return (#xD), line feed (#xA) and tab (#x9) characters.

`xsd:positiveInteger` Corresponds to a positive unbounded integer (0). Since C++ does not

`xsd:token` Represents tokenized strings. Tokens are strings that do not contain the line feed (#xA) nor tab (#x9) characters, that have no leading or trailing spaces (#x20) and that have no internal sequences of two or more spaces. It is recommended to use strings to store `xsd:token` XML schema types. The type declaration is:

```
typedef char *xsd__token;
```

Type `xsd__token` declares a string type which is encoded as

```
<xsd:token xsi:type="xsd:token">...</xsd:token>
```

It is solely the responsibility of the application to make sure the strings do not contain the line feed (#xA) nor tab (#x9) characters, that have no leading or trailing spaces (#x20) and

```
<xsd:unsignedShort xsi:type="xsd:unsignedShort">...</xsd:unsignedShort>
```

Other XML schema types such as `gYearMonth`, `gYear`, `gMonthDay`, `gDay`, `xsd:gMonth`, `QName`, `NOTATION`, etc., can be encoded similarly using a **typedef** declaration.

class xsd__anyURI_: **public** xsd__anySimpleType *f*

Type	Allows Decoding of	Precision Lost?
bool	[xsd:]boolean	no
char* (C string)	any type, see /F28.5	no
wchar_t * (wide string)	any type, see /F28.5	no
double	[xsd:]double	no
	[xsd:]float	no
	[xsd:]long	no
	[xsd:]int	no
	[xsd:]short	no
	[xsd:]byte	no
	[xsd:]unsignedLong	no
	[xsd:]unsignedInt	no
	[xsd:]unsignedShort	no
	[xsd:]unsignedByte	no
	[xsd:]decimal	possibly
	[xsd:]integer	possibly
	[xsd:]positiveInteger	possibly
	[xsd:]negativeInteger	possibly
	[xsd:]nonPositiveInteger	possibly
	[xsd:]nonNegativeInteger	possibly
oat	[xsd:]float	no
	[xsd:]long	no
	[xsd:]int	no
	[xsd:]short	no
	[xsd:]byte	no
	[xsd:]unsignedLong	no
	[xsd:]unsignedInt	no
	[xsd:]unsignedShort	no
	[xsd:]unsignedByte	no
	[xsd:]decimal	possibly
	[xsd:]integer	possibly
	[xsd:]positiveInteger	possibly
	[xsd:]negativeInteger	possibly
	[xsd:]nonPositiveInteger	possibly
	[xsd:]nonNegativeInteger	possibly
long long	[xsd:]long	no
	[xsd:]int	no
	[xsd:]short	no
	[xsd:]byte	no
	[xsd:]unsignedLong	possibly
	[xsd:]unsignedInt	no
	[xsd:]unsignedShort	no
	[xsd:]unsignedByte	no
	[xsd:]integer	possibly
	[xsd:]positiveInteger	possibly
	[xsd:]negativeInteger	possibly
	[xsd:]nonPositiveInteger	possibly
	[xsd:]nonNegativeInteger	possibly

Type	Allows Decoding of	Precision Lost?
long	[xsd:]long	possibly, if **long** is 32 bit
	[xsd:]int	no
	[xsd:]short	no
	[xsd:]byte	no
	[xsd:]unsignedLong	possibly
	[xsd:]unsignedInt	no
	[xsd:]unsignedShort	no
	[xsd:]unsignedByte	no
int	[xsd:]int	no
	[xsd:]short	no
	[xsd:]byte	no
	[xsd:]unsignedInt	possibly
	[xsd:]unsignedShort	no
	[xsd:]unsignedByte	no
short	[xsd:]short	no
	[xsd:]byte	no
	[xsd:]unsignedShort	no
	[xsd:]unsignedByte	no
char	[xsd:]byte	no
	[xsd:]unsignedByte	possibly
unsigned long long	[xsd:]unsignedLong	no
	[xsd:]unsignedInt	no
	[xsd:]unsignedShort	no
	[xsd:]unsignedByte	no
	[xsd:]positiveInteger	possibly
	[xsd:]nonNegativeInteger	possibly
unsigned long	[xsd:]unsignedLong	possibly, if **long** is 32 bit
	[xsd:]unsignedInt	no
	[xsd:]unsignedShort	no
	[xsd:]unsignedByte	no
unsigned int	[xsd:]unsignedInt	no
	[xsd:]unsignedShort	no
	[xsd:]unsignedByte	no
unsigned short	[xsd:]unsignedShort	no
	[xsd:]unsignedByte	no
unsigned char	[xsd:]unsignedByte	no
time_t	[xsd:]dateTime	no(?)

enumeration-type identi er's name, with the usual namespace pre x conventions for identi ers. This can be used to explicitly specify the encoding style. For example:

enum ns1__weekday *f*

11.3.3 Initialized Enumeration Constants

The gSOAP compiler supports the initialization of enumeration constants, as in:

enum ns1__relation *f*

```
<xsd:boolean xsi:type="xsd:boolean">false</xsd:boolean>
```

accessors. This encoding is identical to the **class** instance encoding without inheritance and method declarations, see Section 11.5 for further details. However, the encoding and decoding of **struct**s is more e cient compared to **class** instances due to the lack of inheritance and the requirement by the marshalling routines to check inheritance properties at run time.

Certain type of elds of a **struct** can be (de)serialized as XML attributes. See 11.5.7 for more details.

11.5 Class Instance Encoding and Decoding

A **class** instance is encoded as a SOAP compound data type such that the **class** name forms the data type's element name and schema type and the data member elds are the data type's accessors. Only the data member elds are encoded in the SOAP payload. Class methods are not encoded.

The general form of a **class** declaration is:

```
class
```

Only single inheritance is supported by the gSOAP compiler. Multiple inheritance is not supported, because of the limitations of the SOAP protocol.

```
    int sides;
    enum ns__Color {Red, Green, Blue} color;
    ns__Shape();
    ns__Shape(int sides, enum ns__Green color);
    ~ns__Shape();
};
```

The implementation of the methods of **class** ns__Shape must not be part of the header file and need to be defined elsewhere.

An instance of **class** ns__Shape with name Triangle, 3 sides, and color Green is encoded as:

```
<ns:Shape xsi:type="ns:Shape">
<name xsi:type="string">Triangle</name>
```

```
class update
{ public:
    time_t __item;
    int set(struct soap *soap);
};
```

The setter method assigns the current time:

```
int update::set(struct soap *soap)
{
    this->__item = time(NULL);
    return SOAP_OK;
}
```

Therefore, serialization results in the inclusion of a time stamp in XML.

Caution: a get
The method is not invoked when the element is an `element or has an`

```
    cout << "print(): Derived class instance " << name << " " << num << endl;
}
```

Below is an example CLIENT application that creates a Derived class instance that is passed as the input parameter of the remote method:

```
// CLIENT
#include "soapH.h"
int main()
{
  struct soap soap;
  soap_init(&soap);
  Derived obj1;
  Base *obj2;
  struct methodResponse r;
  obj1.name = "X";
  obj1.num = 3;
  soap_call_method(&soap, url, action, &obj1, r);
  r.obj2->print();
}
...
```

The following example SERVER1 application copies a class instance (Base or Derived class) from the input to the output parameter:

```
// SERVER1
#include "soapH.h"
int main()
{
  soap_serve(soap_new());
}
int method(struct soap *soap, Base *obj1, struct methodResponse &result)
{
  obj1->print();
  result.obj2 = obj1;
  return SOAP_OK;
}
...
```

The following messages are produced by the CLIENT and SERVER1 applications:

```
CLIENT: created a Derived class instance
SERVER1: created a Derived class instance
SERVER1: print(): Derived class instance X 3
CLIENT: created a Derived class instance
CLIENT: print(): Derived class instance X 3
```

Which indicates that the derived class kept its identity when it passed through SERVER1. Note that instances are created both by the CLIENT and SERVER1 by the demarshalling process.

```
// Contents of file "base.h":
class Base
{
  public:
  char *name;
  Base();
  virtual void print();
};
int method(Base *in, Base *out);
```

11.5.7 XML Attributes

```
struct xsd__string
{
    char *__item;
    @ xsd__boolean    ag;
};
```

11.6 Pointer Encoding and Decoding

The serialization of a pointer to a data type amounts to the serialization of the data type in SOAP

Since both a and b elds of P point to the sace integer, the encoding of P is multi-reference:

```
<ns:record xsi:type="ns:record">
  <a href="#1"/>
  <b href="#1"/>
</ns:record>
<id id="1" xsi:type="xsd:int">123</id>
```

Now, the decoding of the content in the R data structure that does not use pointers to integers results in a copy of each multi-reference integer. Note that the two **struct**s resemble the sace XML

This method has a polymorphic input parameter data and a polymorphic output parameter return_. The __type parameters can be one of SOAP_TYPE_xsd__string, SOAP_TYPE_xsd__int, SOAP_TYPE_xsd__ oat, SOAP_TYPE_ns__status, or SOAP_TYPE_ns__widget. The WSDL produced by the gSOAP compiler declares the polymorphic parameters of type `xsd:anyType` which is "too loose" and doesn't allow the gSOAP importer to handle the WSDL accurately. Future gSOAP releases might replace `xsd:anyType` with a `choice`

header le.

11.8 Fixed-Size Arrays

Fixed size arrays are encoded as per SOAP 1.1 one-dimensional array types. Multi-dimensional xed size arrays are encoded by gSOAP as nested one-dimensional arrays in SOAP. Encoding of xed size arrays supports partially transmitted and sparse array SOAP formats.

delete. Such dynamic allocations are exible, but pose a problem for the serialization of data: how does the array serializer know the length of the array to be serialized given only a pointer to the sequence of elements? The application stores the size information somewhere. This information is crucial for the array serializer and has to be made explicitly known to the array serializer by packaging the pointer and array size information within a **struct** or **class**.

11.9.1 SOAP Array Bounds Limits

SOAP encoded arrays use the

To encode the data type as an array, the name of the **struct** or **class** SHOULD NOT have a namespace

The deserializer of a dynamic array can decode

Caution: SOAP 1.2 does not support partially t

__o set eld of a dynamic

The following example header

__SOAPService

f

```
public:
int ID;
char *name;
char *owner;
char *description;
char *homepageURL;
char *endpoint;
char *SOAPAction;
char *methodNamespaceURI;
char
```

```
__ptr = NULL;
__size = 0;
```

```
f
  __ptr = NULL;
  __size = 0;
  __o set = 1;
g
Vector::Vector(int n)
f
  __ptr = ( oat*)malloc(n*sizeof( oat));
  __size = n;
  __o set = 1;
g
Vector::~Vector()
f
  if
```

```
// Contents of file "matrix.h":
class Matrix
{
  public:
  Vector *__ptr;
  int __size;
  int __offset;
  Matrix();
  Matrix(int n, int m);
  ~Matrix();
  Vector& operator[](int i);
};
```

The Matrix type is essentially an array of pointers to arrays which make up the rows of a matrix. The encoding of the two-dimensional dynamic array in SOAP will be in nested form.

11.9.6 Multi-Dimensional Dynamic Arrays

The general form of the **struct** declaration for K-dimensional ($K > 1$) dynamic arrays is:

```
struct some_name
{
  Type *__ptr;
  int __size[K];
  int __offset[K];
  ... // anything that follows here will be ignored
};
```

where Type MUST be a type associated with an XML schema, which means that it must be a **typedef**ed type in case of a primitive type, or a **struct**/**class** name with a namespace prefix for schema association, or another dynamic array. If these conditions are not met, a generic vector XML (de)serialization is used (see Section 11.9.7).

An alternative is to use a **class** with optional methods:

```
class some_name
{
  public:
  Type *__ptr;
  int __size[K];
  int __offset[K];
  method1;
  method2;
  ... // any fields that follow will be ignored
};
```

In the above, K is a constant denoting the number of dimensions of the multi-dimensional array.

To encode the data type as an array, the name of the **struct** or **class** SHOULD NOT have a namespace prefix, otherwise the data type will be encoded and decoded as a generic vector, see Section 11.9.7.

The deserializer of a dynamic array can decode partially t8-30.itiedusti-dimensional arrays

TF84(,r)-333(Kexampl,)-434(dhe)-333(aollo)29(w)ng se a cmtrix

end of the list is reached, the bu ered elements are copied to a newly allocated space on the heap for the dynamic array.

```
  typedef value_type * pointer;
  typedef const value_type * co
  typedef value_type & reference
  typedef const value_type & co
  typedef pointer iterator;
  typedef const_pointer const_itera
protected:
  iterator start;
  iterator  nish;
  size_t size;
public:
  simpleVector() { clear(); }
  ~simpleVector() { delete[] start; }
  void clear() { start =  nish = NULL; }
  iterator begin() { return start; }
  const_iterator begin() const { return start; }
  iterator end() { return  nish; }
  const_iterator end()       { return  nish; }
```

Caution: when parsing XML content the container elements may not be stored in the same order given in the XML content. When gSOAP parses XML it mses theinsert container methods to store

struct ArrayOfstring

```
class SOAP_ENC__base64
{
  unsigned char *__
```

WSDL in order for the gSOAP compiler to generate the (de)serialization routines. Alternatively, the optional DOM parser (dom.c and dom++.cpp) can be used to handle generic XML or arbitrary

```
    soap_print_fault(&soap, stderr);
  else
    printf("Time = %s\n", t);
  return 0;
}
```

To illustrate the manual doc/literal setting, the following client program sets the required properties before the call:

```
#include "soapH.h"
#include "localtime.nsmap" // include generated map file
int main()
{
  struct soap soap;
  char *t;
```

To declare a literal XML \type" to hold XML documents in wide character strings, use:

typedef wchar_t *XML;

Note: only one of the two storage formats can be used. The di erences between the use of regular strings versus wide character strings for XML documents are:

Regular strings for XML documents MUST hold UTF-8 encoded XML documents. That is, the string MUST contain the proper UTF-8 encoding to exchange the XML document in SOAP messages.

Wide character strings for XML documents SHOULD NOT hold UTF-8 encoded XML documents. Instead, the UTF-8 translation is done automatically by the gSOAP runtime marshalling routines.

Here is an example of a remote method speci cation in which the parameters of the remote method uses literal XML encoding to pass an XML document to a service and back:

typedef char *XML;
ns__GetDocument(XML m__XMLDoc, XML &m__XMLDoc_);

The ns__Document is essentially a **struct** that forms the root of the XML document. The use of the underscore in the ns__Document response part of the message avoids the name clash between the **struct**s. Assuming that the namespace mapping table contains the binding of ns to `http://my.org/` and the binding of m to `http://my.org/mydoc.xsd`, the XML message is:

```
<?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:ns="http://my.org/"
  xmlns:m="http://my.org/mydoc.xsd"
  SOAP-ENV:encodingStyle="">
  <SOAP-ENV:Body>
    <ns:GetDocument>
      <XMLDoc xmlns="http://my.org/mydoc.xsd">
```

The rst four elds in SOAP_

```
soap->fault->detail->__type = SOAP_TYPE_ns1__myStackData
```

See Section 17.2 on how to generate WSDL with the proper method-to-header-part bindings.

The `SOAP-ENV:mustUnderstand`

Function
void soap

```
    NULL, NULL);
    /* send the forms as MIME attachments with this invocation */
    if (soap_call_claim__insurance_claim_auto(soap, form1, form2, ...))
      // an error occurred
    else
      // process response
```

where the claim__form

Applications developed with gSOAP can transmit binary DIME attachments with or without

C++ programmers can use an iterator instead, as in:

```
for (soap_multipart::iterator attachment = soap.dime.begin(); attachment != soap.dime.end(); ++at-
tachment)
{
  cout << "DIME attachment:" << endl;
  cout << "Memory=" << (void*)(*attachment).ptr << endl;
  cout << "Size=" << (*attachment).size << endl;
```

elds in the struct/class, but additional elds and methods may appear after the eld declarations. An extended xsd__hexBinary declaration is similar.

The id and type elds contain text. The set the DIME-speci c options eld, you can use the soap_dime_option function:

char

f
: **unsigned char** *

Callback (function pointer)

void *(*soap.fdimereadopen)(

The following example illustrates the client-side initialization of an image attachment struct to stream a le into a DIME attachment:

```
int main()
{
  struct
```

```
soap_init(&soap);
soap.fdimewriteopen = dime_
```

15.5 Streaming Chunked DIME

gSOAP automatically handles inbound chunked DIME attachments (streaming or non-streaming).

The minOccurs and maxOccurs values must be integer literals. A default value can be provided

```
typedef int time__seconds "[1-5]?[0-9] | 60";
```

This defines the following schema type in time.xsd:

```
<simpleType name="seconds">
  <restriction base="xsd:int">
    <pattern value="[1-5]?[0-9]|60"/>
  </restriction base="xsd:int"/>
</simpleType>
```

17.2 Customizing the WSDL and Namespace Mapping Table File Contents With gSOAP Directives

A header le can be augmented with directives for L
automatically generate customized WSDL and namespace mapping tables contents. T
and namespace mapping table les do not need to be modi ed by hand (Sections 8.2.8 and 10. 3spaaddi(,9(3s5

A shortcut to de ne the default quali cation of elements and attributes of a schema:

```
//gsoap namespace-pre x schema form: quali ed
```

or:

```
//gsoap namespace-pre x schema form: unquali ed
```

To document a method, use:

```
//gsoap namespace-pre x service method-documentation: method-name //text
```

where

When literal encoding is required for a particular service method response when the request message is encoded, use:

```
//gsoap namespace-prefix service method-response-encoding: method-name literal
```

or when the `SOAP-ENV:encodingStyle` attribute is different from the SOAP 1.1/1.2 encoding style, use:

```
//gsoap namespace-prefix service method-response-encoding: method-name encoding-style
```

The automatic generation and inclusion of the namespace mapping table requires compiler directives for all namespace prefixes to associate each namespace prefix with a namespace URI. Otherwise, namespace URIs have to be manually added to the table (they appear as http://tempuri.org).

17.3 Transient Data Types

```
soap_done(soap);
free(soap);
```

It is also possible th serialize the tm elds as XML attributes using the @ quali er, see Section 11.5.7.

to include the proper schema de nitions in the WSDL produced by the gSOAP compiler, you should use quali ed **struct**, **class**, and **enum** names with a leading underscore, as in:

```
struct _ns__myStruct
{ ... };
```

This ensures that myStruct is associated with a schema, and therefore included in the appropriate schema in the generated WSDL. The leading underscore prevents the XML serialization of `xsi:type` attributes for this type in the SOAP/XML payload.

17.7 Function Callbacks for Customized I/O and HTTP Handling

gSOAP provides ve callback functions for customized I/O and HTTP Tandling:

Callback (function pointer)

int (*soap.fopen)(**struct** soap *soap, **const char** *endpoint, **const char** *host, **int** port)

Calledatxytoentœbcatedat endpoint. Input parameters host and port are micro-parsed endpoint. return a valid SOAP_INVALID_SOCKET and soap->error

soap

```
char buf[256];
if (lseek(soap->recvfd, 0, SEEK_SET) < 0 || soap_getline(soap, buf, 256)) // go to begin and
```

```
g
...
soap. gnore = myignore;
soap_call_ns__method(&soap, ...); // or soap_serve(&soap)
...
struct Namespace namespaces[] =
f
  f
```

```
struct soap soap;
soap_init(&soap);
...
soap.http_version = "1.0";
```

17.9 HTTP 307 Temporary Redirect Support

The client-side handling of HTTP 307 code "Temporary Redirect" and any of the redirect codes 301, 302, and 303 are not automated in gSOAP. Client application developers may want to consider adding a few lines of code to support redirects. It was decided not to automatically support redirects for the following reasons:

Redirecting a secure HTTPS address to a non-secure HTTP address via 307 creates a security vulnerability.

Cyclic redirects must be detected (e.g. allowing only a limited number of redirect levels).

cting aHTTP POST willaresult in re-serialization and re-post of the entire SOAP request.
The The(dec4050e.request.-497[(mss)ag)-4979mer-497[(e-p)-28(ost)d n nty when050e.req-issu5(a)-97[(te)-49

```
soap_serve(soap);
...
int http_get(struct soap *soap)
{
  soap_response(soap, SOAP_HTML); // HTTP response header with text/html
  soap_
```

```
   exit(1);
g
fprintf(stderr, "Socket connection successfuo3 T(1Fd\n"g)]TJ/F31 9.963 Tf183.0615 0 Td[, m1);
                                                                  argc, count;
                                                                                  g
```

A stand-alone gSOAP Web Service can enforce HTTP authentication upon clients, by checking the soap.userid and soap.passwd strings. These strings are set when a client request contains HTTP authentication headers. The strings SHOULD be checked in each service method (that requires authentication to execute).

Here is an example service method implementation that enforced client authentication:

```
int ns__method(struct soap *soap, ...)
f
   if (!soap->.userid || !soap->.passwd || strcmp(soap->.userid, "guest") || strcmp(soap->.passwd,
"visit"))        return 401; ...
g
```

w/304[1](Wed)-29am33 4009.659nrequires

17.18 Socket Options and Flags

gSOAP's socket communications can be controlled with socket options and ags. The gSOAP run-time environment **struct** soap ags are: **int** soap.socket_ ags to control socket send() and recv() calls, **int** soap.connect_ ags to set client connection socket options, **int** soap.bind_ ags to set server-side port bind socket options, **int** soap.accept_ ags to set server-side request message accept socket options. See the manual pages ob221a.7o5 9.96e16.447 0 0 Td[(ags)]J/F32 9.9a16.447 0(pages)-319(21]TJ5F32 9.

"server.pem", /* key le: required when server must authenticate to clients (see SSL docs on how to obtain this le) */

g
static void dyn_destroy_function(**struct** CRYPTO_dynlock_

or you can add the following line to soapdefs.h:

```
#de ne WITH_OPENSSL
```

and compile with option -DWITH_SOAPDEFS_H to include soapdefs.h

void sigpipe_handle(**int** x) *f g*

Caution: it is important that the WITH_OPENSSL macro MUST be consistently de ned to compile the sources, such as stdsoap2.cpp, soapC.cpp, soapClient.cpp, soapServer.cpp, and all application sources that include stdsoap2.h or soapH.h. If the macros are not consistently used, the application will

There should be a script called CA.sh (and a CA.pl that does the same). This hides all the

Of course the developer using your server cert on her machine will nd that if require_

It is also possible to convert IIS-generated certi cates to PEM format, see http://www.icewarp.com/Knowledgebase/617.l for example.

17.23 SSL Hardware Acceleration

You can specify a hardware engine to enable hardware support for cryptographic acceleration. This can be done once in a server or client with the following statements:

```
static const char *engine = "cswift"+l533*for*enginenamgine
```

The gzip compression is oo28Pgonal to all transpoo2 encodings such as HTTP, SSL, DIME, and can be used with other transpoo2 layers. You can even save and load compressed XML data to/from les.

gSOAP suppoo2s two compression formats: de ate and gzip. The gzip format is used by default.

and may speed up the transmission of compressed SOAP/XML messages. This is accomplished by setting the SOAP_IO

```
  char *path;
  long expire; /* client-side: local time to expire; server-side: seconds to expire */
  unsigned int version;
  short secure;
  short session; /* server-side */
  short env; /* server-side: 1 = got cookie from client */
  short modified; /* server-side: 1 = client cookie was modified */
  struct soap_cookie *next;
} g;
```

Since the cookie database is linked to a soap struct, each thread has a local cookie database in a multi-threaded implementation.

17.27 Server-Side Cookie Support

Server-side cookie support is optional. To enable cookie support, compile all sources with option -DWITH_COOKIES, for example:

```
g++ -DWITH_COOKIES -o myserver ...
```

gSOAP provides the following cookie API for server-side cookie session control:

Function

struct soap_cookie *soap_set_cookie(**struct** soap *soap, **const char** *name, **const char** *value, **const**

The cookie_path value is used to filter cookies intended for this service according to the path prefix rules outlined in RFC2109.

The following example server adopts cookies for session control:

```
int main()
{
  struct soap soap;
  int m, s;
  soap_init(&soap);
  soap.cookie_domain = "...";
  soap.cookie_path = "/"; // the path which is used to filter/set cookies with this destination
  if (argc < 2)
  {
    soap_getenv_cookies(&soap); // CGI app: grab cookies from 'HTTP_COOKIE' env var
    soap_serve(&soap);
  }
  else
  {
    m = soap_bind(&soap, NULL, atoi(argv[1]), 100);
    if (m < 0)
      exit(1);
    for (int i = 1; ; i++)
    {
      s = soap_accept(&soap);
      if (s < 0)
        exit(1);
      soap_serve(&soap);
      soap_end(&soap); // clean up
      soap_free_cookies(&soap); // remove all old cookies from database so no interference occurs
with the arrival of new cookies
    }
  }
  return 0;
}
int ck__demo(struct soap *soap, ...)
{
  int n;
  const char *s;
  s = soap_cookie_value(soap, "demo", NULL, NULL); // cookie returned by client?
  if (!s)
    s = "init-value"; // no: set initial cookie value
  else
    ... // modify 's' to reflect session control
  soap_set_cookie(soap, "demo", s, NULL, NULL);
  soap_set_cookie_
seconds
  return SOAP_OK;
}
```

17.28 Connecting Clients Through Proxy Servers

When a client needs to connect to a Web Service through a proxy server, set the

No canonical XML output

No logging

```
//gsoap ns schema namespace: urn:xmethods-CurrencyExchange
//gsoap ns service method-action: getRate ""
int ns__getRate(.398 re f  30.295 -05char
```

We also compile stdsoap2.c without namespaces:

```
gcc -c -DWITH_
```

```
#include "stdsoap2.h"
#de ne PLUGIN_ID "PLUGIN-1.0" // some name to identify plugin
struct plugin_data // local plugin data
{
  int (*fsend)(struct soap*, const char*, size_t); // to save and use send callback
```

g
// the new send callback
static int plugin_send(**struct** soap *soap, **const char** *buf, size