

Guía Rápida gSOAP

Autor: Manuel Castillo Cagigal

Índice general

1. Introducción	3
2. Como usar gSoap	5
2.1. Obtener gSoap	5
2.2. Herramientas a usar	5
2.3. Generación de las librerías	6
2.4. Uso en nuestro programa	7
2.5. Compilación	9
3. Librerías	11

Capítulo 1

Introducción

La herramienta gSoap realiza una conexión de lenguaje entre el protocolo SOAP/XML y el lenguaje de programación C/C++ facilitando considerablemente el desarrollo de servicios web y aplicaciones cliente/servidor en estos lenguajes.

SOAP (siglas de Simple Object Access Protocol) es un protocolo para el intercambio de estructuras de datos en la implementación de servicios web, este intercambio de datos depende de XML (eXtensible Markup Language) como formato de mensaje. SOAP puede formar la capa básica de una pila de un protocolo de servicio web, proveyendo un fragmentado de mensajes básico a partir del cual puede formarse el servicio web.

Muchas herramientas para C/C++ usan SOAP para implementar los servicios web y ofrecen APIs que requieren el uso de librerías de clases para estructuras de datos específicas de SOAP lo que nos obliga a adaptar la lógica de la aplicación a estas librerías. En cambio gSoap provee a C/C++ las APIs de SOAP de forma transparente ocultando al usuario detalles irrelevantes de las especificaciones de SOAP.

El objetivo de esta guía es mostrar los pasos básicos de esta herramienta, desde que no tenemos nada hasta conseguir las librerías con las clases de los objetos que posteriormente nos servirán para hacer peticiones al servidor de-

seado. Veremos también como compilar nuestros programas con gSoap y una explicación de las librerías que nos crea de forma que podamos entender como comunicarnos con el servidor. Aunque los pasos indicados y la información son bastante genéricos hay que resaltar que están orientados a la versión de gSoap 2.7.11 y en el lenguaje C++.

Capítulo 2

Como usar gSoap

2.1. Obtener gSoap

GSOap es una herramienta de software libre la cual podremos encontrar en su página oficial junto a su manual (<http://www.cs.fsu.edu/~engelen/soap.html>), está disponible en los sistemas operativos Linux, Windows y MaxOS.

Desde linux podemos obtener como un paquete más desde los repositorios, o bien descargando el código que encontraremos en la página antes citada.

La herramienta gSoap es autocontenida por lo que no hará falta descargar ningún paquete más, a no ser que queramos usar OpenSSL por lo que deberemos obtener las librerías correspondientes.

2.2. Herramientas a usar

Una vez instalado gSoap tendremos dos herramientas básicas con las cuales generaremos todas las librerías, con los siguientes ejecutables:

- wsdl2h

- soapcpp2

WSDL son las siglas de Web Services Description Language, un formato XML que se utiliza para describir servicios web, actualmente se usa la versión 2.0. WSDL describe la interfaz pública a los servicios Web. Está basado en XML y describe la forma de comunicación, es decir, los requisitos del protocolo y los formatos de los mensajes necesarios para interactuar con los servicios listados en su catálogo. Las operaciones y mensajes que soporta se describen en abstracto y se ligan después al protocolo concreto de red y al formato del mensaje. En nuestro caso `wSDL2h` (programa cliente) se conecta al servidor especificado, lee el WSDL para determinar que funciones están disponibles en el servidor y los tipos de datos con los que trabajan. Todo esto nos lo entregará en una librería la cual usaremos posteriormente con el compilador `soapcpp2` para crear las librerías de comunicaciones para nuestro programa C/C++.

2.3. Generación de las librerías

En este apartado explicaremos rápidamente como conseguir las librerías a usar en nuestro programa, posteriormente hablaremos de los posibles cambios en estas librerías y de su estructura que es realmente lo complicado de gSoap.

Estos pasos se explican en modo de ejemplo con un teórico servidor que tenemos en casa para el control domótico de la vivienda, en la página web oficial podremos realizar múltiples ejemplos los cuales tambien vienen explicados paso a paso.

Lo primero que debemos hacer es pedir las funciones y las estructuras de datos al servidor el cuál tiene esta información en formato WSDL:

```
$ wSDL2h -o salida.h http://192.168.1.210:8080/sah-ws
```

Esta URL será donde se encuentre la información del formato WSDL en el servidor. Esta instrucción genera la librería con las declaraciones de las APIs y

el tipo de dato de sus parámetros, por defecto generará el código suponiendo que se usará C++, para producir una aplicación en C puro se usa la opción '-c':

```
$ wsdl2h -c -o salida.h http://192.168.1.210:8080/sah-ws
```

La salida de esta instrucción (en nuestro caso sería salida.h) tiene forma de librería en C y como ya se ha mencionado contiene ya las APIs del servidor como si fueran funciones en C, pero todavía no hemos generado los stubs para las C/C++ APIs. Para hacer esto ejecutamos el compilador soapcpp2:

```
$ soapcpp2 -i -C -I(import) salida.h
```

Como vemos en el ejemplo tenemos un parámetro que es (import) esto deberemos cambiarlo por la dirección donde se encuentre la librería stlvector.h para soportar vectores STL. Esta librería viene dentro del directorio donde hayamos instalado gSoap:

```
P.ej: -I/home/control/gsoap/gsoap-2.7/gsoap/import
```

La opción -i indica que queremos objetos C++ proxy y servidor que incluye el cliente. La opción -C indica que sólo queremos los archivos del lado del cliente. Como puede verse el último parámetro, salida.h, será la salida de la herramienta wsdl2h (la cual como ya veremos podremos modificarla).

Ahora ya tendremos en nuestro directorio todo el código necesario para hacer peticiones al servidor, incluidas las tablas XML, este código debe de incluirse en nuestro programa como veremos más adelante.

2.4. Uso en nuestro programa

Como ya hemos comentado tras ejecutar el compilador soapcpp2 obtenemos las librerías necesarias para que nuestro programa pueda comunicarse con el servidor con tan sólo llamar a las funciones del objeto encargado de realizar la comunicación.

Lo primero que tendremos que hacer es incluir las librerías en nuestro programa:

```
#include "soap(nombre\_servicio)Proxy.h"
```

```
#include "(nombre\_servicio).nsmap"
```

El nombre del servicio dependerá del servidor y lo genera automáticamente soapcpp2, un ejemplo en un servidor el cual su servicio se llamara SAHBinding:

```
#include "soapSAHBindingProxy.h"
```

```
#include "SAHBinding.nsmap"
```

La primera de las librerías es la que contiene la clase con la cual crearemos el objeto encargado de realizar la comunicación con el servidor, la segunda hace referencia al mapa de espacios de nombres utilizado.

Ahora veremos un ejemplo de programa que usa estas librerías para comunicarse:

```
main() {
    SAHBindingProxy servicio; //Objeto encargado de comunicarse con el servidor.
    ns1__getAvailableDevicesIDsResponse resp; //Estructura para recibir los datos.
    int numdisp;
    if (servicio.getAvailableDevicesIDs(resp)== SOAP.OK){ //Sin problemas.
        numdisp = resp.return_>_size;
        std::cout << "Tenemos " << numdisp << " elementos." << std::endl;
    }
    else
        servicio.soap_stream_fault(std::cerr); //Informe de error.
    return 0;
}
```

Lo primero que hace el programa es crear el objeto servicio(puede usarse cualquier nombre) este objeto viene de la clase (nombre_servicio)Proxy la cual se encuentra en la librería antes mencionada, realmente esta es la relación que tiene nuestro programa con SOAP ya que toda comunicación son llamadas a funciones de esta clase quedando oculto para nosotros toda lo referente a la comunicación. Posteriormente creamos una variable del tipo ns1__getAvailableDeviceIDsResponse

la cuál será una estructura de datos creada por gSoap para recibir la información proveniente de nuestra función, posteriormente en el capítulo librerías veremos como ver estas estructura e incluso modificarlas. Realmente las funciones creadas por gSoap nos devolverá la información en el último de los parámetros de la función de la clase de comunicaciones ya que el dato que nos devuelve directamente la función es un informe sobre el estado de la conexión, en nuestro caso si no ha habido ningún problema en la comunicación seguiremos adelante y trabajaremos con la información que nos ha devuelto el servidor dentro de la estructura que nosotros tenemos en la variable 'resp'. En caso de error gSoap nos imprimirá por pantalla un informe del error. Lo errores es un tema bastane amplio en gSoap y hay varias maneras de que nos devuelva el informe de errores pero este tema no será tratado en esta guía rápida.

2.5. Compilación

Una vez tenemos nuestro programa acabado deberemos de incluir el código fuente del motor de gSoap en la compilación, este código se encuentra en el archivo stdsoap2.cpp que encontraremos junto a stdsoap2.h en el directorio de gSoap. Una vez copiemos estos archivo en el entorno de trabajo podremos proceder a la compilación. Este es un ejemplo de la llamada al compilador:

```
$ g++ -o IDdisp IDdisp.cpp soapC.cpp stdsoap2.cpp soapSAHBindingProxy.cpp
```

Aquí podemos ver todo en código fuente que necesitamos para compilar nuestro programa:

- El código de nuestro programa (IDdisp.cpp en este ejemplo).
- La implementación de las funciones usadas (soapC.cpp y soapSAHBindingProxy.cpp).
- El código del motor del entorno de gSoap (stdsoap2.cpp).

Capítulo 3

Librerías

En este capítulo se pretende comentar los archivos generados por gSoap y como poder ver en ellos las APIs generadas y las estructuras de datos con las que trabajará nuestro programa además de la oportunidad de modificar el tipo de dato que recibamos en nuestro programa proveniente del objeto de comunicación.

- Salida de wsdl2h:

Esta librería que podemos llamar salida.h sólo hace falta para compilar el resto con la herramienta soapcpp2 pero en ella pueden verse las APIs de nuestro servidor y las estructuras de datos con las que vamos a trabajar y ya que es el paso entre la petición del protocolo WSDL y la compilación final de nuestras librerías nos permite facilmente modificar los tipos de datos con los que vamos a trabajar.

Lo que primero nos encontramos en una cabecera explicando como se ha creado esta librería e información sobre el siguiente paso de compilación. Despues viene las diferentes secciones.

La sección 'Schema Types' contiene información sobre las estructuras de datos que se van usar aunque pueden venir en forma de clases, ya que heredan entre ellas. Es importante repasar estas estructuras porque serán las que usemos

en nuestros programas y es muy común cometer errores de compilación debido a lo enrevesadas que pueden llegar a ser las relaciones entre estas estructuras. Como se ha comentado anteriormente ya que aún no se han compilado las librerías que usaremos directamente en nuestros programas podemos realizar cambios en estas estructuras de datos de forma que cuando se reciban los datos en XML y los pasemos a C/C++ podemos definir a que tipo. Aquí podemos ver un ejemplo:

Código inicial:

```

/// "urn:sah":DeviceStateVariable is a complexType.
class ns1__DeviceStateVariable : public xsd\-\_anyType {
    public:
    /// Element variablename of type xs:string.
        std::string      variablename    1; ///< Required element.
    /// Element value of type xs:anyType.
        xsd__anyType*    value          1;  ///< Required element.
};

```

Código modificado:

```

/// "urn:sah":DeviceStateVariable is a complexType.
class ns1__DeviceStateVariable : public xsd\-\_anyType {
    public:
    /// Element variablename of type xs:string.
        std::string      variablename    1; ///< Required element.
    /// Element value of type xs:anyType.
        std::string      value          1; ///< Required element.
};

```

Como podemos ver se ha pasado del tipo `xsd__anyType*` a `std::string` de forma que cuando usemos la estructura `ns1__DeviceStateVariable` y queramos ver el contenido de la variable `value` podremos tratarla como una cadena. Esto podríamos por ejemplo también hacerlo de un entero a una cadena de forma que vez de recibir el valor numérico `gSoap` nos pasaría los datos a cadena de caracteres, el problema de esto es que podemos provocar serios problemas a la hora de trabajar con los datos como podría ser pasar caracteres a enteros por eso recomiendo que cualquier cambio se haga estandao muy seguro de lo que se

esta haciendo.

Más adelante llegamos a la sección 'Services' en la cual podemos ver las funciones disponibles en nuestro servidor es decir las APIs. Aquí podemos ver también la URL a donde se va a realizar la conexión con el servidor, esta URL puede modificarse en caso de que por ejemplo cambiemos el IP del servidor (NOTA: puede pasar que cuando se crea esta librería no se escriba automáticamente bien esta URL así que si hay errores de comunicación convendría comprobar que esta URL es correcta). Posteriormente viene definidas todas las APIs con sus parámetros con los que trabajarán tanto de entrada como la respuesta del servidor, también están algunas definiciones de estructuras de datos como son las estructuras de las respuestas del servidor.

Esta librería aunque tan sólo se usará para la generación de las demás librerías permite tener una buena de como funciona el servidor y de los servicios que nos va a poder proporcionar.

- soap(nombre_servicio)Proxy.h:

Esta librería, la cual debe ser llamada directamente por nuestro programa, es la que contiene la clase del objeto de comunicaciones que despues usaremos durante todo nuestro programa para comunicarnos con el servidor. Se puede cambiar el nombre de esta clase si queremos pero deberemos de hacerlo tambien en el archivo `soap(nombre_servicio)Proxy.cpp` ya que es donde estan declaradas las funciones de la clase.

Primero encontramos funciones propias de la comunicación con SOAP a las cuales en esta guía no haremos caso. Despues de estas funciones nos encontramos con las APIs de nuestro servidor, aquí podemos ver claramente las estructuras de datos que se vana a usar para los parámetros de las diferentes funciones.

- soapStub.h:

Nos encontraremos definidas detalladamente las clases y estructuras de datos utilizadas para la comunicación con el servidor.

- soapH.h:

Funciones internas de gSoap usadas para la comunicación.

- stdsoap2.h:

Librería con el motor del entorno de gSoap.