C++ Example

This example written in C++ allows developers to manage VoipNow accounts and view information about calls.

How To Install It

System Requirements for Linux

In order to be able to use the SystemAPI C++ example, you must have on your Linux system:

- GCC for Linux (for the demo, we used gcc version 4.8.5)
- gSOAP Toolkit for SOAP Web Services (recommended version: 2.8.32)
- OpenSSL (for the demo, we used OpenSSL version 1.0.1e)

Setup

STEP 1: Please download the files from our GitHub repository:

```
git clone https://github.com/4psa/systemapi-example-c.git
```

You should now find the following files in the systemapi-example-c folder:

File	Description
README.txt	Read me file that provides instructions on how to develop, compile and run SystemAPI applications.
createDemoFiles.sh	Script that automatically generates the stub files and compiles the demo applications.
Demo/DemoAddServiceProvider.cpp	Demo application that adds a new service provider.
Demo/DemoAddOrganization.cpp	Demo application that adds a new organization to a service provider.
Demo/DemoAddUser.cpp	Demo application that adds a new user to an organization.
Demo/DemoAddExtension.cpp	Demo application that adds a new extension to an organization.
Demo/DemoCallCosts.cpp	Demo application that makes a call report.
Demo/RandomChargingPlan.cpp Demo/RandomChargingPlan.h	Utility class that generates a random charging plan ID.
Demo/utils.cpp Demo/utils.h	Miscellaneous utilitary functions.
Demo/Makefile	Makefile that compiles and generates executables for the demo applications.

STEP 2: Please download the schema archive, extract the files and copy them to a directory that can be accessed over the Web. You should be able to find the files at `<PATH_TO_VOIPNOW>/admin/htdocs/soap2/schema/<VERSION>` where `<PATH_TO_VOIPNOW>` can be obtained by running as `root` this bash command:

```
grep "VOIPNOW_ROOT_D" /etc/voipnow/paths.conf | grep -v "#" | awk '{print $2}'
```

STEP 3: After you installed all the prerequisite tools and you have your schema files, you can use the script in the example archive to automatically generate stub files and compile the demo applications.

First you need to make sure that the script has 'execute' permissions by running the following command as root:

```
chmod +x createDemoFiles.sh
```

STEP 4: Now you can run the script with the following command:

```
./createDemoFiles.sh <server_hostname> <PATH_TO_GSOAP> <PATH_TO_SCHEMA_FILES>
```

The parameters have the following significance:

Parameter	Description
`<server_hostname>`	The server's hostname.
`<PATH_TO_GSOAP>`	Complete path to the folder where you extracted your `gsoap`.
`<PATH_TO_SCHEMA_FILES>`	Complete path to the folder where you have the schema files.

STEP 5: After the script has finished running, navigate to the `Demo/` folder from the location where you extracted the example archive. The script has copied to `Demo/` all the necessary stub files that were generated automatically with gsoap.

You can now use the Makefile from `Demo/` to compile the demo applications and generate the executables.

To compile all the demo applications, run from command line:

```
make
```

If you only want to compile one of the demo applications, run from command line:

```
make <TARGET_NAME>
```

where `<TARGET_NAME>` can be one of the following:

Target name	Description
demoaddserviceprovider	compile `DemoAddServiceProvider.cpp`
demoaddorganization	compile `DemoAddOrganization.cpp`
demoadduser	compile `DemoAddUser.cpp`
demoaddextension	compile `DemoAddExtension.cpp`
democallcosts	compile `DemoCallCosts.cpp`

If you want to write your own application, you can use the script to automate the setup process and modify the Makefile to include your application.

STEP 6: You can manually generate the necessary stub files and compile your applications. If you have not done this already, go through every `.wsdl` file in each module folder in your schema folder (`Account/`, `Billing/`, ...), find the line that contains "`CHANGEME`" and replace "`CHANGEME`" with the IP of your server.

STEP 7: After you have installed all the prerequisite tools and you have your schema files, you need to compile the `voipnowservice.wsdl` file from your schema folder. It is recommended that you use `wsdl2h`, that is the application provided by `gSOAP`. You can find it in your `gsoap` directory, i.e. `/<PATH_TO_GSOAP>/bin`.

Navigate to `/<PATH_TO_GSOAP>/bin` and run the followin command to compile the WSDL:

```
./linux386/wsdl2h -I /<PATH_TO_GSOAP>/WS/ -f -u -k -o voip.h voipnowservice.wsdl
```

For more information on the meaning of the parameters we used in the command above please run from command line:

```
./linux386/wsdl2h -help
```

To make things easy, we copied all the 4PSA VoipNow schemes to `/<PATH_TO_GSOAP>/bin`. Otherwise, you must specify the full path to `voipnowservice.wsdl`.

STEP 8: Next you need to generate C++ `stub` files. It is recommended that you use `soapcpp2`, that is the application provided by `gSOAP`. You can find it in the same directory as `wsdl2h`.

Here is the command line to generate `stubs`:

```
./linux386/soapcpp2 -C -L -I /<PATH_TO_GSOAP>/import/ -w -x voip.h
```

For more information on the meaning of the parameters we used in the command above please run from command line:

```
./linux386/soapcpp2 -help
```

STEP 9: Next you must copy to the folder that contains you applications source code (for the demo, this folder is `Demo/`) the following SystemAPI C++ files:

```
Organization.nsmap (you can add any *.nsmap file generated by {{gSoap}} , they are basically the same)
soapAccountProxy.h
soapBillingProxy.h
soapC.cpp
soapChannelProxy.h
soapClient.cpp
soapExensionProxy.h
soapGlobalOpProxy.h
soapH.h
soapOrganizationProxy.h
soapPBXProxy.h
soapReportProxy.h
soapServiceProvider.h
soapStub.h
soapUserProxy.h
```

You must also copy the following files from `/<PATH_TO_GSOAP>/` to the applications folder:

```
stdsoap2.cpp
stdsoap2.h
```

If you decide to write your own applications you must add the following #includes:

```
#include "soapAccountProxy.h"
#include "soapBillingProxy.h"
#include "soapChannelProxy.h"
#include "soapExtensionProxy.h"
#include "soapGlobalOpProxy.h"
#include "soapOrganizationProxy.h"
#include "soapPBXProxy.h"
#include "soapReportProxy.h"
#include "soapServiceProvider.h"
#include "soapUserProxy.h"
#include "Organization.nsmap"
#include "stdsoap2.h"
```

STEP 10: Next you must edit all `soap[MODULE]Proxy.h` files from you applications folder and add the following code after `soap_new()`:

```
soap_ssl_init();
if(soap_ssl_client_context(soap, SOAP_SSL_SKIP_HOST_CHECK, NULL, NULL, NULL, NULL, NULL))
        soap_print_fault(soap, stderr);
```

Each `soap[MODULE]Proxy.h` file will contain a class for that specific module and will set the endpoint (the server that we'll be making the requests to) to the value that was fetched from the `wsdl` file corresponding to that module.

The `nsmap` files will map the namespaces to the proper URLs, so that's why you only need one of these files.

Compile the sources using the following command:

```
g++ -lssl -DDEBUG -DWITH_OPENSSL -o <EXECUTABLE_NAME> <DEMOAPPNAME.cpp> soapC.cpp soapClient.cpp stdsoap2.cpp
```

You can now use the recently created classes to write applications that interact with your VoipNow server.

You will notice that after running the applications, there are three files in your application folder: TEST.LOG, SENT.LOG, and RECV.LOG; they contain useful logs about your application:

- SENT.LOG contains the packages sent to the server
- RECV.LOG contains the packages received from the server
- TEST.LOG contains various logging messages generated by the gSOAP runtime environment

Although everybody can use the SystemAPI C++ example, it is important that the user account used to connect to the VoipNow server is allowed to access the SystemAPI web service. For more information, please read the Access Management section.

How To Use It

Authentication to the SystemAPI

STEP 1: First you must create an object based on the module you are working with. For example, if you want to add a new service provider, you will need to create a ServiceProvider object.

```
ServiceProvider *serviceProvider = new ServiceProvider;
```

STEP 2: Then you need to create the SOAP header and fill in the user credentials.

Token-based Authentication

This type of authentication requires that you provide a valid access token that is generated through an OAuth flow using the key and secret of your application. The code should look like this:

```
std::string accessToken("<EXAMPLE_ACCESS_TOKEN>");
serviceProvider->soap->header = new SOAP_ENV__Header;
serviceProvider->soap->header->ns4__serverInfo = NULL;
serviceProvider->soap->header->ns4__userCredentials = new _ns4__userCredentials;
serviceProvider->soap->header->ns4__userCredentials->accessToken = accessToken;
```

In the examples above, ns4 should be replaced with the right namespace for the header data.

Please note that access tokens expire and can no longer be used for authentication. If your application fails and you get an error like:

```
<faultstring>The authentication data for access to this area or the object identification number is invalid.<
/faultstring><faultactor></faultactor><detail><message>The authentication data for access to this area or the
object identification number is invalid.</message><code>100</code></detail>
```

in RECV.log, the access token you provided for your application has expired and you need to generate a new token.

For more information on how to generate an access token for your application, please check Access Management document.

Examples

The SystemAPI C++ Tool contains demo applications that simulate the following operations:

- add a **Service Provider Account**,
- add an **Organization Account**,
- add a **User Account**,
- add an **Extension Account**,
- get the **Call Costs** for a User Account.

The source code for these demo applications can be found in the Demo/ folder which you extracted from the example archive.

Add a ServiceProvider

The following example shows how to add a new service provider.

The source code makes use of the class "RandomChargingPlan" which you can find in Demo/RandomChargingPlan.*.

You can also use this class when adding a new organization or a new user.

```
int main(int argc, char *argv[])
{
    // make sure that each time you run your application, a valid access token is provided
```

```cpp
    if (argc != 2) {
        std::cerr << "Usage: ./<executable_name> \"<access_token>\"" << std::endl;
        std::cerr << "    example: ./demoaddserviceprovider \"1|V_pmPvEm25-HrqAzERx_nvJbBvNs~q3F|1|v-gntT4GFH-
UCUX0EM2_r9XTVDtw~qCF\"" << std::endl;
        exit(EXIT_FAILURE);
    }

    // We will add a new Service Provider, so we need a ServiceProvider object
    ServiceProvider *serviceProvider = new ServiceProvider;
    if (NULL == serviceProvider) {
        std::cerr << "Failed creating a ServiceProvider object" << std::endl;
        exit(EXIT_FAILURE);
    }

    // Authentication Data - passed from command line
    std::string accessToken(argv[1]);

    // filling in the header with the user credentials
    serviceProvider->soap->header = new SOAP_ENV__Header;
    if (NULL == serviceProvider->soap->header) {
        std::cerr << "Failed creating a SOAP_ENV_HEADER object" << std::endl;
        exit(EXIT_FAILURE);
    }
    serviceProvider->soap->header->ns4__serverInfo = NULL;
    serviceProvider->soap->header->ns4__userCredentials =
            new _ns4__userCredentials;
    if (NULL == serviceProvider->soap->header->ns4__userCredentials) {
        std::cerr << "Failed creating an _ns4__userCredentials object" << std::endl;
        exit(EXIT_FAILURE);
    }
    serviceProvider->soap->header->ns4__userCredentials->accessToken = accessToken;

    // creating 2 objects for the request and for the response
    _ns7__AddServiceProvider *request = new _ns7__AddServiceProvider;
    if (NULL == request) {
        std::cerr << "Failed creating an AddServiceProvider object" << std::endl;
        exit(EXIT_FAILURE);
    }
    _ns7__AddServiceProviderResponse *response =
            new _ns7__AddServiceProviderResponse;
    if (NULL == response) {
        std::cerr << "Failed creating an AddServiceProviderResponse object" << std::endl;
        exit(EXIT_FAILURE);
    }

    /*
     * information about the new Service Provider
     * (name, login, password, country and charging plan)
     **/
    std::stringstream name_ss;
    std::stringstream login_ss;
    std::stringstream pass_ss;
    std::string name;
    std::string login;
    std::string pass;
    std::string country(COUNTRY);
    std::string chargingPlanID;

    srand(time(NULL));

    // filling in the information about the new organization (name, login, password, country and charging plan)
    name_ss << "ServiceProviderCPP_" << rand() % 1000;
    login_ss << "Admin_" << rand() % 1000;
    pass_ss << "Pass_" << rand() % 1000;

    name_ss >> name;
    login_ss >> login;
    pass_ss >> pass;

    request->name = &name;
    request->login = &login;
```

```
        request->password = &pass;
        request->country = &country;

        // get a random charging plan ID using RandomChargingPlan class
        RandomChargingPlan *rcp = new RandomChargingPlan(accessToken);
        if (NULL == rcp) {
            std::cerr << "Failed creating a RandomChargingPlan object" << std::endl;
            exit(EXIT_FAILURE);
        }

        chargingPlanID = rcp->getRandomChargingPlan();

        if (chargingPlanID != NO_CHARGING_PLAN_FOUND)
            request->ns5__chargingPlanID = &chargingPlanID;

        // making the request and getting the response
        int errCode = serviceProvider->__ns25__AddServiceProvider(request, response);
        if (SOAP_OK == errCode) {
            // no error
            std::cout << "OK adding service provider" << std::endl;
        } else {
            // error found
            soap *s = new soap;
            if (NULL == s) {
                std::cerr << "Failed creating a soap object" << std::endl;
                exit(EXIT_FAILURE);
            }
            s->error = errCode;
            soap_print_fault(s, stderr);
            std::cerr << "Please check the log files for more information" << std::endl;
        }
        return 0;
}
```

Adding Other Account Types

These examples can be found in the package downloaded with the SystemAPI C++ Tool.

If you wish to add other account types (organizations, extensions or users), there is not much to change from the program listed above. First of all, you will need to create the proper object, so instead of writing:

```
ServiceProvider *serviceProvider = new ServiceProvider;
```

You will need to write:

for adding an organization:

```
Organization *organization = new Organization;
```

for adding an extension:

```
Extension *extension = new Extension;
```

for adding an user:

```
User *user = new User;
```

The parameters that you need to fill in for the request are the same as before (name, login, password, country), except you also need to specify a parent ID. If you add an organization, the parent ID will represent the ID of the service provider that will own the organization; if you add a new user, the parent ID will represent the ID of the organization; if you add an extension, the parent ID will represent the ID of the user. In short, you will simply need to add this:

```
std::string parentID("1344");//instead of 1344, put a real id
request->parentID = &parentID;
```

The last thing you need to modify is the request name, which will switch from `AddServiceProvider` to `AddOrganization`, `AddUser` or `AddExtension`. For example:

```
int errCode = extension->__ns28__AddExtension(request, response);
```

Keep in mind that you may need to change `ns28` to the proper namespace that will be generated by `gsoap`. Also, each appearance of the `serviceProvider` object will have to be properly changed to the name of the object created (**extension**, **user** or **organization** in our case).

Call Costs

Here's another example that shows how to make a **CallReport** request.

```
int main(int argc, char *argv[])
{
    // make sure that each time you run your application, a valid access token is provided
    if (argc != 2) {
        std::cerr << "Usage: ./<executable_name> \"<access_token>\"" << std::endl;
        std::cerr << "    example: ./democallcosts \"1|V_pmPvEm25-HrqAzERx_nvJbBvNs~q3F|1|D~nbBUf87k~7I12F79T-
nJnHU12Y.4Aq\"" << std::endl;
        exit(EXIT_FAILURE);
    }

    // We will make a Call report, so we need a Report object
    Report *report = new Report;
    if (NULL == report) {
        std::cerr << "Failed creating a Report object" << std::endl;
        exit(EXIT_FAILURE);
    }

    // Authentication Data - passed from command line
    std::string accessToken(argv[1]);
    // filling in the header with the user credentials
    report->soap->header = new SOAP_ENV__Header;
    if (NULL == report->soap->header) {
        std::cerr << "Failed creating a SOAP_ENV__Header object" << std::endl;
        exit(EXIT_FAILURE);
    }
    report->soap->header->ns4__serverInfo = NULL;
    report->soap->header->ns4__userCredentials = new _ns4__userCredentials;
    if (NULL == report->soap->header->ns4__userCredentials) {
        std::cerr << "Failed creating an _ns4__userCredentials  object" << std::endl;
        exit(EXIT_FAILURE);
    }
    report->soap->header->ns4__userCredentials->accessToken = accessToken;

    // creating 2 objects for the request and for the response
    _ns21__CallReport *request = new _ns21__CallReport;
    if (NULL == request) {
        std::cerr << "Failed creating a CallReport object" << std::endl;
        exit(EXIT_FAILURE);
    }
    _ns21__CallReportResponse *response = new _ns21__CallReportResponse;
    if (NULL == response) {
        std::cerr << "Failed creating a CallReportResponse object" << std::endl;
        exit(EXIT_FAILURE);
    }

    // making the request and getting the response
    int errCode = report->__ns32__CallReport(request, response);
    if (SOAP_OK == errCode) {
        // no error
        std::cout << "OK retrieving call report\n";
        std::ifstream logfile("RECV.log");
        std::stringstream ssBuffer;
        std::string fileContent;
        ssBuffer << logfile.rdbuf();
        fileContent = ssBuffer.str();
        size_t posResult = fileContent.rfind(RESPONSE_BEGIN);
        std::cout << std::endl << std::endl << "Result is " << std::endl <<
```

```cpp
            fileContent.substr(posResult) << std::endl;
    } else {
        // error found
        soap *s = new soap;
        if (NULL == s) {
            std::cerr << "Failed creating a soap object" << std::endl;
            exit(EXIT_FAILURE);
        }
        s->error = errCode;
        soap_print_fault(s, stderr);
        std::cerr << "Please check the log files for more information" << std::endl;
    }
    return 0;
}
```