



## Using C/C++ and gSOAP with Avid Interplay WS

Revision 1.1 – April 7, 2008

---

<b>Overview</b>	<b>2</b>
Distribution and Licensing of gSOAP	2
General Conventions Used in this Document	2
Locating the WSDL and XSD files	2
<b>Creating a gSOAP Client with Microsoft Visual Studio .NET Step-by-Step</b>	<b>3</b>
Create a New Win32 Console Project	3
Compile the WSDL File for the Interplay WS Assets Service	5
Alternate Code Generation with gSOAP	7
Write Code to Use the Interplay WS Assets Service	8
<b>Advanced Topics</b>	<b>10</b>
Changing the Endpoint URL	10
Enabling Secure Transmission over HTTPS	11
Enabling MTOM Support	15

## Overview

The Interplay WS API provides [SOAP web services](#) for interacting with Avid Interplay. The first release provides operations for exchanging and modifying metadata in the Avid Interplay Engine. This document demonstrates how to create a C or C++ client for Interplay WS using Microsoft Visual Studio .NET and the [gSOAP](#) web services development toolkit.

## Distribution and Licensing of gSOAP

Although much of gSOAP is licensed under a lenient [gSOAP Public License](#), some important parts are licensed under either the [GPL](#) or [Genivia commercial license](#). Please be aware of these restrictions before developing a product using gSOAP. You can find more information at the [gSOAP web site](#).

## General Conventions Used in this Document

### Server name and port

The server name and port of the Interplay WS services will be different for each site. For the purposes of this documentation, we will be using the *localhost* server and port *80* (the default port). You should modify your work to reflect the actual hostname and port of your Interplay WS installation.

For simplicity, this documentation will mainly use *http*. Instructions for using *https* can be found near the end of this document. For security purposes, we recommend you use *https* in production environments.

### Workgroup name and server mapping

The Interplay WS application must be configured with workgroup name and server mappings. This can be done via the Avid Interplay Framework Configuration client. For the purposes of this documentation, we will assume that the workgroup name *WGA* is mapped to our Interplay Engine server.

## Locating the WSDL and XSD files

Since Interplay WS is a [SOAP](#) web service, it uses a [Web Services Description Language \(WSDL\)](#) file to describe its interface. This WSDL also references an [XML Schema Document \(XSD\)](#) to define the XML types used in the messages. In development, your SOAP toolkit will use the WSDL file to generate client code to access the service.

The WSDL for Interplay WS Assets can be found at:

<http://iws-srv/services/Assets?wsdl>

(Remember to change the server name and port to match your environment)

This WSDL will also reference the XSD, which can be found at:

<http://iws-srv/services/Assets?xsd=assets.xsd>

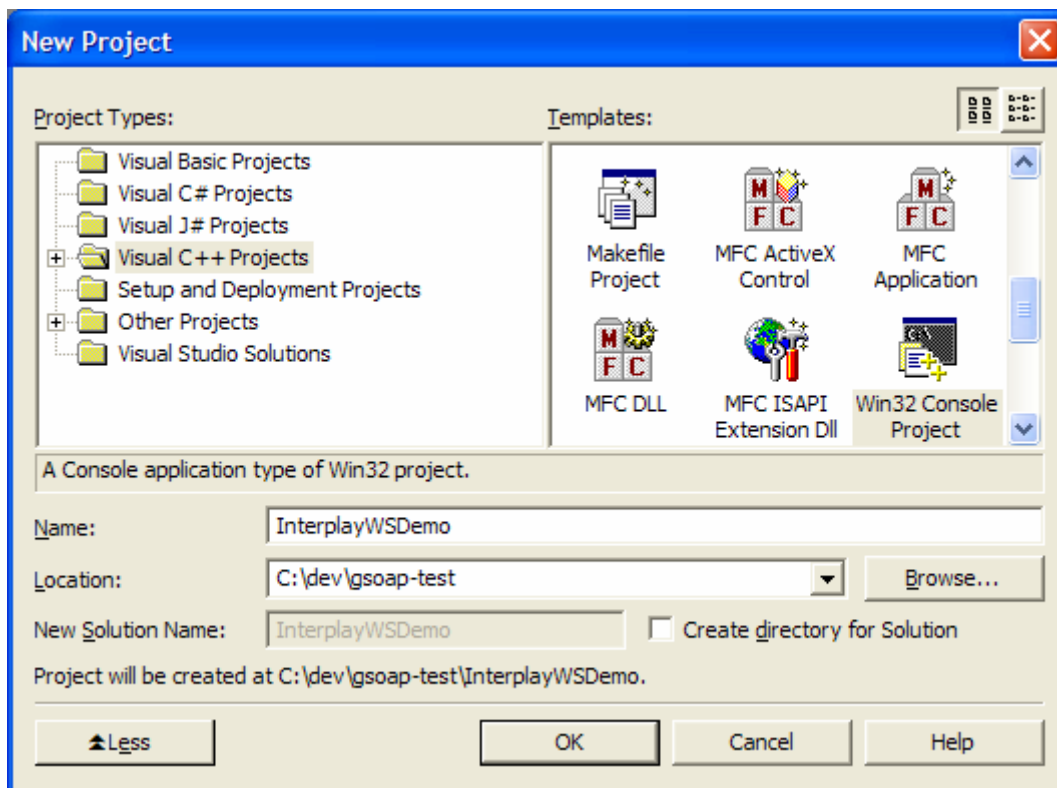
## Creating a gSOAP Client with Microsoft Visual Studio .NET Step-by-Step

For this tutorial, we will create a simple C++ Console application that gets the children of the *Projects/Rainforest* folder.

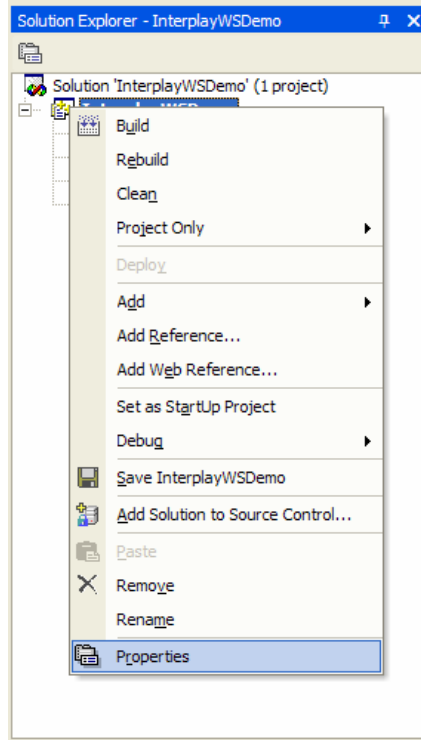
### Create a New Win32 Console Project

1. Create a new project by choosing *File -> New -> Project...*
2. Choose a Visual C++ Win32 Console Application, and enter the name and location of the project.

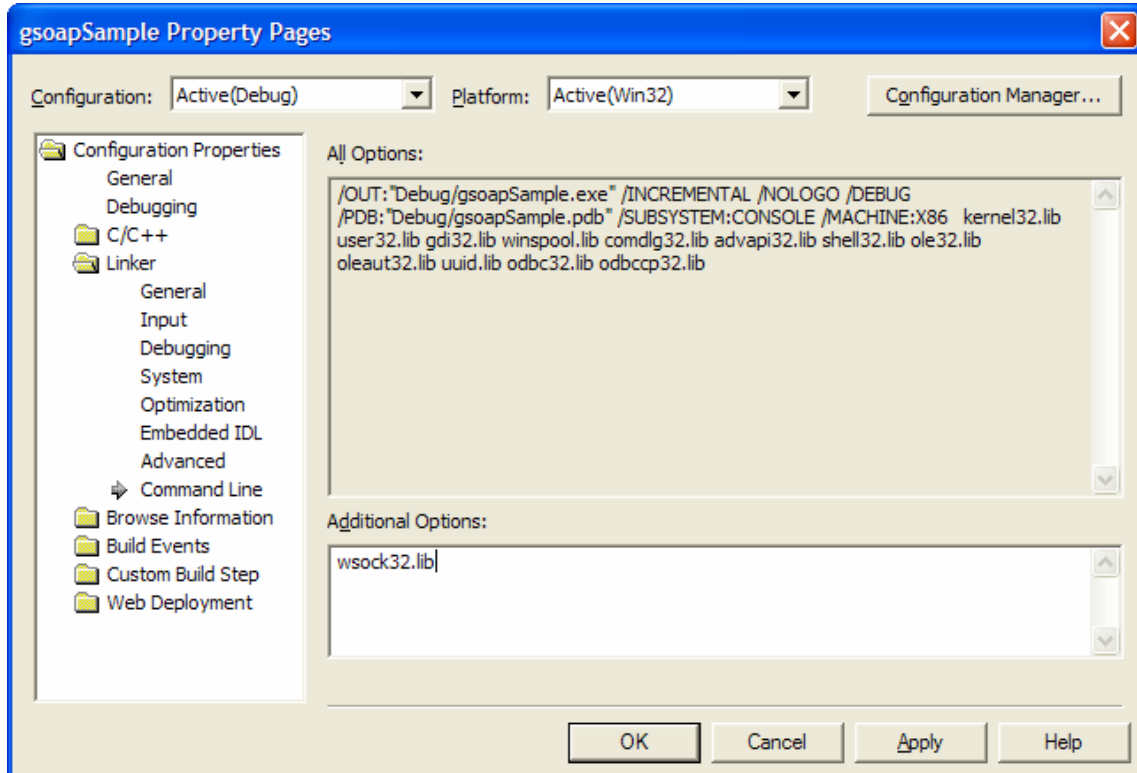
**Note:** It is assumed that the project path is `C:\dev\gsoap-test\InterplayWSDemo`. If your chosen project path is different, then be sure to make the appropriate changes when stepping through this sample.



3. Once created, right-click on the project in the Solution Explorer and select *Properties*.



4. Add `wsock32.lib` to *Configuration Properties* -> *Linker* -> *Command Line*.



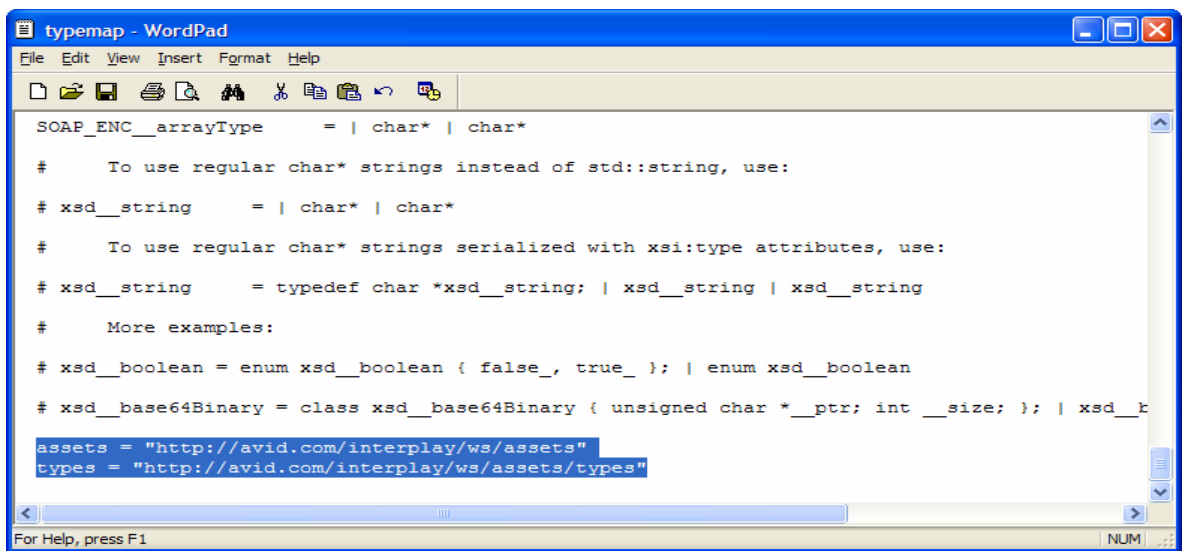
## Compile the WSDL File for the Interplay WS Assets Service

1. Create a new directory called `assets` in the project's path and copy the `typemap.dat` file from `C:\<Path to gSOAP>` into it.

**Note:** It is assumed that your path to gSOAP is `C:\gSOAP`. If your gSOAP installation is located elsewhere, then be sure to make the appropriate changes when stepping through this sample.

2. Open the new copy of the `typemap.dat` file and setup the namespaces that will be used by adding the following lines to the file:

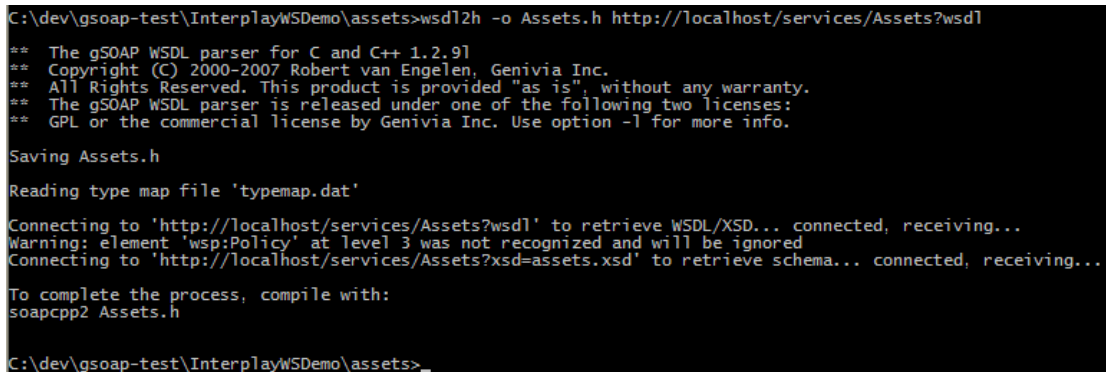
```
assets = "http://avid.com/interplay/ws/assets"  
types = "http://avid.com/interplay/ws/assets/types"
```



```
typemap - WordPad  
File Edit View Insert Format Help  
SOAP_ENC__arrayType = | char* | char*  
  
# To use regular char* strings instead of std::string, use:  
# xsd__string = | char* | char*  
  
# To use regular char* strings serialized with xsi:type attributes, use:  
# xsd__string = typedef char *xsd__string; | xsd__string | xsd__string  
  
# More examples:  
  
# xsd__boolean = enum xsd__boolean { false_, true_ }; | enum xsd__boolean  
  
# xsd__base64Binary = class xsd__base64Binary { unsigned char *_ptr; int __size; }; | xsd__b  
  
assets = "http://avid.com/interplay/ws/assets"  
types = "http://avid.com/interplay/ws/assets/types"
```

3. Open a new command prompt and navigate to your `assets` directory (`C:\dev\gsoap-test\InterplayWSDemo\assets`).
4. Run gSOAP's WSDL parser and provide the name of the header file that is to be generated along with the URL to the WSDL file (see *Locating the WSDL and XSD File* above):

```
wsdl2h -o Assets.h http://localhost/services/Assets?wsdl
```



```
C:\dev\gsoap-test\InterplayWSDemo\assets>wsdl2h -o Assets.h http://localhost/services/Assets?wsdl  
** The gSOAP WSDL parser for C and C++ 1.2.91  
** Copyright (C) 2000-2007 Robert van Engelen, Genivia Inc.  
** All Rights Reserved. This product is provided "as is", without any warranty.  
** The gSOAP WSDL parser is released under one of the following two licenses:  
** GPL or the commercial license by Genivia Inc. Use option -l for more info.  
  
Saving Assets.h  
Reading type map file 'typemap.dat'  
  
Connecting to 'http://localhost/services/Assets?wsdl' to retrieve WSDL/XSD... connected, receiving..  
Warning: element 'wsp:Policy' at level 3 was not recognized and will be ignored  
Connecting to 'http://localhost/services/Assets?xsd=assets.xsd' to retrieve schema... connected, receiving..  
  
To complete the process, compile with:  
soapcpp2 Assets.h  
  
C:\dev\gsoap-test\InterplayWSDemo\assets>
```

5. Run gSOAP's compiler and provide the header file that was just generated:

```
soapcpp2 -C -IC:\gSOAP\import Assets.h
```

**Note:** The `-I` argument is used to import `stlvector.h`, which is used for STL support within gSOAP. The `-C` (uppercase) argument is used to generate client code only. This sample is generating C++ code with STL support. In order to generate code in either pure C or C++ without STL support, see *Alternate Code Generation with gSOAP*.

```
C:\dev\gsoap-test\InterplayWSDemo\assets>soapcpp2 -C -IC:\gSOAP\import Assets.h
** The gSOAP Stub and Skeleton Compiler for C and C++ 2.7.91
** Copyright (C) 2000-2007, Robert van Engelen, Genivia Inc.
** All Rights Reserved. This product is provided "as is", without any warranty.
** The gSOAP compiler is released under one of the following three licenses:
** GPL, the gSOAP public license, or the commercial license by Genivia Inc.

Saving soapStub.h
Saving soapH.h
Saving soapC.cpp
Saving soapClient.cpp
Saving soapClientLib.cpp
Using assets service name: AssetsPortBinding
Using assets service style: document
Using assets service encoding: literal
Using assets service location: http://localhost/services/Assets
Using assets schema namespace: http://avid.com/interplay/ws/assets
Saving soapAssetsPortBindingProxy.h client proxy
Saving AssetsPortBinding.CheckIn.req.xml sample SOAP/XML request
Saving AssetsPortBinding.CheckIn.res.xml sample SOAP/XML response
Saving AssetsPortBinding.CheckInAAF.req.xml sample SOAP/XML request
Saving AssetsPortBinding.CheckInAAF.res.xml sample SOAP/XML response
Saving AssetsPortBinding.CreateFolder.req.xml sample SOAP/XML request
Saving AssetsPortBinding.CreateFolder.res.xml sample SOAP/XML response
Saving AssetsPortBinding.Duplicate.req.xml sample SOAP/XML request
Saving AssetsPortBinding.Duplicate.res.xml sample SOAP/XML response
Saving AssetsPortBinding.GetAttributes.req.xml sample SOAP/XML request
Saving AssetsPortBinding.GetAttributes.res.xml sample SOAP/XML response
Saving AssetsPortBinding.GetChildren.req.xml sample SOAP/XML request
Saving AssetsPortBinding.GetChildren.res.xml sample SOAP/XML response
Saving AssetsPortBinding.GetLatest.req.xml sample SOAP/XML request
Saving AssetsPortBinding.GetLatest.res.xml sample SOAP/XML response
Saving AssetsPortBinding.LinkToMOB.req.xml sample SOAP/XML request
Saving AssetsPortBinding.LinkToMOB.res.xml sample SOAP/XML response
Saving AssetsPortBinding.Move.req.xml sample SOAP/XML request
Saving AssetsPortBinding.Move.res.xml sample SOAP/XML response
Saving AssetsPortBinding.Search.req.xml sample SOAP/XML request
Saving AssetsPortBinding.Search.res.xml sample SOAP/XML response
Saving AssetsPortBinding.SetAttributes.req.xml sample SOAP/XML request
Saving AssetsPortBinding.SetAttributes.res.xml sample SOAP/XML response
Saving AssetsPortBinding.nsmap namespace mapping table

Compilation successful

C:\dev\gsoap-test\InterplayWSDemo\assets>_
```

6. Copy the `stdsoap2.h` and `stdsoap2.cpp` files from `C:\gSOAP` into the `assets` directory (`C:\dev\gsoap-test\InterplayWSDemo\assets`).

## Alternate Code Generation with gSOAP

gSOAP has the ability to generate code in C, C++ without STL, and C++ with STL. We've already shown you how to perform the latter, so we'll demonstrate how to generate code for the other two scenarios. If you just want to write code in C++ with STL support, skip to the next section.

### C:

```
wsd12h -c -o Assets.h http://localhost/services/Assets?wsdl  
soapcpp2 -c -C Assets.h
```

### C++ without STL:

```
wsd12h -s -o Assets.h http://localhost/services/Assets?wsdl  
soapcpp2 -C Assets.h
```

## Write Code to Use the Interplay WS Assets Service

This sample code is written in C++ with STL, but is still a good reference if you have chosen to use C or C++ without STL.

1. Include the proxy class header file and nsmmap file generated by gSOAP.

```
#include "assets/soapAssetsPortBindingProxy.h"  
#include "assets/AssetsPortBinding.nsmmap"
```

2. Create a couple of useful type definitions used for iterating through the results of the *GetChildren* response.

```
typedef std::vector<types__AssetDescriptionType*>::iterator AssetDescriptionTypeItr;  
typedef std::vector<types__AttributeType*>::iterator AttributeTypeItr;
```

3. Get an implementation of the *Assets* port. For this example, we are writing this and all other code in the *main* function of our program.

```
AssetsPortBinding port;
```

4. Set the *UserCredentials* to pass in the operation.

```
types__UserCredentialsType creds;  
creds.Username = "jsmith";  
creds.Password = "secret";
```

5. Allocate space for the *Header* and add the *UserCredentials* to it.

```
port.soap->header = (struct SOAP_ENV__Header*)soap_malloc(port.soap,  
    sizeof(struct SOAP_ENV__Header));  
port.soap->header->types__UserCredentials = &creds;
```

6. Create and populate the parameters for the *GetChildren* operation. See the Interplay WS Reference Guide for more information on Interplay URIs.

```
types__GetChildrenType body;  
body.InterplayURI = "interplay://WGA/Projects/Rainforest";  
body.setIncludeFolders = true;  
body.setIncludeFiles = true;  
body.setIncludeMOBs = true;
```

7. Execute the *GetChildren* operation via the *Assets* port. This returns an error code, which should be processed in your own application.



```
types__GetChildrenResponseType response;  
port.__assets__GetChildren(&body, &response);
```

8. Handle the results. For this example, we'll just output the Interplay URI and the returned attributes. *See the Interplay WS Reference Guide for more info on attributes.*

```
std::vector<types__AssetDescriptionType*> children = response.Results->AssetDescription;  
for ( AssetDescriptionTypeItr child = children.begin(); child != children.end(); child++ )  
{  
    std::cout << (*child)->InterplayURI << std::endl;  
  
    std::vector<types__AttributeType*> attributes = (*child)->Attributes->Attribute;  
  
    for ( AttributeTypeItr attr = attributes.begin(); attr != attributes.end(); attr++)  
    {  
        std::cout << (*attr)->Group << "." << (*attr)->Name  
            << " = " << (*attr)->__item << std::endl;  
    }  
  
    std::cout << std::endl;  
}
```

9. Compile and run the program. You should see the results in the console window.

## Advanced Topics

### Changing the Endpoint URL

In some cases, you may want to change the endpoint that your client code uses. By default, the code will use whichever endpoint URL was specified in the WSDL to create the service. To make your client communicate with an Interplay WS service on a different host, you need to change the endpoint URL.

Changing the endpoint URL in gSOAP is a simple task.

#### C++:

```
AssetsPortBinding port;  
port.endpoint = "http://new-endpoint-url/";
```

#### C:

```
/* In C you pass the new endpoint URL directly to SOAP function calls */  
char* endpoint = "http://new-endpoint-url/";  
soap_call__assets__GetChildrenResponse(soap, endpoint, body, response);
```

Another common reason to change the endpoint is for debugging purposes. You can use a tool such as [TCPMon](#) to create a relay to the Interplay WS host. You would then configure your client code to send requests to TCPMon on your local machine. TCPMon could then intercept the sent and received messages so you can view their contents. If TCPMon is listening on port 81 on your local machine, your code should look like this:

#### C++:

```
AssetsPortBinding port;  
port.endpoint = "http://localhost:81/services/Assets";
```

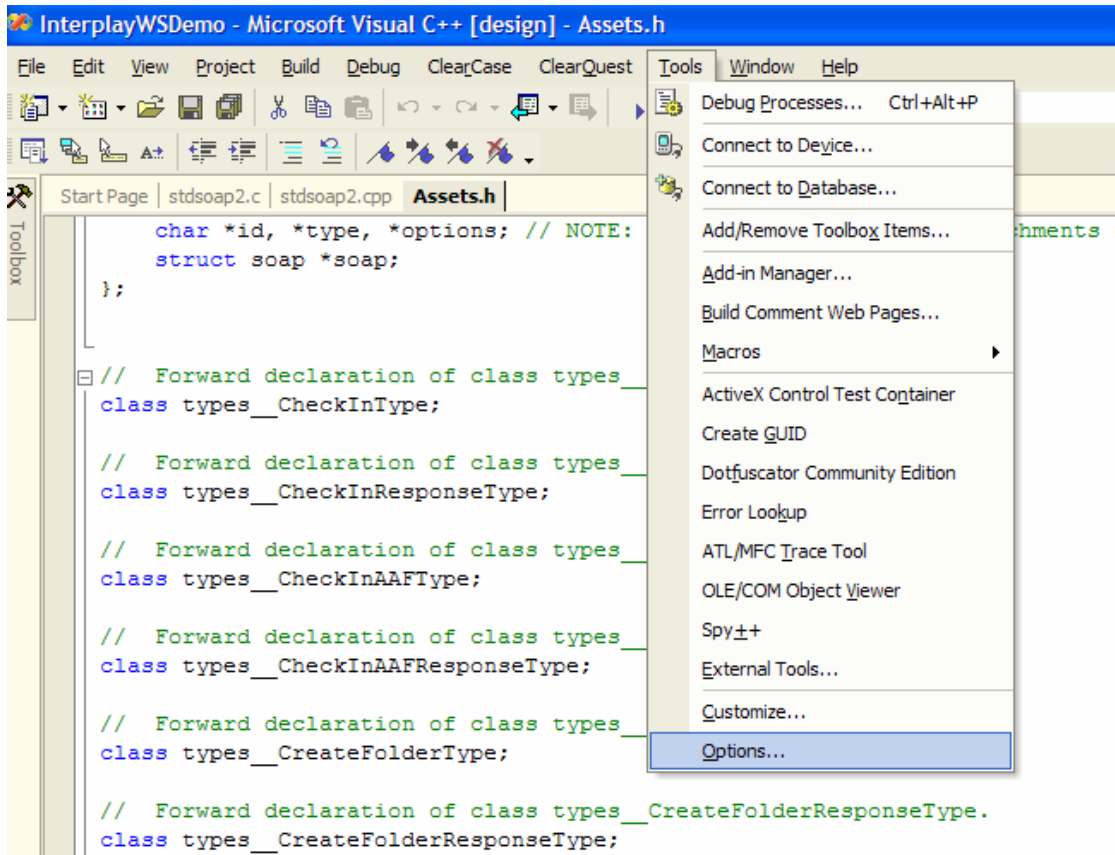
#### C:

```
char* endpoint = "http://localhost:81/services/Assets";  
soap_call__assets__GetChildrenResponse(soap, endpoint, body, response);
```

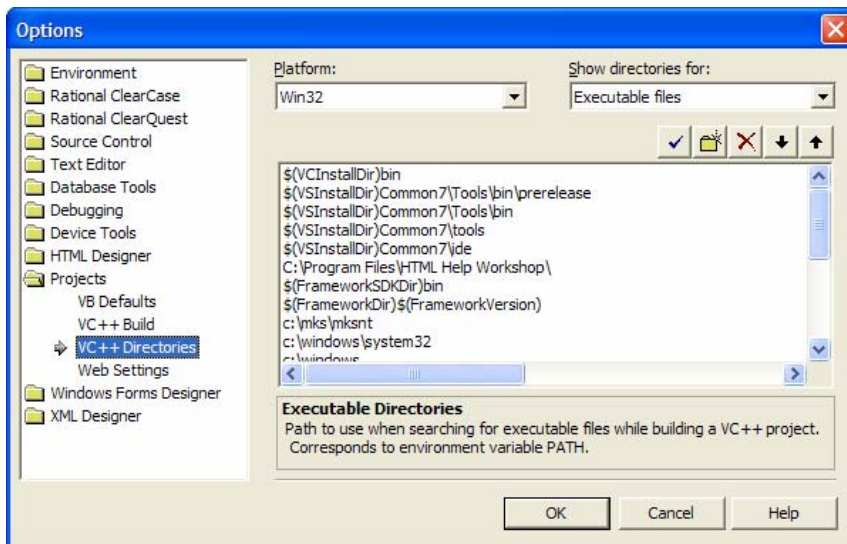
## Enabling Secure Transmission over HTTPS

The Interplay WS service also allows for encrypted transmissions via HTTPS. In order to make use of this feature you must first download and install the [OpenSSL Toolkit](#). Once you have done so you can start transmitting over HTTPS with gSOAP by following the steps below.

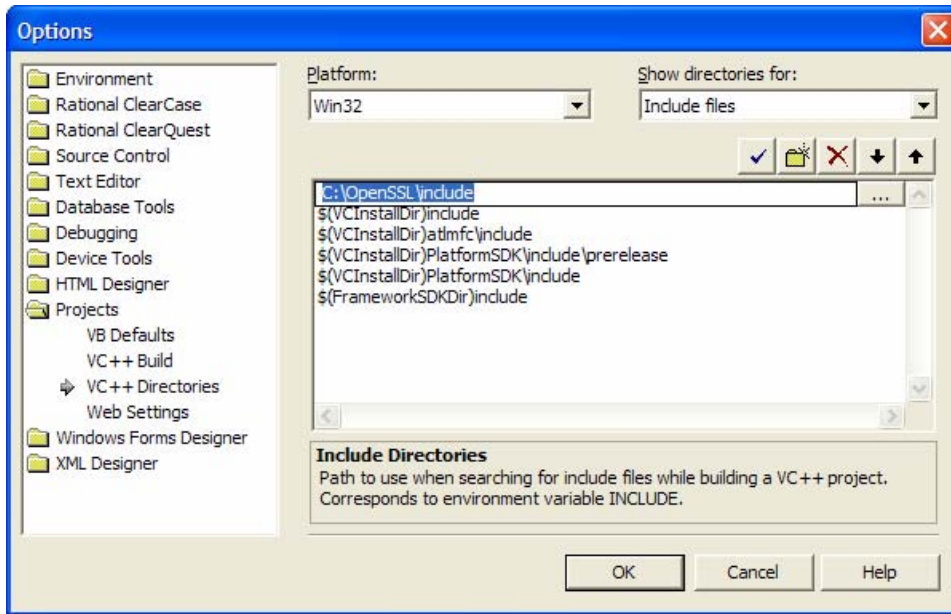
1. Add the OpenSSL include and lib directories to your project. Choose *Tools -> Options*.



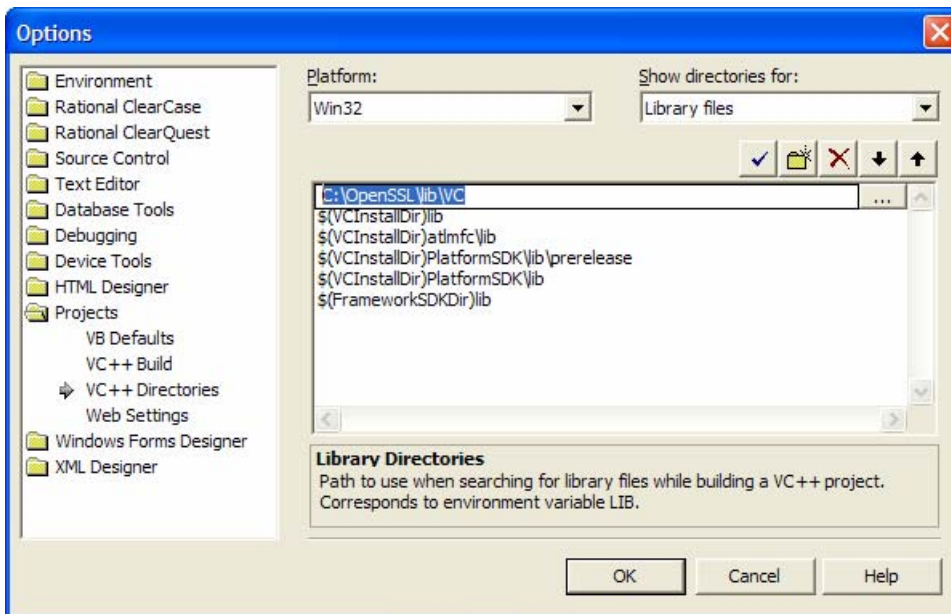
2. Select *Projects -> VC++ Directories* from the list on the left.



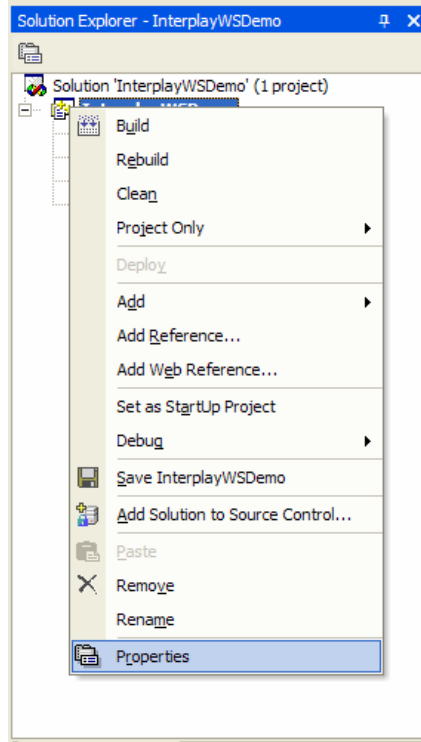
3. Select *Include files* from the list of directories on the right and add the OpenSSL Toolkit's `include` directory.



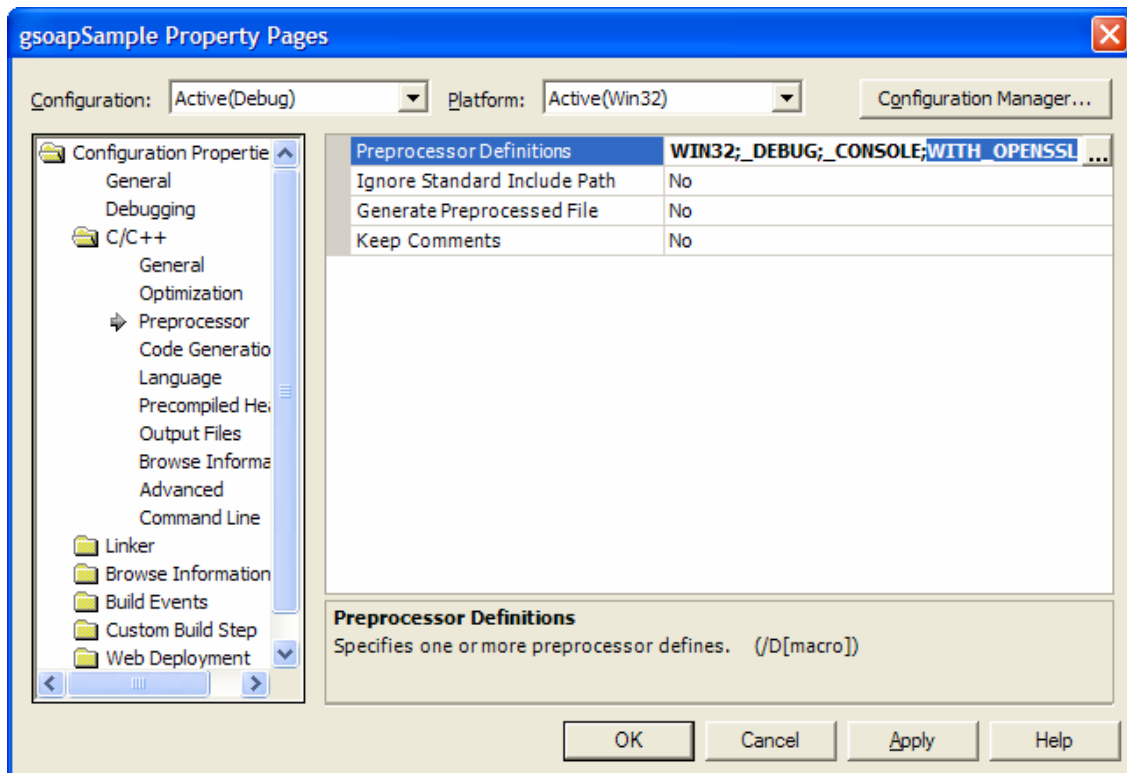
4. Select *Library files* from the list of directories on the right and add the OpenSSL Toolkit's `lib\VC` directory.



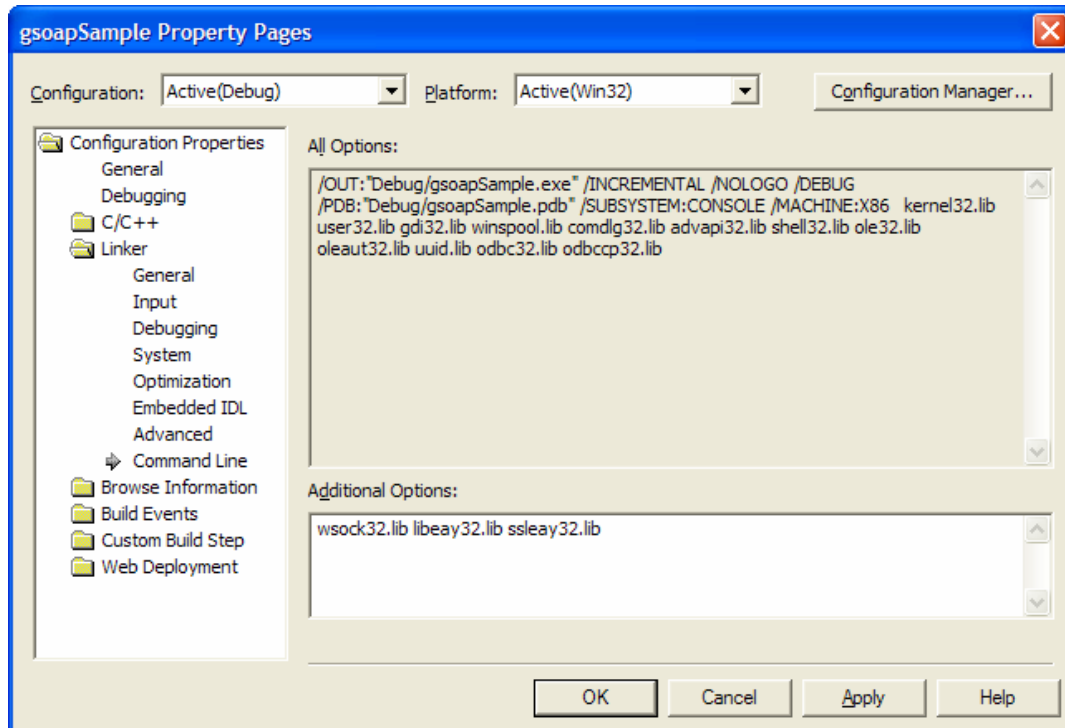
5. Right-click the project in the Solution Explorer and select *Properties*.



6. Add the preprocessor definition WITH\_OPENSSL to *Configuration Properties* -> *C/C++* -> *Preprocessor*.



7. Add `libeay32.lib` and `ssleay32.lib` to *Configuration Properties* -> *Linker* -> *Command Line*.



8. The following piece of code shows how to setup support for SSL in gSOAP and change the port so that it uses HTTPS instead of HTTP. The Interplay WS uses a *self-signed certificate*.

```
if ( soap_ssl_client_context(port.soap,
                             SOAP_SSL_NO_AUTHENTICATION,
                             NULL, // keyfile
                             NULL, // password to read keyfile
                             NULL, // cacert file to store trusted certificates
                             NULL, // capath to directory with trusted certificates
                             NULL, // file containing random data for seed
                             ) )
{
    soap_print_fault(port.soap, stderr);
    return 1;
}

port.endpoint = "https://iws-srv/services/Assets";
```

## **Enabling MTOM Support**

MTOM is the recommended best practice for efficiently transporting binary files via SOAP. For samples using MTOM refer to the “mtom” and “mtom-streaming” samples that ship with gSOAP. These can be found in “samples” folder in the root directory of your gSOAP installation.