OpenADR 2.0b Open Source Virtual End Node – C++ Library User's Manual

1026753

Open Source BSD 3-Clause License

ELECTRIC POWER RESEARCH INSTITUTE

OpenADR 2.0b Open Source Virtual End Node – C++ Library User's Manual

1026753

Technical Update, October 2014

OpenADR 2.0b Open Source Virtual End Node – C++ Library User's Manual. EPRI, Palo Alto, CA: 2014. 1026753.

DISCLAIMER OF WARRANTIES AND LIMITATION OF LIABILITIES

Software:	Software Title (Software Acronym) Version #
Developed for:	Electric Power Research Institute (EPRI) 3420 Hillview Ave. Palo Alto, CA 94304
Support*	This software is provided by EPRI "AS IS" and without customer support beyond such embodiments within the distribution of this software that may or may not provide such support.
Copyright	Copyright © 2014 Electric Power Research Institute, Inc. All Rights Reserved. Permission to use, copy, modify, and distribute this software for any purpose with or without fee is hereby granted, provided that the above copyright notice and this permission notice appear in all copies.
Developed by:	Nebland Software, LLC 859 Richborough Rd Green Bay, WI
Ordering Information:	The embodiments of this Program and supporting materials may be ordered from Electric Power Software Center (EPSC) 9625 Research Drive Charlotte, NC 28262 Phone 1-800-313-3774 Email askepri@epri.com
Disclaimer:	**THIS SOFTWARE IS PROVIDED BY EPRI AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL <COPYRIGHT HOLDER> BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.**

ELECTRIC POWER RESEARCH INSTITUTE

PRODUCT DESCRIPTION

This application was designed to support the role of a virtual end node (VEN) as defined in the OpenADR Alliance's OpenADR 2.0 Profile B Specification, updated July 1, 2013. OpenADR is a machine-to-machine interface that defines the information model, transport and security mechanisms, and the manner in which data is exchanged between two end points. OpenADR 2.0 is an open specification that defines how information is communicated between an electricity service provider and customers, but it does not purport to define how either end point uses the information. This VEN library is one example of how the specification can be applied. This open source library, written in C++, was developed as a library designed to provide users with a toolkit for the OpenADR 2.0b interface, its information model, XML payloads, interactions with a VTN, and many other uses.

The intent of developing the source code and making it available to the open source community is to provide the electric power industry with a research tool to demonstrate and test the OpenADR 2.0 communication specification in different use cases to identify and correct potential gaps in the OpenADR 2.0 specification that can only be identified through a wide range of implementations.

ABSTRACT

The open source OpenADR Virtual End Node library was developed to provide stakeholders with source code and a library that the industry could use to advance automated demand response research. The VEN library was developed to conform with OpenADR 2.0, profile B and supports the HTTP Pull data model. The VEN library, developed using C++. This user manual describes how to install and use EPRI's open source OpenADR VEN library.

Keywords

Demand Response
Automated Demand Response
OpenADR 2.0b
Profile B
Pull
Poll

Virtual Top Node (VTN)
Virtual End Node (VEN)

CONTENTS

LIST OF FIGURES

(Optional)

LIST OF TABLES

(Optional)

1 BACKGROUND

This project is an open source C++ implementation of the OpenADR 2.0b pull specification for clients. The goal of the project is to create a simple library that can be used to create a compliant VEN. The library by itself is not compliant and cannot be compliance tested.

2 INSTALL AND CONFIGURE OADR LIBRARY SOURCE

The OpenADR ("OADR") libraries have dependencies that must be configured before building can be completed. This section focuses on correctly addressing those dependencies and the overall structure of the OADR libraries on disk.

OADR Prerequisites

This project relies on the following open source projects:

- **libcurl:** HTTP communication (version 7.34.0 or later);
- **OpenSSL:** used by libcurl for HTTP/s communication (version 1.0.1d or later);
- **Xerces C++:** C++ XML parser; and
- **Google Test ("gtest")**: Google C++ unit test library.

The build system assumes that these libraries (excluding gtest) are installed. The library and development headers (sometimes referred to as a dev package) must be installed. Please refer to your operating system documentation for instructions on installing these prerequisites. The installation of these libraries under Ubuntu 14.04 and gtest is described below.

In addition to the above dependencies, the following dependencies ship with the library:

- **CodeSynthesis XSD Headers and Library:** definitions needed for XSD-generated code (used to serialize/deserialize XML objects); and
- **CodeSynthesis-Generated Source:** C++ representation of OADR XSD objects.

Installing the dependencies on Ubuntu 14.04

On a fresh install of Ubuntu 14.04, the following command will install the dependencies:

```
sudo apt-get install g++ scons libxerces-c-dev libcurl4-openssl-dev
```

Installing Google Test

Google Test is an automated testing framework developed by Google. It is used in the OADR libraries to build the final testing executable. Information about gtest and how to expand its use to cover additional test cases can be found at https://code.google.com/p/googletest/; explanations of its use in the OADR libraries is included in the developer section of this manual.

1. Download Google Test from here: https://code.google.com/p/googletest/downloads/detail?name=gtest-1.7.0.zip&can=2&q
2. Unzip the file **gtest-1.7.0.zip** and **cd** to the top level directory. Choose a location accessible to the user that will be logged in during builds.
3. Make the directory to hold the library by typing the following command:

   ```
   mkdir lib
   ```
4. Change to that directory by typing:

   ```
   cd lib
   ```
5. Build the gtest library by executing the following command:

   ```
   g++ -isystem ../include -I ../ -pthread -c ../src/gtest-all.cc
   ```
6. Call archive to create a library file that can be accessed by your oadr build tools by executing:

   ```
   ar -rv libgtest.a gtest-all.o
   ```

The final step will create a file called **libtest.a** in the current directory. This library is used by the oadrtest program. To compile oadrtest, the build system uses an environment variable to locate the gtest library. The following section describes how to create the environment variable.

Adding Google Test to the Environment for Build Tools to Access

The only external dependency that the OADR library build tools require is a path to your installation of gtest. During the build process, the build tool will use the environment variable GTEST_PATH to find the gtest library and header files. To add the variable to the current terminal, run the following command:

```
export GTEST_PATH=/path/to/gtest/install/dir/gtest-1.7.0
```

To make this change permanent, developers should place the gtest environment variable creation in their ***.bashrc*** file. Consult your operating system documentation for more information.

If you are unable to build the test program, double-check that the gtest variable is correctly set and the gtest library is properly built.

Download the Source

Stable releases of this project are available on SourceForge. EPRI project members may also receive access to the current source before it is made available to the general public.

OADR Library Directory Structure Information

The library consists of the following directories:

- oadr
- oadrsd
- oadrtest
- oadrsample
- schema
- xsd-4.0.0

Each of these directories is covered in turn below.

oadr

oadr implements an OADR 2.0b VEN library intended to be used to create a compliant VEN. The library by itself is not compliant but can be used to create a compliant VEN.

oadrsd

oadrsd is a library used to serialize and deserialize OADR XML messages. This library consists solely of the source generated by the CodeSynthesis XSD compiler. You are free to use the OADR library under the BSD 3-Clause license, but you may not separate oadrsd by itself in a different project under the BSD license. If oadrsd is separated from this project, the GPL license V2 takes precedence. EPRI purchased a license from Code Synthesis that allows the distribution of the CodeSynthesis runtime and generated source alongside the EPRI-created project, and allows the combined project to be distributed under the BSD 3-Clause license.

oadrtest

A set of unit tests utilizing the Google gtest library. The unit tests are not comprehensive.

oadrsample

A sample program demonstrating how the library could be used to implement a VEN.

schema

The OpenADR Alliance schema files for OADR 2.0a and 2.0b.

xsd-4.0.0

The CodeSynthesis runtime and include files, version 4.0.0.

3 SCONS

SCons is used to build the libraries, test, and sample programs. The following sections discuss SCons installation, building the programs, and using a toolchain other than the default.

Installing SCONS

SCons (*S*oftware *Cons*truction tool) is a build system built on top of the Python programming language. Used in place of Make, CMake, nmake, etc., its purpose is to add programmability and readability to highly complex builds, while offering advanced automation for continuous integration. The SCons website at http://www.scons.org holds extensive information about the project.

The instructions below are a general guide to installing SCons on Linux. Please refer to the instructions for your system for installing SCONS.

Building the libraries with SCONS

All SCons commands must be run from the oadrlib directory.

To create a debug build of the libraries and programs, run

```
scons
```

This will create build object files, libraries, and executables in a Debug directory under each of the respective projects.

To create a release build, run:

```
scons mode=release
```

To clean the projects, run:

```
scons --clean
```

Issues you might encounter:

1. The OADRSD library is large: building can take a significant amount of time and disk space.
2. Linking the libraries into oadrtest is a memory-intensive operation: the machine doing the build should have more than 2 GB of memory.
3. SCons places object files in the Debug subdirectories of each project, *and* in the Release subdirectories, depending upon build mode. Should disk space be at a premium, make certain that the files in one of these folders is deleted.

Changing the Toolchain

By default, SCons will use the g++ compiler that's in your path. There are two options for specifying an alternative compiler:

1. Add the following to the Environment object in all SCcons build files, where **env** is an environment object:

   ```
   env.Replace(CC = “path/to/gcc”)
   env.Replace(CXX= “path/to/g++”)
   ```

2. When running from the command line, precede the **scons** command with:

   ```
   CC=/path/to/gcc CXX=/path/to/g++/.
   ```

   For example:

   ```
   CC=/path/to/gcc CXX=/path/to/g++ scons
   ```

4 RUNNING THE TESTS

If everything builds without error, the library can be tested by running the oadrtest program. This program links against the oadr and oadrsd libraries (found in oadr/debug and oadrsd/debug respectively), which won't be in your library path by default. Before you can run the tests, you must add the two directories to the library search path.

To add the files to your path, run:

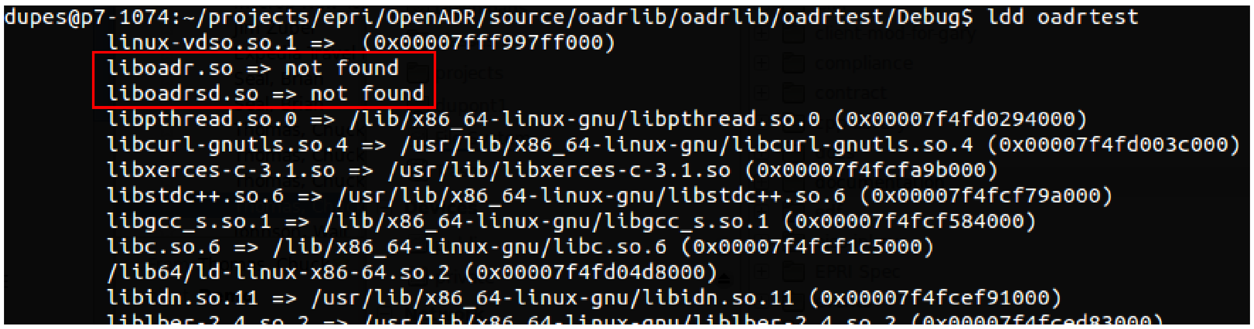
export LD_LIBRARY_PATH=/home/user/path/to/oadrlib/oadrsd/debug:/home/user/path/to/oadrlib/oadr/debug

Be sure to use the actual paths to your oadr and oadrsd directories.

To test if the library paths were correctly added, from the directory **oadrtest/debug**, run:

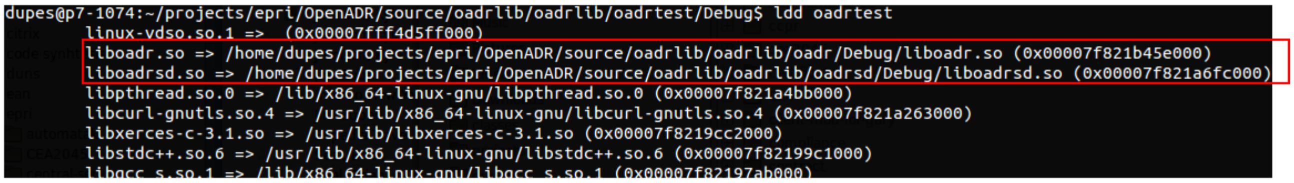
ldd oadrtest.out

If the library path isn't set correctly, running this command will show ***file not found*** next to the oadr and oadrsd entries.



```
dupes@p7-1074:~/projects/epri/OpenADR/source/oadrlib/oadrlib/oadrtest/Debug$ ldd oadrtest
        linux-vdso.so.1 =>  (0x00007fff997ff000)
        liboadr.so => not found
        liboadrsd.so => not found
        libpthread.so.0 => /lib/x86_64-linux-gnu/libpthread.so.0 (0x00007f4fd0294000)
        libcurl-gnutls.so.4 => /usr/lib/x86_64-linux-gnu/libcurl-gnutls.so.4 (0x00007f4fd003c000)
        libxerces-c-3.1.so => /usr/lib/libxerces-c-3.1.so (0x00007f4fcfa9b000)
        libstdc++.so.6 => /usr/lib/x86_64-linux-gnu/libstdc++.so.6 (0x00007f4fcf79a000)
        libgcc_s.so.1 => /lib/x86_64-linux-gnu/libgcc_s.so.1 (0x00007f4fcf584000)
        libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007f4fcf1c5000)
        /lib64/ld-linux-x86-64.so.2 (0x00007f4fd04d8000)
        libidn.so.11 => /usr/lib/x86_64-linux-gnu/libidn.so.11 (0x00007f4fcef91000)
        liblber-2.4.so.2 => /usr/lib/x86_64-linux-gnu/liblber-2.4.so.2 (0x00007f4fced83000)
```

If the paths are set correctly, output from **ldd** will be similar to the following:

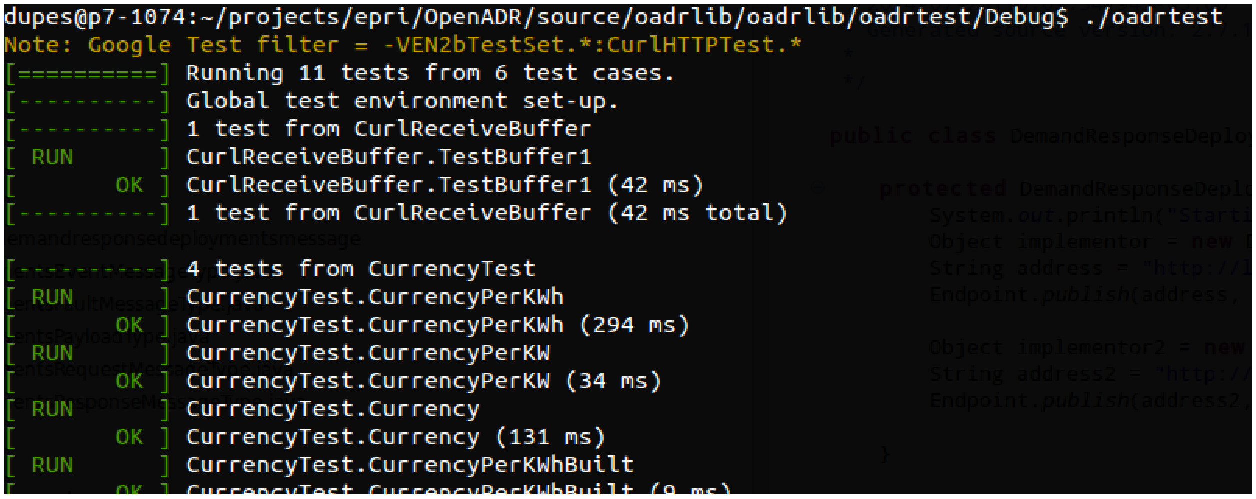


```
dupes@p7-1074:~/projects/epri/OpenADR/source/oadrlib/oadrlib/oadrtest/Debug$ ldd oadrtest
        linux-vdso.so.1 =>  (0x00007fff4d5ff000)
        liboadr.so => /home/dupes/projects/epri/OpenADR/source/oadrlib/oadrlib/oadr/Debug/liboadr.so (0x00007f821b45e000)
        liboadrsd.so => /home/dupes/projects/epri/OpenADR/source/oadrlib/oadrlib/oadrsd/Debug/liboadrsd.so (0x00007f821a6fc000)
        libpthread.so.0 => /lib/x86_64-linux-gnu/libpthread.so.0 (0x00007f821a4bb000)
        libcurl-gnutls.so.4 => /usr/lib/x86_64-linux-gnu/libcurl-gnutls.so.4 (0x00007f821a263000)
        libxerces-c-3.1.so => /usr/lib/libxerces-c-3.1.so (0x00007f8219cc2000)
        libstdc++.so.6 => /usr/lib/x86_64-linux-gnu/libstdc++.so.6 (0x00007f82199c1000)
        libgcc_s.so.1 => /lib/x86_64-linux-gnu/libgcc_s.so.1 (0x00007f82197ab000)
```

Once the path is set, you can run the unit tests from the oadrtest/debug directory with:

./oadrtest.out

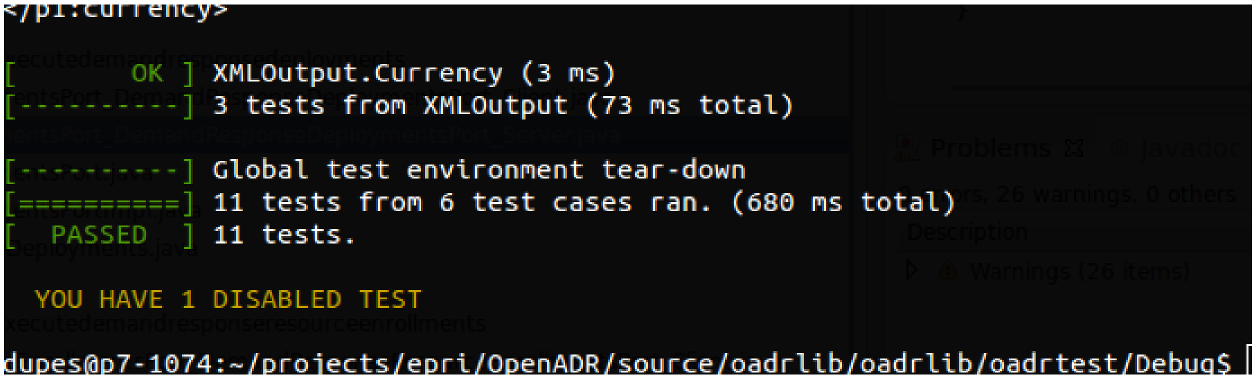
Output from the running the program should be look like the following image (note that part of the output was cut out to save space):



```
dupes@p7-1074:~/projects/epri/OpenADR/source/oadrlib/oadrlib/oadrtest/Debug$ ./oadrtest
Note: Google Test filter = -VEN2bTestSet.*:CurlHTTPTest.*
[==========] Running 11 tests from 6 test cases.
[----------] Global test environment set-up.
[----------] 1 test from CurlReceiveBuffer
[ RUN      ] CurlReceiveBuffer.TestBuffer1
[       OK ] CurlReceiveBuffer.TestBuffer1 (42 ms)
[----------] 1 test from CurlReceiveBuffer (42 ms total)

[----------] 4 tests from CurrencyTest
[ RUN      ] CurrencyTest.CurrencyPerKWh
[       OK ] CurrencyTest.CurrencyPerKWh (294 ms)
[ RUN      ] CurrencyTest.CurrencyPerKW
[       OK ] CurrencyTest.CurrencyPerKW (34 ms)
[ RUN      ] CurrencyTest.Currency
[       OK ] CurrencyTest.Currency (131 ms)
[ RUN      ] CurrencyTest.CurrencyPerKWhBuilt
[       OK ] CurrencyTest.CurrencyPerKWhBuilt (9 ms)
```

• • •



```
</p1:currency>

[       OK ] XMLOutput.Currency (3 ms)
[----------] 3 tests from XMLOutput (73 ms total)

[----------] Global test environment tear-down
[==========] 11 tests from 6 test cases ran. (680 ms total)
[  PASSED  ] 11 tests.

  YOU HAVE 1 DISABLED TEST

dupes@p7-1074:~/projects/epri/OpenADR/source/oadrlib/oadrlib/oadrtest/Debug$
```

In addition to these automated tests, unit tests exist to test ECC and RSA certificates against an SSL-configured webserver. Instructions for configuring a web server to run these tests are not included in this manual.

5 SAMPLE PROGRAM

The sample program is located in sample/debug and is called ***sample.out***. The sample program connects to a VTN, performs registration (including registering a status report), polls the VTN every 10 seconds, and opts into any event it receives. Instructions for configuring a VTN to communicate with the sample VEN are not included in this manual.

To control which VTN the VEN connects to, edit the VTN URL and VEN name settings at the top of main.cpp.

As shipped, the sample program does not use SSL. The top of VENImpl.cpp has instructions for enabling SSL. The certificates files must be in PEM format.

6 VEN API AND LIBRARY ARCHITECTURE

This section provides an overview of the library architecture and the API. For an example of how the library can be used to create a VEN, see the sample program (discussed in the previous section).

The main class that programmers will interact with is oadr/ven/VEN2b. VEN2b is constructed with a pointer to an ***IHttp*** object, a base URL, a VEN Name string, a VEN ID string, and a registration ID string:

VEN2b(**IHttp** *http, **string** baseURL, **string** venName, **string** venID = "", **string** registrationID = "");

The base URL parameter should include the entire base path, up to the service name. For example, the EPRI VTN (running on the local machine), has the following base URL:

http://localhost:8080/OpenADR2/Simple/2.0b

The VEN name string is used by the VTN to identify the VEN during registration. The VEN ID and registration ID should be populated if the VEN has previously registered with the VTN. If the VEN has not previously communicated, or if the VEN should reregister without the existing VEN ID and registration ID, these fields can be left blank.

The IHttp object handles all HTTP/s communication. The object was written as an interface to make it easier to write unit tests for the VEN2b object, and to make it easier to replace the HTTP communication with a different library. The library libcurl is currently the only supported HTTP library.

Constructing a VEN

A new VEN can be constructed like this:

```
try
{
        HttpCurl *curl = new HttpCurl();
        VEN2b *ven = new VEN2b(curl, vtnURL, venName);
}
Catch (CurlException &exception)
{
}
```

A Note on Exception Handling

The call to create a new HttpCurl and all calls to functions on the VEN2b object *must* be wrapped with try/catch. Any errors encountered with communication, such as a hostname lookup failure, communication failures, or trouble initializing a libcurl function, is reported via exception handling.

A CurlException is the object thrown for all encountered exceptions. Future versions of the library may replace the exception object with something more generic.

VEN API

The VEN2b object implements a 2.0b HTTP pull VEN. Function calls on the VEN will send a request to the server and wait synchronously for the reply. Most functions return an auto_ptr to an object that extends Oadr2bRequst. See the file **VEN2b.h** for a list of functions supported by the VEN API.

Here's an example function signature:

auto_ptr<**RequestEvent**> requestEvent(**string** requestID = "", **unsigned int** replyLimit = 0);

The name of the function and the return object represents the OpenADR message type that is sent to the server. In this example, the returned object and function name are requestEvent**.** This function sends an oadrRequestEvent message to the VTN and expects an oadrDistributeEvent message in response. See the OpenADR 2.0b documentation for a list of request/reply messages used in each of the four services.

A Note on auto_ptr

Most of the VEN functions return an auto_ptr. Auto pointers (auto_ptr) can be used like normal pointers. The main advantage of auto pointers is garbage collection: as soon as an auto pointer variable goes out of scope, the object is cleaned up.

Similarly to the previous example, the function queryRegistration sends a queryRegistration message to the server and expects a createdPartyRegistration message in response.

OADR2b Request Objects

Each oadr2b request object holds the XML of the request and response, an object representation of the request and response (stored in an oadrPayload object found in the oadrsd library), a string for the expected request and response type (i.e., requestEvent/distributeEvent for the RequestEvent object), the http response code and description (i.e., 200 OK) as returned from the HTTP server, and the OADR response code and description which are also based on HTTP codes.

To determine if a response was successful, the response code (see the responseCode function) should be examined. A successful query will return a 200 response. Anything else is considered to be an error. If the server encountered an error, the HTTP response code (httpResponseCode**)** and http response description (httpResponseDescription) fields may provide information. These fields represent the HTTP layer code and message from the server.

The request and response objects are both of type oadrPayload. The reason for this is that all OpenADR 2.0b payloads have oadrPayload as the root element. Inside oadrPayload is an oadrSignedObject which will contain one of a number of objects, including oadrDistributeEvent, oadrCreatdEvent, oadrResponse, etc (see the XML schema files for a full list of available objects). Each potential object has a **.present()** function available for testing what type of object is contained with the oadrPayload. See the **VENImpl::poll** method for an example. This function checks for the presence of each of the possible objects that can be returned from a poll query.

The objects are complex and cumbersome to navigate. Using an IDE such as Eclipse to manage the project helps immensely. Code completion makes it much easier to use the objects.

Implementing a Poll VEN

An example of the following description of a VEN can be seen in the ***sample*** project.

The first message that a VEN might send to the server is an oadrQueryRegistration message (via the VEN2b.queryRegistration function). This is an optional message, but can be used to determine if the VEN and VTN are properly configured to communicate with each other. After a successful response is received by the VEN, the VEN can proceed with registration.

Registration begins with the VEN sending an oadrCreatePartyRegistration message (via the **VEN2b.createPartyRegistration** function). If registration is successful, the VEN2b object will save the VEN ID and registration ID. These fields will be used for all future communication. A proper VEN implementation should persist these values and reload them when the VEN starts. (A discussion of whether a VEN should reregister or use existing IDs can be found below.)

After receiving an OK response to an oadrCreatePartyRegistration message, the VEN should:

1) Register reports;
2) Send a poll message and process the server’s registerReports message;
3) Request events and respond accordingly with a createdEvent message and (optionally) a createOpt message; and
4) Continually poll at the frequency specified in the oacrCreatedPartyRegistration message.

Refer the sample program for an example. In particular, look at the **VENImpl::registerVEN** function.

The sample program demonstrates every message exchange except those used for oadrUpdateReport messages. (OpenADR reporting is complex and is not included in this first version of the library.)

Another complex area is parsing, scheduling, and responding to events. The sample code shows how to parse an event and respond with an opt in message for each event (see **VENImpl::processMessage(oadrDistributeEventType& message)**).

The VEN does not demonstrate how to schedule and react to an event. A compliant VEN will schedule events and react to them according to the start time, ramp duration, and other parameters, and adjust settings (i.e., shed load) appropriately.

7 DEVELOPING WITH ECLIPSE

This section covers use of the Eclipse IDE. The OADR objects are complex and difficult to use. Programming using an IDE with code completion makes it much easier it much easier to use the library.

Downloading and installing Eclipse

Though your distribution may have an Eclipse package, it is recommended to download a new copy from the Eclipse website. Any recent version of Eclipse with the CDT plugin will suffice. Eclipse can be downloaded from here:

https://www.eclipse.org/downloads/packages/eclipse-ide-cc-developers/lunar

To unzip/untar the file, run:

```
tar –xzf /path/to/eclipse.tar.gz
```

Running this command will create an eclipse directory. From the command line change to the eclipse directory, and run Eclipse:

```
./eclipse
```

In a following section the Google Test library path will be added to the build environment in Eclipse. If Eclipse isn't executed from the command line, the Eclipse environment will not have access to the GTEST_PATH environment variable that was created in the first section.

Loading the Project into Eclipse

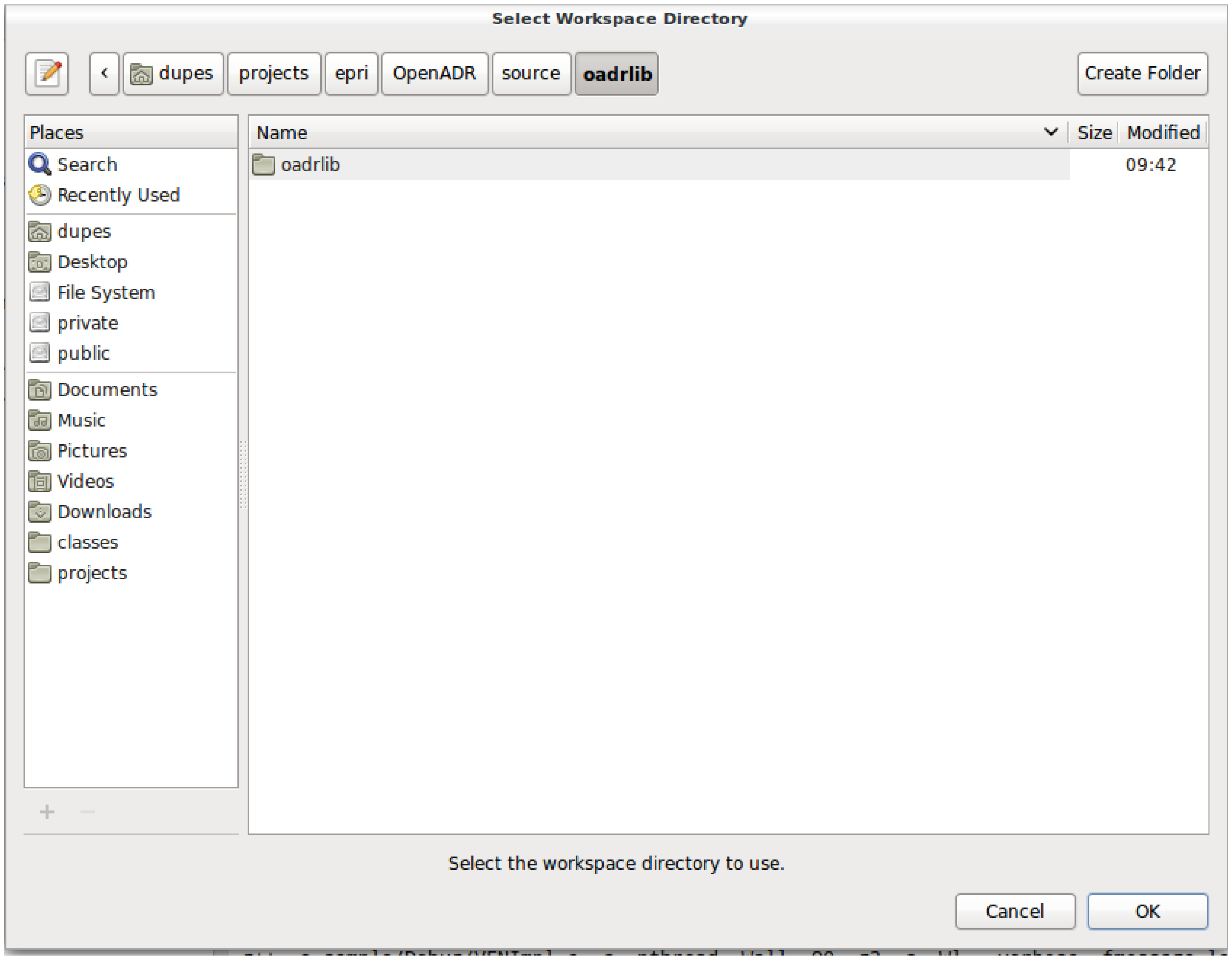
The Eclipse IDE is used to manage this project. Since Eclipse creates project files that are specific for a system, it's not possible to ship the library with Eclipse project files that are suitable for every system. The following instructions show how to load the project and configure Eclipse so that code completion and other such IDE niceties work as expected.

The IDE is used for development only: it is not used to build the libraries. Eclipse manages its own set of directories separate from the build system. At times, the Eclipse index or directories may not be in sync with the build system, causing Eclipse to indicate errors even though the source builds as expected. If the source builds but Eclipse is indicating errors, double-check the instructions below to ensure the project is configured correctly and rebuild the index. To rebuild the index, right click on the project and select index, followed by rebuild all.

Step 1: Open the Workspace

After launching, Eclipse will ask for a workspace directory. Select the parent directory of oadrlib as the workspace directory. See Figure 1.

Note that this and the following steps will cause Eclipse to create a number of files and directories in the workspace directory. You may want to create a new workspace directory and copy oadrlib into the workspace directory (as is depicted in Figure 1).



Select the workspace directory to use.

Figure 1

Opening the Workspace. The parent directory of oadrlib is the workspace directory.

Step 2: Create a New Project

Next, the source can be loaded into Eclipse by creating a new project.

1) Select New->project

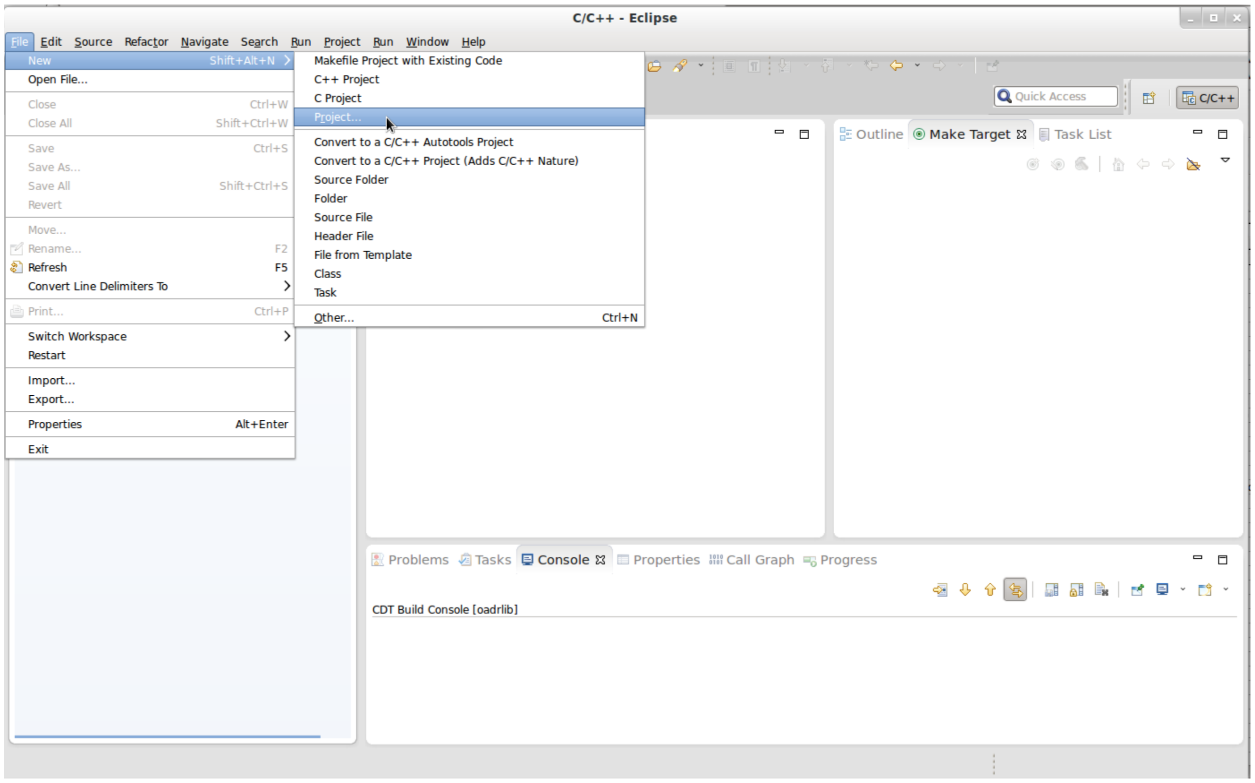


Figure 2

Creating a New Project

2) Under **C/C++**, select **C++ Project** and click **next.**

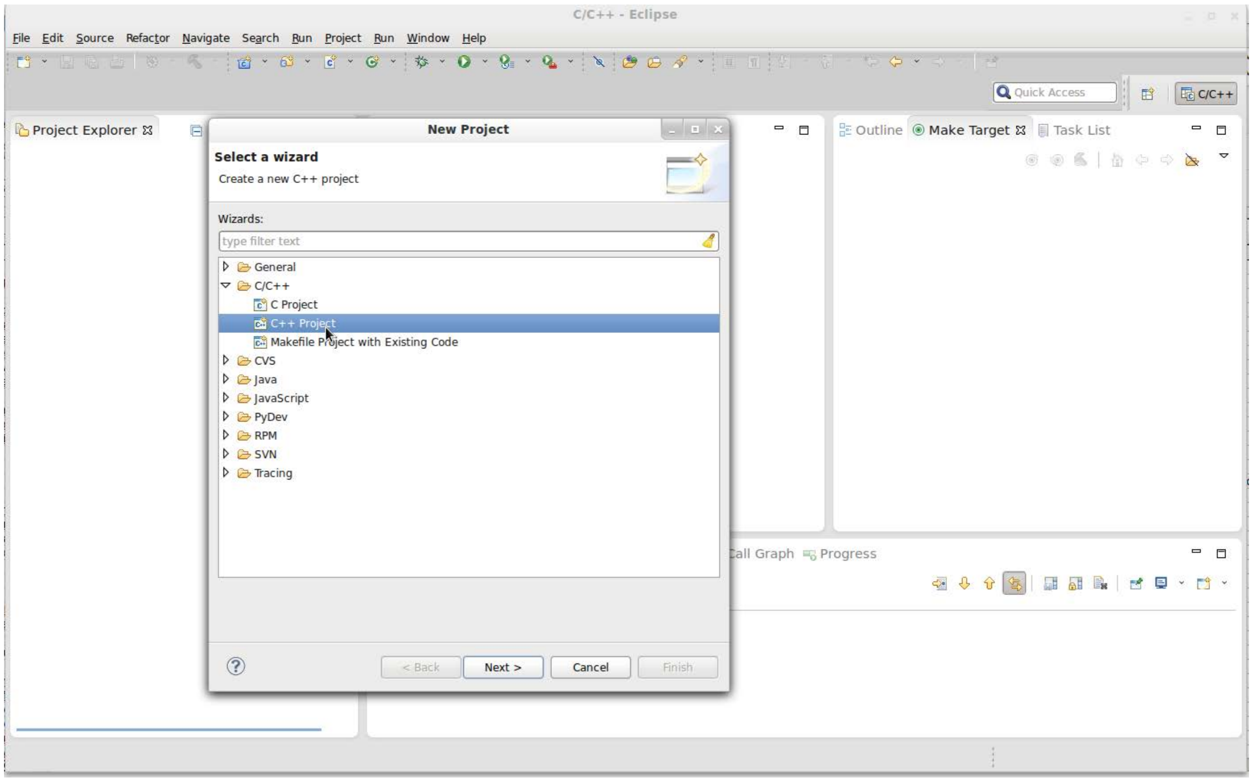
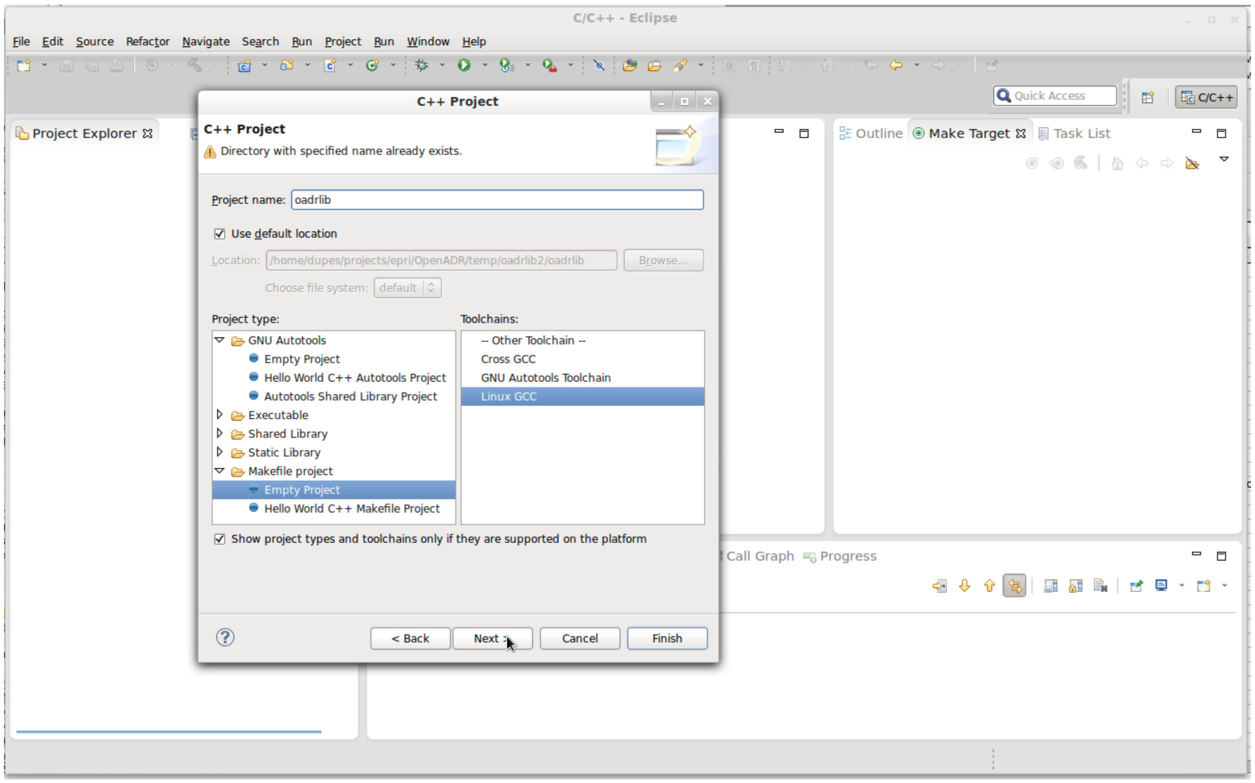


Figure 3

Selecting C++ Project

3) Enter Project Settings

Next to Project Name, enter oadrlib. Under **Makefile project**, select **Empty Project**. Under **Toolchains**, select **Linux GCC**. Eclipse will warn that a "Directory with the specified name already exists." This warning is expected. Click **Finish** and the project will load.


C++ Project

Directory with specified name already exists.

Project name: oadrlib

☑ Use default location

Location: /home/dupes/projects/epri/OpenADR/temp/oadrlib2/oadrlib

Choose file system: default

Project type:
- GNU Autotools
  - Empty Project
  - Hello World C++ Autotools Project
  - Autotools Shared Library Project
- Executable
- Shared Library
- Static Library
- Makefile project
  - Empty Project
  - Hello World C++ Makefile Project

Toolchains:
- -- Other Toolchain --
- Cross GCC
- GNU Autotools Toolchain
- Linux GCC

☑ Show project types and toolchains only if they are supported on the platform

Figure 4

Entering Project Settings

Step 3: Add gtest to the Eclipse Include Path

After loading the project, Eclipse will begin indexing the source files; this will take a few minutes. After building the index, Eclipse will indicate that it cannot find the gtest header files and show warnings and errors on any code that references gtest. Performing this step will add gtest to Eclipse, eliminating the errors and warnings.

1) Right click the **oadrlib** project and select **Properties**

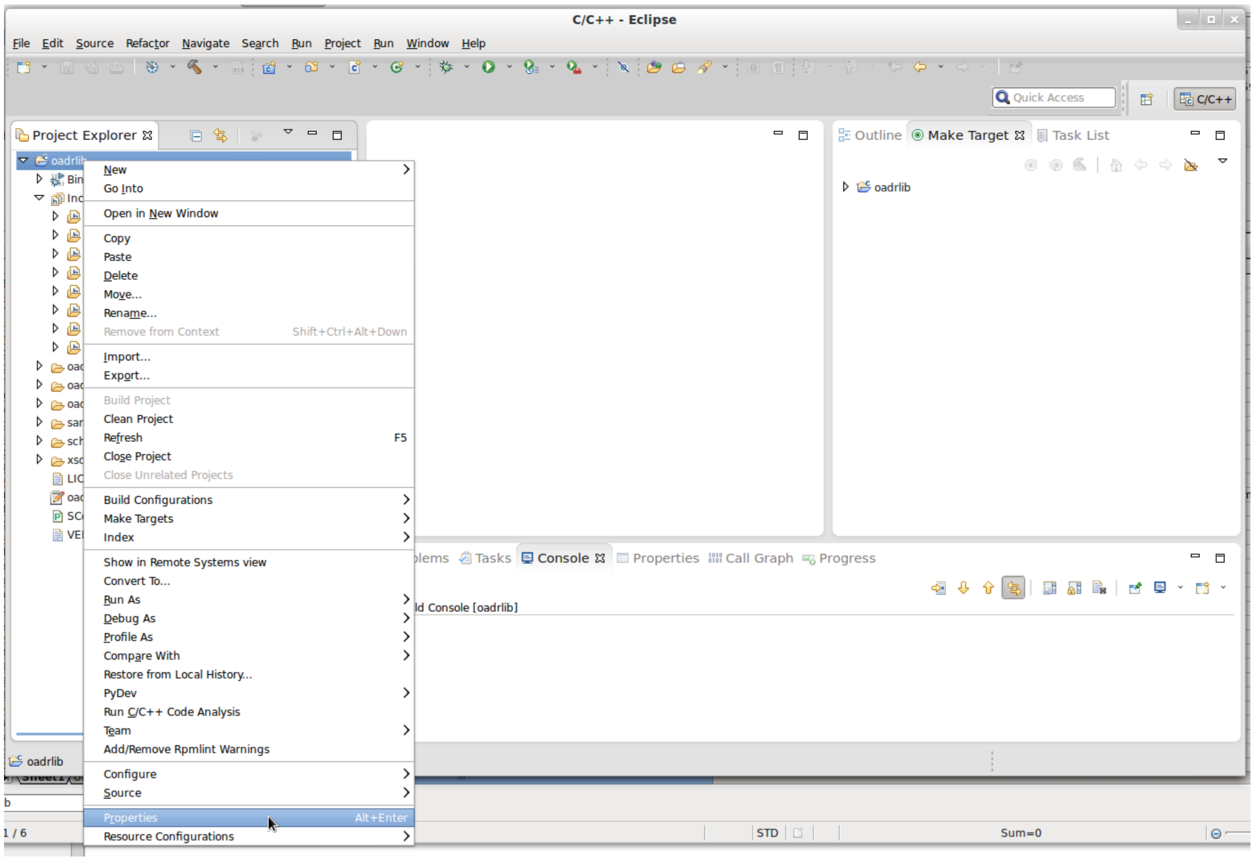


Figure 5

Right Clicking oadrlib and Selecting Properties

2) Under **C/C++ General**, select **Paths and Symbols.** On the **Includes** tab, select **GNU C++.** Click the **Add** button on the right side of the window.

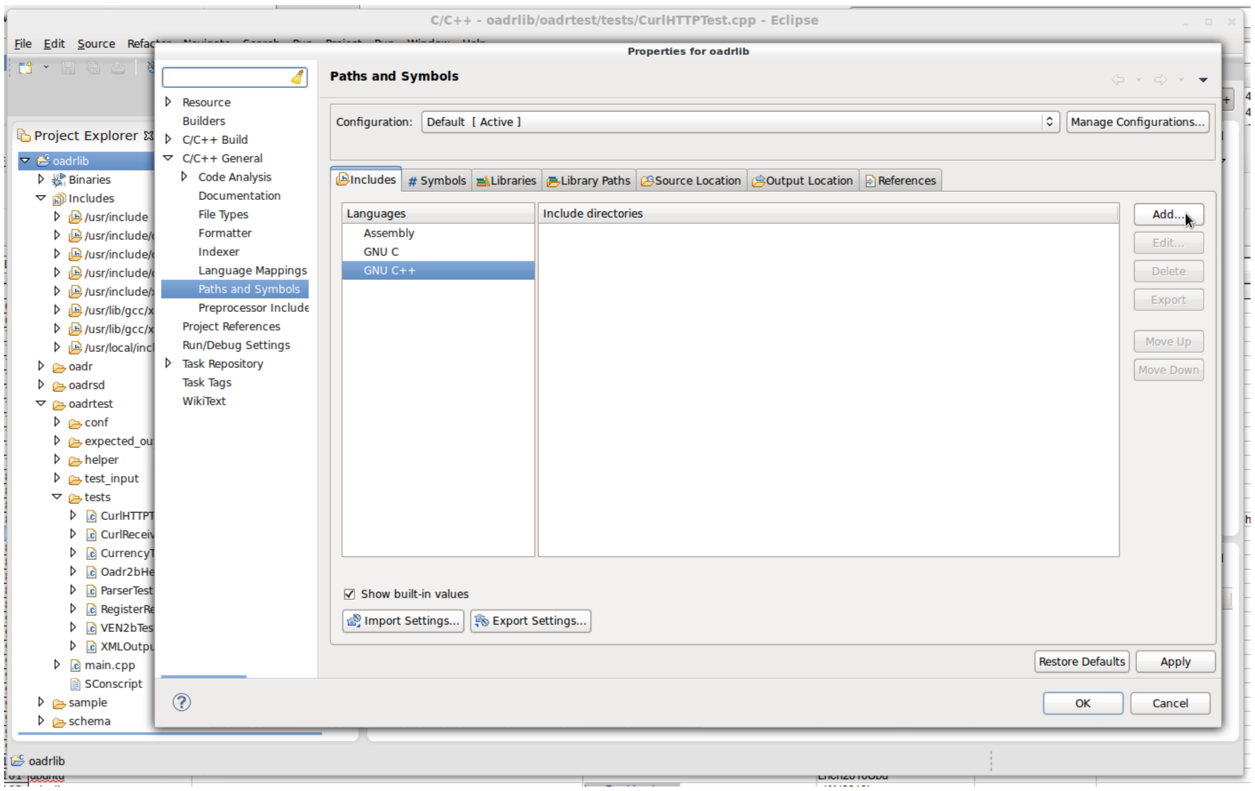
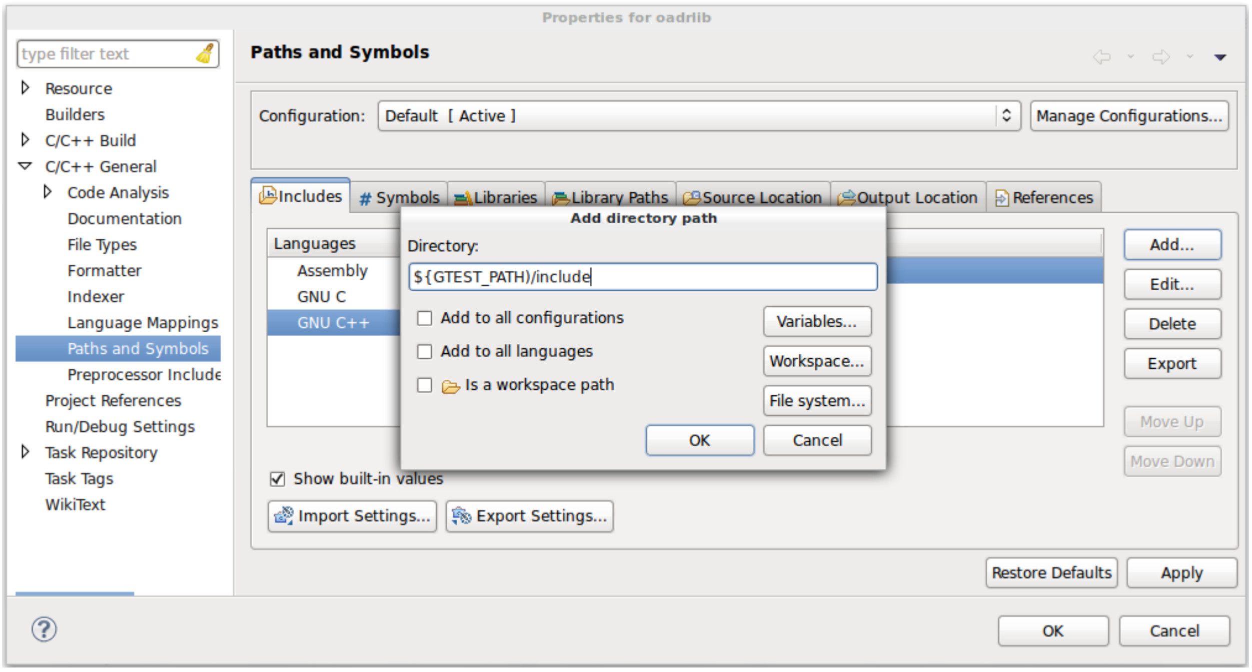


Figure 6

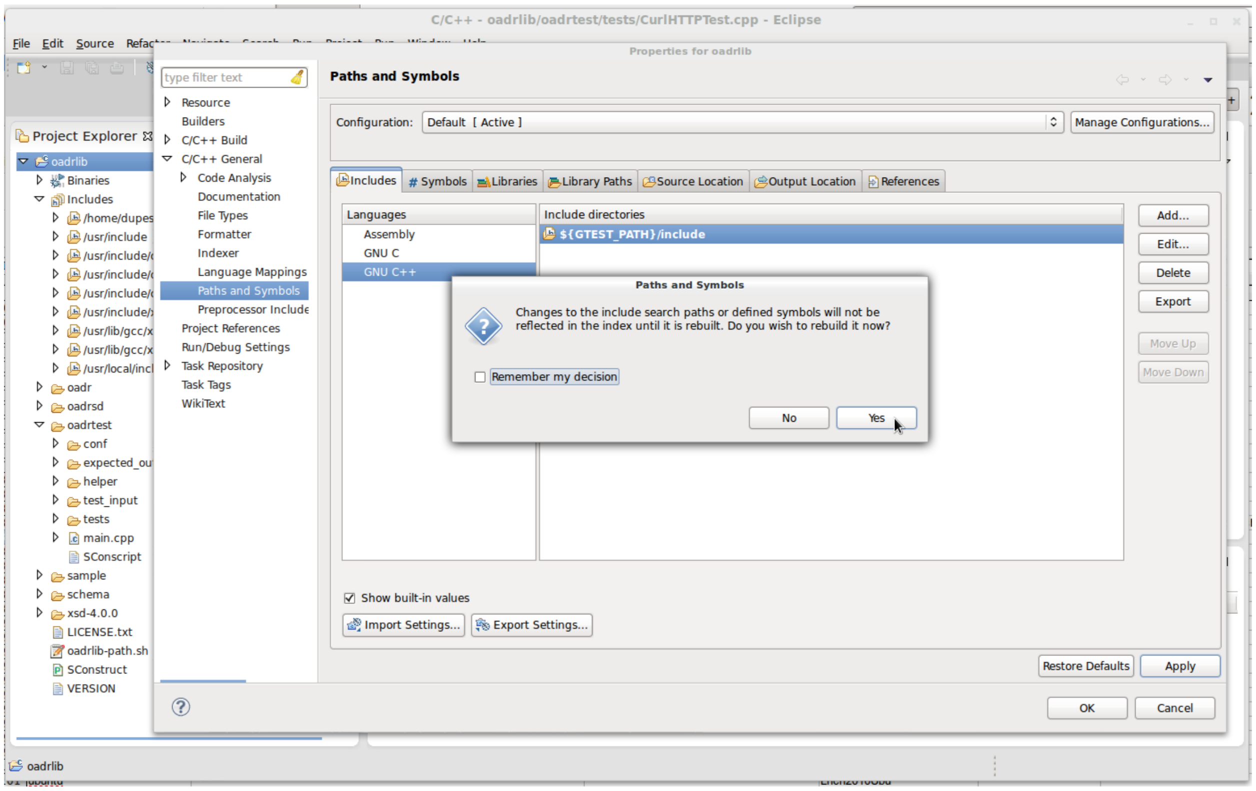
Browsing to C/C++ General->Paths and Symbols, Select GNU C++, and clicking Add.

3) In the popup window, enter **${GTEST_PATH}/include** and select **OK**.



Directory: ${GTEST_PATH}/include

4) Select **Apply** in the bottom right corner. Eclipse will prompt you to rebuild the index: select **Yes** and then select **OK** to close the dialog.



C/C++ - oadrlib/oadrtest/tests/CurlHTTPTest.cpp - Eclipse

Properties for oadrlib

Paths and Symbols

Configuration: Default [Active]

Include directories: ${GTEST_PATH}/include

Paths and Symbols

Changes to the include search paths or defined symbols will not be reflected in the index until it is rebuilt. Do you wish to rebuild it now?

☐ Remember my decision

☑ Show built-in values

Step 4: Creating Eclipse Build targets

At this point, the C++ index will begin running and will take several minutes to complete. Until indexing is finished, you may notice that Eclipse reports errors and warnings in the oadrtest program.

8 LICENSE

1026753