openPASS / World_OSI

Setup guide

Date	24.07.2019
Author	Reinhard Biegel

1 Introduction

The World_OSI provides an implementation of the WorldInterface using Open Simulation Interface (OSI) as a "backend storage" [1]. OSI already provides data structures for representing various objects in traffic simulation environments. As these objects are specified rather sensor centric, the OSI source currently needs some additions to satisfy the needs of the World_OSI module (see OSI WorldInterface Pull Request [2]).

To allow unit-testing of the implementation, an additional layer has been introduced on top of OSI, whose classes represent the objects needed in the simulator's world (WorldData). The OSI objects are instantiated there. An architectural diagram showing WorldData and its interactions is currently not available.

Another benefit of the additional WorldData layer is keeping the area of contact to the data backend small, to allow easy adaption if OSI switches from protobuf to another storage technology (open discussion).

The current use of protobuf provides easy serialization and deserialization of all stored data out-of-the-box.

2 Prerequisites

Before being able to compile and run the World_OSI module, make sure to have all dependencies installed. You can follow the guide for setting up the openPASS environment located in `Documentation/openPASS_Setup_StepByStep.pdf` which is located in the openPASS source repository. Please note to check out the intech branch, where the OSI changes will be located for now. For the OSI world demonstration an additional project file OpenPASS_OSI_UseCase has been added.

The World_OSI has been developed on Linux x86_64 using Qt 5.12.3, gcc 7.1.0, 7.3.0 and 8.1.0. On Windows systems a MinGW environment with the same compiler versions has been used. There are no known issues regarding different gcc versions.

3 Dependencies

The World_OSI module introduces an additional third-party dependency on the OSI library, and as a consequence on Google protobuffers.

3.1 CMake

For compilation of protobuf and OSI, the CMake build system is required. Please download and install a binary distribution of CMake [4]. Version 3.9.4 has been used during testing. Please feel free to try a more recent version (latest is 3.15.0).

3.2 Protobuf

If not already installed on the system, the protobuf library and headers have to be built prior to being able to compile OSI. Protobuf sources are located at [3]. This guide will give some instructions to compile version 3.6.1, but any other version should be compatible (including 2.x).

As most users seem to work on Windows environments, the following instructions will target these systems.

3.2.1 Protobuf compilation

1. Download `protobuf-cpp-3.6.1.zip` and extract the archive. We will use `C:\OpenPASS\thirdParty\sources` as target directory.

2. Open a command line with MinGW environment at the extraction directory

3. Create the build directory

```
> cd cmake
> mkdir build
> cd build
```

4. Run CMake (tests are disabled here due to some compiler warnings being treated as errors, may vary with compiler version)

```
> cmake -G "MinGW Makefiles" -DCMAKE_BUILD_TYPE=Release
  -DCMAKE_INSTALL_PREFIX=C:/OpenPASS/thirdParty
  -Dprotobuf_BUILD_SHARED_LIBS=ON
  -Dprotobuf_BUILD_TESTS=OFF ..
```

5. Compilation

```
> mingw32-make -j3
```

6. Installation

```
> mingw32-make install
```

7. Final steps
   a. Copy the MinGW runtime libraries from the MinGW installation directory to the `thirdParty/bin` directory `(libgcc_s_seh-1.dll, libwinpthread-1.dll and libstdc++6.dll).`
   b. Copy the `libprotobuf.dll` from `bin` to `lib` subfolder

3.3 Open Simulation Interface

The OSI project is hosted on Github. Please check out the source of the mentioned pull request and compile OSI using the following commands executed from the source directory.

1. Create build directory

```
> mkdir build
> cd build
```

2. Run CMake. As I could not make CMake find protobuf in the custom installation directory, we specify the paths manually here.

   ```
   > cmake -G "MinGW Makefiles" -DCMAKE_BUILD_TYPE=Release
     -DCMAKE_INSTALL_PREFIX=C:/OpenPASS/thirdParty
     -DProtobuf_INCLUDE_DIR=C:/OpenPASS/thirdParty/include
     -DProtobuf_PROTOC_EXECUTABLE=C:/OpenPASS/thirdParty/bin/protoc.exe
     -DProtobuf_LIBRARIES=C:/OpenPASS/thirdParty/lib ..
   ```

3. Adding linker flags for protobuf
   Edit the file `C:\OpenPASS\thirdParty\source\open-simulation-interface\build\CMakeFiles\open_simulation_interface.dir\linklibs.rsp`

   Add `"-LC:/OpenPASS/thirdParty/lib -lprotobuf"` to the end of the line.

   If anybody knows how to avoid this step, please drop a note.

4. Compilation

   ```
   > mingw32-make -j3
   ```

5. Installation

   ```
   > make install
   ```

6. Final steps
   a. Include path fix: rename `thirdParty/include/osi3` to `osi`
   b. Copy `libopen_simulation_interface.dll` from `lib/osi3` subfolder to `lib`

The OSI class documentation is part of the source code and can be compiled using Doxygen. Instructions are located in the OSI `Readme.md`. A Pre-compiled version is located here: https://opensimulationinterface.github.io/open-simulation-interface/index.html. Note that this version of the documentation doesn't include the extensions from the OSI WorldInterface pull request.

4 Compiling World_OSI

Please adapt the variable `EXTRA_LIB_PATH` and `EXTRA_INCLUDE_PATH` to your needs. These are set in `global.pri`. Using the paths from this guide, the required values would have to be `C:/OpenPASS/thirdParty/lib` and `C:/OpenPASS/thirdParty/include` respectively.

5 Simulation

The proposed demo scenario can be simulated using the configuration files from `OpenPass_Source_Code/openPASS_Resource/OpenPass_OSI_UseCase`.

Remember to copy the required third party libraries (OSI, protobuf, Qt, MinGW) to the Slave's directory.

6 References

[1] https://github.com/OpenSimulationInterface/open-simulation-interface

[2] https://github.com/OpenSimulationInterface/open-simulation-interface/pull/244

[3] https://github.com/protocolbuffers/protobuf/releases

[4] https://cmake.org/files/v3.9/