Open Source SSAM Computation Engine API

New Global Systems for Intelligent Transportation Management

Phone and Fax: (662) 324-4084
li.zhang@ngsim.com

75 Cavalier Blvd.
Florence, KY 41042

May 1, 2017

Contents

Document Purpose

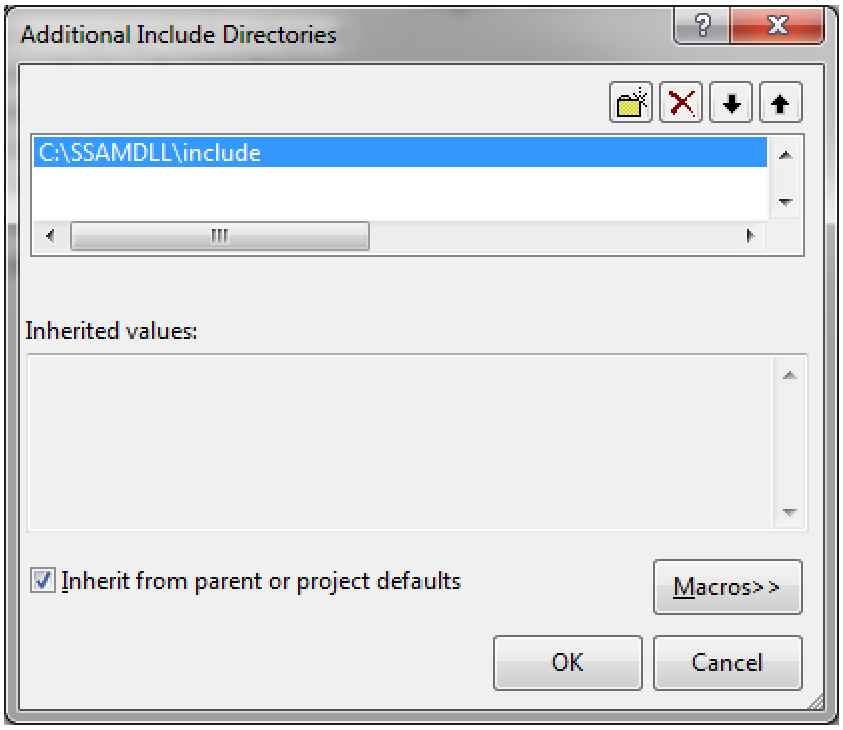
This document provides the details to use the Application Programming Interface (API) of computation engine of Surrogate Safety Assessment Model (SSAM) -3.0. SSAM is safety analysis tool. The API is provided as a DLL package. This document describes how to configure a project to use the API and the usage of API functions. API provides interface and provide functions for other program, for example, microscopic simulation software to call SSAM computation engine. The API may be of use to simulation software developers, researchers, transportation engineers, and safety engineers. SSAM API have been tested with ETFOMM, an open source microscopic simulation software sponsored by US DOT. Please refer to ETFOMM.org for details.

1. Locate SSAM API DLL

1.1. Download SSAMDLL.zip and place the extracted folder to C:\.

2. Configure project properties

2.1. Add "C:\SSAMDLL\include" folder to "Project Properties"->"C\C++"->"General"->"Additional Include Directories":


Additional Include Directories

C:\SSAMDLL\include

Inherited values:

[✓] Inherit from parent or project defaults

Figure 1 Add "include" in "Additional Include Directories"

2.2. Add “C:\SSAMDLL\lib” folder to “Project Properties”->”Linker”->”General”->”Additional Library Directories”:

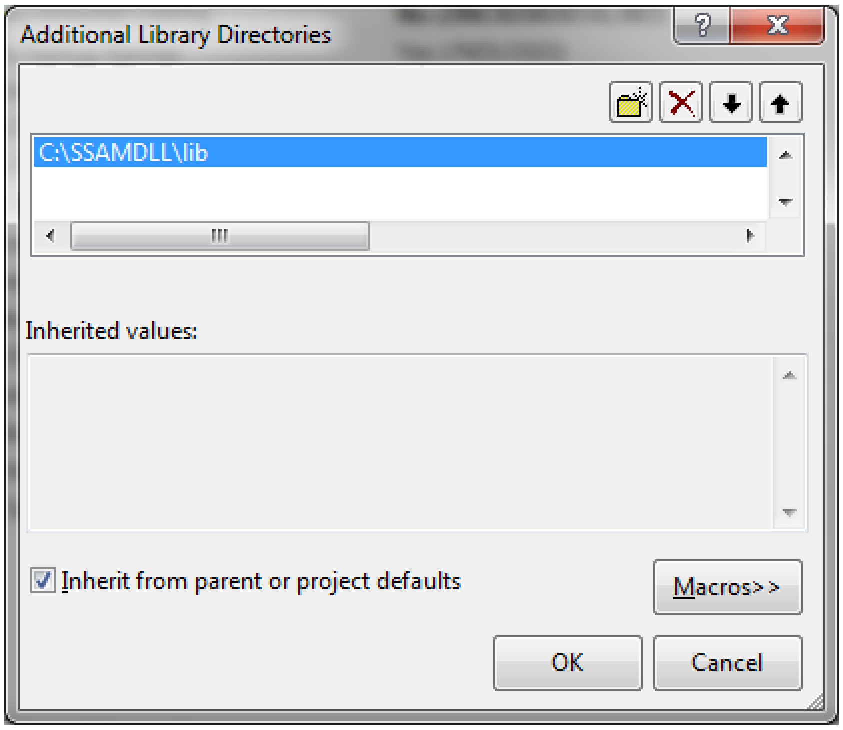

Additional Library Directories
C:\SSAMDLL\lib
Inherited values:
Inherit from parent or project defaults
Macros>>
OK
Cancel

Figure 2 Add "lib" in "Additional Library Directories"

2.3. Add SSAMDLL.lib to “Project Properties”->”Linker”->”Input”->”Additional Dependencies”:

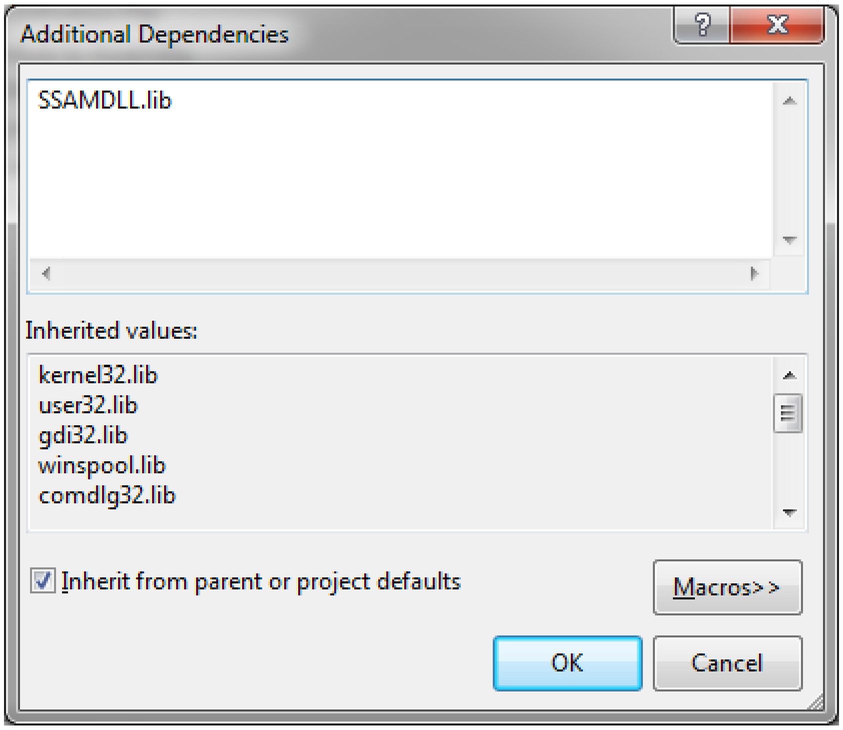


Figure 3 Add "SSAMDLL.lib" to "Additional Dependencies"

2.4. Copy “SSAMDLL.dll” from “bin” folder and place it in the same folder as the executable file of the created project.

3. Use SSAM API in source code

There are three ways to use SSAM API in source code: by a TRJ file, by a complete list of TRJ data, or by a list of TRJ data at each time step. First part of this section will show the common usage of the three methods, and the different usages will be explained in the second part.

3.1. Common Usage

3.1.1. Include SSAM.h in the program:

```
#include "SSAM.h"
```

3.1.2. Create an object of the SSAM class:

```
SSAMFuncs::SSAM SSAMRunner;
```

3.1.3. Before starting SSAM analysis, set max TTC and max PET:

```
SSAMRunner.SetMaxTTC(ttc);
SSAMRunner.SetMaxPET(pet);
```

3.1.4. After SSAM analysis is completed, get conflict list and summaries:

```
std::list<SP_Conflict> conflicts(SSAMRunner.GetConflictList());
std::list<SP_Summary> summaries(SSAMRunner.GetSummaries());
```

3.2. Different Usage

3.2.1. Run SSAM analysis by a TRJ file

3.2.1.1. Create a string to contain the TRJ file name:

```
std::string TRJFile("trjfile.trj");
```

3.2.1.2. Pass TRJ file name to SSAM Runner:

```
SSAMRunner.AddTrjFile(TRJFile);
```

3.2.1.3. Run SSAM analysis:

```
SSAMRunner.Analyze();
```

3.2.2. Run SSAM analysis by a complete list of TRJ data

3.2.2.1. Assign a project name for the SSAM Runner:

```
std::string projName("projectName");
```

3.2.2.2. Create a list to contain the input records:

```
std::list<TrjRecord> trjData;
```

3.2.2.3. Create input records:

3.2.2.3.1.Create one format record:

```
SSAMFuncs::InputFormat inputFormat;
inputFormat.m_Endian = 'L';
inputFormat.m_Version = version;
inputFormat.m_ZOption = zOption;
TrjRecord rec;
rec.SetRecordType (TrjRecord::FORMAT);
rec.SetFormat(inputFormat);
trjData.push_back(rec);
```

3.2.2.3.2.Create one dimension record:

```
SP_TrjDimensions dimensions = std::make_shared<TrjDimensions>();
dimensions->SetUnits(unit);
dimensions->SetScale(scale);
dimensions->SetMinX(minx);
dimensions->SetMinY(minY);
dimensions->SetMaxX(maxX);
dimensions->SetMaxY(maxY);
TrjRecord rec;
rec.SetRecordType (TrjRecord::DIMENSIONS);
rec.SetDimensions(dimensions);
trjData.push_back(rec);
```

3.2.2.3.3.Iterate through vehicle trajectories, for each time step:

3.2.2.3.3.1. Create one time step record:

```
TrjRecord rec;
rec.SetRecordType(TrjRecord::TIMESTEP);
rec.SetTimestep(timeStep);
trjData.push_back(rec);
```

3.2.2.3.3.2. Iterate through vehicle trajectories of this time step, for each vehicle, create one vehicle record:

```
SP_Vehicle veh = std::make_shared<Vehicle>();
veh->SetTimeStep(vehicleInfo.timestep);
veh->SetVehicleID(vehicleInfo.ID);
veh->SetLinkID(vehicleInfo.link);
veh->SetLaneID(vehicleInfo.lane);
veh->SetScale(scale);
veh->SetLength(vehicleInfo.length);
veh->SetWidth(vehicleInfo.width);
veh->SetSpeed(vehicleInfo.velocity);
veh->SetAcceleration(vehicleInfo.acceleration);
veh->SetPosition(vehicleInfo.frontB.x, vehicleInfo.frontB.y,
      vehicleInfo.rearB.x, vehicleInfo.rearB.y);
veh->SetFrontZ(vehicleInfo.frontB.z);
veh->SetRearZ(vehicleInfo.rearB.z);
TrjRecord rec;
rec.SetRecordType (TrjRecord::VEHICLE);
rec.SetVehicle(veh);
trjData.push_back(rec);
```

3.2.2.4.Pass input records to SSAM Runner:

```
SSAMRunner.AddTrjDataList(<ProjectName>, &trjData);
```

3.2.2.5.Run SSAM analysis:

```
SSAMRunner.Analyze();
```

3.2.3.Run SSAM analysis by a list of TRJ data at each time step of simulation

3.2.3.1.Assign a project name for the SSAM Runner:

```
SSAMRunner.SetTrjSrcName(<projectName>);
```

3.2.3.2.Initialize SSAM Runner:

```
SSAMRunner.Initialize();
```

3.2.3.3.Generate format record and set to SSAM Runner:

```
SSAMFuncs::InputFormat inputFormat;
inputFormat.m_Endian = 'L';
inputFormat.m_Version = version;
inputFormat.m_ZOption = zOption;
SSAMRunner.SetFormat(inputFormat);
```

3.2.3.4.Generate dimensions record and set to SSAM Runner:

```
SSAMFuncs::SP_Dimensions& pDimensions;
pDimensions->SetUnits(unit);
pDimensions->SetScale(scale);
pDimensions->SetMinX(minX);
pDimensions->SetMinY(minY);
pDimensions->SetMaxX(maxX);
pDimensions->SetMaxY(maxY);
SSAMRunner.SetDimensions(pDimensions);
```

3.2.3.5.At each time step of simulation:

3.2.3.5.1.Set time step to SSAM Runner:

```
SSAMRunner.SetTimeStep(timeStep);
```

3.2.3.5.2.Iterate through vehicle trajectories of current time step, for each vehicle, create one vehicle record and set to SSAM Runner:

```
SP_Vehicle veh = std::make_shared<Vehicle>();
veh->SetTimeStep(vehicleInfo.timestep);
veh->SetVehicleID(vehicleInfo.ID);
veh->SetLinkID(vehicleInfo.link);
veh->SetLaneID(vehicleInfo.lane);
veh->SetScale(scale);
veh->SetLength(vehicleInfo.length);
veh->SetWidth(vehicleInfo.width);
veh->SetSpeed(vehicleInfo.velocity);
```

```
veh->SetAcceleration(vehicleInfo.acceleration);
veh->SetPosition(vehicleInfo.frontB.x, vehicleInfo.frontB.y,
      vehicleInfo.rearB.x, vehicleInfo.rearB.y);
veh->SetFrontZ(vehicleInfo.frontB.z);
veh->SetRearZ(vehicleInfo.rearB.z);
 SSAMRunner.SetVehicle(veh);
```

3.2.3.5.3.After one simulation run is done, close the run in SSAM Runner:

```
SSAMRunner.CloseRun();
```

3.2.3.5.4.After all simulation runs are done, terminate SSAM Runner:

```
SSAMRunner.Terminate();
```