SUMO - HowTo... Add a global output File

Daniel Krajzewicz

Benjamin Maier

Eric Nicolay

SUMO - HowTo... Add a global output File

Daniel Krajzewicz
Benjamin Maier
Eric Nicolay

$Revision: 3772 $

Table of Contents

1. Introduction 1
    Document Information 1
    Document Scope 1
    Document Audience 1
    Document Structure 1
2. Description of the Goal 2
3. Implementation 3
    Output Management within the Simulation 3
    Using the Stream within the Simulation 4
        Making the Output optional 4
        Adding a global Access 5
        Adding writing Functions 5

Chapter 1. Introduction

Document Information

This document is a part of *SUMO* - *S*imulation of *U*rban *MO*bility. SUMO is an open source microscopic road traffic simulation. It can be found under http://sumo.sourceforge.net/.

This document fits to version 0.9.5. We will try to keep it up-to-date with the next versions.

Document Scope

SUMO contains several simulation-wide output options which, f.e., collect information from all instances of vehicles or streets and save them into a single file. Their implementation is hidden between different layers, structures, and methods so adding a new output may be not very straightforward. To ease the development of such new outputs, this HowTo shows the implementation of a new simulation-wide output by example.

Document Audience

This document contains information for *developers* who want to extend SUMO.

Document Structure

The first chapter describes what the described extension shall do (the goal).

The second one described how to extend SUMO to achieve the goal.

Chapter 2. Description of the Goal

For the TrafficOnline project, vehicles should be equipped with cellphones which report their states into an output. The output to implement shall be accessable from vehicle devices. Each vehicle may contain several devices and all shall write to a single file as soon as an event occures or after a given time has passed.

Chapter 3. Implementation

For the TrafficOnline project, vehicles should be equipped with cellphones which report their states into an output.

Output Management within the Simulation

All simulation-wide outputs are managed by the `SUMOFrame` and `MSNet`. `MSNet` holds the list of implemented, optional, simulation-wide outputs in an enumeration defined in `MSNet.h`:

```
...
    /**
     * @enum MSNetOutputs
     * List of network-wide outputs
     */
    enum MSNetOutputs {
        /// netstate output
        OS_NETSTATE = 0,
        /// emissions output
        OS_EMISSIONS,
        /// trip information output
        OS_TRIPDURATIONS,
        /// route information output
        OS_VEHROUTE,
        /// maximum value
        OS_MAX
    };
...
```

You may notice, that the names of the entries correspond with the simulation-wide outputs that were available so far: "netstate", "emissions-output", "tripinfo-output", and "vehroute-output". We add our new handle to the list changing this part of MSCORN like following, first:

```
...
    /**
     * @enum MSNetOutputs
     * List of network-wide outputs
     */
    enum MSNetOutputs {
        /// netstate output
        OS_NETSTATE = 0,
        /// emissions output
        OS_EMISSIONS,
        /// trip information output
        OS_TRIPDURATIONS,
        /// route information output
        OS_VEHROUTE,
        /// TrafficOnline-SS2 output  // <-- new entry
        OS_DEVICE_TO_SS2,             // <-- new entry
        /// maximum value
        OS_MAX
    };
...
```

At first, we have to add the command-line option which will force the simulation to generate our new output by adding the following line to the method "`void SUMOFrame::fillOptions(OptionsCont &oc)`":

```
oc.doRegister("ss2-output", new Option_FileName());
```

This means that the user can enable the output using the option `--ss2-output=<FILENAME>`. The application can determine whether this option is set using oc.isSet("ss2-output") and can retrieve the file name - if the option is set - using oc.getString("ss2-output").

Then we have to add procedures which build and close this new output. Most of the simulation wide output files are opened within "`std::vector<OutputDevice*> SUMOFrame::buildStreams(const OptionsCont &oc)`". We add the following line to this method:

```
ret[MSNet::OS_DEVICE_TO_SS2] = buildStream(oc, "ss2-output");
```

`OutputDevice *SUMOFrame::buildStream(const OptionsCont &oc, const std::string &optionName)` will build our output device as soon as the user wants the output to be generated. If the option, `--ss2-output` in this case, is not given, buildStream returns 0 (the output stream is not build). The returned pointer to the stream - independent whether a stream was built or not - will be set into the returned list of output streams to use, "`ret`". This list is given to `MSNet` later and all opened streams will be closed and deleted as soon as the simulation ends.

Using the Stream within the Simulation

Making the Output optional

In order not to create all outputs at every simulation run, but only those wished by the user, the list of currently wished outputs is managed by the `MSCORN`-class; CORN is an acronym for - COmpute Or Not. To add a new, optional output, we at first add the MSCORN-internal handle to the list of `MSCORN`-functions. Looking into `MSCORN.h`, you will find the following enumeration:

```
class MSCORN {
public:
    /**
     * @enum Function
     * A list of functions
     */
    enum Function {
        CORN_OUT_TRIPDURATIONS,
        CORN_OUT_EMISSIONS,
        CORN_OUT_VEHROUTES,

        CORN_VEH_REALDEPART,
        CORN_MEAN_VEH_TRAVELTIME,
        ...
```

You may notice, that the names of these first entries correspond with the simulation-wide outputs, "tripinfo-output", "emissions-output", and "vehroute-output", too. We add our new handle to the list changing this part of `MSCORN` like following, first:

```
class MSCORN {
public:
    /**
     * @enum Function
     * A list of functions
     */
    enum Function {
        CORN_OUT_TRIPDURATIONS,
        CORN_OUT_EMISSIONS,
        CORN_OUT_VEHROUTES,
```

```
        CORN_OUT_DEVICE_TO_SS2, // <-- new handle

        CORN_VEH_REALDEPART,
        CORN_MEAN_VEH_TRAVELTIME,
        ...
```

We inform `MSCORN` that our new output is wished as soon as the simulation is initialised. This is done by extending the method "`void MSNet::initialiseSimulation()`" by the following lines:

```
    // ... the same for TrafficOnline-SS2 information
    if ( myOutputStreams[OS_DEVICE_TO_SS2]!=0 ) {
        MSCORN::setWished(MSCORN::CORN_OUT_DEVICE_TO_SS2);
    }
```

That means that the output will only be set as being wished if it was previously created.

Adding a global Access

To ease accessing the output stream, we'll save a pointer to it within the `MSCORN`-class. To do this, we extend `MSCORN` by adding a static pointer to the stream and a method to assign it. The changes in `MSCORN.h` are:

```
...
public:
...
    static void setVehicleDeviceTOSS2Output(OutputDevice *s);
...
private:
    static OutputDevice *myVehicleDeviceTOSS2Output;
...
```

And in `MSCORN.cpp`:

```
...
OutputDevice *MSCORN::myVehicleDeviceTOSS2Output = 0;
...
void
MSCORN::setVehicleDeviceTOSS2Output(OutputDevice *s)
{
    myVehicleDeviceTOSS2Output = s;
}
...
```

Furthermore, we have to use our new assignment method. This is done in "`void MSNet::closeBuilding(...)`":

```
...
   MSCORN::setVehicleDeviceTOSS2Output(streams[OS_DEVICE_TO_SS2]);
...
```

Note

The usage of pointer to the stream within `MSCORN` is not the most pretty way to use the stream. One could also use it directly within `MSNet`.

Adding writing Functions

Now we have static interfaces for the output stream we can use. As the format of the data to generate is well defined, we'll use interfaces which allow only to access the stream with the set of values that shall be written. In `MSCORN`, we add a function for writing our messages. In `MSCORN.h`, we add:

```
...
    static void saveTOSS2_CalledPositionData(SUMOTime time,
        int callID, const std::string &pos, int quality);
...
```

And in `MSCORN.cpp`:

```
...
void
MSCORN::saveTOSS2_CalledPositionData(SUMOTime time, int callID,
                                     const std::string &pos,
                                     int quality)
{
    if(myVehicleDeviceTOSS2Output!=0) {
        myVehicleDeviceTOSS2Output->getOStream()
            << "01;" << time << ';' << callID << ';' << pos
            << ';' << quality << "\n";
    }
}
...
```

Please notice the usage of the `OutputDevice` (`getOStream()`).

Now we can write data into the stream from our devices. Data shall be written as soon as a call begins. To do this, as soon as one of our cell phone devices switches to the state "`STATE_CONNECTED`", we'll jump into `void MSCORN::saveTOSS2_CalledPositionData(SUMOTime time, int callID, const std::string &pos, int quality);` with our current values:

```
...
    // TOL_SPEC_SS2 (1.2)
    if( m_State==STATE_CONNECTED
        &&
        MSCORN::wished(MSCORN::CORN_OUT_DEVICE_TO_SS2)) {

        MSCORN::saveTOSS2_CalledPositionData(
            MSNet::getInstance()->getCurrentTimeStep(), gCallID,
            myVehicle.getLane().edge().getID(), 0);
    }
...
```

That's the position where we write our data into the stream for this example.

That's all folks!