Scenargie® 2.2

Programmers Guide

Space-Time Engineering, LLC.
October 2017

Contents

Preface

This document describes the structure of Scenargie and APIs to customize simulation modes. Please refer the related documents, “Scenargie Base Simulator User Guide” and “Scenargie Base Simulator Model Reference”.

Related documents

Installation Guide
Visual Lab User Guide
Base Simulator User Guide
Base Simulator Model Reference
Dot Eleven Module User Guide
LTE Module User Guide
Sensor Module for BLE User Guide
ITS Extension Module User Guide
Multi-Agent Extension Module User Guide
Multi-Agent Extension Module Model Reference
Fast Urban Propagation Module User Guide
High Fidelity Propagation Module User Guide
Trace Analyzer User Guide
Emulation Module User Guide

1. Scenargie Overview

This chapter describes the overview of Scenargie simulator.

1.1. Architecture

Scenargie is a discrete event simulator. Figure 1-1 shows the conceptual figure for simulation execution by discrete event simulator.

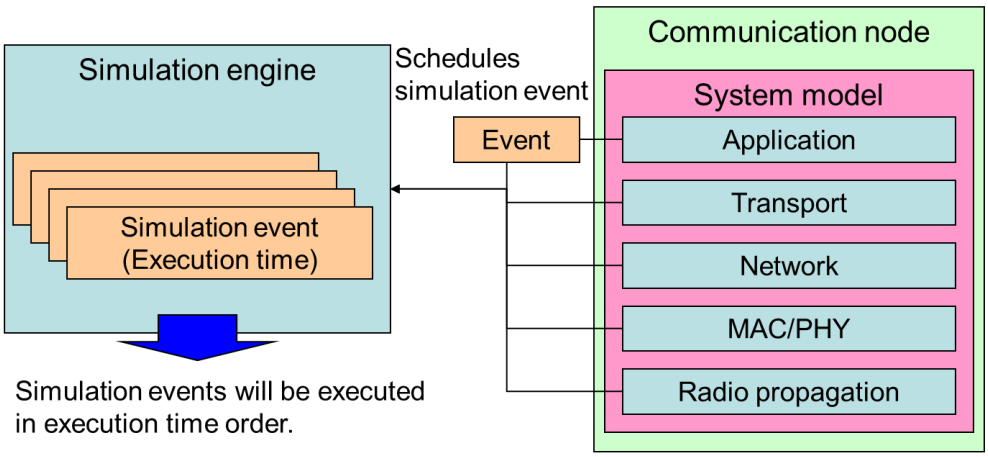

Figure 1-1. Simulation execution by discrete event simulator.

In a discrete event simulator, one set of procedures is defined as a simulation event. A simulation event is scheduled into a simulation engine with the event execution time. The simulation engine executes the scheduled simulation events in execution time order. During simulation event execution, new simulation event can be scheduled and the scheduled simulation event can be canceled according to a simulator model. For example, the one set of procedures that one packet is sent at certain time in application layer is a simulation event.

1.2. System structure

Figure 1-2 shows the system structure of Scenargie. The network simulator includes not only multiple communication nodes as simulator target but GIS data access, trace and statistics output functions. In addition, the network simulator has capability to interact with GUI (Scenargie Visual Lab).

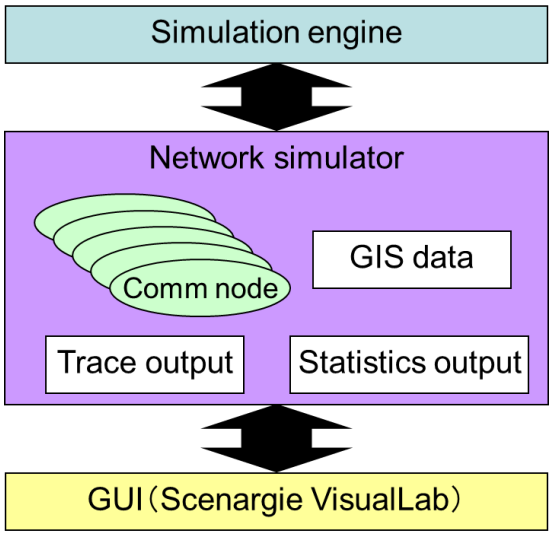

Figure 1-2 System structure of Scenargie.

1.3. Class relationships

1.3.1. Class relationship about simulation engine

Figure 1-3**Error! Reference source not found.** shows class relationship about simulation engine. Simulation engine (SimulationEngine class) is instantiated once in Scenargie program, and handles the scheduled simulation events (SimulationEvent class). Each node (NetworkNode class) has an interface to the simulation engine and conducts operations about simulation engine such as scheduling new simulation event.

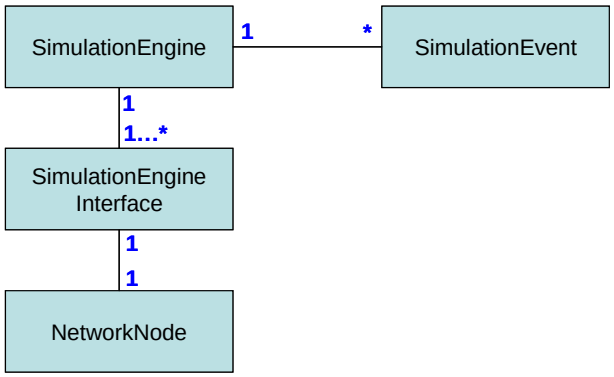

Figure 1-3 Class relationship about simulation engine.

1.3.2. Class relationship about system model

Figure 1-4**Error! Reference source not found.** shows class relationship about system model. Network simulator (NetworkSimulator class) includes multiple nodes (NetworkNode class). Each node has an application layer (ApplicationLayer class), a transport layer (TransportLayer class), and a network layer (NetworkLayer class). The application layer has multiple applications (Application class) and the transport layer has TCP (TcpProtocol class) and UDP (UdpProtocol class). The network layer includes a MAC layer (MacLayer class) per network interface.

For wireless simulation, there is a radio propagation environment (SimplePropagationModel class) in a network simulator. Each node accesses to the radio propagation environment via an interface (SimplePropagationModelForNode class). SimplePropagationModel class is instantiated one or multiple times depending on system model.

Note that NetworkSimulator class, NetworkNode class, Application class, NetworkLayer class, and MacLayer class are base classes. For instantiation, an inherited class is required.

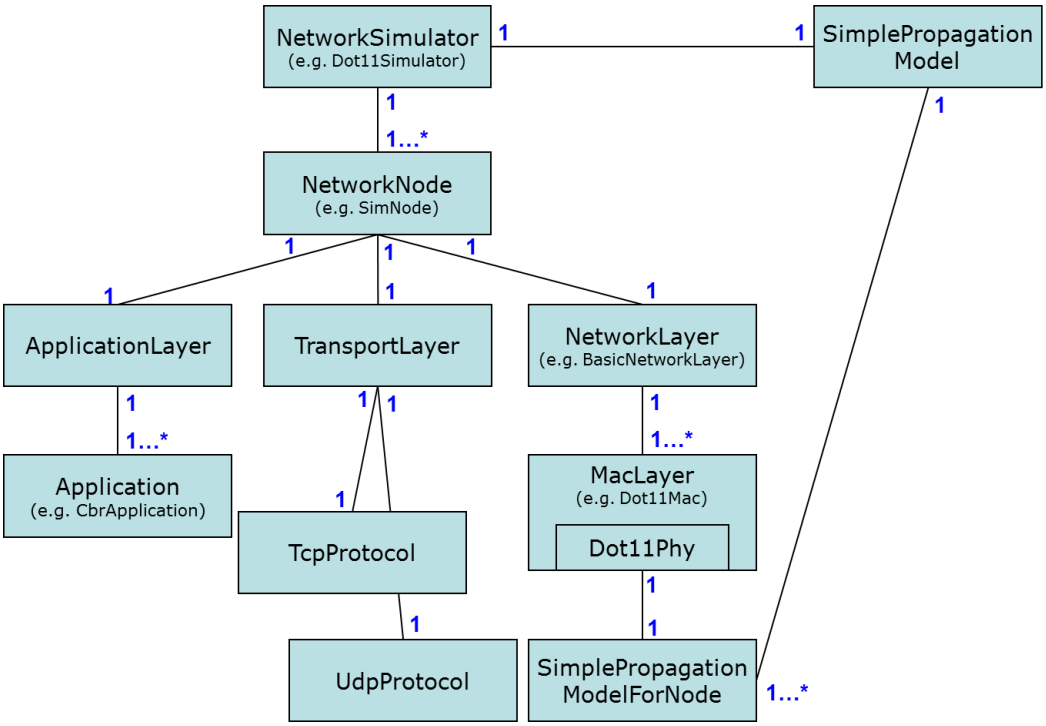

Figure 1-4 Simulation execution by discrete event simulator.

1.3.3. Instantiation

Simulation engine and network simulator are instantiated in main function. The following example shows that a simulation engine and a network simulator are instantiated in main function.

base/sim.cpp

```
int main(int argc, char* argv[])
{
...
    shared_ptr<SimulationEngine> theSimulationEnginePtr(
        new SimulationEngine(
            theParameterDatabaseReader,
            runSequentially,
            numberParallelThreads));

    BasicNetworkSimulator theNetworkSimulator(
        theParameterDatabaseReaderPtr,
        theSimulationEnginePtr,
        runSeed,
        runSequentially);
...
}
```

1.4. Interfaces between layers

Figure 1-5**Error! Reference source not found.** and Table 1-1**Error! Reference source not found.** shows the interfaces between layers to send and receive a packet. As you can see, a predefined function is called to do an action. These functions are pure virtual functions in abstract classes. You can override the functions as you want in an inherited class.

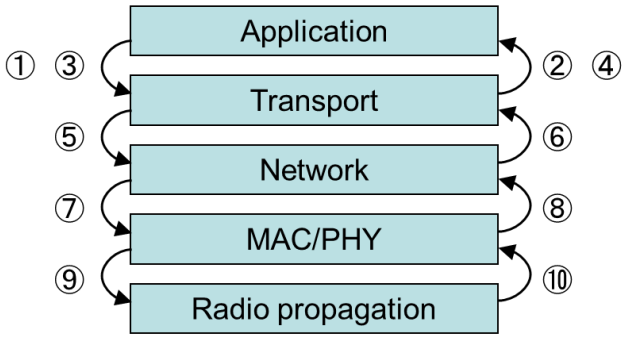

Figure 1-5 Interfaces between layers.

Table 1-1 Functions to interface between layers.

scensim_transport.h/cpp

	Class	Function	Description
①	UdpProtocol	SendPacket ()	Sends a UDP packet to network layer.
②	UdpProtocol:: PacketForAppFromTransportLayerHandler	ReceivePacket ()	Receives a UDP packet from network layer.
③	TcpConnection	SendDataBlock ()	Sends TCP data block.
④	TcpConnection:: AppTcpEventHandler	ReceiveDataBlock ()	Receives TCP data block.
⑥	ProtocolPacketHandler	ReceivePacketFromNetworkLayer ()	Receives a packet from network layer.

scensim_network.h/cpp

	Class	Function	Description
⑤	NetworkLayer	ReceivePacketFromUpperLayer ()	Receives a packet from upper layer (transport layer).

⑧	NetworkLayer	ReceivePacketFromMac ()	Receives a packet from MAC layer.

scensim_mac.h

	Class	Function	Description
⑦	MacLayer	NetworkLayerQueueChangeNotification ()	Gets a transmission queue change notification from network layer.

scensim_prop.h

	Class	Function	Description
⑨	SimplePropagationModelForNode	TransmitSignal	Transmits a signal.
⑩	SimplePropagationModelForNode::SignalHandler	ProcessSignal	Starts or ends a signal reception.

1.5. Scenario file

As “Base Simulator User Guide” describes, several text-based setting files are utilized as a scenario in Scenargie. Table 1-2**Error! Reference source not found.** shows the relationship between scenario files and classed to load a scenario file.

Table 1-2 Relationship between scenario files and classes to load.

Scenario file (extension)	Class	Source file
Configuration file (.config)	ParameterDatabaseReader	scensim_paramio.h
Mobility file (.mob)	TraceFileMobilityModel	scensim_mobility.h
Bit error table file /block error table file (.ber/.bler)	BitOrBlockErrorRateCurveDatabase	scensim_bercurves.h
Statistics configuration file (.statconfig)	ReadStatConfigFile (Global function)	scensim_stats.cpp
Static route file (.routes)	ReadStaticRoutingTableFile (Global function)	scensim_network.h
Antenna file (.ant)	AntennaPatternDatabase	scensim_proploss.h
Material file (.material)	GisSubsystem	scensim_gis.h
Shape file (.shp)	GisSubsystem	scensim_gis.h

2. Common Functions

This chapter describes classes and methods of common functions to customize a system model. Scenargie is written in standard C++. All source code for system models will be disclosed upon software license purchase. You can modify any part of the system model. Chapter 3 describes the system model and chapter 4 describes APIs list for customization. Please note that a method to build a simulator is shown in “Scenargie Base Simulator User Guide”.

2.1. Node ID

Node is a fundamental unit in Scenargie simulation. Nodes include not only communication objects but also GIS objects. A node is identified with Node ID (Type: NodeId). NodeId is unsigned integer data type as follows and can theoretically handle about 4.3 billion nodes. Node ID for communication objects is typically assigned from one in ascending order. However, Scenargie can handle unused number. Regarding GIS objects, starting number and range of Node ID is predefined for each GIS object type such as road and building. In addition, Node ID 0 and UINT_MAX are reserved for ANY_NODEID and INVALID_NODEID respectively.

scensim_nodeid.h

```
typedef unsigned int NodeId;

const NodeId InvalidNodeId = UINT_MAX;
const NodeId INVALID_NODEID = InvalidNodeId;
const NodeId AnyNodeId = 0;
const NodeId ANY_NODEID = AnyNodeId;

const NodeId GISOBJECT_ROAD_START_NODEID = 100000000;
const NodeId GISOBJECT_INTERSECTION_START_NODEID = 101000000;
const NodeId GISOBJECT_BUILDING_START_NODEID = 102000000;
const NodeId GISOBJECT_WALL_START_NODEID = 102500000;
const NodeId GISOBJECT_RAIL_START_NODEID = 103000000;
const NodeId GISOBJECT_WAY_STATION_START_NODEID = 104000000;
const NodeId GISOBJECT_NODE_STATION_START_NODEID = 104500000;
const NodeId GISOBJECT_SIGNAL_START_NODEID = 105000000;
const NodeId GISOBJECT_BUSSTOP_START_NODEID = 105500000;
const NodeId GISOBJECT_AREA_START_NODEID = 106000000;
const NodeId GISOBJECT_PARK_START_NODEID = 106500000;
const NodeId GISOBJECT_ENTRANCE_START_NODEID = 107000000;
const NodeId GISOBJECT_SERVICEAREA_START_NODEID = 108000000;
const NodeId GISOBJECT_GENERIC_POLYGON_START_NODEID = 109000000;
```

2.2. Simulation time

2.2.1. SimTime

Simulation time in Scenargie is handled with SimTime. SimTime is long long int data type as follows. The unit of integer is nanosecond. Therefore, simulation can run a long simulation scenario with nanosecond accuracy. Reserved words for second, millisecond, and microsecond are provided for convenience.

scensim_time.h

```
typedef long long int SimTime;

const SimTime NANO_SECOND = 1;
const SimTime MICRO_SECOND = 1000 * NANO_SECOND;
const SimTime MILLI_SECOND = 1000 * MICRO_SECOND;
const SimTime SECOND = 1000 * MILLI_SECOND;

const SimTime ZERO_TIME = 0;
const SimTime INFINITE_TIME = LLONG_MAX;
```

2.2.2. Obtaining simulation time

SimulationEngineInterface::CurrentTime() can be used to get current time in a model as follows.

```
const SimTime currentTime =
    simulationEngineInterfacePtr->CurrentTime();
```

2.3. Simulation event

As described in Chapter 1, a system model schedules a series of procedures as a simulation event with an execution time to simulation engine. The simulation engine executes the scheduled events in execution time order. This section describes how to make, schedule, reschedule, and cancel a simulation event.

2.3.1. Making and scheduling a simulation event

The following shows two steps to execute an original simulation event; 1) Defining a simulation event class and 2) Scheduling a simulation event.

1) Defining a simulation event class

A simulation event class should be defined by inheriting SimulationEvent class.

scensim_engine.h

```
class SimulationEvent {
public:
    virtual ~SimulationEvent() { }
    virtual void ExecuteEvent() = 0;
};
```

The following example shows that original simulation is defined by inheriting SimulationEvent class. A series of procedures to be executed should be written in ExecuteEvent() function.

```
class MyEvent : public SimulationEvent {
public:
    void ExecuteEvent() { cout << “Hello” << endl; }
};
```

Generally, a simulation event will be executed in the object that scheduled the event. Therefore, a self-pointer will be passed when a simulation event is instantiated. Then, the

simulation event calls a function with the pointer. The following is an example of calling a function in self-object in a protocol model.

```
class MyMacLayer : public MacLayer {

    void MyFunction() { cout << “executing … MyFunction” << endl; }

    class MyEvent2 : public SimulationEvent {
    public:
        MyEvent2(MyMacLayer* initMacPtr) : macPtr(initMacPtr) {}
        void ExecuteEvent() {macPtr->MyFunction(); }
    private:
        MyMacLayer* macPtr;
    };
...
```

2) Scheduling a simulation event

A simulation event with the execution time will be scheduled to simulation engine via SimulationEngineInterface.

The following example shows that MyMacLayer::MyFunction() is executed 60 seconds after current time.

```
MyMacLayer::MyMacLayer {
...
    const SimTime currentTime =
        simEngineInterfacePtr->CurrentTime();

    const SimTime eventTime = currentTime + (60 * SECOND);

     simEngineInterfacePtr->ScheduleEvent(
        new MyEvent2(this), eventTime);
...
```

2.3.2. Rescheduling and cancel a simulation event

The previous subsection describes how to make and schedule a simulation event if the event cannot be rescheduled or cancelled. This subsection describes how to reschedule or cancel a simulation event.

If a simulation event may be rescheduled or cancelled, an event ticket (EventRechduleTicket class) needs to be issued in advance. Rescheduling and cancelling will be conducted with the event ticket. In addition, an event ticket cannot be copied. The pointer to the event ticket can be stored in a container such as vector or map.

Scheduling a simulation event with event ticket:

```
EventRescheduleTicket myEventTicket;

simEngineInterfacePtr->ScheduleEvent(
    new MyEvent2(this), eventTime, myEventTicket);
```

The following is an example of rescheduling and cancelling a simulation event. The simulation event should not be executed yet to reschedule or cancel.

The following example shows that a scheduled simulation event is rescheduled to 3 seconds after current time.

```
if (!myEventTicket.IsNull()) {
    const SimTime newEventTime =
        simEngineInterfacePtr->CurrentTime() + (3 * SECOND);

    simEngineInterfacePtr-> RescheduleEvent(
        myEventTicket, newEventTime);
}
```

The following example shows that the simulation event is cancelled.

```
if (!myEventTicket.IsNull()) {
    simEngineInterfacePtr-> CancelEvent(myEventTicket);
}
```

2.3.3. Error in simulation event execution

The following error can be happened in simulation event execution.

```
void ScenSim::SimEngineThreadPartition::ScheduleEvent(const
boost::shared_ptr<ScenSim::SimulationEvent>&, const ScenSim::NodeId&, const
ScenSim::SimTime&, ScenSim::EventRescheduleTicket&):
Assertion `eventTime >= currentTime' failed.
```

The above error happens when the simulation event is scheduled with a past time. To avoid this error, a simulation event should be scheduled with a future time including current time. That can be done to specify an event time with current time plus relative time (currentTime + alpha).

2.4. Packet

Scenargie defines Packet class for packet type. Packets are generated in application layer or other layer, and the packets are delivered to destination nodes via each layer. The following describes how to create a packet, how to manipulate (add/read/delete) headers, and how to add extra information to a packet.

2.4.1. Creating a packet

Packet::CreatePacket() with an instance of SimulationEngineInterface and packet payload is utilized to create a packet. Multiple CreatePacket() functions are defined according to packet payload type. Packet payload type includes structure, vector<unsigned char>, string, and unsigned char* (Refer Chapter 4 for details).

Scenargie provides two types of memory allocation for a packet. One is that all payload size is allocated. The other is that requisite minimum payload size is allocated (virtual payload). The former means that the created packet is equivalent to a real packet on the physical system. Therefore, the packets can be utilized for emulation where a packet goes through physical network and simulated network (Emulation feature for Scenargie requires an extension module). The latter (virtual payload) is good for simulation only and minimizes memory consumption. To enable virtual payload, a flag is passed as an argument into Packet::CreatePacket(). In addition, there is a function in TCP to enable virtual payload. The following example shows that CBR packet is created. Fourth argument of Packet::CreatePacket() is a flag to enable or disable virtual payload function. When the flag is true, virtual payload is enabled. When the flag is false, the feature is disabled.

scensim_app_cbr.h

```
class CbrApplication: public Application {
...
    struct CbrPayloadType {
        unsigned int sequenceNumber;
        SimTime sendTime;

        CbrPayloadType(
            const unsigned int initSequenceNumber,
            const SimTime initSendTime)
            :
            sequenceNumber(initSequenceNumber),
            sendTime(initSendTime)
        {}
    };//CbrPayloadType//
...
};
```

```
void CbrSourceApplication::SendPacket() {
...
    currentPacketSequenceNumber++;

    CbrPayloadType cbrAppPayload(
        currentPacketSequenceNumber,
        simulationEngineInterfacePtr->CurrentTime());

    unique_ptr<Packet> packetPtr =
        Packet::CreatePacket(
            *simulationEngineInterfacePtr,
            cbrAppPayload,
            packetPayloadSizeBytes,
            useVirtualPayload);
...
}
```

2.4.2. Adding a header

Packet::AddPlainStructHeader() or Packet::AddRawHeader() is utilized to add a header to a packet. The former function is utilized when a header is predefined as a structure. The latter function is utilized when a header is a byte stream. The following is an example of adding a UDP header and an IP header.

Adding a UDP header:

scensim_transport.h/cpp

```
struct UdpHeader {
    UdpHeader(
        unsigned short int initSourcePort,
        unsigned short int initDestinationPort,
        unsigned short int initLength)
        :
        sourcePort(initSourcePort),
        destinationPort(initDestinationPort),
        length(initLength),
        unused(0)
    {}
    unsigned short int sourcePort;
    unsigned short int destinationPort;
    unsigned short int length;
    unsigned short int unused;
};
```

```
void UdpProtocol::SendPacket() {
...
    packetPtr->AddPlainStructHeader(
        UdpHeader(
            HostToNet16(sourcePort),
            HostToNet16(destinationPort),
            HostToNet16(static_cast<unsigned short int>(
                packetPtr->LengthBytes() + sizeof(UdpHeader)))));
...
}
```

Adding an IP header:

scensim_network.cpp

```
void BasicNetworkLayer::ReceivePacketFromUpperLayer() {
...
    IpHeaderModel
        header(
            trafficClass,
            packetPtr->LengthBytes(),
            hopLimit,
            protocol,
            sourceAddress,
            destinationAddress);

    packetPtr->AddRawHeader(
        header.GetPointerToRawBytes(), header.GetNumberOfRawBytes());
    packetPtr->AddTrailingPadding(header.GetNumberOfTrailingBytes());
...
}
```

2.4.3. Reading and deleting a header

Packet::GetAndReinterpretPayloadData() or Packet::GetRawPayloadDat() is utilized to read information from a header. The former function is utilized when a header is a structure. The latter function is utilized for a header is a byte stream. An argument of the functions is offset bytes from the beginning of the packet and to the position to start reading. When a header is read from the beginning, the argument is not necessary.

Packet::DeleteHeader() is utilized to delete a header. An argument of the function is a size of bytes to be deleted. The following is an example of reading and deleting a UDP header and an IP header.

Reading and deleting a UDP header:

scensim_transport.cpp

```
void UdpProtocol::ReceivePacketFromNetworkLayer() {
...
    UdpHeader anUdpHeader =
        packetPtr->GetAndReinterpretPayloadData<UdpHeader>();
    const unsigned short int sourcePort =
        NetToHost16(anUdpHeader.sourcePort);
    const unsigned short int destinationPort =
        NetToHost16(anUdpHeader.destinationPort);

    packetPtr->DeleteHeader(sizeof(UdpHeader));
...
}
```

Reading and deleting an IP header:

secnesim_network.cpp

```
void BasicNetworkLayer::ReceivePacketFromMac() {
...
    IpHeaderOverlayModel ipHeader(
        packetPtr->GetRawPayloadData(), packetPtr->LengthBytes());
...
    ipHeader.StopOverlayingHeader();
    packetPtr->DeleteHeader(ipHeaderLength);
...
}
```

2.4.4. Reading payload information

Reading payload information in application layer is same as reading a header. The following is an example of reading payload information in CBR application.

scensim_app_cbr.h

```
void CbrSinkApplication::ReceivePacket() {
...
    CbrPayloadType cbrPayload =
        packetPtr->GetAndReinterpretPayloadData<CbrPayloadType>();
...
}
```

2.4.5. Adding extra information to a packet

In Scenargie, all information including application payload and headers in each layer is stored within a packet like an actual packet. Therefore, the information required for simulation model is also stored within the packet. However, you can add extra information separately from payload or headers in the packet. For example, you can add the time when a packet is inserted into a transmission queue. Please note that the feature requires extra computational power to manipulate a packet, and adding information within a packet is preferable in terms of simulation runtime performance.

There are three steps to add extra information to a packet; 1) Creating a container to add extra information, 2) Adding extra information to a packet, 3) Obtaining extra information from a packet. The following describes each step.

1) Creating a container to add extra information

A container to add extra information can be created by inheriting ExtrinsicPacketInformation class. As ExtrinsicPacketInformation::Clone() is a pure virtual function which copies itself and returns the self- pointer, it is required to implement the function for your own inherited class.

scensim_packet.h

```
class ExtrinsicPacketInformation {
public:
    virtual ~ExtrinsicPacketInformation() { }

    virtual shared_ptr<ExtrinsicPacketInformation> Clone() = 0;

};
```

The following shows the example of creating a container to store information in SimTime.

```
class MyInformation : public ExtrinsicPacketInformation {
public:
    MyInformation(const SimTime& initEnqueuedTime)
    : enqueuedTime (initEnqueuedTime) {}
    shared_ptr<ExtrinsicPacketInformation> Clone() {
        return shared_ptr<ExtrinsicPacketInformation>(
            new MyInformation(*this));
    }
    SimTime GetEnqueuedTime () const { return enqueuedTime; }
private:
    SimTime enqueuedTime;
};
```

2) Adding extra information to a packet

Packet::AddExtrinsicPacketInformation() is utilized to add extra information to a packet. It is required to predefine an identifier in ExtrinsicPacketInfoId for the extra information. ExtrinsicPacketInfoId is an alias of std::string.

The following is an example of adding current time to a packet as extra information.

```
const ExtrinsicPacketInfoId enqueuedTimeInfoId = “enqueuedTime”;
...
const SimTime currentTime =
    simulationEngineInterfacePtr->CurrentTime();

packetPtr->AddExtrinsicPacketInformation(
    enqueuedTimeInfoId,
    shared_ptr<ExtrinsicPacketInformation>(
        new MyInformation(currentTime)));
...
```

3) Obtaining extra information from a packet

Packet::GetExtrinsicPacketInformation() is utilized to obtain extra information from a packet. As the function assumes that the information has already been added to the packet, it is required to check that the information is actually added before calling the function. Packet::CheckExtrinsicPacketInformationExist() is utilized to check the information is added.

The following is an example of obtaining extra information from a packet.

```
if(packetPtr->CheckExtrinsicPacketInformationExist(enqueuedTimeInfoId)) {

    MyInformation& myInformation =
      packetPtr->GetExtrinsicPacketInformation<MyInformation>(
senqueuedTimeInfoId);

    const SimTime enqueuedTime = myInformation.GetEnqueuedTime();
}
```

2.4.6. Packet ID

When a packet is created (Packet::CreatePacket() is called), Packet ID (PacketId class) as a unique ID is generated and added to the packet. Packet ID consists of node ID and sequence ID. Node ID is an ID of the node that created the packet. Sequence ID is a unique ID per packet of the node. The packet ID remains in copying packet unless new packet ID is assigned manually.

Packet::GetPacketId() is utilized to get packet ID from a packet. PakcetId::GetSourceNodeId() and PakcetId::GetSourceNodeSequenceNumber () are utilized to get node ID and sequence ID from packet ID.

2.5. Random number generator

Seed of random number generator (hereafter called “run seed”) can be set as a parameter to run a scenario. “run seed” is the global seed and is shared in all simulation models. If “run seed” does not change, the simulation results never change. In each node, a random number is generated by using node level random seed (hereafter called “node seed”). “node seed” is created by hashing node ID and “run seed”. Therefore, “node seed” is a unique value that is determined by node ID and “run seed”, but “node seed” and “run seed” are hardly correlated. In each interface, “interface seed” is generated by interface index and node in the same way of generating node seed. In each model such as application and MAC, seed is generated by hashing an arbitrary number and “node seed” or “interface seed”.

Seed for mobility (hereafter called “mobility seed”) in addition to “run seed” can be set for mobility model. If same “mobility seed” is used and multiple “run seed” are used for scenarios, serval simulation can be conducted with keeping same mobility behavior but different communication behavior. If “mobility seed” is not set, the value of “run seed” is used for mobility model.

In Scenargie, general and high quality random number generators utilize boost::rand48 and boost::mt19937 in boost library respectively.

Type of random seed is uint32_t as follows.

randomnumbergen.h

```
typedef uint32_t RandomNumberGeneratorSeed;
```

To generate a random number, it is required to initialize a random number generator with a seed. Then, the random number generator returns a uniform distributed random number. The followings show each step.

1) Setting a seed for random number generator

RandomNumberGenerator::SetSeed() is utilized to set a seed for random number generator. A seed can be generated with HashInputsToMakeSeed(). As “node seed” is automatically generated in the constructor of NetworkNode class, you can get “node seed” with NetworkNode::GetNodeSeed().

2) Generating a random number

RandomNumberGenerator::GenerateRandomDouble() is utilized to generate a uniform distributed random number (double) between 0 and 1 [0,1). RandomNumberGenerator::GenerateRandomInt() is utilized to generate a uniform distributed random number (integer) between specified integer range.

The following examples shows that random number is generated and used for picking a random position in RandomWaypointMobilityModel class and for picking a random backoff slot in Dot11Mac class.

scensim_mobility.h

```
class RandomWaypointMobilityModel : public ObjectMobilityModel {
...
    static const long int SEED_HASHING_INPUT = 35620163;
...
};
```

```
RandomWaypointMobilityModel::RandomWaypointMobilityModel(
...
    const RandomNumberGeneratorSeed nodeSeed =
        HashInputsToMakeSeed(runSeed, theNodeId);
    RandomNumberGenerator aRandomNumberGenerator(
        HashInputsToMakeSeed(
            nodeSeed, theInterfaceId, SEED_HASHING_INPUT));
...
}
```

```
Vertex GetRandomPositionInPolygon (
...
    randomPosition.x =
        minRect.minX +
        (minRect.maxX - minRect.minX) *
        aRandomNumberGenerator.GenerateRandomDouble();
    randomPosition.y =
        minRect.minY +
        (minRect.maxY - minRect.minY) *
        aRandomNumberGenerator.GenerateRandomDouble();
...
}
```

dot11_mac.h (Dot Eleven Module)

```
class Dot11Mac : public MacLayer {
...
    RandomNumberGenerator aRandomNumberGenerator;
...
};
```

```
Dot11Mac::Dot11Mac() {
...
    aRandomNumberGenerator(
        HashInputsToMakeSeed(nodeSeed, initInterfaceIndex)),
...
}
```

```
void Dot11Mac::RecalcRandomBackoff() {
...
    accessCategoryInfo.currentNumOfBackoffSlots =
        aRandomNumberGenerator.GenerateRandomInt(0,
        accessCategoryInfo.currentContentionWindowSlots);
...
}
```

2.6. Parameter

All parameters in a configuration file are utilized with ParameterDatabaseReader class. Here, the way to specify parameters in a configuration file is described in “Scenargie Base Simulator User Guide”.

Loading of configuration file and instantiation of ParameterDatabaseReader class are conducted in a main function. The reference of ParameterDatabaseReader is passed to each simulation model and the reference is utilized to access the parameters.

The following is an example of instantiating ParameterDatabaseReader in Base Simulator (base/sim.cpp).

base/sim.cpp

```
int main(int argc, char* argv[])
{
...
    MainFunctionArgvProcessingBasicParallelVersion1(
        argc,
        argv,
        configFileName,
        isControlledByGui,
        numberParallelThreads,
        runSequentially,
        seedIsSet,
        runSeed);

    shared_ptr<ParameterDatabaseReader> theParameterDatabaseReaderPtr(
        new ParameterDatabaseReader(configFileName));
    ParameterDatabaseReader& theParameterDatabaseReader =
        *theParameterDatabaseReaderPtr;
...
}
```

ParameterDatabaseReader class has the following function to check that the specified parameter exists. Multiple functions are defined according to scope (global, node, interface/instance). Please refer “Scenargie Base Simulator User Guide” for details of scope.

ParameterDatabaseReader::ParameterExists()

The followings are functions to read a parameter. Multiple functions are defined according to scope and type of parameter value.

ParameterDatabaseReader::ReadBool(): Read bool type parameter.
ParameterDatabaseReader::ReadInt(): Read int type parameter.
ParameterDatabaseReader::ReadBigInt(): Read long long int type parameter.
ParameterDatabaseReader::ReadNonNegativeInt(): Read unsigned int type parameter.
ParameterDatabaseReader::ReadNonNegativeBigInt(): Read unsigned long long int type parameter.
ParameterDatabaseReader::ReadDouble(): Read double type parameter.
ParameterDatabaseReader::ReadTime(): Read SimTime type parameter.
ParameterDatabaseReader::ReadString(): Read string type parameter.

The followings are examples of reading parameters with global, node or interface scope.

scensim_netsim.h

```
NetworkSimulator::NetworkSimulator() {
...
string antennaFileName;
if (theParameterDatabaseReader.ParameterExists("custom-antenna-file")) {
    antennaFileName =
    theParameterDatabaseReader.ReadString("custom-antenna-file");
}//if//
...
}
```

scensim_network.h

```
BasicNetworkLayer::ConstructNetworkLayer (
...
    if (theParameterDatabaseReader.ParameterExists(
        "network-hop-limit", theNodeId)) {
        const int hopLimitInt =
            theParameterDatabaseReader.ReadInt(
            "network-hop-limit", theNodeId);
...
}
```

```
void BasicNetworkLayer::CreateMacOnInterfaceIfNotCustom(
...
    if (theParameterDatabaseReader.ParameterExists(
        "mac-protocol", theNodeId, theInterfaceId)) {
        macProtocolString =
            theParameterDatabaseReader.ReadString(
            "mac-protocol", theNodeId, theInterfaceId);
...
}
```

2.7. Statistics output

Please refer “Scenargie Base Simulator User Guide” to add and output original statistic.

2.8. Trace output

Please refer “Scenargie Base Simulator User Guide” to add and output original trace.

3. System Model

This chapter describes the class structures at each layer and how to customize models. First, network simulator and communication nodes are explained. Then, each layer is explained. APIs at each layer are listed in Chapter 4.

3.1. Network simulator and communication nodes

In Scenargie, one network simulator has multiple communication nodes and each node has a protocol stack of system models as described in Figure 1-4. A network simulator is created by inheriting NetworkSimulator class. A communication node is created by inheriting NetworkNode class class.

The following example shows that a network simulator and communication nodes are created in Dot Eleven Module. You can create original network simulator and communication node by inheriting NetworkSimulator class and NetworkNode class.

dot11/sim.cpp （Dot Eleven Module）

```
class Dot11Simulator : public NetworkSimulator {
public:
    Dot11Simulator(
        const shared_ptr<ParameterDatabaseReader>&
            initParameterDatabaseReaderPtr,
        const shared_ptr<SimulationEngine>& initSimulationEnginePtr,
        const RandomNumberGeneratorSeed& initRunSeed,
        const bool initRunSequentially);
...
Dot11Simulator::Dot11Simulator(
    const shared_ptr<ParameterDatabaseReader>&
        initParameterDatabaseReaderPtr,
    const shared_ptr<SimulationEngine>& initSimulationEnginePtr,
    const RandomNumberGeneratorSeed& initRunSeed,
    const bool initRunSequentially)
    :
    NetworkSimulator(
        initParameterDatabaseReaderPtr,
        initSimulationEnginePtr,
        initRunSeed,
        initRunSequentially)
    {
...
```

```
class SimNode : public NetworkNode {
public:
    SimNode(
        const ParameterDatabaseReader& initParameterDatabaseReader,
        const GlobalNetworkingObjectBag& globalNetworkingObjectBag,
        const shared_ptr<SimulationEngineInterface>&
            simulationEngineInterfacePtr,
        const NodeId& theNodeId,
        const RandomNumberGeneratorSeed& runSeed,
        const shared_ptr<ObjectMobilityModel>&
            nodeMobilityModelPtr);
    ~SimNode() {}
...
}
...
SimNode::SimNode(
    const ParameterDatabaseReader& theParameterDatabaseReader,
    const GlobalNetworkingObjectBag& theGlobalNetworkingObjectBag,
    const shared_ptr<SimulationEngineInterface>&
        initSimulationEngineInterfacePtr,
    const NodeId& initNodeId,
    const RandomNumberGeneratorSeed& initRunSeed,
    const shared_ptr<ObjectMobilityModel>& initNodeMobilityModelPtr)
    :
    NetworkNode(
        theParameterDatabaseReader,
        theGlobalNetworkingObjectBag,
        initSimulationEngineInterfacePtr,
        initNodeMobilityModelPtr,
        initNodeId,
        initRunSeed),
    nodeMobilityModelPtr(initNodeMobilityModelPtr)
{
}
...
```

3.2. Application layer

As described in Chapter 1, each node has application layer and application layer has multiple applications according to simulation scenarios. The following describes that existing application classes and class structure, and how to add original application.

3.2.1. Overview

Application class is an abstract class and each application is created by inheriting Application class.

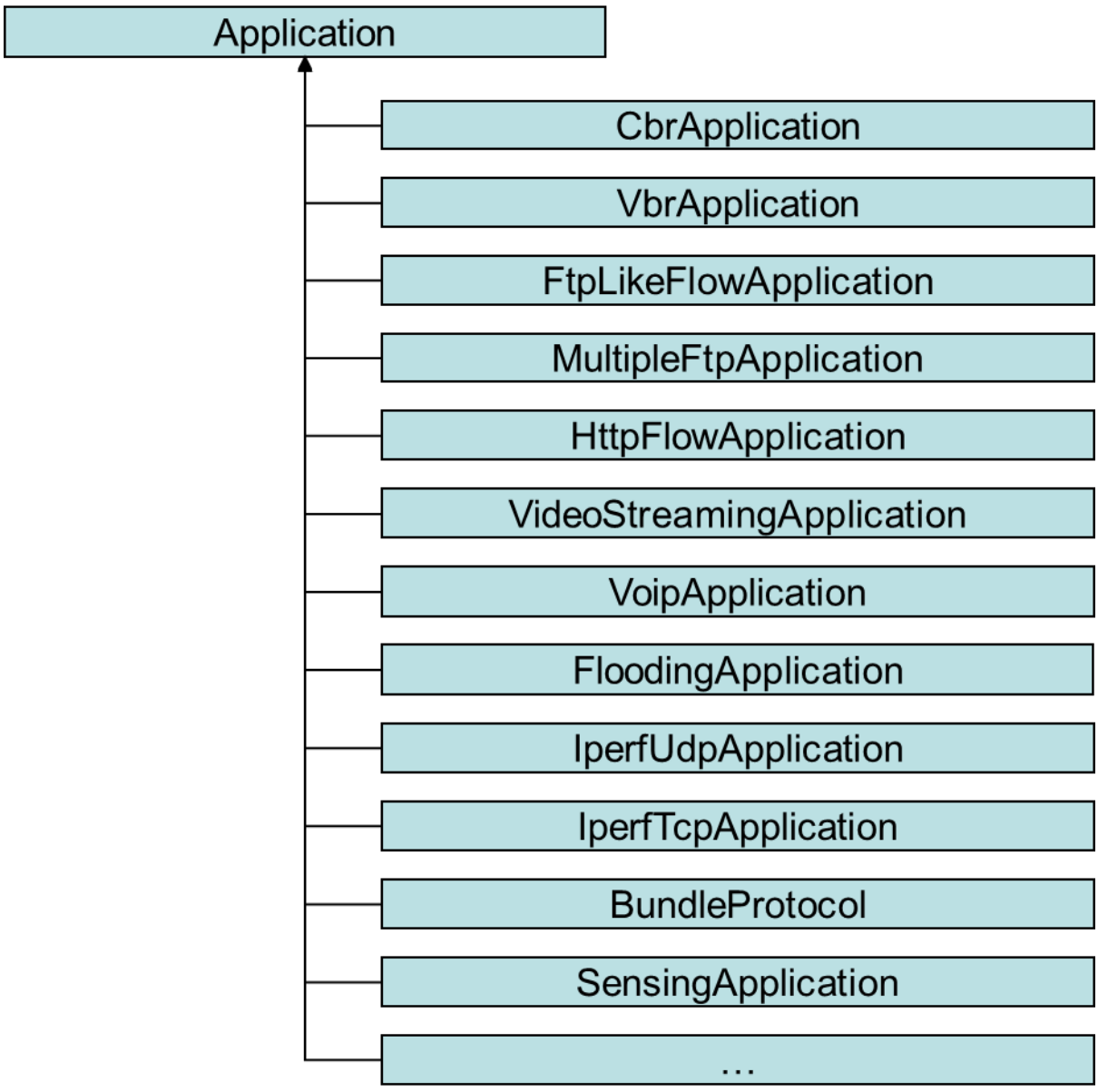

Figure 3-1**Error! Reference source not found.** shows the examples of the existing applications inheriting Application class such as CBR and VBR. Some application is separated into two applications for sender and receiver.

ApplicationLayer::AddApp() is utilized to add an application to a node after an application is instantiated. All existing applications are automatically read and added to each node when the constructor of NetworkNode class loads application settings in a configuration file (.config).

In each application, an event to send a packet at specified time is scheduled to send a packet. An application passes a packet to UDP or passes a data block to TCP by a pointer to transport layer for transmission. At receiver side, a packet handler with port number should be registered in a transport layer. The transport layer receives a packet from network layer and passes the packet to the application layer by the registered packet handler with target port number.

The following explains how to create and add original application.

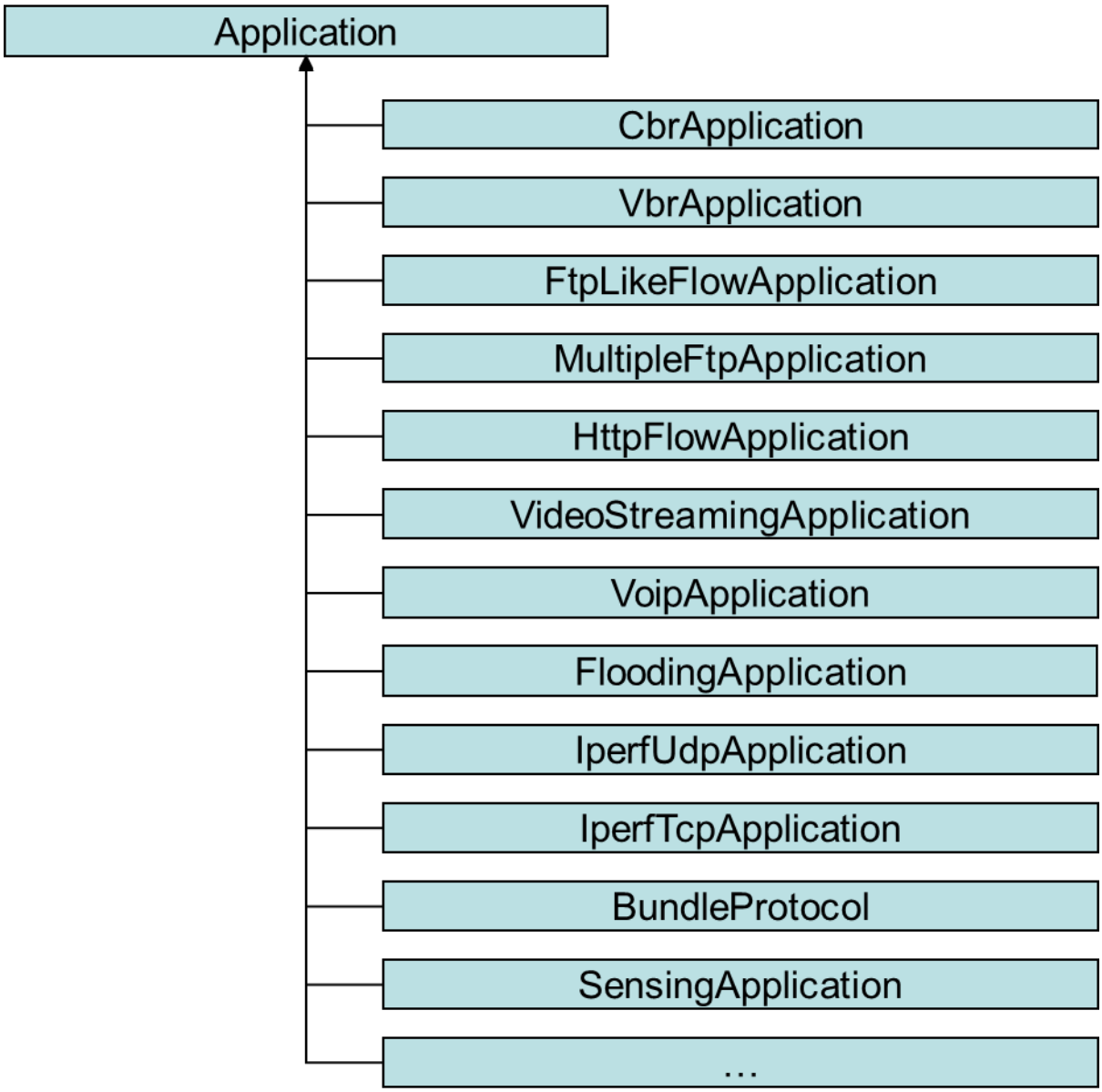

Figure 3-1 Class structure of application model.

3.2.2. Creating an application

There are the existing applications that use UDP or TCP as transport layer. This section describes an example to create an application with UDP. Two methods exist to define an application. One is to define two classes for sender and receiver. The other is to define one class for both sender and receiver. The former is suitable for unicast applications and the latter is

suitable for broadcast applications. In the existing applications, CBR application is a case of the former and Flooding application is a case of the latter.

The following example describes how to create a UDP application by referring the existing application, CBR application. CbrApplication class is a class inheriting Application class, and defines common types and loads parameters. CbrSourceApplication and CbrSinkApplication that inherit CbrApplication are utilized to send and receive a packet respectively.

3.2.2.1. Creating a base class for application

A base class for application is not always required. However, a base class can aggregate common procedures for sender and receiver classes such as reading parameters. The following items are required to implement in a base class.

- Reading parameters
- Definition of packet payload type

The following shows a code fragment of CbrApplication class. CbrApplication inherits Application class. CbrPayloadType as application payload is defined and parameters are read in CbrApplication class. Payload type should be defined and be shared between sender and receiver applications to communicate sender and receiver sides. In addition, you can create a bigger payload packet by padding regardless of predefined payload type size.

scensim_app_cbr.h

```
class CbrApplication: public Application {
...
    struct CbrPayloadType {
        unsigned int sequenceNumber;
        SimTime sendTime;

        CbrPayloadType(
            const unsigned int initSequenceNumber,
            const SimTime initSendTime)
            :
            sequenceNumber(initSequenceNumber),
            sendTime(initSendTime)
        {}
    };//CbrPayloadType//
...
```

```
CbrApplication::CbrApplication()
...
    cbrStartTime =
        initParameterDatabaseReader.ReadTime(
            parameterPrefix + "-start-time", sourceNodeId,
            theApplicationId);

    cbrEndTime =
        initParameterDatabaseReader.ReadTime(
            parameterPrefix + "-end-time", sourceNodeId, theApplicationId);

    cbrPriority = static_cast<PacketPriority>(
        parameterDatabaseReader.ReadNonNegativeInt(
            parameterPrefix + "-priority", sourceNodeId, theApplicationId));
...
```

3.2.2.2. Creating a class for sender

The following items are required to implement in a class for sender.

- Definition of a sender application class that inherits a base class for application
- Definition of a simulation event to send a packet
- Registration of simulation event
- Procedure of sending a packet

The following shows a code fragment of CbrSourceApplication. CbrSourceApplication inherits CbrApplication. CbrEvent as simulation event to send a packet is defined in CbrSoruceApplication class. When CbrEvent is executed, Cb r SourceApplication::SendPacket() is called.

scensim_app_cbr.h

```
class CbrSourceApplication:
     public CbrApplication,
     public enable_shared_from_this<CbrSourceApplication> {
...
    class CbrEvent: public SimulationEvent {
    public:
        explicit
        CbrEvent(const shared_ptr<CbrSourceApplication>&
            initCbrApplicationPtr)
            : cbrApplicationPtr(initCbrApplicationPtr) {}
        virtual void ExecuteEvent() { cbrApplicationPtr->SendPacket(); }

    private:
        shared_ptr<CbrSourceApplication> cbrApplicationPtr;

    };//CbrEvent//
...
```

The following example shows a registration of simulation event and a procedure of sending a packet in CbrSourceApplication. After the application is instantiated, CbrSourceApplication::CompleteInitialization() should be call to complete the initialization of the application. In the function, an initial simulation event to send a packet will be scheduled. In the case of CbrSourceApplication, CbrEvent with specified packet transmission start time is scheduled. A pointer to self is passed to create an event. If smart pointer (shared_ptr) is used, shared_from_this() is passed instead of regular pointer (this). It is required to inherit enable_shared_from_this<ClassName> to use shared_from_this() in application class definition.

When CbrEvent is executed, CbrSourceApplication::SendPacket is called. In SendPacket() function, a payload is filled with required information, and a packet is created with Packet::CreatePacket(). After destination is specified, UdpProtocol::SendPacket() is called to pass the packet to UDP. After the packet is transmitted, CbrEvent is scheduled to send next packet periodically.

scensim_app_cbr.h

```
void CbrSourceApplication::CompleteInitialization() {
. . .
    if (cbrStartTime < cbrEndTime) {
        simulationEngineInterfacePtr->ScheduleEvent(
            unique_ptr<SimulationEvent>(
                new CbrEvent(shared_from_this())),
                    cbrStartTime);
    }//if//
}//CompleteInitialization//
```

```
void CbrSourceApplication::SendPacket()
...
    CbrPayloadType cbrAppPayload(
        currentPacketSequenceNumber,
        simulationEngineInterfacePtr->CurrentTime());

    unique_ptr<Packet> packetPtr =
        Packet::CreatePacket(
            *simulationEngineInterfacePtr,
            cbrAppPayload,
            packetPayloadSizeBytes,
            useVirtualPayload);
...
    transportLayerPtr->udpPtr->SendPacket(
        packetPtr, sourceAddress, 0, destAddress,
        destinationPortId, cbrPriority);
...
    const SimTime nextPacketTime =
        simulationEngineInterfacePtr->CurrentTime() + packetInterval;

    if (nextPacketTime < cbrEndTime) {
        simulationEngineInterfacePtr->ScheduleEvent(
            unique_ptr<SimulationEvent>(
                new CbrEvent(shared_from_this())),
                nextPacketTime);
    }
...
}
```

3.2.2.3. Creating a class for receiver

The following items are required to implement in a class for receiver.

- Definition of a receiver application class that inherits a base class for application
- Definition of a packet handler
- Registration of packet handler

- Procedure of receiving a packet

The following shows a code fragment of CbrSinkApplication. CbrSinkApplication also inherits CbrApplication like CbrSourceApplication. A packet handler inhering UdpProtocol::PacketForAppFromTransportLayerHandler should be created for UDP applications. When a packet is coming in at application layer, the packet handler calls ReceivePacket() function. Therefore, a procedure of receiving a packet should be implemented in ReceivePacket() function in a class for receiver. In the case of CbrSinkApplication, CbrSinkApplication::ReceivePacket() is called.

scensim_app_cbr.h

```
class CbrSinkApplication:
    public CbrApplication,
    public enable_shared_from_this<CbrSinkApplication> {
...
    class PacketHandler:
        public UdpProtocol::PacketForAppFromTransportLayerHandler {
    public:
        PacketHandler(const shared_ptr<CbrSinkApplication>&
            initCbrSinkPtr) : cbrSinkPtr(initCbrSinkPtr) { }

        void ReceivePacket(
            unique_ptr<Packet>& packetPtr,
            const NetworkAddress& sourceAddress,
            const unsigned short int sourcePort,
            const NetworkAddress& destinationAddress,
            const PacketPriority& priority)
        {
            cbrSinkPtr->ReceivePacket(packetPtr);
        }

    private:
        shared_ptr<CbrSinkApplication> cbrSinkPtr;

    };//PacketHandler//
...
};
```

The following shows a registration of packet handler and a procedure of receiving a packet. A registration of packet handler is conducted in CbrSinkApplication::CompleteInitialization() as a part of initialization. A packet handler is created and is registered with receiver port number. This process enables application to receive a packet from transport layer. After receiving a packet from transport layer, the application executes receiving process. The following example shows that CbrSinkApplication::ReceivePacket() does receiving process. In this application, an

end-to-end delay is calculated at receiving packet. If sender adds more information in a packet, receiver can conduct further action according to the information in the packet. For example, receiver sends another packet to sender.

scensim_app_cbr.h

```
void CbrSinkApplication::CompleteInitialization() {
    packetHandlerPtr =
        shared_ptr<PacketHandler>(
            new PacketHandler(shared_from_this()));
   assert(transportLayerPtr->udpPtr->PortIsAvailable(destinationPortId));

    transportLayerPtr->udpPtr->OpenSpecificUdpPort(
        NetworkAddress::anyAddress,
        destinationPortId,
        packetHandlerPtr);
...
}

void CbrSinkApplication::ReceivePacket(unique_ptr<Packet>& packetPtr) {
    CbrPayloadType cbrPayload =
        packetPtr->GetAndReinterpretPayloadData<CbrPayloadType>();

    SimTime delay =
        simulationEngineInterfacePtr->CurrentTime() –
        cbrPayload.sendTime;
...
}
```

3.2.2.4. Creating a class for sender and receiver

If one class is utilized for sender and receiver, all required functions for sender and receiver described in the previous section should be implemented in one class. The following items are required to implement in a class.

- Definition of an application class that inherits a base class for application
- Definition of a simulation event to send a packet
- Definition of a packet handler
- Registration of simulation event
- Registration of packet handler
- Procedure of sending a packet
- Procedure of receiving a packet

Only one application per node needs to be initialized because a common port number for receiving packets is predefined. This method is suitable for broadcast application. When multiple applications are applied to a node, a port number for receiving per application should be unique. This style is adopted in FloodingApplication and some other applications of Scenargie.

3.2.3. Adding an application to a node

There are two methods to add an application to a node. One is to add an application to target sender node and target receiver node like CBR application. The other one is to add an application to all nodes regardless of whether the node is sender or receiver. In both cases, an application is instantiated based on the parameters in a configuration file. After that, ApplicationLayer::AddApp() is utilized to add an application to application layer. The followings describe each method.

3.2.3.1. Adding an application (1)

ApplicationMaker class is utilized to add applications. ApplicationMaker class is automatically instantiated in the constructor of NetworkNode class. ApplicationMaker reads a configuration file and add applications to nodes.

The following four items are required to implement in ApplicationMaker class to add an original application.

- Definition of application type
- Setting of parameter to identify an application
- Definition and implementation of parser for application
- Call of parser

The following shows an example of definition of original application type. Newly added sample code in ApplicationMaker class is written in blue.

scensim_application.h

```
class ApplicationMaker {
...
    enum ApplicationType {
        APPLICATION_CBR,
...
        APPLICATION_MY_APP

        //Add new app
    };
...
}
```

The following shows an example of setting a parameter to identify the original application. Newly added sample code in ApplicationMaker class is written in blue.

scensim_application.cpp

```
ApplicationMaker::ApplicationMaker()
{
...
    appSpecificParameterNames[APPLICATION_CBR] = "cbr-destination";
...
    appSpecificParameterNames[APPLICATION_MY_APP] =
        "my-app-destination";

    //add new app
    //Add application specification paramter for user application
    // e.g. appSpecificParameterNames[APPLICATION_USERAPP] =
      "userapp-destination";
...
}
```

The following shows an example of definition and implementation of a parser for the original application. Newly added sample code in ApplicationMaker class is written in blue. When source node ID is equal to self-node ID, MySourceApplication is instantiated and is added to an application layer.

scensim_application.h

```
class ApplicationMaker {
...
    void ReadCbrFromConfig(
        const ParameterDatabaseReader& theParameterDatabaseReader,
        const ApplicationInstanceInfo& applicationInstanceId);
...
    void ReadMyAppFromConfig(
        const ParameterDatabaseReader& theParameterDatabaseReader,
        const ApplicationInstanceInfo& applicationInstanceId);

    //Add new app
...
}
```

scensim_application.cpp

```
void ApplicationMaker::ReadMyAppFromConfig(
    const ParameterDatabaseReader& theParameterDatabaseReader,
    const ApplicationInstanceInfo& applicationInstanceId)
{
    const NodeId& sourceNodeId =
        applicationInstanceId. nodeIdWithParameter;
    const InterfaceOrInstanceId& instanceId =
        applicationInstanceId.instanceId;
    const NodeId destinationNodeIdOrAnyNodeId =
        App_ConvertStringToNodeIdOrAnyNodeId(
            theParameterDatabaseReader.ReadString(
                "my-app-destination", sourceNodeId, instanceId));

    const unsigned short defaultDestinationPortId =
        applicationInstanceId.GetDefaultDestinationPortNumber();

    if (sourceNodeId == nodeId) {
        shared_ptr<MySourceApplication> appPtr(
            new MySourceApplication (
                theParameterDatabaseReader,
                simulationEngineInterfacePtr,
                instanceId,
                sourceNodeId,
                destinationNodeIdOrAnyNodeId,
                defaultDestinationPortId));

        appLayerPtr->AddApp(appPtr);
        appPtr->CompleteInitialization();
    }
...
}
```

The following shows an example of call of a parser for the original application. Newly added sample code in ApplicationMaker class is written in blue.

scensim_application.cpp

```
void ApplicationMaker:: ReadApplicationIntancesFromConfig(
{
...
    switch(application) {
    case APPLICATION_CBR:
        (*this).ReadCbrFromConfig(
            theParameterDatabaseReader, applicationInstanceId);
     break;
...
    case APPLICATION_MY_APP:
        (*this).ReadMyAppFromConfig(
            theParameterDatabaseReader, applicationInstanceId);
        break;
    default:
        assert(false && "Implement application parser!");
    }//switch//
...
}//ReadApplicationLineFromConfig//
```

3.2.3.2. Adding an application (2)

The method in described in the previous section adds an application to only sender or receiver nodes according to configuration file. This is suitable for unicast applications which sender and receiver nodes are limited. On the other hand, when all nodes have common application, another method to add applications can be utilized as follows.

The following example shows that an application is added to all nodes. In a constructor of node (for example, definition of node class written in sim.cpp), an application is instantiated and is added to an application layer. The parameters for the application can be retrieved through ParameterDatabaseReader in the application class. Note that CompleteInitialization() function for registration of handler should be called after calling ApplicationLayer::AddApp() because ApplicationLayer::AddApp() sets a pointer of transport layer.

```
SimNode::SimNode()
{
...
   shared_ptr<MyApp> myAppPtr(
       new MyApp (
           simulationEngineInterfacePtr,
           theParameterDatabaseReader,
           ・・・));
    (*this).GetAppLayerPtr()->AddApp(myAppPtr);
    myAppPtr->CompleteInitialization();
...
```

3.3. Transport layer

This section describes transport layer. Scenargie provides TCP and UDP as a transport layer protocol. Transport layer is instantiated in the constructor of BasicNetworkLayer class and the pointers of transport layer are passed to Application class. Each application can send a packet or data block with a pointer to UDP or TCP. The TCP implementation is ported from BSD9.

The following shows the definition of transport layer.

scensim_transport.h

```
class TransportLayer {
public:
    TransportLayer(
        const ParameterDatabaseReader& theParameterDatabaseReader,
        const shared_ptr<SimulationEngineInterface>&
            simulationEngineInterfacePtr,
        const shared_ptr<NetworkLayer>& networkLayerPtr,
        const NodeId& theNodeId,
        const RandomNumberGeneratorSeed& nodeSeed);

    shared_ptr<UdpProtocol> udpPtr;
    shared_ptr<TcpProtocol> tcpPtr;

    shared_ptr<NetworkLayer> GetNetworkLayerPtr() const {
        return networkLayerPtr; }

    void DisconnectProtocolsFromOtherLayers()
    {
        udpPtr->DisconnectFromOtherLayers();
        tcpPtr->DisconnectFromOtherLayers();
        networkLayerPtr.reset();
    }

private:
    shared_ptr<NetworkLayer> networkLayerPtr;

};//TransportLayer//
```

TCP and UDP are implemented by inheriting ProtocolPacketHandler class to establish an interface to network layer (Figure 3-2). If you want to add an original transport protocol, a class inheriting ProtocolPacketHandler should be defined and registered to network layer by BasicNetworkLayer::RegisterPacketHandlerForProtocol() with protocol number.

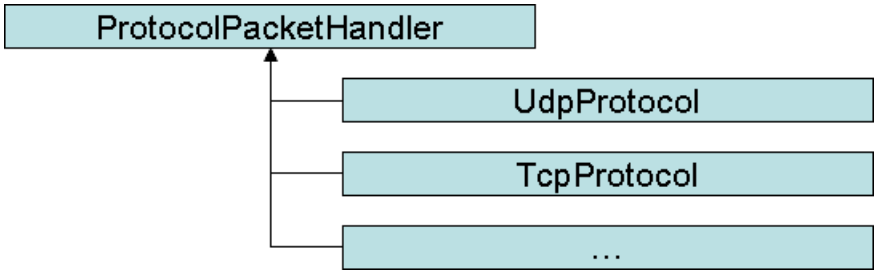

Figure 3-2 Class structure of transport layer.

scensim_network.h

```
class ProtocolPacketHandler {
public:
    virtual ~ProtocolPacketHandler() { }

    virtual void DisconnectFromOtherLayers() { }

    virtual void ReceivePacketFromNetworkLayer(
        unique_ptr<Packet>& packetPtr,
        const NetworkAddress& sourceAddress,
        const NetworkAddress& destinationAddress,
        const PacketPriority trafficClass,
        const NetworkAddress& lastHopAddress,
        const unsigned char hopLimit,
        const unsigned int interfaceIndex) = 0;

    virtual void GetPortNumbersFromPacket(
        const Packet& aPacket,
        const unsigned int transportHeaderOffset,
        bool& portNumbersWereRetrieved,
        unsigned short int& sourcePort,
        unsigned short int& destinationPort) const = 0;

};
```

scensim_transport.cpp

```
void UdpProtocol::ConnectToNetworkLayer(
    const shared_ptr<NetworkLayer>& initNetworkLayerPtr)
{
    this->networkLayerPtr = initNetworkLayerPtr;

    networkLayerPtr->RegisterPacketHandlerForProtocol(
        IP_PROTOCOL_NUMBER_UDP, shared_from_this());
}
```

bsd9tcpglue.cpp

```
void TcpProtocolImplementation::ConnectToNetworkLayer(
    const shared_ptr<NetworkLayer>& initNetworkLayerPtr)
{
    assert(networkLayerPtr == nullptr);

    networkLayerPtr = initNetworkLayerPtr;
    networkLayerPtr->RegisterPacketHandlerForProtocol(
        IP_PROTOCOL_NUMBER_TCP, tcpProtocolPtr->shared_from_this());

}//ConnectToNetworkLayer//
```

3.4. Network layer

This section describes network layer. Scenargie supports IP (Internet Protocol) as network layer protocol. Internet protocol is created by inheriting Network Layer class that is an abstract class. When you want to add an original network layer, a class inheriting NetworkLayer class should be defined like BasicNetworkLayer (Figure 3-3).

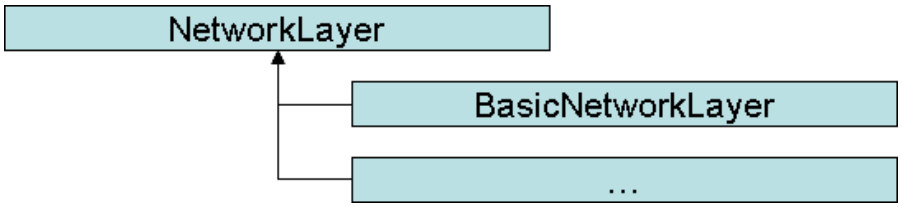

Figure 3-3 Class structure of network layer.

As NetworkLayer class is a pure abstract class, all defined functions must be implemented. In particular, NetworkLayer::ReceivePacketFromUpperLayer () for receiving a packet from upper layer and NetworkLayer::ReceivePacketFromMac() for receiving a packet from MAC layer are important functions in network layer. MacLayer::NetworkLayerQueueChangeNotification() is utilized to notify queue change to MAC layer after inserting a packet to transmission queue. ProtocolPacketHandler::ReceivePacketFromNetworkLayer() with preregistered handler is utilized to pass a packet to an upper layer.

The following shows a code fragment of BasicNetworkLayer class.

scensim_network.h/cpp

```
class BasicNetworkLayer:
    public NetworkLayer,
    public enable_shared_from_this<BasicNetworkLayer> {
...
```

```
void BasicNetworkLayer::ReceivePacketFromUpperLayer(
    unique_ptr<Packet>& packetPtr,
    const NetworkAddress& initialSourceAddress,
    const NetworkAddress& destinationAddress,
    PacketPriority trafficClass,
    const unsigned char protocol)
{
...
    IpHeaderModel
        header(
            trafficClass,
            packetPtr->LengthBytes(),
            hopLimit,
            protocol,
            sourceAddress,
            destinationAddress);

    packetPtr->AddRawHeader(
        header.GetPointerToRawBytes(), header.GetNumberOfRawBytes());
    packetPtr->AddTrailingPadding(header.GetNumberOfTrailingBytes());

    (*this).InsertPacketIntoAnOutputQueue(
        packetPtr, interfaceIndex, nextHopAddress, trafficClass);

}
```

```
void BasicNetworkLayer::InsertPacketIntoAnOutputQueue(
    unique_ptr<Packet>& packetPtr,
    const unsigned int interfaceIndex,
    const NetworkAddress& nextHopAddress,
    const PacketPriority initialTrafficClass,
    const EtherTypeFieldId etherType)
{
...
    outputQueue.Insert(
        packetPtr, nextHopAddress, trafficClass,
        enqueueResult, packetToDropPtr, etherType);
...
    interface.macLayerPtr->NetworkLayerQueueChangeNotification();
}
```

```
void BasicNetworkLayer::ReceivePacketFromMac(
    const unsigned int interfaceIndex,
    unique_ptr<Packet>& packetPtr,
    const NetworkAddress& lastHopAddress,
    const EtherTypeFieldId etherType)
{
...
       map<unsigned  char,  shared_ptr<ProtocolPacketHandler>  >::iterator
mapIter =
           protocolPacketHandlerMap.find(protocolNum);

       if (mapIter != protocolPacketHandlerMap.end()) {
           mapIter->second->ReceivePacketFromNetworkLayer(
               packetPtr,
               sourceAddress,
               destinationAddress,
               trafficClass,
               lastHopAddress,
               currentHopLimit,
               interfaceIndex);
       }
...
}
```

3.5. MAC/PHY layer

This section describes MAC layer. Scenargie does not provide any abstract class for PHY layer, but does provide an abstract class for MAC layer. Therefore, MAC layer includes PHY layer in terms of class structure. Scenargie provides the APIs for signal transmission and receiving in a wireless environment to simulate wireless systems.

The following shows the overview of the existing MAC layer. The following is a code fragment of the definition of MacLayer class. MacLayer class includes several pure virtual functions such as NetworkLayerQueueChangeNotification() to notify queue change. Figure 3-4**Error! Reference source not found.** shows a class structure of MAC layer.

scensim_mac.h

```
class MacLayer {
public:
    virtual ~MacLayer() { }
    // Network Layer Interface:
    virtual void NetworkLayerQueueChangeNotification() = 0;
    virtual void DisconnectFromOtherLayers() = 0;
    virtual GenericMacAddress GetGenericMacAddress() const
        { assert(false); abort(); return GenericMacAddress(); }
...
};//MacLayer//
```

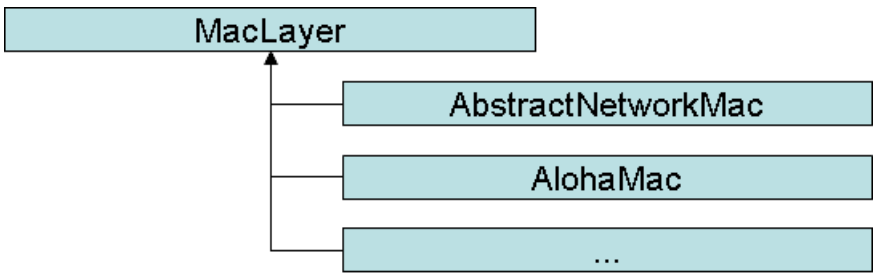

Figure 3-4 Class structure of MAC layer.

It is required to implement an original class inheriting MacLayer to create an original MAC layer. NetworkLayerQueueChangeNotification() function is the interface from network layer to MAC layer. As a pointer to network layer is passed to a MAC layer, the pointer is utilized as an interface from Mac layer to network layer. In BasicNetworkLayer class, ReceivePacketFromMac() function is utilized to pass a packet from MAC layer to network layer. Other than those above, it is necessary to pass a pointer of MAC layer to a network layer and to register a transmission queue to network layer.

The following example shows a code fragment of AlohaMac class. Aloha MAC is one of multi access system for wireless network such as unslotted aloha and slotted aloha. The following code shows that a class inheriting MacLayer class is defined, and transmission queue is created and is registered to network layer in the constructor of the class. This process creates a transmission queue per network interface and binds network interface and MAC layer. When network layer enqueues a packet to the transmission queue, NetworkLayerQueueChangeNotification() is called to notify queue change from network layer to MAC layer. MAC layer starts a process to send a packet after receiving the notification from network layer. Though the process varies based on system model, a packet will be retrieved from transmission queue and will be passed to PHY layer. ReceivePacketFromMac() function in network layer is utilized to pass a packet from MAC layer to network layer.

aloha_mac.h

```
class AlohaMac :
    public MacLayer, public enable_shared_from_this<AlohaMac> {
...
    //Notification from network layer
    void NetworkLayerQueueChangeNotification();
...
};

void AlohaMac::NetworkLayerQueueChangeNotification()
{
    if (macState == IDLE_STATE) {
        (*this).TransmitNextDataFrameIfNecessary();
    }//if//
}//NetworkLayerQueueChangeNotification//
```

```
AlohaMac::AlohaMac(
...
    //Creating transmittion queue
        networkOutputQueuePtr(
            new FifoInterfaceOutputQueue(
                theParameterDatabaseReader,
                initInterfaceId,
                simulationEngineInterfacePtr)),
...
{
    //Registration of transmission queue
    networkLayerPtr->SetInterfaceOutputQueue(
        interfaceIndex, networkOutputQueuePtr);
...
}
```

```
void AlohaMac::RetrievePacketFromNetworkLayer(bool& wasRetrieved)
{
...
    //Dequeuing a packet from transmission queue
        networkOutputQueuePtr->DequeuePacket(
            (*this).currentDataPacketPtr,
            nextHopAddress,
            notUsed1, notUsed2);
...
}
```

```
void AlohaMac::ProcessReceivedDataFrame(const Packet& aFrame)
{
...
    //Sending a packet to network layer
    networkLayerPtr->ReceivePacketFromMac(
        interfaceIndex, dataPacketPtr, lastHopAddress);
...
}
```

Other MAC layer models such as Dot11Mac in Dot Eleven Module have same interfaces to network layer as AlohaMac.

3.6. Radio propagation

This section describes pathloss model and antenna model as radio propagation model.

3.6.1. Pathloss model

Figure 3-5**Error! Reference source not found.** shows class structure of pathloss model.

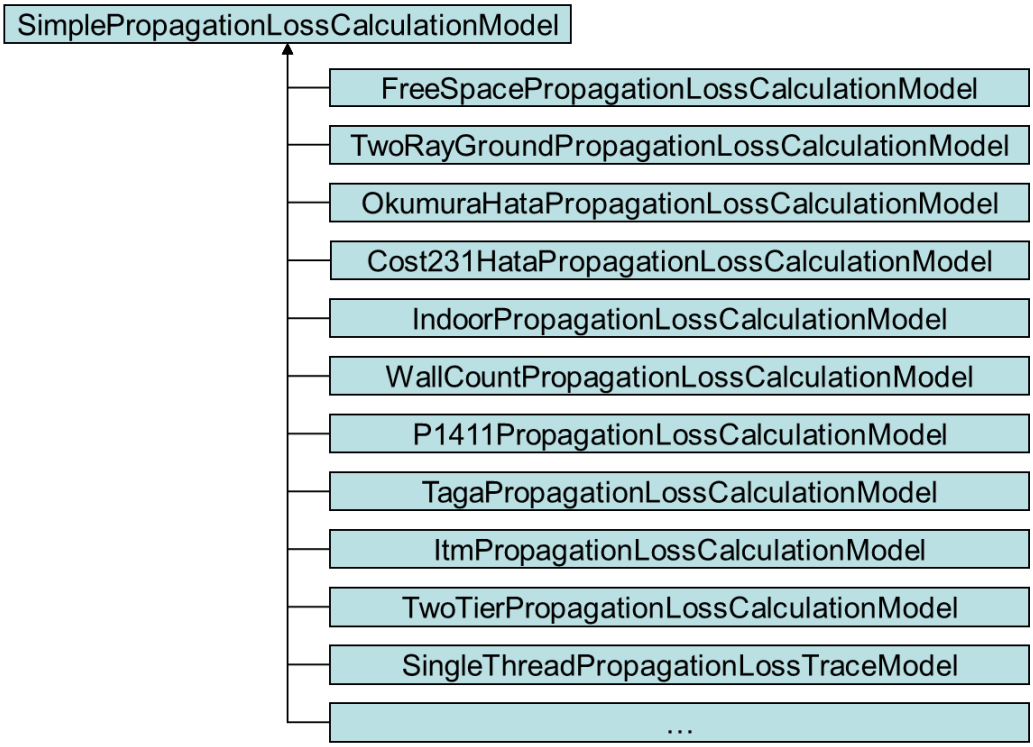

Figure 3-5 Class structure of pathloss model.

Every pathloss model must inherit SimplePropagationLossCalculationModel class. The following examples show a code fragment of SimplePropagationLossCalculationModel class and FreeSpacePropagationLossCalculationModel class.

scensim_proploss.h

```
class SimplePropagationLossCalculationModel {
...
    virtual double CalculatePropagationLossDb(
        const ObjectMobilityPosition& txAntennaPosition,
        const ObjectMobilityPosition& rxAntennaPosition,
        const double& xyDistanceSquaredMeters) const = 0;
...
};
```

```
class FreeSpacePropagationLossCalculationModel:
    public SimplePropagationLossCalculationModel {
...
    double CalculatePropagationLossDb(
        const ObjectMobilityPosition& txPosition,
        const ObjectMobilityPosition& rxPosition,
        const double& xyDistanceSquaredMeters) const override;
...
};

double
FreeSpacePropagationLossCalculationModel::CalculatePropagationLossDb(
    const ObjectMobilityPosition& txPosition,
    const ObjectMobilityPosition& rxPosition,
    const double& xyDistanceSquaredMeters) const
{
...
}
```

It is required to inherit SimplePropagationLossCalculationModel class and implement a virtual function, SimplePropagationLossCalculationModel::CalculatePropagationLossDb(), to add an original pathloss model. Then, it is required to add statements to use the original pathloss model in CreatePropagationLossCalculationModel() function as follows.

scensim_prop.cpp

```
shared_ptr<SimplePropagationLossCalculationModel>
CreatePropagationLossCalculationModel(
    const shared_ptr<SimulationEngine>& simulationEnginePtr,
    const ParameterDatabaseReader& theParameterDatabaseReader,
    const shared_ptr<GisSubsystem>& gisSubsystemPtr,
    const string& propModelName,
    const double& carrierFrequencyMhz,
    const double& maximumPropagationDistanceMeters,
    const bool propagationDelayIsEnabled,
    const unsigned int numberThreadsForDataParallelPropCalculation,
    const InterfaceOrInstanceId& instanceId,
    const RandomNumberGeneratorSeed& runSeed)
{
...
    else if (propModelName == "freespace") {

        return shared_ptr<SimplePropagationLossCalculationModel>(
            new FreeSpacePropagationLossCalculationModel(
                carrierFrequencyMhz,
                maximumPropagationDistanceMeters,
                propagationDelayIsEnabled,
                numberThreadsForDataParallelPropCalculation));

    }
    else if (propModelName == "mypathlossmodel") {
        return shared_ptr<SimplePropagationLossCalculationModel>(
            new MyPathlossModel (
                carrierFrequencyMhz,
                maximumPropagationDistanceMeters,
                propagationDelayIsEnabled,
                numberThreadsForDataParallelPropCalculation));
...
```

3.6.2. Antenna model

Figure 3-6**Error! Reference source not found.** shows class structure of antenna model.

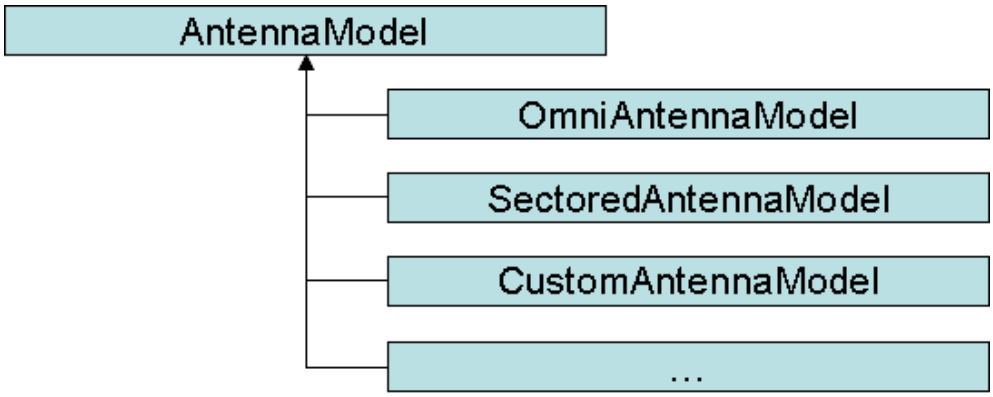

Figure 3-6 Class structure of antenna model.

Every antenna model must inherit AntennaModel class. The following examples show a code fragment of AntennaModel class and SectoredAntennaModel class.

scensim_proploss.h

```
class AntennaModel {
public:
    AntennaModel() { }
    virtual ~AntennaModel() { }

    virtual bool IsOmniDirectional() const = 0;

    virtual double GetOmniGainDbi() const = 0;

    virtual double GainInDbForThisDirection(
        const double& azimuthFromBoresightClockwiseDegrees = 0.0,
        const double& elevationFromBoresightDegrees = 0.0,
        const double& currentAntennaRotation = 0.0) const = 0;
...
};
```

```
class SectoredAntennaModel: public AntennaModel {
...
    virtual double GainInDbForThisDirection(
        const double& azimuthFromBoresightDegrees,
        const double& elevationFromBoresightDegrees,
        const double& currentAntennaRotation = 0.0) const override
    {
...
}
```

It is required to inherit AntennaModel class and implement three virtual functions to add an original pathloss model. Then, it is required to define antenna model name and add statements to use the original antenna model as follows.

scenargiesim.cpp

```
shared_ptr<AntennaModel> CreateAntennaModel(
    const ParameterDatabaseReader& theParameterDatabaseReader,
    const NodeId& theNodeId,
    const InterfaceOrInstanceId& theInterfaceId,
    const AntennaPatternDatabase& anAntennaPatternDatabase)
{
...
    else if (antennaModelString == "sectored") {
        const double antennaGainDbi =
           theParameterDatabaseReader.ReadDouble("max-antenna-gain-dbi",
theNodeId, theInterfaceId);

        return                                  (shared_ptr<AntennaModel>(new
SectoredAntennaModel(antennaGainDbi)));
    }
    else if (antennaModelString == "myantennamodel") {
        return (shared_ptr<AntennaModel>(
            new MyAntennaModel()));
    }
...
}
```

Original antenna patterns in a file can be utilized as custom antenna model. Please refer “Scenargie Base Simulator User Guide” for details.

3.7. Mobility model

This section describes mobility model. Figure 3-7 shows class structure of mobility model.

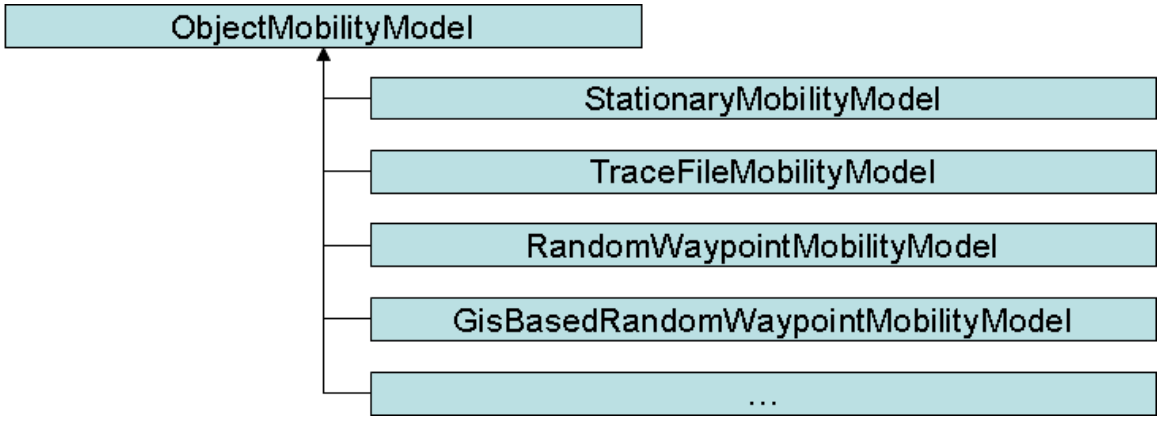

Figure 3-7 Class structure of mobility model.

Every mobility model must inherit ObjectMobilityModel class. The following example shows a code fragment of ObjectMobilityModel class and TraceFileMobilityModel class.

scensim_proploss.h

```
class ObjectMobilityModel {
...
    virtual void GetUnadjustedPositionForTime(
        const SimTime& snapshotTime,
        ObjectMobilityPosition& position) = 0;
...
};
```

```
class TraceFileMobilityModel: public ObjectMobilityModel {
...
    virtual void GetUnadjustedPositionForTime(
        const SimTime& snapshotTime,
        ObjectMobilityPosition& position);
...
}
...
void TraceFileMobilityModel::GetUnadjustedPositionForTime(
    const SimTime& snapshotTime,
    ObjectMobilityPosition& position)
{
...
}
```

It is required to inherit ObjectMobilityModel class and implement a virtual function, ObjectMobilityModel::GetUnadjustedPositionForTime() to add an original mobility model. Then, it is required to define mobility model name and add statements to use the original mobility model in CreateAntennaMobilityModel fucntion as follows.

scensim_mobility.cpp

```
shared_ptr<ObjectMobilityModel> CreateAntennaMobilityModel(
    const ParameterDatabaseReader& theParameterDatabaseReader,
    const NodeId& theNodeId,
    const InterfaceOrInstanceId& theInterfaceId,
    const RandomNumberGeneratorSeed& mobilitySeed,
    InorderFileCache& mobilityFileCache,
    const shared_ptr<GisSubsystem>& theGisSubsystemPtr)
{
...
    if (mobilityModelString == "trace-file") {
...
    }
    else if (mobilityModelString == "mymobilitymodel ") {

        return shared_ptr<ObjectMobilityModel>(
            new MyMobilityModel(
                theParameterDatabaseReader,
                nodeId,
                interfaceId,
                ...
            );
    }
...
```

3.8. GIS data access

This section describes GIS data access. Scenargie supports Shape format by ESRI as GIS data and handles Shape file data in simulations. For example, GisBasedRandomWaypointMobilityModel enables communication nodes to move based on road information, and IndoorPropagationLossCalculationModel and P1411PropagationLossCalculationModel calculate pathloss by using building information. GisSubsystem loads GIS data and provides the APIs to access GIS data. GisSubsystem is instantiated in the constructor of NetworkSimulator class. Each model can access GIS data through a pointer to GisSubsystem.

The following example shows a code fragment of NetworkSimulator.

scensim_netsim.h

```
NetworkSimulator::NetworkSimulator(
    const shared_ptr<ParameterDatabaseReader>&
        initParameterDatabaseReaderPtr,
    const shared_ptr<SimulationEngine>& initSimulationEnginePtr,
    const RandomNumberGeneratorSeed& initRunSeed,
    const bool initRunSequentially)
    :
    theSimulationEnginePtr(initSimulationEnginePtr),
    runSeed(initRunSeed),
    mobilitySeed(initRunSeed),
    theParameterDatabaseReaderPtr(initParameterDatabaseReaderPtr),
    theGisSubsystemPtr(
       new GisSubsystem(
       *theParameterDatabaseReaderPtr, initSimulationEnginePtr)),
    timeStepEventSynchronizationStep(INFINITE_TIME),
    runSequentially(initRunSequentially),
    nextSynchronizationTimeStep(0)
{
```

scensim_mobility.cpp

```
shared_ptr<ObjectMobilityModel> CreateAntennaMobilityModel(
    const ParameterDatabaseReader& theParameterDatabaseReader,
    const NodeId& theNodeId,
    const InterfaceOrInstanceId& theInterfaceId,
    const RandomNumberGeneratorSeed& runSeed,
    InorderFileCache& mobilityFileCache,
    const shared_ptr<GisSubsystem>& theGisSubsystemPtr)
{
...
    else if (mobilityModelString == "gis-based-random-waypoint") {

        assert(theGisSubsystemPtr != nullptr);

        return shared_ptr<ObjectMobilityModel>(
            new GisBasedRandomWaypointMobilityModel(
                theParameterDatabaseReader,
                mobilityObjectId,
                theNodeId,
                theInterfaceId,
                runSeed,
                mobilityFileCache,
                mobilityGranularityMeters,
                theGisSubsystemPtr));

    }
...
}
```

4. API lists

This chapter describes the APIs by category to customize simulation models.

4.1. Simulation engine related APIs

Source file: scensim_engine.h

4.1.1. SimulationEvent

Abstract class for simulation event

Return type	Function(argument)	Description
virtual void	**ExecuteEvent** ()=0	Executes simulation event. (pure virtual function)

4.1.2. EventRescheduleTicket

Simulation event ticket class

Return type	Function(argument)	Description
void	**Clear** ()	Invalidates simulation event ticket.
bool	**IsNull** () const	Checks that simulation event is invalid.

4.1.3. SimulationEngineInterface

Simulation engine interface class

Return type	Function(argument)	Description
void	**ShutdownThisInterface** ()	Shuts down this simulation engine interface from simulation engine.
SimTime	**CurrentTime** () const	Gets current time.
NodeId	**GetNodeId** () const	Gets Node ID.
void	**ScheduleEvent** (const shared_ptr< SimulationEvent > &eventPtr, const SimTime &eventTime, EventRescheduleTicket &eventTicket)	Schedules a reschedulable simulation event. (For smart pointer)
void	**ScheduleEvent** (const shared_ptr< SimulationEvent > &eventPtr, const SimTime &eventTime)	Schedules a simulation event. (For smart pointer)
void	**ScheduleEvent** (unique_ptr< SimulationEvent > &eventPtr, const SimTime &eventTime)	Schedules a simulation event. (For unique pointer)
void	**CancelEvent** (EventRescheduleTicket &eventTicket)	Cancels a simulation event.
void	**RescheduleEvent** (EventRescheduleTicket &eventTicket, const SimTime &eventTime)	Reschedules a simulation event.
void	**ScheduleExternalEventAtNode** (const NodeId &destinationNodeId, unique_ptr< SimulationEvent > &eventPtr, const SimTime &eventTime)	Schedules an external simulation event at target node.

void	**ScheduleExternalEventAtPartition** (const unsigned int destinationPartitionIndex, unique_ptr< SimulationEvent > &eventPtr, const SimTime &eventTime)	Schedules an external simulation event at target partition.
unsigned long long int	**GenerateAndReturnNewLocalSequenceNumber** ()	Generates and returns new local (node level) sequence number.
bool	**TraceIsOn** (const TraceTag traceTag) const	Checks that trace output for the specified trace tag is on.
void	**OutputTrace** (const string &modelName, const string &modelInstanceId, const string &eventName, const string &stringToOutput) const	Outputs trace in text mode.
bool	**BinaryOutputIsOn** () const	Checks that trace output in binary mode is activated.
template<typename T > void	**OutputTraceInBinary** (const string &modelName, const string &modelInstanceId, const string &eventName, const T &data) const	Outputs trace in binary mode. (With event specific information)
void	**OutputTraceInBinary** (const string &modelName, const string &modelInstanceId, const string &eventName) const	Output trace in binary mode. (Without event specific information)
bool	**ParallelismIsOn** () const	Checks that parallel calculation mode is activated.
shared_ptr< CounterStatistic >	**CreateCounterStat** (const string &statName, const bool useBigCounter=false)	Creates counter type statistic.
shared_ptr< RealStatistic >	**CreateRealStat** (const string &statName, const bool useBigReal=false)	Create real type statistic.

shared_ptr< RealStatistic >	**CreateRealStatWithDbConversion** (const string &statName, const bool useBigReal=false)	Creates real type statistic. (Records liner value and converts it to dB)

4.1.4. SimulationEngine

Simulation engine class

Return type	Function(argument)	Description
	SimulationEngine (const ParameterDatabaseReader &theParameterDatabaseReader, const bool initIsRunningSimulationSequentially =true, const unsigned int numberPartitionThreads=1)	Constructor of SimulationEngine class.
shared_ptr < SimulationEngine Interface >	**GetSimulationEngineInterface** (const ParameterDatabaseReader &theParameterDatabaseReader, const NodeId &nodeId, const size_t startingPartitionIndex=0)	Creates and returns simulation engine interface for target node.
void	**RunSimulationSequentially** (const SimTime &runUntilTime)	Runs simulation in single thread.
void	**RunSimulationInParallel** (const SimTime &stopTime)	Runs simulation in parallel mode. (For parallel simulation mode)
void	**PauseSimulation** ()	Pauses simulation.
void	**ClearAnyImpendingPauseSimulation** ()	Restarts simulation.
unsigned int	**GetNumberPartitionThreads** () const	Gets number of partition threads.
SimTime	**CurrentTime** () const	Gets current time.
bool	**SimulationIsPaused** () const	Checks that simulation is paused.
bool	**SimulationIsDone** () const	Checks that simulation is completed.
void	**ShutdownSimulator** ()	Shuts down simulation.
void	**EnableTraceAtANode** (const NodeId &nodeId, const TraceTag traceTag)	Enables trace output at target node.
void	**DisableTraceAtANode** (const NodeId &nodeId, const	Disables trace output at target node.

	TraceTag traceTag)	
RuntimeStatistics System &	**GetRuntimeStatisticsSystem** ()	Gets runtime statistics subsystem.
TraceSubsystem &	**GetTraceSubsystem** ()	Gets trace subsystem.
void	**SetTraceSubsystem** (const shared_ptr< TraceSubsystem > &newTraceSubsystemPtr)	Sets trace subsystem.

4.2. Packet related APIs

Source file: scensim_packet.h

4.2.1. Packet

Packet class

Return type	Function(argument)	Description
	Packet (const Packet &right)	Constructor of Packet class.
const PacketId	**GetPacketId** () const	Gets Packet ID.
void	**SetPacketId** (const PacketId &newPacketId)	Sets Packet ID.
void	**AddRawHeader** (const unsigned char rawHeader[], const unsigned int sizeBytes)	Adds raw header.
template<typename T > void	**AddPlainStructHeader** (const T &header)	Adds a structured header.
template<typename T > void	**AddPlainStructHeaderWithTrailingAlignmentBytes** (const T &header, const unsigned int numTrailingAlignmentBytes)	Adds a structured header with alignment bytes.
void	**AddTrailingPadding** (const unsigned int paddingLengthBytes)	Adds trailing padding.
void	**RemoveTrailingPadding** (const unsigned int paddingLengthBytes)	Removes trailing padding.
void	**DeleteHeader** (const unsigned int bytesToDelete)	Deletes a header by specified bytes.
unsigned int	**LengthBytes** () const	Gets packet length in bytes.
unsigned int	**ActualLengthBytes** () const	Gets actual packet length (packet length excluding virtual payload size) in bytes.
const unsigned	**GetRawPayloadData** () const	Gets raw payload data.

char *		
unsigned char *	**GetRawPayloadData** ()	Gets raw payload data.
const unsigned char *	**GetRawPayloadData** (const unsigned int byteOffset, const unsigned int length) const	Gets raw payload data with specified offset.
unsigned char *	**GetRawPayloadData** (const unsigned int byteOffset, const unsigned int length)	Gets raw payload data with specified offset.
template<typename T > const T &	**GetAndReinterpretPayloadData** (const int byteOffset=0) const	Gets structured payload data.
template<typename T > T &	**GetAndReinterpretPayloadData** (const int byteOffset=0)	Gets structured payload data.
void	**AddExtrinsicPacketInformation** (const ExtrinsicPacketInfoId &extrinsicPacketInfoId, const shared_ptr< ExtrinsicPacketInformation > &infoPtr)	Adds extrinsic packet information.
template<typename T > T &	**GetExtrinsicPacketInformation** (const ExtrinsicPacketInfoId &extrinsicInfoId) const	Gets extrinsic packet information.
bool	**CheckExtrinsicPacketInformation Exist** (const ExtrinsicPacketInfoId &extrinsicInfoId) const	Checks that extrinsic packet information exists.
void	**MakeLocalCopyOfExtrinsicPacketInfo** ()	Copies extrinsic packet information to self-packet.
template<typename T > static unique_ptr< Packet >	**CreatePacket** (SimulationEngineInterface &simEngineInterface, const T &payload)	Creates a packet. (Designates structured payload)
template<typename T > static unique_ptr< Packet >	**CreatePacketWithExtraHeaderSpace** (SimulationEngineInterface &simEngineInterface, const T	Creates a packet with extra header space. (Designates structured payload)

	&payload, const unsigned int extraAllocatedBytesForHeaders)	
template<typename T > static unique_ptr< Packet >	**CreatePacket** (SimulationEngineInterface &simEngineInterface, const T &payload, const unsigned int totalPayloadLength, const bool initUseVirtualPayload=false)	Creates a packet. (Designates structured payload, packet length, virtual payload use)
template<typename T > static unique_ptr< Packet >	**CreatePacketWithExtraHeaderSpace** (SimulationEngineInterface &simEngineInterface, const T &payload, const unsigned int totalPayloadLength, const unsigned int extraAllocatedBytesForHeaders, const bool initUseVirtualPayload=false)	Creates a packet with extra header space. (Designates structured payload, packet length, virtual payload use)
static unique_ptr< Packet >	**CreatePacket** (SimulationEngineInterface &simEngineInterface, const vector< unsigned char > &payload)	Creates a packet. (Designates a vector of bytes)
static unique_ptr< Packet >	**CreatePacketWithExtraHeaderSpace** (SimulationEngineInterface &simEngineInterface, const vector< unsigned char > &payload, const unsigned int extraAllocatedBytesForHeaders)	Creates a packet with extra header space. (Designates a vector of bytes)
static unique_ptr< Packet >	**CreatePacket** (SimulationEngineInterface &simEngineInterface, const vector< unsigned char > &payload, const unsigned int totalPayloadLength, const bool initUseVirtualPayload=false)	Creates a packet. (Designates a vector of bytes, packet length, virtual payload use)
static unique_ptr<	**CreatePacketWithExtraHeaderSpa**	Creates a packet with extra header

Packet >	**ce** (SimulationEngineInterface &simEngineInterface, const vector< unsigned char > &payload, const unsigned int totalPayloadLength, const unsigned int extraAllocatedBytesForHeaders, const bool initUseVirtualPayload=false)	space. (Designates a vector of bytes, packet length, virtual payload use)
static unique_ptr< Packet >	**CreatePacket** (SimulationEngineInterface &simEngineInterface, const string &payload)	Creates a packet. (Designates a payload in string)
static unique_ptr< Packet >	**CreatePacketWithExtraHeaderSpace** (SimulationEngineInterface &simEngineInterface, const string &payload, const unsigned int extraAllocatedBytesForHeaders)	Creates a packet with extra header space. (Designates a payload in string)
static unique_ptr< Packet >	**CreatePacket** (SimulationEngineInterface &simEngineInterface, const string &payload, const unsigned int totalPayloadLength, const bool initUseVirtualPayload=false)	Creates a packet. (Designates a payload in string, packet length, virtual payload use)
static unique_ptr< Packet >	**CreatePacketWithExtraHeaderSpace** (SimulationEngineInterface &simEngineInterface, const string &payload, const unsigned int totalPayloadLength, const unsigned int extraAllocatedBytesForHeaders, const bool initUseVirtualPayload=false)	Creates a packet with extra header space. (Designates a payload in string, packet length, virtual payload use)
static unique_ptr< Packet >	**CreatePacket** (SimulationEngineInterface	Creates a packet. (Designates a payload in char,

	&simEngineInterface, const unsigned char *payload, const unsigned int payloadLength)	packet length)
static unique_ptr< Packet >	**CreatePacketWithExtraHeaderSpace** (SimulationEngineInterface &simEngineInterface, const unsigned char *payload, const unsigned int payloadLength, const unsigned int extraAllocatedBytesForHeaders)	Creates a packet with extra header space. (Designates a payload in char, packet length)
static unique_ptr< Packet >	**CreatePacket** (SimulationEngineInterface &simEngineInterface)	Creates a packet. (No payload)
static unique_ptr< Packet >	**CreatePacketWithExtraHeaderSpace** (SimulationEngineInterface &simEngineInterface, const unsigned int extraAllocatedBytesForHeaders)	Creates a packet with extra header space. (No payload)
static unique_ptr< Packet >	**CreatePacketWithoutSimInfo** (const unsigned char data[], const unsigned int size)	Creates a packet without simulation information (packet ID). (Designates packet data in unsigned char)
template<typename T > static unique_ptr< Packet >	**CreatePacketWithoutSimInfo** (const T &payload)	Creates a packet without simulation information (packet ID). (Designates a structured packet data)

4.2.2. ExtrinsicPacketInformation

Abstract class for extrinsic packet information

Return type	Function(argument)	Description
virtual shared_ptr < ExtrinsicPacketInf	**Clone** ()=0	Clones information. （pure virtual function）

ormation >		

4.2.3. PacketId

Packet ID type

Return type	Function(argument)	Description
	PacketId (const NodeId nodeId, const unsigned long long int sequenceNumber)	Constructor of PacketIdTyp class.
NodeId	**GetSourceNodeId** () const	Gets source node ID.
unsigned long long int	**GetSourceNodeSequenceNumber** () const	Gets source node sequence number.
string	**ConvertToString** () const	Converts packet ID to string.

4.3. Random number related APIs

Source file: randomnumbergen.h

4.3.1. RandomNumberGenerator

Random number generator with boost::rand48

Return type	Function(argument)	Description
	RandomNumberGenerator (const RandomNumberGeneratorSeed &seed)	Constructor of RandomNumberGenerator.
void	**SetSeed** (const RandomNumberGeneratorSeed &seed)	Sets a seed for random number generator.
int32_t	**GenerateRandomInt** (const int32_t lowestValue, const int32_t highestValue)	Generates and returns random number in integer within specified range.
double	**GenerateRandomDouble** ()	Generates and returns random number in double from 0 to 1, [0:1).

4.3.2. HighQualityRandomNumberGenerator

Random number generator with boost::mt19937

Return type	Function(argument)	Description
	HighQualityRandomNumberGenerator (const RandomNumberGeneratorSeed &seed)	Constructor of HighQualityRandomNumberGenerator.
void	**SetSeed** (const	Sets a seed for random number generator.

	RandomNumberGeneratorSeed &seed)	
int32_t	**GenerateRandomInt** (const int32_t lowestValue, const int32_t highestValue)	Generates and returns random number in integer within specified range.
double	**GenerateRandomDouble** ()	Generates and returns random number in double from 0 to 1, [0:1).

4.3.3. Utility functions

Utility functions related to random number generation

Return type	Function(argument)	Description
RandomNumber GeneratorSeed	**HashInputsToMakeSeed** (const RandomNumberGeneratorSeed seed, const unsigned long long int hashingInput)	Generates a random seed. (Hash key is one unsinged long long int value)
template<typename T> RandomNumber GeneratorSeed	**HashInputsToMakeSeed** (const RandomNumberGeneratorSeed seed, const T hashingInput)	Generates a random seed. (Hash key is one template value)
template<typename T1, typename T2> RandomNumber GeneratorSeed	**HashInputsToMakeSeed** (const RandomNumberGeneratorSeed seed, const T1 hashingInput1, const T2 hashingInput2)	Generates a random seed. (Hash keys are two template values)
template<typename T1, typename T2, typename T3> RandomNumber GeneratorSeed	**HashInputsToMakeSeed** (RandomNumberGeneratorSeed Seed, const T1 hashingInput1, const T2 hashingInput2, const T3 hashingInput3)	Generates a random seed. (Hash keys are three template value)
template<typename T> inline	**HashInputsToMakeSeed**(RandomNumberGeneratorSeed	Generates a random seed. (Hash keys are one string and one

RandomNumber GeneratorSeed	seed, const std::string& hashingInput1, const T hashingInput2)	template value)
double	**ConvertToExponentialDistribution** (const double& randomDouble)	Convert uniform distributed value, [0:1), to exponential distributed value.
double	**ConvertToGuassianDistribution** (const double& uniformRandom1, const double& uniformRandom2)	Convert uniform distributed value, [0:1), to Guassian distributed value. (Generates one random number)
void	**ConvertToGaussianDistribution** (const double& uniformRandom1, const double& uniformRandom2, double& gaussianRandom1, double& gaussianRandom2)	Convert uniform distributed value, [0:1), to Guassian distributed value. (Generates two random numbers)

4.4. Parameter related APIs

Source file: scensim_parmio.h

4.4.1. ParameterDatabaseReader

Parameter database class

Return type	Function(argument)	Description
	ParameterDatabaseReader (const string &ParameterFileName)	Constructor of ParameterDatabaseReader class.
void	**DisableUnusedParameterWarning** ()	Disables warning for unused parameters.
bool	**ParameterExists** (const string &parameterName) const	Checks that the specified parameter exists. (For global parameter)
bool	**ParameterExists** (const string &parameterName, const InterfaceOrInstanceId &instanceId) const	Checks that the specified parameter exists. (For instance parameter)
bool	**ParameterExists** (const string &parameterName, const NodeId &nodeId) const	Checks that the specified parameter exists. (For node parameter)
bool	**ParameterExists** (const string &parameterName, const NodeId &nodeId, const InterfaceOrInstanceId &interfaceId) const	Checks that the specified parameter exists. (For interface parameter)
bool	**ReadBool** (const string &parameterName) const	Gets bool type parameter value. (For global parameter)
bool	**ReadBool** (const string &parameterName, const InterfaceOrInstanceId &instanceId) const	Gets bool type parameter value. (For instance parameter)
bool	**ReadBool** (const string &parameterName, const	Gets bool type parameter value. (For node parameter)

	NodeId &nodeId) const	
bool	**ReadBool** (const string &parameterName, const NodeId &nodeId, const InterfaceOrInstanceId &interfaceId) const	Gets bool type parameter value. (For interface parameter)
int	**ReadInt** (const string &parameterName) const	Gets int type parameter value. (For global parameter)
int	**ReadInt** (const string &parameterName, const InterfaceOrInstanceId &instanceId) const	Gets int type parameter value. (For instance parameter)
int	**ReadInt** (const string &parameterName, const NodeId &nodeId) const	Gets int type parameter value. (For node parameter)
int	**ReadInt** (const string &parameterName, const NodeId &nodeId, const InterfaceOrInstanceId &interfaceId) const	Gets int type parameter value. (For interface parameter)
long long int	**ReadBigInt** (const string &parameterName) const	Gets long long int type parameter value. (For global parameter)
long long int	**ReadBigInt** (const string &parameterName, const InterfaceOrInstanceId &instanceId) const	Gets long long int type parameter value. (For instance parameter)
long long int	**ReadBigInt** (const string &parameterName, const NodeId &nodeId) const	Gets long long int type parameter value. (For node parameter)
long long int	**ReadBigInt** (const string &parameterName, const NodeId &nodeId, const InterfaceOrInstanceId &interfaceId) const	Gets long long int type parameter value. (For interface parameter)
unsigned int	**ReadNonNegativeInt** (	Gets unsigned int type parameter

	const string &parameterName) const	value. (For global parameter)
unsigned int	**ReadNonNegativeInt** (const string &parameterName, const InterfaceOrInstanceId &instanceId) const	Gets unsigned int type parameter value. (For instance parameter)
unsigned int	**ReadNonNegativeInt** (const string &parameterName, const NodeId &nodeId) const	Gets unsigned int type parameter value. (For node parameter)
unsigned int	**ReadNonNegativeInt** (const string &parameterName, const NodeId &nodeId, const InterfaceOrInstanceId &interfaceId) const	Gets unsigned int type parameter value. (For interface parameter)
unsigned long long int	**ReadNonNegativeBigInt** (const string &parameterName) const	Gets unsigned long long int type parameter value. (For global parameter)
unsigned long long int	**ReadNonNegativeBigInt** (const string &parameterName, const InterfaceOrInstanceId &instanceId) const	Gets unsigned long long int type parameter value. (For instance parameter)
unsigned long long int	**ReadNonNegativeBigInt** (const string &parameterName, const NodeId &nodeId) const	Gets unsigned long long int type parameter value. (For node parameter)
unsigned long long int	**ReadNonNegativeBigInt** (const string &parameterName, const NodeId &nodeId, const InterfaceOrInstanceId &interfaceId) const	Gets unsigned long long int type parameter value. (For interface parameter)
double	**ReadDouble** (const string &parameterName) const	Gets double type parameter value. (For global parameter)
double	**ReadDouble** (const string &parameterName, const InterfaceOrInstanceId &instanceId) const	Gets double type parameter value. (For instance parameter)
double	**ReadDouble** (	Gets double type parameter value.

	const string &parameterName, const NodeId &nodeId) const	(For node parameter)
double	**ReadDouble** (const string &parameterName, const NodeId &nodeId, const InterfaceOrInstanceId &interfaceId) const	Gets double type parameter value. (For interface parameter)
SimTime	**ReadTime** (const string &parameterName) const	Gets SimTime parameter value. (For global parameter)
SimTime	**ReadTime** (const string &parameterName, const InterfaceOrInstanceId &instanceId) const	Gets SimTime parameter value. (For instance parameter)
SimTime	**ReadTime** (const string &parameterName, const NodeId &nodeId) const	Gets SimTime parameter value. (For node parameter)
SimTime	**ReadTime** (const string &parameterName, const NodeId &nodeId, const InterfaceOrInstanceId &interfaceId) const	Gets SimTime parameter value. (For interface parameter)
string	**ReadString** (const string &parameterName) const	Gets string type parameter value. (For global parameter)
string	**ReadString** (const string &parameterName, const InterfaceOrInstanceId &instanceId) const	Gets string type parameter value. (For instance parameter)
string	**ReadString** (const string &parameterName, const NodeId &nodeId) const	Gets string type parameter value. (For instance parameter)
string	**ReadString** (const string &parameterName, const NodeId &nodeId, const InterfaceOrInstanceId &interfaceId) const	Gets string type parameter value. (For interface parameter)
string	**GetContainingNodeIdSetNameFor**	Gets group name specified by

	(const NodeId &nodeId) const	is-member-of parameter for the specified node.
void	**MakeSetOfAllNodeIds** (set< NodeId > &setOfNodeIds) const	Gets all node IDs.
void	**MakeSetOfAllCommNodeIds** (set< NodeId > &setOfNodeIds) const	Gets all communication node IDs.
bool	**CommNodeIdExists** (const NodeId &nodeId) const	Checks that the specified communication node exists.
void	**MakeSetOfAllNodeIdsWithParameter** (const string &parameterName, set< NodeId > &setOfNodeIds) const	Gets node ID set of nodes that have the specified parameter.
void	**MakeSetOfAllNodeIdsWithParameter** (const string &parameterName, const string &parameterValue, set< NodeId > &setOfNodeIds) const	Gets node ID set of nodes that have the specified parameter value.
void	**MakeSetOfAllInterfaceIdsForANode** (const NodeId &nodeId, set< InterfaceOrInstanceId > &setOfInterfaces) const	Gets all interface IDs for the specified node.
void	**MakeSetOfAllInterfaceIdsForANode** (const NodeId &nodeId, const string &parameterName, set< InterfaceOrInstanceId > &setOfInterfaces) const	Gets all interface IDs with the specified parameter for specified node.
void	**MakeSetOfAllInterfaceIds** (const string &parameterName, set< InterfaceOrInstanceId > &setOfInterfaces) const	Gets all interface IDs with the specified parameter.
void	**MakeSetOfAllInterfaceIds** (const string &parameterName, const string &parameterMustBeEqual, set< InterfaceOrInstanceId >	Gets all interface IDs with the specified parameter value.

	&setOfInterfaces) const	
void	**MakeSetOfAllInstanceIdsForANode** (const NodeId &nodeId, const string &parameterName, set< InterfaceOrInstanceId > &setOfInstances) const	Gets all instance IDs for the specified node.
void	**MakeSetOfAllInstanceIds** (const string &parameterName, set< InterfaceOrInstanceId > &setOfInstances) const	Gets all instance IDs with the specified parameter.

4.5. BER (Bit Error Rate) related APIs

Source file: scensim_bervurves.h

4.5.1. BitOrBlockErrorRateCurveDatabase

BER/BLER curve database

Return type	Function(argument)	Description
	BitOrBlockErrorRateCurveDatabase (const string &berFileName)	Constructor of BitOrBlockErrorRateCurveDatabase class.
void	**LoadBerCurveFile** (const string &berFileName)	Loads BER curve file.
void	**LoadBlockErrorRateCurveFile** (const string &blerFileName)	Loads BLER curve file.
shared_ptr< BitErrorRateCurve >	**GetBerCurve** (const string &curveFamilyName, const string &modeName)	Gets BER curve with the specified BER curve family name and mode name.
shared_ptr< BlockErrorRateCurve >	**GetBlockErrorRateCurve** (const string &curveFamilyName, const string &modeName) const	Gets BLER curve with the specified BLER curve family name and mode name.

4.5.2. BitErrorRateCurve

BER curve

Return type	Function(argument)	Description
	BitErrorRateCurve (const string &initCurveFamilyName, const string &initModeName, long long int initBitsPerSecondForMode)	Constructor of BitErrorRateCurve class.
string	**GetCurveFamilyName** () const	Gets BER curve family name.
string	**GetModeName** () const	Gets mode name.
void	**AddDataPoint** (	Adds BER data point (SNR and bit

	const double &snrValue, const double &bitErrorRateForThatSnr)	error rate).
double	**CalculateBitErrorRate** (const double &signalToNoiseAndInterferenceRatio) const	Caluclates bit error rate for the specified SINR.

4.5.3. BlockErrorRateCurve

BLER curve

Return type	Function(argument)	Description
	BlockErrorRateCurve (const string &initCurveFamilyName, const string &initModeName)	Constructor of BlockErrorRateCurve class.
string	**GetCurveFamilyName** () const	Gets BLER curve family name.
string	**GetModeName** () const	Gets mode name.
void	**AddDataPoint** (const double &snrValue, const double &blockErrorRateForThatSnr)	Adds BLER data point (SNR and block error rate).
double	**CalcBlockErrorRate** (const double &signalToNoiseAndInterferenceRatio) const	Calculates block error rate for the specified SINR.

4.6. Statistics related APIs

Source file: scensim_stats.h/cpp

4.6.1. CounterStatistic

Counter type statistic

Return type	Function(argument)	Description
bool	**IsEnabled** () const	Checks that this statistic is enabled.
void	**IncrementCounter** (const unsigned long long int incrementNumber=1)	Increments counter.
void	**UpdateCounter** (const long long int newCounterValue)	Updates counter.

4.6.2. RealStatistic

Real type statistic

Return type	Function(argument)	Description
bool	**IsEnabled** () const	Checks that this statistic is enabled.
void	**RecordStatValue** (const double &value)	Records statistic value.

4.7. Utility functions

Source file: scensim_support.h

4.7.1. Utility functions

Return type	Function(argument)	Description
unsigned short int	**ConvertToUShortInt** (const unsigned int value)	Converts unsigned int value to unsigned short int value.
unsigned short int	**ConvertToUShortInt** (const unsigned int value, const string& failureMessage)	Converts unsigned int value to unsigned short int value with failure message.
unsigned char	**ConvertToUChar** (const unsigned int value)	Converts unsigned int value to unsigned char value.
unsigned char	**ConvertToUChar**(const unsigned int value, const string& failureMessage)	Converts unsigned int value to unsigned char value with failure message.
unsigned int	**RoundToUint** (const double& x)	Rounds double value to unsigned int value.
unsigned int	**RoundUpToUint** (const double& x)	Rounds up double value to unsigned int value.
int	**RoundToInt** (const double& x)	Rounds double value to int value.
unsigned int	**DivideAndRoundUp**(const unsigned int x, const unsigned int y)	Divides and rounds up to unsigned int value.
template<typename T> T	**MinOf3** (const T& x1, const T& x2, const T& x3)	Returns minimum of tree values.
double	**ConvertToNonDb** (const double& dB)	Converts decibel value (dB) value to linear value.
double	**ConvertToDb** (const double& nonDb)	Converts linear value to decibel value (dB).
double	**ConvertIntToDb** (const unsigned int value)	Converts integer value to decibel value (dB).
double	**ConvertYMetersToLatitudeDegrees** (	Converts Y coordinate in meters to latitude in degrees.

	const double& latitudeOriginDegrees, const double& yMeters)	
double	**ConvertXMetersToLongitudeDegrees** (const double& latitudeOriginDegrees, const double& longitudeOriginDegrees, const double& xMeters)	Converts X coordinates in meters to longitude in degrees.
double	**ConvertLatitudeDegreesToYMeters** (const double latitudeOriginDegrees, const double latitudeDegrees)	Converts latitude in degrees to Y coordinate in meters.
double	**ConvertLongitudeDegreesToXMeters** (const double latitudeOriginDegrees, const double longitudeOriginDegrees, const double longitudeDegrees)	Converts longitude in degrees to X coordinate in meters.
template<typename T> T	**CalcInterpolatedValue** (const double& x1, const T& y1, const double& x2, const T& y2, const double x)	Calculates interpolated value for the specified two points.
void	**ConvertStringToLowerCase** (string& aString)	Converts string to lower case.
string	**MakeLowerCaseString** (const string& aString)	Converts and returns lower case string.
bool	**StringIsAllLowerCase** (const string& aString)	Checks that the specified string is all lower case.
void	**ConvertStringToUpperCase** (string& aString)	Converts string to upper case.
string	**MakeUpperCaseString** (const string& aString)	Converts and returns upper case string.
bool	**StringIsAllUpperCase** (const string& aString)	Checks that the specified string is all upper case.

bool	**IsEqualCaseInsensitive** (const string& left, const string& right)	Checks that two strings are equal ignoring case.
template<typename T> string	**ConvertToString** (const T& aT)	Converts to string.
void	**ConvertStringToInt** (const string& aString, int& intValue, bool& success)	Converts string to int value.
void	**ConvertStringToNonNegativeInt** (const string& aString, unsigned int& uintValue, bool& success)	Converts string to unsigned int value.
void	**ConvertStringToBigInt** (const string& aString, long long int& intValue, bool& success)	Converts string to long long int value.
void	**ConvertStringToDouble** (const string& aString, double& doubleValue, bool& success)	Converts string to double value.

4.8. Network simulator related APIs

Source file: scensim_netsim.h

4.8.1. NetworkSimulator

Base class for network simulator

Return type	Function(argument)	Description
	NetworkSimulator (const shared_ptr< ParameterDatabaseReader > &initParameterDatabaseReaderPtr, const shared_ptr< SimulationEngine > &initSimulationEnginePtr, const RandomNumberGeneratorSeed &runSeed, const bool initRunSequentially=true)	Constructor of NetworkSimulator class.
void	**DeleteAllNodes** ()	Deletes all nodes.
void	**GetListOfNodeIds** (vector< NodeId > &nodeIds)	Gets all node IDs.
virtual NetworkAddress	**LookupNetworkAddress** (const NodeId &nodeId) const	Looks up network address for the target node ID.
virtual void	**LookupNetworkAddress** (const NodeId &nodeId, NetworkAddress &networkAddress, bool &success) const	Looks up network address for the target node ID with success flag.
virtual NodeId	**LookupNodeId** (const NetworkAddress &aNetworkAddress) const	Looks up node ID for the target network address.
virtual void	**LookupNodeId** (const NetworkAddress &aNetworkAddress, NodeId &nodeId, bool &success) const	Looks up node ID for the target network address with success flag.

virtual unsigned int	**LookupInterfaceIndex** (const NodeId &nodeId, const InterfaceId &interfaceName) const	Looks up interface index for the specified node ID and interface name.
virtual void	**CreateNewNode** (const ParameterDatabaseReader &theParameterDatabaseReader, const NodeId &nodeId, const string &nodeTypeName="")	Creates new node without mobility model.
virtual void	**CreateNewNode** (const ParameterDatabaseReader &theParameterDatabaseReader, const NodeId &nodeId, const shared_ptr< ObjectMobilityModel > &nodeMobilityModelPtr, const string &nodeTypeName="")	Creates new node.
virtual void	**DeleteNode** (const NodeId &nodeId)	Deletes node.
void	**InsertApplicationIntoANode** (const NodeId &nodeId, const shared_ptr< Application > &appPtr)	Inserts an application into the target node.
shared_ptr < MacLayerInterfaceForEmulation >	**GetMacLayerInterfaceForEmulation** (const NodeId &nodeId) const	Gets MAC layer interface for emulation.
const GlobalNetworkingObjectBag &	**GetGlobalNetworkingObjectBag** () const	Gets global networking object bag.
virtual double	**CalculatePathlossFromNodeToLocation** (const NodeId &nodeId, const PropagationInformationType &informationtType, const size_t interfaceIndex, const double &positionXMeters, const double &positionYMeters, const double &positionZMeters,	Calculates pathloss from the specified node to the target location.

	PropagationStatisticsType &propagationStatistics)	
virtual void	**CalculatePathlossFromNodeToNode** (const NodeId &txNodeId, const NodeId &rxNodeId, const PropagationInformationType &informationType, const unsigned int txInterfaceIndex, const unsigned int rxInterfaceIndex, PropagationStatisticsType &propagationStatistics)	Calculates pathloss from the specified node to node.
virtual void	**OutputNodePositionsInXY** (const SimTime lastOutputTime, std::ostream &nodePositionOutStream) const	Outputs node X Y positions in stream (For GUI)
virtual void	**OutputTraceForAllNodePositions** (const SimTime &lastOutputTime) const	Outputs all node positions in trace.
void	**OutputAllNodeIds** (std::ostream &outStream) const	Outputs all node IDs in stream. (For GUI)
void	**OutputRecentlyAddedNodeIdsWithTypes** (std::ostream &outStream)	Outputs newly added node IDs in stream. (For GUI)
void	**OutputRecentlyDeletedNodeIds** (std::ostream &outStream)	Outputs newly deleted node IDs in stream. (For GUI)
void	**RunSimulationUntil** (const SimTime &simulateUpToTime)	Runs simulation until the specified time.
void	**AddPropagationCalculationTraceIfNecessary** (const InterfaceId &channelId, const shared_ptr< SimplePropagationLossCalculationModel > &propagationCalculationModelPtr)	Sets pathloss trace output if necessary.

virtual const ObjectMobilityPosition	**GetNodePosition** (const NodeId &nodeId)	Gets node position for the target node.
virtual const ObjectMobilityPosition	**GetAntennaLocation** (const NodeId &nodeId, const unsigned int interfaceIndex)	Gets antenna location for the target node.
SimTime	**GetTimeStepEventSynchronizationStep** ( ) const	Gets synchronization interval for time step event.
RandomNumber GeneratorSeed	**GetMobilitySeed** () const	Gets seed for mobility.
void	**AddNode** (const shared_ptr< NetworkNode > &aNodePtr)	Adds the specified node to simulator.
void	**RemoveNode** (const NodeId &nodeId)	Removes the specified node from simulator.
protected		
virtual void	**CompleteSimulatorConstruction** ()	Completes simulator construction.
virtual bool	**SupportMultiAgent** () const	Checks that the simulator supports multi agent simulation capability.
void	**SetupStatOutputFile** ()	Sets up statistics output file.
void	**CheckTheNecessityOfMultiAgentSupport** ()	Checks that the multi agent simulation capability is required.
virtual void	**ExecuteTimestepBasedEvent** ()	Executes time step based events.

4.9. Network node related APIs

Source file: scensim_netsim.h

4.9.1. NetworkNode

Abstract class for node

Return type	Function(argument)	Description
	NetworkNode (const ParameterDatabaseReader &theParameterDatabaseReader, const GlobalNetworkingObjectBag &theGlobalNetworkingObjectBag, const shared_ptr< SimulationEngineInterface > &initSimulationEngineInterfacePtr, const shared_ptr< ObjectMobilityModel > &initNodeMobilityModelPtr, const NodeId &theNodeId, const RandomNumberGeneratorSeed &runSeed, const bool dontBuildStackLayers=false)	Constructor of NetworkNode class.
NodeId	**GetNodeId** ()	Gets node ID.
string	**GetNodeTypeName** () const	Gets node type name.
void	**SetNodeTypeName** (const string &typeName)	Sets node type name.
NetworkAddress	**GetPrimaryNetworkAddress** () const	Gets primary network address.
RandomNumber GeneratorSeed	**GetNodeSeed** () const	Gets node seed.
virtual shared_ptr< NetworkLayer >	**GetNetworkLayerPtr** () const	Gets a pointer to network layer.

virtual const NetworkLayer &	**GetNetworkLayerRef** () const	Gets a reference to network layer.
virtual shared_ptr < TransportLayer >	**GetTransportLayerPtr** () const	Gets a pointer to transport layer.
virtual shared_ptr < ApplicationLayer >	**GetAppLayerPtr** () const	Gets a pointer to application layer.
virtual const ObjectMobilityPosition	**GetCurrentLocation** () const	Gets current location.
virtual double	**CalculatePathlossToLocation** (const PropagationInformationType &informationType, const size_t interfaceIndex, const double &positionXMeters, const double &positionYMeters, const double &positionZMeters, PropagationStatisticsType &propagationStatistics) const	Calculates pathloss to the specified location.
virtual void	**CalculatePathlossToNode** (const PropagationInformationType &informationType, const unsigned int interfaceIndex, const ObjectMobilityPosition &rxAntennaPosition, const AntennaModel &rxAntennaModel, PropagationStatisticsType &propagationStatistics) const	Calculates pathloss to the specified node.
virtual bool	**HasAntenna** (const InterfaceId &channelId) const	Checks that this node has an antenna with the specified channel ID.
virtual shared_ptr< AntennaModel >	**GetAntennaModelPtr** (const unsigned int interfaceIndex) const	Gets a pointer for antenna model.

virtual ObjectMobilityPosition	**GetAntennaLocation** (const unsigned int interfaceIndex) const	Gets antenna location with the specified interface.
virtual void	**OutputTraceForNodePosition** (const SimTime &lastOutputTime) const	Outputs trace for node position.
void	**OutputTraceForAddNode** () const	Outputs trace for node addition event.
void	**OutputTraceForDeleteNode** () const	Outputs trace for node deletion event.
virtual void	**CreateDynamicApplication** (const ParameterDatabaseReader &theParameterDatabaseReader, const GlobalNetworkingObjectBag &theGlobalNetworkingObjectBag, const NodeId &sourceNodeId, const InterfaceOrInstanceId &instanceId)	Creates an application dynamically.

4.10. Application layer related APIs

Source file: scensim_application.h

4.10.1. ApplicationLayer

Application layer holding applications

Return type	Function(argument)	Description
	ApplicationLayer (const shared_ptr< NetworkAddressLookupInterface > &networkAddressLookupInterfacePtr , const shared_ptr< SimulationEngineInterface > &simulationEngineInterfacePtr, const shared_ptr< TransportLayer > &transportLayerPtr, const shared_ptr< ObjectMobilityModel > &nodeMobilityModelPtr, const NodeId &initNodeId, const RandomNumberGeneratorSeed &initNodeSeed)	Constructor of ApplicationLayer class.
void	**AddApp** (const shared_ptr< Application > &appPtr)	Adds an application to application layer.
void	**DisconnectFromOtherLayers** ()	Disconnects the holding smart pointers.
template<typename T > shared_ptr< T >	**GetApplicationPtr** (const ApplicationId &applicationId)	Gets a pointer to the specified application.
unsigned short int	**GetNewApplicationInstanceNumber** ()	Gets new application instance number.

4.10.2. Application

Base class for application

Return type	Function(argument)	Description
	Application (const shared_ptr< SimulationEngineInterface > &initSimEngineInterfacePtr, const ApplicationId initApplicationId)	Constructor of ApplicationLayer class.
void	**DisconnectFromOtherLayers** ()	Disconnects the holding smart pointers.
shared_ptr < MacAndPhyInfoInterface >	**GetMacAndPhyInfoInterface** (const InterfaceId &interfaceId)	Gets MAC and PHY info interface.
Type	Member variable	Description
ApplicationId	**applicationId**	Application ID.
shared_ptr < SimulationEngineInterface >	**simulationEngineInterfacePtr**	A pointer to simulation engine interface.
shared_ptr < NetworkAddressLookupInterface >	**networkAddressLookupInterfacePtr**	A pointer to network address lookup interface.
shared_ptr< TransportLayer >	**transportLayerPtr**	A pointer to transport layer.
shared_ptr< RandomNumberGenerator >	**aRandomNumberGeneratorPtr**	A pointer to random number generator.
shared_ptr< ObjectMobilityModel >	**nodeMobilityModelPtr**	A pointer to mobility model.

shared_ptr< ApplicationLayer >	**applicationLayerPtr**	A pointer to application layer.

4.11. Transport layer related APIs

Source file: scensim_network.h

4.11.1. ProtocolPacketHandler

Interface class between transport layer and network layer.

Return type	Function(argument)	Description
virtual void	**ReceivePacketFromNetworkLayer** (unique_ptr< Packet > &packetPtr, const NetworkAddress &sourceAddress, const NetworkAddress &destinationAddress, const PacketPriority trafficClass, const NetworkAddress &lastHopAddress, const unsigned char hopLimit, const unsigned int interfaceIndex)=0	Receives a UDP packet from network layer. (pure virtual function)
virtual void	**GetPortNumbersFromPacket** (const Packet &aPacket, const unsigned int transportHeaderOffset, bool &portNumbersWereRetrieved, unsigned short int &sourcePort, unsigned short int &destinationPort) const =0	Gets source and destination port numbers from the specified packet. (pure virtual function)

Source file: scensim_transport.h

4.11.2. TransportLayer

Transport layer

Return type	Function(argument)	Description

	TransportLayer (const ParameterDatabaseReader &theParameterDatabaseReader, const shared_ptr< SimulationEngineInterface > &simulationEngineInterfacePtr, const shared_ptr< NetworkLayer > &networkLayerPtr, const NodeId &nodeId, const RandomNumberGeneratorSeed &nodeSeed)	Constructor of TransportLayer class.
shared_ptr< NetworkLayer >	**GetNetworkLayerPtr** () const	Gets a pointer to network layer.
void	**DisconnectProtocolsFromOtherLayers** ()	Disconnects the holding smart pointers.
Type	Member variable	Description
shared_ptr< UdpProtocol >	**udpPtr**	A pointer to UDP.
shared_ptr< TcpProtocol >	**tcpPtr**	A pointer to TCP.

4.11.3. UdpProtocol

UDP protocol

Return type	Function(argument)	Description
	UdpProtocol (const shared_ptr< SimulationEngineInterface > &initSimulationEngineInterfacePtr)	Constructor of UdpProtocol class.
void	**DisconnectFromOtherLayers** ()	Disconnects the holding smart pointers.
void	**ConnectToNetworkLayer** (const shared_ptr< NetworkLayer >	Registers UDP protocol (handler) to network layer.

	&networkLayerPtr)	
void	**SendPacket** (unique_ptr< Packet > &packetPtr, const unsigned short int sourcePort, const NetworkAddress &destinationAddress, const unsigned short int destinationPort, const PacketPriority &priority)	Sends a UDP packet to network layer without destination address.
void	**SendPacket** (unique_ptr< Packet > &packetPtr, const NetworkAddress &sourceAddress, const unsigned short int sourcePort, const NetworkAddress &destinationAddress, const unsigned short int destinationPort, const PacketPriority &priority)	Sends a UDP packet to network layer with destination address.
bool	**PortIsAvailable** (const NetworkAddress &portAddress, const int portNumber) const	Checks that the specified port number for the specified network address is available.
bool	**PortIsAvailable** (const int portNumber) const	Checks that the specified port number is available.
void	**OpenSpecificUdpPort** (const NetworkAddress &address, const unsigned short int portNumber, const shared_ptr< PacketForAppFromTransportLayerHandler > &packetHandlerPtr)	Opens the specified port number and registers a handler for application (PacketForAppFromTransportLayerHandler).
virtual void	**ReceivePacketFromNetworkLayer** (unique_ptr< Packet > &packetPtr, const NetworkAddress	Receives a UDP packet from network layer.

	&sourceAddress, const NetworkAddress &destinationAddress, const PacketPriority trafficClass, const NetworkAddress &lastHopAddress_notused, const unsigned char hopLimit_notused, const unsigned int interfaceIndex)	
virtual void	**GetPortNumbersFromPacket** (const Packet &aPacket, const unsigned int transportHeaderOffset, bool &portNumbersWereRetrieved, unsigned short int &sourcePort, unsigned short int &destinationPort) const	Gets source and destination port numbers from the specified packet.

4.11.4. UdpProtocol::PacketForAppFromTransportLayerHandler

Handler to pass a packet to application layer from UDP protocol

Return type	Function(argument)	Description
virtual void	**ReceivePacket** (unique_ptr< Packet > &packetPtr, const NetworkAddress &sourceAddress, const unsigned short int sourcePort, const NetworkAddress &destinationAddress, const PacketPriority &priority)=0	Receives a packet from UDP protocol. (pure virtual function)

4.11.5. TcpProtocol

TCP protocol

Return type	Function(argument)	Description
	TcpProtocol (const ParameterDatabaseReader &theParameterDatabaseReader, const shared_ptr< SimulationEngineInterface > &initSimulationEngineInterfacePtr, const NodeId &nodeId, const RandomNumberGeneratorSeed &nodeSeed)	Constructor of TcpProtocol class.
void	**ConnectToNetworkLayer** (const shared_ptr< NetworkLayer > &networkLayerPtr)	Connects TCP protocol (handler) to network layer.
void	**DisconnectFromOtherLayers** ()	Disconnects the holding smart pointers.
void	**CreateOutgoingTcpConnection** (const NetworkAddress &localAddress, const unsigned short int localPort, const NetworkAddress &destinationAddress, const unsigned short int destinationPort, const PacketPriority &priority, const shared_ptr< TcpConnection::AppTcpEventHandler > &appEventHandlerPtr, shared_ptr< TcpConnection > &newTcpConnectionPtr)	Creates an outgoing TCP connection with destination address.
void	**CreateOutgoingTcpConnection** (const unsigned short int localPort, const NetworkAddress &destinationAddress, const unsigned short int destinationPort, const PacketPriority &priority, const shared_ptr< TcpConnection::AppTcpEventHandler > &appEventHandlerPtr,	Creates an outgoing TCP connection without destination address.

	shared_ptr< TcpConnection > &newTcpConnectionPtr)	
bool	**PortIsAvailable** (const int portNumber) const	Checks that the specified port number is available.
void	**OpenSpecificTcpPort** (const NetworkAddress &address, const unsigned short int portNumber, const shared_ptr< ConnectionFromTcpProtocolHandler > &connectionHandlerPtr)	Opens the specified port number and registers a handler to an application (ConnectionFromTcpProtocolHandler).
void	**DisconnectConnectionHandlerForPort** (const NetworkAddress &address, const unsigned short int portNumber)	Disconnects TCP connection (Opens port number).
virtual void	**ReceivePacketFromNetworkLayer** (unique_ptr< Packet > &packetPtr, const NetworkAddress &sourceAddress, const NetworkAddress &destinationAddress, const PacketPriority trafficClass, const NetworkAddress &lastHopAddress_notused, const unsigned char hopLimit_notused, const unsigned int interfaceIndex_notused)	Receives a packet from network layer.
virtual void	**GetPortNumbersFromPacket** (const Packet &aPacket, const unsigned int transportHeaderOffset, bool &portNumbersWereRetrieved, unsigned short int &sourcePort, unsigned short int &destinationPort)	Gets source and destination port numbers from the specified packet.

	const	

4.11.6. ConnectionFromTcpProtocolHandler

Handler to set TCP connection from TCP protocol to application

Return type	Function(argument)	Description
virtual void	**HandleNewConnection** (const shared_ptr< TcpConnection > &connectionPtr)=0	Sets new TCP connection to application. (pure virtual function)

4.11.7. TcpConnection

TCP connection

Return type	Function(argument)	Description
bool	**IsConnected** () const	Checks that TCP connection is connected.
void	**EnableVirtualPayload** ()	Enables virtual payload function.
void	**SetPacketPriority** (const PacketPriority &priority)	Sets packet priority.
void	**SetAppTcpEventHandler** (const shared_ptr< AppTcpEventHandler > &newAppTcpEventHandlerPtr)	Sets a handler to application (AppTcpEventHandler).
void	**ClearAppTcpEventHandler** ()	Clears a handler to application (AppTcpEventHandler).
void	**SendDataBlock** (shared_ptr< vector< unsigned char > > &dataBlockPtr, const unsigned int	Sends TCP data block with virtual payload.

	dataLength)	
void	**SendDataBlock** (shared_ptr< vector< unsigned char > > &dataBlockPtr)	Sends TCP data block.
unsigned long long int	**GetNumberOfReceivedBytes** () const	Gets number of received bytes.
unsigned long long int	**GetNumberOfSentBytes** () const	Gets number of sent bytes.
unsigned long long int	**GetNumberOfDeliveredBytes** () const	Gets number of delivered (acked data) bytes.
unsigned long long int	**GetCurrentNumberOfUnsentBufferedBytes** () const	Gets current number of unsent buffered bytes.
unsigned long long int	**GetCurrentNumberOfAvailableBufferBytes** () const	Gets current number of available buffer bytes.
NetworkAddress	**GetForeignAddress** () const	Gets foreign (destination) address.
void	**Close** ()	Closes a TCP connection.

4.11.8. TcpConnection::AppTcpEventHandler

Handler to send and receive TCP block between TCP connection and application

Return type	Function(argument)	Description
virtual void	**DoTcpIsReadyForMoreDataAction** ()=0	Sends data block to TCP connection. (pure virtual function)
virtual void	**ReceiveDataBlock** (const unsigned char dataBlock[], const unsigned int dataLength, const unsigned int actualDataLength, bool &stallIncomingDataFlow)=0	Receives TCP data block from TCP connection. (pure virtual function)
virtual void	**DoTcpRemoteHostClosedAction** ()	Does remote host process when TCP connection is closed.
virtual void	**DoTcpLocalHostClosedAction** ()	Does local host process when TCP connection is closed.

4.12. Network layer related APIs

Source file: scensim_network.h/cpp

4.12.1. NetworkLayer

Abstract class for network layer

Return type	Function(argument)	Description
virtual void	**DisconnectFromOtherLayers** ()=0	Disconnects the holding smart pointers. (pure virtual function)
virtual NodeId	**GetNodeId** () const =0	Gets node ID. (pure virtual function)
virtual shared_ptr< RoutingTable >	**GetRoutingTableInterface** ()=0	Gets a pointer to routing table. (pure virtual function)
virtual NetworkAddress	**GetPrimaryNetworkAddress** () const =0	Gets primary network address. (pure virtual function)
virtual unsigned int	**NumberOfInterfaces** () const =0	Gets number of interfaces. (pure virtual function)
virtual unsigned int	**LookupInterfaceIndex** (const NetworkAddress &interfaceAddress) const =0	Looks up interface index for the specified network address. (pure virtual function)
virtual unsigned int	**LookupInterfaceIndex** (const InterfaceId &interfaceName) const =0	Looks up interface index for the specified interface name. (pure virtual function)
virtual InterfaceId	**GetInterfaceId** (const unsigned int interfaceIndex) const =0	Gets interface ID. (pure virtual function)
virtual NetworkAddress	**GetNetworkAddress** (const unsigned int interfaceIndex) const =0	Gets network address. (pure virtual function)

virtual NetworkAddress	**GetSubnetAddress** (const unsigned int interfaceIndex) const =0	Gets subnet address. (pure virtual function)
virtual NetworkAddress	**GetSubnetMask** (const unsigned int interfaceIndex) const =0	Gets subnet mask. (pure virtual function)
virtual unsigned int	**GetSubnetMaskBitLength** (const unsigned int interfaceIndex) const =0	Gets subnet mask bit length. (pure virtual function)
virtual NetworkAddress	**MakeBroadcastAddressForInterface** (const unsigned int interfaceIndex) const =0	Makes broadcast address. (pure virtual function)
virtual void	**SetInterfaceIpAddress** (const size_t interfaceIndex, const NetworkAddress &newInterfaceAddress, const unsigned int subnetMaskLengthBits)=0	Sets network address. (pure virtual function)
virtual void	**SetInterfaceGatewayAddress** (const unsigned int interfaceIndex, const NetworkAddress &newGatewayAddress)=0	Sets gateway address. (pure virtual function)
virtual void	**ClearInterfaceIpInformation** (const unsigned int interfaceIndex)=0	Clears interface information. (pure virtual function)
virtual void	**RegisterPacketHandlerForProtocol** (const unsigned char protocolNum, const shared_ptr< ProtocolPacketHandler > &packetHandlerPtr)=0	Registers transport protocol (handler). (pure virtual function)
virtual void	**RegisterOnDemandRoutingProtocolInterface** (const shared_ptr< OnDemandRoutingProtocolInterface > interfacePtr)=0	Registers on demand routing protocol. (pure virtual function)

virtual void	**RegisterNetworkAddressInterface** (const shared_ptr< NetworkAddressInterface > &interfacePtr)=0	Registers network address interface. (pure virtual function)
virtual void	**ReceivePacketFromUpperLayer** (unique_ptr< Packet > &packetPtr, const NetworkAddress &destinationAddress, PacketPriority trafficClass, const unsigned char protocol)=0	Receives a packet from upper layer without destination address. (pure virtual function)
virtual void	**ReceivePacketFromUpperLaye**r (unique_ptr< Packet > &packetPtr, const NetworkAddress &sourceAddress, const NetworkAddress &destinationAddress, PacketPriority trafficClass, const unsigned char protocol)=0	Receives a packet from upper layer with destination address. (pure virtual function)
virtual void	**ReceiveOutgoingBroadcastPacket** (unique_ptr< Packet > &packetPtr, const unsigned int outgoingInterfaceIndex, PacketPriority trafficClass, const unsigned char protocol)=0	Receives outgoing broadcast packet. (pure virtual function)
virtual void	**ReceiveOutgoingPreformedNetworkPacket** (unique_ptr< Packet > &packetPtr)=0	Receives outgoing preformed (IP header attached) packet. (pure virtual function)
virtual void	**ReceiveRoutedNetworkPacketFromRoutingProtocol** (unique_ptr< Packet > &packetPtr, const unsigned int interfaceIndex, const NetworkAddress &nextHopAddress)=0	Receives a packet from routing protocol. (pure virtual function)

virtual void	**GetInterfaceIndexForOneHopDestination** (const NetworkAddress &destinationAddress, bool &success, unsigned int &interfaceIndex) const =0	Gets interface index for next hop address. (pure virtual function)
virtual void	**GetNextHopAddressAndInterfaceIndexForDestination** (const NetworkAddress &destinationAddress, bool &success, NetworkAddress &nextHopAddress, unsigned int &interfaceIndex) const =0	Gets next hop address and interface index. (pure virtual function)
virtual NetworkAddress	**GetSourceAddressForDestination** (const NetworkAddress &destinationAddress) const =0	Gets source address for the specified destination address. (pure virtual function)
virtual void	**GetNextHopAddressAndInterfaceIndexForNetworkPacket** (const Packet &aPacket, bool &success, NetworkAddress &nextHopAddress, unsigned int &interfaceIndex) const =0	Gets next hop address and interface index for the specified IP packet. (pure virtual function)
virtual void	**ReceivePacketFromMac** (const unsigned int macIndex, unique_ptr< Packet > &packetPtr, const NetworkAddress &lastHopAddress, const EtherTypeFieldId etherType=ETHERTYPE_IS_NOT_SPECIFIED)=0	Receives a packet for MAC layer. (pure virtual function)
virtual void	**ReceiveUndeliveredPacketFromMac** (const unsigned int macIndex, unique_ptr< Packet > &packetPtr, const NetworkAddress &nextHopAddress)=0	Receives an undelivered packet from MAC layer. (pure virtual function)

virtual void	**SetInterfaceMacLayer** (const unsigned int interfaceIndex, const shared_ptr< MacLayer > &macLayerPtr)=0	Sets MAC layer. (pure virtual function)
virtual shared_ptr< MacLayer >	**GetMacLayerPtr** (const unsigned int interfaceIndex) const =0	Gets a pointer to MAC layer. (pure virtual function)
virtual shared_ptr < NetworkInterface Manager >	**GetNetworkInterfaceManagerPtr** (const unsigned int interfaceIndex) const =0	Gets a pointer to network interface manager. (pure virtual function)
virtual void	**SetInterfaceOutputQueue** (const unsigned int interfaceIndex, const shared_ptr< InterfaceOutputQueue > &outputQueuePtr)=0	Sets an interface output queue. (pure virtual function)
virtual void	**ProcessLinkIsUpNotification** (const unsigned int interfaceIndex)=0	Processes a notification that link is up from MAC layer (STA side). (pure virtual function)
virtual void	**ProcessLinkIsDownNotification** (const unsigned int interfaceIndex)=0	Processes a notification that link is down from MAC layer (STA side). (pure virtual function)
virtual void	**ProcessNewLinkToANodeNotification** (const unsigned int interfaceIndex, const GenericMacAddress &macAddress)=0	Processes a notification that new link to a node is established from MAC layer (AP side). (pure virtual function)
virtual bool	**MacSupportsQualityOfService** (const unsigned int interfaceIndex) const =0	Checks that MAC layer supports QoS capability. (pure virtual function)
virtual shared_ptr < MacQualityOfServiceControlInterface >	**GetMacQualityOfServiceInterface** (const unsigned int interfaceIndex) const =0	Gets a pointer to MAC QoS interface. (pure virtual function)

virtual void	**InsertPacketIntoAnOutputQueue** (unique_ptr< Packet > &packetPtr, const unsigned int interfaceIndex, const NetworkAddress &nextHopAddress, const PacketPriority trafficClass, const EtherTypeFieldId etherType=ETHERTYPE_IS_NOT_SPECIFIED)=0	Inserts a packet into an output queue. (pure virtual function)
virtual void	**SetupDhcpServerAndClientIfNecessary** (const ParameterDatabaseReader &theParameterDatabaseReader, const shared_ptr< ApplicationLayer > &appLayerPtr)=0	Sets up DHCP server/client if necessary. (pure virtual function)

4.12.2. BasicNetworkLayer

IP network layer

Return type	Function(argument)	Description
void	**DisconnectFromOtherLayers** ()	Disconnects the holding smart pointers.
NodeId	**GetNodeId** () const	Gets node ID.
shared_ptr< RoutingTable >	**GetRoutingTableInterface** ()	Gets a pointer to routing table.
bool	**IsANetworkAddressForThisNode** (const NetworkAddress &anAddress) const	Checks that the specified network address is for this node.
void	**CheckIfNetworkAddressIsForThisNode** (const NetworkAddress &anAddress, bool &addressIsForThisNode, unsigned int &interfaceIndex) const	Checks that the specified network address is for this node and gets target interface index.
NetworkAddress	**GetPrimaryNetworkAddress** ()	Gets primary network address.

	const	
NetworkAddress	**GetNetworkAddress** (const unsigned int interfaceIndex) const	Gets network address.
NetworkAddress	**GetSubnetAddress** (const unsigned int interfaceIndex) const	Gets subnet address.
NetworkAddress	**GetSubnetMask** (const unsigned int interfaceIndex) const	Gets subnet mask.
unsigned int	**GetSubnetMaskBitLength** (const unsigned int interfaceIndex) const	Gets subnet mask bit length.
NetworkAddress	**MakeBroadcastAddressForInterface** (const unsigned int interfaceIndex) const	Makes broadcast address.
void	**SetInterfaceIpAddress** (const unsigned int interfaceIndex, const NetworkAddress &newInterfaceAddress, const unsigned int subnetMaskLengthBits)	Sets network address.
void	**SetInterfaceGatewayAddress** (const unsigned int interfaceIndex, const NetworkAddress &newGatewayAddress)	Sets gateway address.
void	**ClearInterfaceIpInformation** (const unsigned int interfaceIndex)	Clears interface information.
void	**RegisterPacketHandlerForProtocol** (const unsigned char protocolNum, const shared_ptr< ProtocolPacketHandler > &packetHandlerPtr)	Registers transport protocol (handler).
void	**RegisterOnDemandRoutingProtocolInterface** (const shared_ptr<	Registers on demand routing protocol.

	OnDemandRoutingProtocolInterface > interfacePtr)	
void	**RegisterNetworkAddressInterface** (const shared_ptr< NetworkAddressInterface > &interfacePtr)	Registers network address interface.
void	**GetInterfaceIndexForOneHopDestination** (const NetworkAddress &destinationAddress, bool &success, unsigned int &interfaceIndex) const	Gets interface index for next hop address.
void	**GetNextHopAddressAndInterfaceIndexForDestination** (const NetworkAddress &destinationAddress, bool &success, NetworkAddress &nextHopAddress, unsigned int &interfaceIndex) const	Gets next hop address and interface index.
NetworkAddress	**GetSourceAddressForDestination** (const NetworkAddress &destinationAddress) const	Gets source address for the specified destination address.
void	**GetNextHopAddressAndInterfaceIndexForNetworkPacket** (const Packet &aPacket, bool &success, NetworkAddress &nextHopAddress, unsigned int &interfaceIndex) const	Gets next hop address and interface index for the specified IP packet.
unsigned int	**LookupInterfaceIndex** (const NetworkAddress &interfaceAddress) const	Looks up interface index for the specified network address.
unsigned int	**LookupInterfaceIndex** (const InterfaceId &interfaceName) const	Looks up interface index for the specified interface name.
InterfaceId	**GetInterfaceId** (const unsigned int interfaceIndex) const	Gets interface ID.

unsigned int	**NumberOfInterfaces** () const	Gets number of interfaces.
void	**SetInterfaceMacLayer** (const unsigned int interfaceIndex, const shared_ptr< MacLayer > &macLayerPtr)	Sets MAC layer.
void	**ReceivePacketFromUpperLayer** (unique_ptr< Packet > &packetPtr, const NetworkAddress &destinationAddress, PacketPriority trafficClass, const unsigned char protocol)	Receives a packet from upper layer without destination address.
void	**ReceivePacketFromUpperLayer** (unique_ptr< Packet > &packetPtr, const NetworkAddress &sourceAddress, const NetworkAddress &destinationAddress, PacketPriority trafficClass, const unsigned char protocol)	Receives a packet from upper layer with destination address.
void	**ReceiveOutgoingBroadcastPacket** (unique_ptr< Packet > &packetPtr, const unsigned int outgoingInterfaceIndex, PacketPriority trafficClass, const unsigned char protocol)	Receives outgoing broadcast packet.
void	**ReceiveOutgoingPreformedNetworkPacket** (unique_ptr< Packet > &packetPtr)	Receives outgoing preformed (IP header attached) packet.
void	**ReceiveRoutedNetworkPacketFromRoutingProtocol** (unique_ptr< Packet > &packetPtr, const unsigned int interfaceIndex, const NetworkAddress &nextHopAddress)	Receives a packet from routing protocol.

void	**ReceivePacketFromMac** (const size_t macIndex, Packet *&packetPtr, const NetworkAddress &lastHopAddress)	Receives a packet for MAC layer.
void	**ReceiveUndeliveredPacketFromMac** (const unsigned int macIndex, unique_ptr< Packet > &packetPtr, const NetworkAddress &nextHopAddress)	Receives an undelivered packet from MAC layer.
shared_ptr< InterfaceOutputQueue >	**GetInterfaceOutputQueue** (const unsigned int interfaceIndex) const	Gets an interface output queue.
shared_ptr< MacLayer >	**GetMacLayerPtr** (const unsigned int interfaceIndex) const	Gets a pointer to MAC layer.
void	**SetInterfaceOutputQueue** (const unsigned int interfaceIndex, const shared_ptr< InterfaceOutputQueue > &outputQueuePtr)	Sets an interface output queue.
void	**ProcessLinkIsUpNotification** (const unsigned int interfaceIndex)	Processes a notification that link is up from MAC layer (STA side).
void	**ProcessLinkIsDownNotification** (const unsigned int interfaceIndex)	Processes a notification that link is down from MAC layer (STA side).
void	**ProcessNewLinkToANodeNotification** (const unsigned int interfaceIndex, const GenericMacAddress &newNodeMacAddress)	Processes a notification that new link to a node is established from MAC layer (AP side).
bool	**MacSupportsQualityOfService** (const unsigned int interfaceIndex) const	Checks that MAC layer supports QoS capability.

shared_ptr < MacQualityOfServiceControlInterface >	**GetMacQualityOfServiceInterface** (const unsigned int interfaceIndex) const	Gets a pointer to MAC QoS interface.
void	**InsertPacketIntoAnOutputQueue** (unique_ptr< Packet > &packetPtr, const unsigned int interfaceIndex, const NetworkAddress &nextHopAddress, const PacketPriority trafficClass, const EtherTypeFieldId etherType=ETHERTYPE_IS_NOT_SPECIFIED)	Inserts a packet into an output queue.
void	**SetupDhcpServerAndClientIfNecessary** (const ParameterDatabaseReader &theParameterDatabaseReader, const shared_ptr< ApplicationLayer > &appLayerPtr)	Sets up DHCP server/client if necessary.
static shared_ptr < BasicNetworkLayer >	**CreateNetworkLayer** (const ParameterDatabaseReader &theParameterDatabaseReader, const GlobalNetworkingObjectBag &theGlobalNetworkingObjects, const shared_ptr< SimulationEngineInterface > &simEngineInterfacePtr, const NodeId &initNodeId, const RandomNumberGeneratorSeed &nodeSeed)	Creates BasicNetworkLayer instance.

4.13. Network address related APIs

Source file: scensim_netaddress_ipv4.h、scensim_netaddress_ipv6.h

4.13.1. NetworkAddress

Definition of network address

Return type	Function(argument)	Description
	NetworkAddress (const uint32_t initIpAddress)	Constructor of NetworkAddress class.
	NetworkAddress (const uint64_t &ipAddressHighBits, const uint64_t &ipAddressLowBits)	Constructor of NetworkAddress class. (For IPv6)
	NetworkAddress (const NetworkAddress &subnetAddress, const NetworkAddress &hostIdentifierOnlyAddress)	Constructor of NetworkAddress class.
void	**SetAddressFromString** (const string &stringToConvert, const NodeId &nodeId, bool &success)	Sets network address from string with node ID.
void	**SetAddressFromString** (const string &stringToConvert, bool &success)	Sets network address from string.
string	**ConvertToString** () const	Converts network address to string.
void	**SetToTheBroadcastAddress** ()	Sets the broadcast address.
void	**SetToAnyAddress** ()	Sets any address (all bits are 0).
bool	**IsTheBroadcastAddress** () const	Checks that the address is the broadcast address (all bits are 1).
bool	**IsABroadcastAddress** (const NetworkAddress &subnetMask) const	Checks that the address is a broadcast address.
bool	**IsAMulticastAddress** () const	Checks that the address is a multicast address.
bool	**IsLinkLocalAddress** () const	Checks that the address is a link

		local address. (For IPv6)
bool	**IsAnyAddress** () const	Checks that the address is any address (all bits are 0).
bool	**IsTheBroadcastOrAMulticastAddress** ( ) const	Checks that the address is the broadcast address (all bits are 1) or a multicast address.
bool	**IsABroadcastOrAMulticastAddress** (const Ipv4NetworkAddress &subnetMask) const	Checks that the address is a broadcast address or a multicast address.
uint64_t	**GetRawAddressLowBits** () const	Gets lower 64 bits of raw address. (For IPv6)
uint64_t	**GetRawAddressHighBits** () const	Gets higher 64 bits of raw address. (For IPv6)
uint32_t	**GetRawAddressLow32Bits** () const	Gets lower 32 bits of raw address.
unsigned int	**GetMulticastGroupNumber** () const	Gets multicast group number.
void	**SetWith32BitRawAddress** (const uint32_t newipAddress)	Sets new network address.
bool	**IsInSameSubnetAs** (const NetworkAddress &address, const NetworkAddress &subnetMask) const	Checks that the network address is in same subnet.
NetworkAddress	**MakeSubnetAddress** (const NetworkAddress &subnetMask) const	Makes subnet address.
NetworkAddress	**MakeAddressWithZeroedSubnetBits** (const NetworkAddress &subnetMask) const	Makes network address with zeroed subnet bits.
static NetworkAddress	**ReturnTheBroadcastAddress** ()	Gets the broadcast address (all bits are 1).
static NetworkAddress	**MakeABroadcastAddress** (const NetworkAddress	Gets a broadcast address.

	&subnetAddress, const NetworkAddress &subnetMask)	
static NetworkAddress	**ReturnAnyAddress** ()	Gets any address (all bits are 0).
static NetworkAddress	**MakeSubnetMask** (const unsigned int numberPrefixBits)	Gets subnet mask.
static bool	**IsIpv4StyleAddressString** (const string &addressString)	Checks that description of IP address is IPv4 style.
Type	Member function	Description
static const unsigned int	**numberBits**	Number of address bits. (IPv4: 32, IPv6: 128)
static const NetworkAddress	**anyAddress**	Any address.
static const NetworkAddress	**broadcastAddress**	Broadcast address.
static const NetworkAddress	**invalidAddress**	Invalid address.

4.14. IP header related APIs

Source file: scensim_ipv4.h、scensim_ipv6.h

4.14.1. IpHeaderModel

Definition of IP header

Return type	Function(argument)	Description
	IpHeaderModel (const unsigned char trafficClass, const unsigned int payloadLengthBeforeIp, const unsigned char hopLimit, const unsigned char nextHeaderTypeCode, const NetworkAddress &sourceAddress, const NetworkAddress &destinationAddress)	Constructor of IpHeaderModel class.
const unsigned char *	**GetPointerToRawBytes** () const	Gets a pointer to header raw data.
unsigned int	**GetNumberOfRawBytes** () const	Gets IP header length in bytes.
unsigned int	**GetNumberOfTrailingBytes** ()	Gets trailing length in bytes.
unsigned char	**GetNextHeaderProtocolCode** () const	Gets next header protocol number. (For IPv6)
void	**SetFinalNextHeaderProtocolCode** (const unsigned char nextHeader)	Sets final next header protocol number. (For IPv6)
void	**AddBindingUpdateExtensionHeader** (const unsigned short sequenceNumber, const unsigned short lifetimein4SecUnits, const unsigned short bindingId)	Adds binding update extension header. (For IPv6)
void	**AddHomeAddressDestinationOptionsHeader** (const NetworkAddress	Adds home address destination options header.

	&homeAddress)	(For IPv6)
bool	**HasIpsecEspOverhead** () const	Checks that the IP header includes IPsec ESP. (For IPv6)
void	**AddIpsecEspOverhead** ()	Adds IPsec ESP. (For IPv6)

4.14.2. IpHeaderOverlayModel

IP header overlay

Return type	Function(argument)	Description
	IpHeaderOverlayModel (const unsigned char *initHeaderPtr, const size_t initPacketLength)	Constructor of IpHeaderOverlayModel class.
	IpHeaderOverlayModel (unsigned char *initHeaderPtr, const size_t initPacketLength)	Constructor of IpHeaderOverlayModel class.
void	**StopOverlayingHeader** () const	Stops IP header overlaying.
void	**GetHeaderTotalLengthAndNextHeaderProtocolCode** (unsigned int &headerLength, unsigned char &protocolCode) const	Gets total header length and next protocol number.
unsigned int	**GetLength** () const	Gets total header length.
unsigned char	**GetTrafficClass** () const	Gets traffic class.
unsigned short int	**GetFlowLabel** () const	Gets flow label. (For IPv6)
unsigned char	**GetNextHeaderProtocolCode** () const	Gets next header protocol number.
unsigned char	**GetHopLimit** () const	Get hop limit.
NetworkAddress	**GetSourceAddress** () const	Gets source address.
NetworkAddress	**GetDestinationAddress** () const	Gets destination address.
void	**SetTrafficClass** (const unsigned char trafficClass)	Gets traffic class.

void	**SetFlowLabel** (unsigned short int flowLabel)	Sets flow label. (For IPv6)
void	**SetHopLimit** (const unsigned char hopLimit)	Sets hop limit.
void	**SetSourceAddress** (const NetworkAddress &sourceAddress)	Sets source address.
void	**SetDestinationAddress** (const NetworkAddress &destinationAddress)	Sets destination address.
bool	**MobilityExtensionHeaderExists** () const	Checks that an extension header for mobile IP exists.
bool	**MobileIpBindingUpdateHeaderExists** () const	Checks that a binding update header exists.
const MobileIpBindingUpdateExtensionHeader &	**GetMobileIpBindingUpdateHeader** () const	Gets a binding update header. (For IPv6)
bool	**HomeAddressDestinationOptionsHeaderExists** () const	Checks that a home address destination options header exists.
NetworkAddress	**GetHomeAddressFromDestinationOptionsHeader** () const	Gets a home address options header. (For IPv6)

4.15. Routing table related APIs

Source file: scensim_network.h

4.15.1. RoutingTable

Routing table

Return type	Function(argument)	Description
void	**AddOrUpdateRoute** (const NetworkAddress &destinationAddress, const NetworkAddress &nextHopAddress, const unsigned int nextHopInterfaceIndex)	Adds or updates a routing entry without subnet mask.
void	**AddOrUpdateRoute** (const NetworkAddress &destinationAddress, const NetworkAddress &destinationAddressSubnetMask, const NetworkAddress &nextHopAddress, const unsigned int nextHopInterfaceIndex)	Adds or updates a routing entry.
void	**DeleteRoute** (const NetworkAddress &destinationAddress)	Deletes a routing entry without subnet mask.
void	**DeleteRoute** (const NetworkAddress &destinationAddress, const NetworkAddress &destinationAddressSubnetMask)	Deletes a routing entry.
void	**LookupRoute** (const NetworkAddress &destinationAddress, bool &foundRoute, NetworkAddress &nextHopAddress, unsigned int	Looks up routing table.

	&nextHopInterfaceIndex) const	

4.16. Output queue related APIs

Source file: scensim_queues.h

4.16.1. InterfaceOutputQueue

Abstract class for output queue

Return type	Function(argument)	Description
virtual bool	**InsertWithFullPacketInformationModeIsOn** () const	Checks that full packet information mode for inserting is on.
virtual void	**Insert** (unique_ptr< Packet > &packetPtr, const NetworkAddress &nextHopAddress, const PacketPriority priority, EnqueueResultType &enqueueResult, unique_ptr< Packet > &packetToDropPtr, const EtherTypeFieldId etherType=ETHERTYPE_IS_NOT_ SPECIFIED)=0	Inserts a packet into the output queue. (pure virtual function)
virtual void	**InsertWithFullPacketInformation** (unique_ptr< Packet > &packetPtr, const NetworkAddress &nextHopAddress, const NetworkAddress &sourceAddress, const unsigned short int sourcePort, const NetworkAddress &destinationAddress, const unsigned short int destinationPort, const unsigned char protocolCode, const PacketPriority priority, const unsigned short int ipv6FlowLabel, EnqueueResultType &enqueueResult, unique_ptr<	Inserts a packet into an output queue with full packet information. (pure virtual function)

	Packet > &packetToDropPtr)	
virtual PacketPriority	**MaxPossiblePacketPriority** () const	Gets max packet priority that the output queue can handle.
virtual bool	**IsEmpty** () const =0	Checks that the queue is empty. (pure virtual function)
virtual void	**DequeuePacket** (unique_ptr< Packet > &packetPtr, NetworkAddress &nextHopAddress, PacketPriority &priority, EtherTypeFieldId &etherType)=0	Dequeues a packet from the output queue. (pure virtual function)

4.17. MAC layer related APIs

Source file: scensim_mac.h

4.17.1. MacLayer

Abstract class for MAC layer

Return type	Function(argument)	Description
virtual void	**NetworkLayerQueueChangeNotification** ()=0	Gets a transmission queue change notification from network layer. (pure virtual function)
virtual void	**DisconnectFromOtherLayers** ()=0	Disconnects the holding smart pointers. (pure virtual function)
virtual GenericMacAddress	**GetGenericMacAddress** () const	Gets generic mac address.
virtual shared_ptr < MacQualityOfServiceControlInterface >	**GetQualityOfServiceInterface** () const	Gets a pointer to QoS control interface.
virtual shared_ptr < MacLayerInterfaceForEmulation >	**GetMacLayerInterfaceForEmulation** ()	Gets a pointer to MAC layer interface for emulation.
virtual shared_ptr < MacAndPhyInfoInterface >	**GetMacAndPhyInfoInterface** ()	Gets a pointer to MAC/PHY information interface.

4.17.2. MacAddressResolver

Abstract class for MAC address resolver

Return type	Function(argument)	Description
virtual void	**GetMacAddress** (const NetworkAddress &aNetworkAddress, const NetworkAddress &networkAddressMask, bool &wasFound, MacAddress &resolvedMacAddress)=0	Gets a MAC address for the specified network address. (pure virtual function)
virtual void	**GetNetworkAddressIfAvailable** (const MacAddress &macAddress, const NetworkAddress &subnetNetworkAddress, bool &wasFound, NetworkAddress &resolvedNetworkAddress)=0	Gets a network address for the specified MAC address. (pure virtual function)

4.18. Radio propagation related APIs

Source file: scensim_prop.h

4.18.1. SimplePropagationModelForNode

Interface class for radio propagation model

Return type	Function(argument)	Description
void	**TurnOnSignalsGetFramePtrSupport** ()	Turns on the function to obtain a pointer to frame. (For MIMO channel model)
bool	**PropagationDelayIsOn** () const	Checks that propagation delay is enabled.
void	**DisconnectThisInterface** ()	Disconnects this interface from channels.
bool	**IAmDisconnected** () const	Checks that this interface is disconnected from channels.
SimTime	**DisconnectTime** () const	Gets time when this interface is disconnected from channels.
bool	**IAmNotReceivingSignals** () const	Checks that this interface is not receiving any signals.
void	**StopReceivingSignals** ()	Stops receiving signals.
void	**StartReceivingSignals** ()	Starts receiving signals.
virtual NodeId	**GetNodeId** () const	Gets node ID.
virtual unsigned int	**GetInterfaceIndex** () const	Gets interface index.
InterfaceOrInstanceId	**GetInstanceId** ()	Gets channel instance ID.
virtual const AntennaModel &	**GetAntennaModel** () const	Gets a reference of antenna model.
virtual const shared_ptr < AntennaModel >	**GetAntennaModelPtr** ()	Gets a pointer to antenna model.

virtual ObjectMobilityModel &	**GetMobilityModel** () const	Gets a reference of mobility model.
virtual shared_ptr < ObjectMobilityModel >	**GetMobilityModelPtr** () const	Gets a pointer to mobility model.
virtual const ObjectMobilityPosition	**GetCurrentMobilityPosition** () const	Gets current position.
virtual bool	**ReceivedSignalPowerIncludesMyAntennaGain** () const	Checks that received signal power includes self antenna gain.
unsigned int	**GetBaseChannelNumber** () const	Gets base channel number.
unsigned int	**GetChannelCount** () const	Gets total number of channels.
double	**GetCarrierFrequencyMhz** (const unsigned int channelNumber) const	Gets carrier frequency for the specified channel in MHz.
double	**GetCarrierFrequencyMhz** () const	Gets carrier frequency for current channel in MHz.
double	**GetChannelBandwidthMhz** (const unsigned int channelNumber) const	Gets channel bandwidth for the specified channel in MHz.
double	**GetChannelBandwidthMhz** () const	Gets channel bandwidth for current channel in MHz.
virtual void	**TransmitSignal** (const double &txPowerDbm, const SimTime &duration, FrameType *&framePtr)	Transmits signal. (For single channel)
virtual void	**TransmitSignal** (const vector< unsigned int > &channelNumbers, const double &txPowerDbm, const SimTime &duration, FrameType *&framePtr)	Transmits signal. (For multiple channels)

double	**CalculatePathlossToLocation** (const double &positionXMeters, const double &positionYMeters, const double &positionZMeters) const	Calculates pathloss to the specified location.
void	**CalculatePathlossToLocation** (const PropagationInformationType &informationType, const double &positionXMeters, const double &positionYMeters, const double &positionZMeters, PropagationStatisticsType &propagationStatistics) const	Calculates pathloss to the specified location and gets path information.
unsigned int	**GetCurrentChannelNumber** () const	Gets current channel number.
const unsigned int	**CurrentlyReceivingFramesOnBondedChannels** ( ) const	Checks that currently receiving frames are on bonded channels.
const vector< unsigned int > &	**GetCurrentChannelNumberSet** () const	Gets current channel number sets. (For channel bonding)
bool	**IsOnChannel** (const unsigned int channelNum) const	Checks that the specified channel is in use.
bool	**ChannelIsBeingUsed** (const unsigned int channelNumber) const	Checks that the specified channel is used. (For emulation)
void	**SwitchToChannelNumber** (const unsigned int channelNumber, const bool doNotCalcInterferenceLevel=false)	Switches to the specified channel. (For single channel)
void	**SwitchToChannelNumbers** (const vector< unsigned int > channelNumberSet, const bool doNotCalcInterferenceLevel=false)	Switches to the specified channel. (For multiple channels)
void	**SetRelativeAntennaAttitudeAzimuth** (	Sets relative antenna attitude azimuth in degrees.

	const double &azimuthDegrees)	
SimTime	**GetLastTransmissionEndTime** () const	Gets the end time of last transmission.
const FrameType *	**GetCurrentlyTransmittingFramePtr** () const	Gets a pointer to currently transmitting frame.
double	**GetLastTransmissionPowerDbm** () const	Gets power of last transmission in dBm.
SimTime	**GetLastChannelSwitchTime** () const	Gets time of last channel switch.
void	**RegisterSignalHandler** (SignalHandler *initSignalHandlerPtr)	Registers a handler for signal start.
void	**RegisterSignalEndHandler** (SignalHandler *aSignalHandlerPtr)	Registers a hander for signal end.
void	**UnregisterSignalHandler** ()	Unregisters a handle for signal start.
void	**UnregisterSignalEndHandler** ()	Unregisters a handle for signal end.
void	**ReceiveIncomingSignal** (const IncomingSignal &aSignal)	Starts receiving signal.
void	**ReceiveIncomingSignalEnd** (const IncomingSignal &aSignal)	Completes receiving signal.
virtual void	**SetMobilityModel** (const shared_ptr< ObjectMobilityModel > &newMobilityModelPtr)	Sets mobility model.
unsigned int	**CurrentThreadPartitionIndex** () const	Gets current thread partition index.
shared_ptr < SimplePropagationModel < FrameType > >	**GetPropagationModel** () const	Gets propagation model.
double	**GetDistanceMetersTo** (const NodeId &otherNodeId) const	Gets distance to the specified node in meters.

4.18.2. SimplePropagationModelForNode::IncomingSignal

Signal

Return type	Function(argument)	Description
	IncomingSignal (const unsigned int channelNumber, const NodeId &sourceNodeId, const SimTime &startTime, const SimTime &duration, const double transmittedPowerDbm, const double receivedPowerDbm, const shared_ptr< const FrameType > &framePtr, const bool initIsANoiseFrame)	Constructor of IncomingSignal class for single channel. (For smart pointer)
	IncomingSignal (const vector< unsigned int > &channelNumbers, const NodeId &sourceNodeId, const SimTime &startTime, const SimTime &duration, const double transmittedPowerDbm, const double receivedPowerDbm, const shared_ptr< const FrameType > &framePtr, const bool initIsANoiseFrame)	Constructor of IncomingSignal class. (For smart pointer)
	IncomingSignal (const unsigned int channelNumber, const NodeId &sourceNodeId, const SimTime &startTime, const SimTime &duration, const double transmittedPowerDbm, const double receivedPowerDbm, const FrameType *framePtr, const bool initIsANoiseFrame)	Constructor of IncomingSignal class for single channel. (For raw pointer)
	IncomingSignal (	Constructor of IncomingSignal class

	const vector< unsigned int > &channelNumbers, const NodeId &sourceNodeId, const SimTime &startTime, const SimTime &duration, const double transmittedPowerDbm, const double receivedPowerDbm, const FrameType *framePtr, const bool initIsANoiseFrame)	for single channel. (For raw pointer)
NodeId	**GetSourceNodeId** () const	Gets source node ID.
unsigned int	**GetChannelNumber** () const	Gets channel number.
unsigned int	**GetNumberBondedChannels** () const	Gets number of bonded channels.
unsigned int	**GetBondedChannelNumber** (const unsigned int channelIndex)	Gets channel number for the specified channel index.
bool	**IsOnChannel** (const unsigned int channelNum) const	Checks that the specified channel is in use.
bool	**ChannelIntersectionIsEmpty** (const vector< unsigned int > &receivedChannels) const	Checks that the specified channels have nothing in common.
SimTime	**GetStartTime** () const	Gets start time of signal transmission.
SimTime	**GetDuration** () const	Gets duration of signal transmission.
double	**GetTransmittedPowerDbm** () const	Gets transmitted power in dBm.
double	**GetReceivedPowerDbm** () const	Gets received power in dBm.
double	**GetPathlossDb** () const	Gets pathloss in dB.
bool	**HasACompleteFrame** () const	Checks that the signal has a complete frame.
bool	**HasAFrame** () const	Checks that the signal has a frame.
const FrameType &	**GetFrame** () const	Gets a reference of frame.
shared_ptr< const FrameType >	**GetFrame** () const	Gets a smart pointer to frame.

4.18.3. SimplePropagationModelForNode::SignalHandler

Signal handler

Return type	Function(argument)	Description
virtual void	**ProcessSignal** (const IncomingSignal &aSignal)=0	Starts or ends signal reception. (pure virtual function)

4.18.4. SimplePropagationModel

Propagation environment

Return type	Function(argument)	Description
	SimplePropagationModel (const ParameterDatabaseReader &theParameterDatabaseReader, const RandomNumberGeneratorSeed &runSeed, const shared_ptr< SimulationEngine > &simulationEnginePtr, const shared_ptr< GisSubsystem > &gisSubsystemPtr, const InterfaceOrInstanceId &instanceId=nullInstanceId, const bool takeOwnershipOfFrames=false)	Constructor of SimplePropagationModel class.
void	**TurnOnSignalsGetFramePtrSupport** ()	Turns on the function to obtain a pointer to frame. (For MIMO channel model)
bool	**PropagationDelayIsOn** () const	Checks that propagation delay is enabled.
InterfaceOrInstanceId	**GetInstanceId** ()	Gets channel instance ID.

void	**DisconnectNodeInterface** (const unsigned int channelNumber, const shared_ptr< SimplePropagationModelForNode< FrameType > > &interfacePtr)	Disconnects the node interface from the specified channel.
void	**StopReceivingSignalsAtNode** (const unsigned int channelNumber, const shared_ptr< SimplePropagationModelForNode< FrameType > > &interfacePtr)	Stops receiving signals at the specified node.
void	**StartReceivingSignalsAtNod**e (const unsigned int channelNumber, const shared_ptr< SimplePropagationModelForNode< FrameType > > &interfacePtr)	Starts receiving signals at the specified node.
void	**AddNodeToChannel** (const unsigned int channelNumber, const shared_ptr< SimplePropagationModelForNode< FrameType > > &nodeInterfacePtr)	Adds the specified node to the specified channel.
void	**InvalidateCachedInformationFor** (const SimplePropagationModelForNode< FrameType > &nodeInfo)	Invalidates cached information for the specified node.
void	**DeleteNodeFromChanne**l (const unsigned int channelNumber, const shared_ptr< SimplePropagationModelForNode< FrameType > > &nodeInterfacePtr)	Deletes the node interface from the specified channel.
virtual shared_ptr < SimplePropagationModelForNode < FrameType > >	**GetNewPropagationModelInterface** (const shared_ptr< SimulationEngineInterface > &simEngineInterfacePtr, const shared_ptr< AntennaModel > &antennaModelPtr, const	Gets new propagation model interface for the specified interface index.

	shared_ptr< ObjectMobilityModel > &antennaMobilityModelPtr, const NodeId &nodeId, const InterfaceId &interfaceId, const unsigned int interfaceIndex)	
virtual shared_ptr < SimplePropagationModelForNode < FrameType > >	**GetNewPropagationModelInterface** (const shared_ptr< SimulationEngineInterface > &simEngineInterfacePtr, const shared_ptr< AntennaModel > &antennaModelPtr, const shared_ptr< ObjectMobilityModel > &antennaMobilityModelPtr, const NodeId &nodeId)	Gets new propagation model interface.
double	**GetCarrierFrequencyMhz** (const unsigned int channelNumber) const	Gets carrier frequency for the specified channel in MHz.
double	**GetChannelBandwidthMhz** (const unsigned int channelNumber) const	Gets channel bandwidth for the specified channel in MHz.
unsigned int	**GetBaseChannelNumber () const**	Gets base channel number.
unsigned int	**GetChannelCount** () const	Gets total number of channels.
bool	**ChannelIsBeingUsed** (const unsigned int channelNumber) const	Checks that the specified channel is used. (For emulation)
shared_ptr < SimplePropagationLossCalculationModel >	**GetPropagationCalculationModel** () const	Gets a pointer to pathloss calculation model.

void	**CalculatePathlossToNode** (const PropagationInformationType &informationType, const ObjectMobilityPosition &txAntennaPosition, const ObjectMobilityModel::MobilityObjectId &txObjectId, const AntennaModel &txAntennaModel, const ObjectMobilityPosition &rxAntennaPosition, const ObjectMobilityModel::MobilityObjectId &rxObjectId, const AntennaModel &rxAntennaModel, const unsigned int channelNumber, PropagationStatisticsType &propagationStatistics) const	Calculates pathloss to the specified node.
void	**CalculatePathlossToLocation** (const PropagationInformationType &informationType, const SimTime &currentTime, const SimplePropagationModelForNode< FrameType > &transmittingNodeInfo, const double &positionXMeters, const double &positionYMeters, const double &positionZMeters, const unsigned int channelNumber, PropagationStatisticsType &propagationStatistics) const	Calculates pathloss to the specified location.
void	**SwitchToChannelNumber** (const shared_ptr< SimplePropagationModelForNode< FrameType > > &switchingNodeInfoPtr, const unsigned int channelNumber, const bool doNotCalcInterferrenceLevel)	Switches to specified channel. (For single channel)

void	**SwitchToChannelNumbers** (const shared_ptr< SimplePropagationModelForNode< FrameType > > &receivingNodeInfoPtr, const vector< unsigned int > &channelNumbers, const bool doNotCalcInterferenceLevel)	Switches to specified channels. (For multiple channels)
protected		
void	**ScheduleSignalEventAtNode** (SimulationEngineInterface &simEngineInterface, const NodeId &txNodeId, const vector< unsigned int > &channelNumbers, const shared_ptr< SimplePropagationModelForNode< FrameType > > &receivingNodeInfoPtr, const double &transmitPowerDbm, const double &receivedPowerDbm, const SimTime &currentTime, const SimTime &propagationDelay, const SimTime &duration, const FrameType *framePtr, const bool isANoiseFrame=false)	Schedules signal receiving start or end event. (For channel bonding)
void	**ScheduleSignalEventAtNode** (SimulationEngineInterface &simEngineInterface, const NodeId &txNodeId, const unsigned int channelNumber, const shared_ptr< SimplePropagationModelForNode< FrameType > > &receivingNodeInfoPtr, const double &transmitPowerDbm, const double &receivedPowerDbm, const SimTime &currentTime, const	Schedules signal receiving start or end event.

	SimTime &propagationDelay, const SimTime &duration, const FrameType *framePtr, const bool isANoiseFrame=false)	
bool	**NodeCanHearSignal** (const SimplePropagationModelForNode< FrameType > &receivingNodeInfo, const IncomingSignal &aSignal) const	Checks that the specified node listens the specified signal.
void	**SendSignalStartEventToNode** (SimplePropagationModelForNode< FrameType > &receivingNodeInfo, IncomingSignal *aSignalPtr)	Sends signal receiving start event to the specified node.
void	**SendSignalEndEventToNode** (SimplePropagationModelForNode< FrameType > &receivingNodeInfo, IncomingSignal *aSignalPtr)	Sends signal receiving end event to the specified node.
void	**DeleteIncomingSignal** (SimplePropagationModelForNode< FrameType > &receivingNodeInfo, IncomingSignal *aSignalPtr)	Deletes incoming signal form the designated node.
void	**TransmitSignalToSingleNode** (SimulationEngineInterface &simEngineInterface, const NodeId &txNodeId, const unsigned int txInterfaceIndex, const vector< unsigned int > &channelNumbers, const ObjectMobilityPosition &txAntennaPosition, const AntennaModel &txAntennaModel, const double &transmitPowerDbm, const SimTime &currentTime, const SimTime &duration, const FrameType *framePtr, const shared_ptr<	Transmits signal to the specified node. (For multiple channels)

	SimplePropagationModelForNode< FrameType > > &receivingNodeInfoPtr, const bool isANoiseFrame=false)noPropagation Delay=false)	
void	**TransmitSignalToSingleNode** (SimulationEngineInterface &simEngineInterface, const NodeId &txNodeId, const unsigned int txInterfaceIndex, const unsigned int txChannelNumber, const ObjectMobilityPosition &txAntennaPosition, const AntennaModel &txAntennaModel, const double &transmitPowerDbm, const SimTime &currentTime, const SimTime &duration, const FrameType *framePtr, const shared_ptr< SimplePropagationModelForNode< FrameType > > &receivingNodeInfoPtr, const bool isANoiseFrame=false)	Transmits signal to the specified node. (For single channel)
void	**TransmitSignalInLocalPartition** (SimulationEngineInterface &simEngineInterface, const NodeId &txNodeId, const unsigned int txInterfaceIndex, const unsigned int txChannelNumber, const ObjectMobilityPosition &txAntennaPosition, const AntennaModel &txAntennaModel, const double &transmitPowerDbm, const SimTime &currentTime, const SimTime &duration, const FrameType *framePtr)	Transmits signal. (For single channel)

void	**TransmitSignalInLocalPartition** (SimulationEngineInterface &simEngineInterface, const NodeId &txNodeId, const unsigned int txInterfaceIndex, const vector< unsigned int > &channelNumbers, const ObjectMobilityPosition &txAntennaPosition, const AntennaModel &txAntennaModel, const double &transmitPowerDbm, const SimTime &currentTime, const SimTime &duration, const FrameType *framePtr)	Transmits signal. (For multiple channels)
void	**TransmitSignalInLocalPartitionUtilizingMultipleThreads** (SimulationEngineInterface &simEngineInterface, const NodeId &txNodeId, const unsigned int txChannelNumber, const ObjectMobilityPosition &txAntennaPosition, const AntennaModel &txAntennaModel, const double &transmitPowerDbm, const SimTime &currentTime, const SimTime &duration, const FrameType *framePtr)	Transmits signal. (For single channel) (For multithreading)
void	**TransmitSignalInLocalPartitionUtilizingMultipleThreads** (SimulationEngineInterface &simEngineInterface, const NodeId &txNodeId, const vector< unsigned int > &channelNumbers, const ObjectMobilityPosition &txAntennaPosition, const AntennaModel &txAntennaModel, const double &transmitPowerDbm,	Transmits signal. (For multiple channel) (For multithreading)

	const SimTime &currentTime, const SimTime &duration, const FrameType *framePtr)	
void	**TransmitSignal** (SimulationEngineInterface &simEngineInterfaceForTxNode, SimplePropagationModelForNode< FrameType > &transmittingNodeInfo, const unsigned int channelNumber, const double &transmitPowerDbm, const SimTime &duration, FrameType *framePtr)	Transmits signal.
void	**TransmitSignal** (SimulationEngineInterface &simEngineInterfaceForTxNode, SimplePropagationModelForNode< FrameType > &transmittingNodeInfo, const vector< unsigned int > &channelNumbers, const double &transmitPowerDbm, const SimTime &duration, FrameType *framePtr)	Transmits signal. (For bonded channels)
void	**TransmitChannelInterferenceSignals** (SimulationEngineInterface &simEngineInterface, const NodeId &txNodeId, const unsigned int txInterfaceIndex, const unsigned int txChannelNumber, const ObjectMobilityPosition &txAntennaPosition, const AntennaModel &txAntennaModel, const double &transmitPowerDbm, const SimTime &currentTime, const SimTime &duration)	Transmits channel interference signals. (For single channel)
void	**TransmitChannelInterferenceSignals** (	Transmits channel interference signals.

	SimulationEngineInterface &simEngineInterface, const NodeId &txNodeId, const unsigned int txInterfaceIndex, const vector< unsigned int > &txChannelNumbers, const ObjectMobilityPosition &txAntennaPosition, const AntennaModel &txAntennaModel, const double &transmitPowerDbm, const SimTime &currentTime, const SimTime &duration)	

4.19. Pathloss related APIs

Source file: scensim_proploss.h/cpp

4.19.1. SimplePropagationLossCalculationModel

Abstract class for pathloss model

Return type	Function(argument)	Description
	SimplePropagationLossCalculationModel (const double &carrierFrequencyMhz, const double &maximumPropagationDistanceMeters=DBL_MAX, const bool propagationDelayIsEnabled=false, const int numberDataParallelThreads=0)	Constructor of SimplePropagationLossCalculationModel class.
double	**GetCarrierFrequencyMhz** () const	Gets carrier frequency in MHz.
virtual double	**CalculatePropagationLossDb** (const ObjectMobilityPosition &txAntennaPosition, const MobilityObjectId &txObjectId, const ObjectMobilityPosition &rxAntennaPosition, const MobilityObjectId &rxObjectId, const double &xyDistanceSquaredMeters) const	Calculates pathloss between the specified nodes in dB.
virtual double	**CalculatePropagationLossDbParallelVersion** (const ObjectMobilityPosition &txAntennaPosition, const ObjectMobilityModel::MobilityObjectId &txObjectId, const ObjectMobilityPosition &rxAntennaPosition, const	Calculates pathloss between the specified nodes in dB. (For multithreading)

	ObjectMobilityModel::MobilityObjectId &rxObjectId, const double &xyDistanceSquaredMeters, const int calculationThreadId) const	
virtual void	**SetTimeTo** (const SimTime &time)	Sets time for trace based (pre calculated) model.
virtual bool	**PropagationLossIsSymmetricValue** () const	Checks that pathloss values are symmetric for Tx and Rx.
virtual boo l	**SupportMultipointCalculation** () const	Checks that the pathloss calculation model supports multipoint calculation.
virtual void	**CacheMultipointPropagationLossDb** (const SimTime &currentTime, const ObjectMobilityPosition &txAntennaPosition, const vector< ObjectMobilityPosition > &rxAntennaPositions)	Caches multipoint propagation loss in dB.
bool	**IsCloserThanMaxPropagationDistance** (const ObjectMobilityPosition &txAntennaPosition, const ObjectMobilityPosition &rxAntennaPosition) const	Checks that the distance of the specified nodes is closer than max propagation distance.
void	**CalculateOrRetrieveTotalLossDbAndPropDelay** (SignalLossCache &aSignalLossCache, const SimTime &currentTime, const unsigned int channelNumber, const NodeId &txNodeId, const unsigned int txInterfaceIndex, const ObjectMobilityPosition &txAntennaPosition, const AntennaModel &txAntenna, const	Calculates or retrieves total propagation loss and propagation delay.

	NodeId &rxNodeId, const unsigned int rxInterfaceIndex, ObjectMobilityModel &rxMobilityModel, const AntennaModel &rxAntenna, double &totalLossDb, SimTime &propagationDelay)	
double	**CalculateTotalAntennaGainDbi** (const ObjectMobilityPosition &txAntennaPosition, const AntennaModel &txNodeAntenna, const ObjectMobilityPosition &rxAntennaPosition, const AntennaModel &rxNodeAntenna) const	Calculates total antenna gain in dBi.
double	**CalculateAntennaGainDbi** (const ObjectMobilityPosition &antennaPosition, const AntennaModel &nodeAntenna, const double destX, const double destY, const double destZ) const	Calculates antenna gain in dBi.
virtual void	**CalculatePropagationPathInformation** (const PropagationInformationType &informationType, const ObjectMobilityPosition &txAntennaPosition, const MobilityObjectId &txObjectId, const AntennaModel &txAntennaModel, const ObjectMobilityPosition &rxAntennaPosition, const MobilityObjectId &rxObjectId, const AntennaModel &rxAntennaModel, PropagationStatisticsType &propagationStatistics) const	Calculates pathloss and ray path information.

void	**ClearParallelCalculationSet** ()	Clears pathloss information container for multithreading.
void	**AddToParallelCalculationSet** (const ObjectMobilityPosition &txAntennaPosition, const AntennaModel *txAntennaModelPtr, const size_t &rxNodeIndex, const ObjectMobilityPosition &rxAntennaPosition, const shared_ptr< AntennaModel > &rxAntennaModelPtr)	Adds pathloss information container for multithreading.
void	**CalculateLossesDbAndPropDelaysInParallel** ()	Calculates total propagation loss in dB and propagation delay. (For multithreading)
unsigned int	**GetNumberOfParallelCalculations** () const	Gets number of parallel calculations. (For multithreading)
double	**GetTotalLossDb** (const size_t jobIndex) const	Gets total propagation loss in dB. (For multithreading)
SimTime	**GetPropagationDelay** (const size_t JobIndex) const	Gets propagation delay. (For multithreading)
NodeId	**GetTxNodeId** (const size_t jobIndex) const	Gets Tx node ID. (For multithreading)
ObjectMobilityPosition &	**GetTxAntennaPosition** (const size_t jobIndex) const	Gets Tx antenna position. (For multithreading)
unsigned int	**GetRxNodeIndex** (const size_t jobIndex) const	Gets Rx node index. (For multithreading)
ObjectMobilityPosition &	**GetRxAntennaPosition** (const size_t jobIndex) const	Gets Rx antenna position. (For multithreading)
virtual bool	**RayPathTraceIsEnabled** () const	Checks that ray path trace function is enabled. (For HFPM)
virtual double	**CalculateTotalLossDbWithRayPathTrace** (	Calculates total propagation loss in dB with ray path trace.

	const ObjectMobilityPosition &txAntennaPosition, const MobilityObjectId &txNodeId, const AntennaModel &txNodeAntenna, const ObjectMobilityPosition &rxAntennaPosition, const MobilityObjectId &rxNodeId, const AntennaModel &rxNodeAntenna, double &totalLossDb)	(For HFPM)
protected		
virtual double	**CalculatePropagationLossDb** (const ObjectMobilityPosition &txAntennaPosition, const ObjectMobilityPosition &rxAntennaPosition, const double &xyDistanceSquaredMeters) const =0	Calculates propagation loss in dB for specified two nodes. (pure virtual function)

4.20. Antenna related APIs

Source file: scensim_proploss.h

4.20.1. AntennaModel

Abstract class for antenna model

Return type	Function(argument)	Description
virtual bool	**IsOmniDirectional** () const =0	Checks that the antenna is omni directional. (pure virtual function)
virtual double	**GetOmniGainDbi** () const =0	Gets gain in dBi for omni antenna. (pure virtual function)
virtual double	**GainInDbForThisDirection** (const double &azimuthFromBoresightClockwiseDegrees=0.0, const double &elevationFromBoresightDegrees=0.0, const double &currentAntennaRotation=0.0) const =0	Gets gain for this direction in dBi (pure virtual function)
virtual bool	**SupportsQuasiOmniMode** () const	Checks that the antenna supports quasi omni mode.
virtual bool	**IsInQuasiOmniMode** () const	Checks that the antenna is in quasi omni mode.
virtual void	**SwitchToQuasiOmniMode** ()	Switches to quasi omni mode.
virtual void	**SwitchToDirectionalMode** ()	Switches to directional mode.

4.21. Mobility related APIs

Source file: scensim_proploss.h, scensim_mobility.h

4.21.1. ObjectMobilityPosition

Definition of object position

Return type	Function(argument)	Description
	ObjectMobilityPosition (const SimTime &initLastMoveTime, const SimTime &initEarliestNextMoveTime, const double &initXPositionMeters, const double &initYPositionMeters, const double &initTheHeightFromGroundMeters, const bool &initTheHeightContainsGroundHeightMeters, const double &initAttitudeAzimuthDegrees, const double &initAttitudeElevationDegrees, const double &initVelocityMetersPerSecond, const double &initVelocityAzimuthDegrees, const double &initVelocityElevationDegrees)	Constructor of ObjectMobilityPosition class.
SimTime	**LastMoveTime** () const	Gets time of last move.
SimTime	**EarliestNextMoveTime** () const	Gets time of earliest next move.
double	**X_PositionMeters** () const	Gets X position in meters.
double	**Y_PositionMeters** () const	Gets Y position in meters.
double	**HeightFromGroundMeters** () const	Gets height from ground in meters.
bool	**TheHeightContainsGroundHeightMeters** () const	Checks that the height contains ground height.

double	**AttitudeAzimuthFromNorthClockwiseDegrees** () const	Gets attitude azimuth from north clockwise in degrees.
double	**AttitudeElevationFromHorizonDegrees** () const	Gets attitude elevation from horizon in degrees.
double	**VelocityMetersPerSecond** () const	Gets velocity (speed) in m/s.
double	**VelocityAzimuthFromNorthClockwiseDegrees** () const	Gets velocity azimuth from north clockwise in degrees.
double	**VelocityElevationFromHorizonDegrees** () const	Gets velocity elevation from horizon in degrees.
void	**SetLastMoveTime** (const SimTime &lastMoveTime)	Sets time of last move.
void	**SetEarliestNextMoveTime** (const SimTime &nextMoveTime)	Sets time of earliest next move.
void	**SetX_PositionMeters** (const double &newXPosition)	Sets X position in meters.
void	**SetY_PositionMeters** (const double &newYPosition)	Sets Y position in meters.
void	**SetHeightFromGroundMeters** (const double &newHeight)	Sets height from ground in meters.
void	**SetTheHeightContainsGroundHeightMeters** (const bool newTheHeightContainsGroundHeightMeters)	Sets a flag that a height contains ground.
void	**SetAttitudeFromNorthClockwiseDegrees** (const double &newAttitude)	Sets attitude azimuth from north clockwise in degrees.
void	**SetAttitudeElevationFromHorizonDegrees** (const double &newElevation)	Sets attitude elevation from horizon in degrees.
void	**SetVelocityMetersPerSecond** (const double &newVelocity)	Sets velocity (speed) in m/s.
void	**SetVelocityFromNorthClockwiseDegrees** (const double &newVelocityAzimuth)	Sets velocity azimuth from north clockwise in degrees.
void	**SetVelocityElevationFromHorizon**	Sets velocity elevation from horizon

	Degrees (const double &newVelocityElevation)	in degrees.

4.21.2. ObjectMobilityModel

Abstract class for mobility model.

Return type	Function(argument)	Description
	ObjectMobilityModel (const ParameterDatabaseReader &theParameterDatabaseReader, const NodeId &nodeId, const InterfaceOrInstanceId &interfaceId)	Constructor of ObjectMobilityModel class.
void	**GetPositionForTime** (const SimTime &snapshotTime, ObjectMobilityPosition &position)	Gets position at the specified time.
virtual void	**GetUnadjustedPositionForTime** (const SimTime &snapshotTime, ObjectMobilityPosition &position)=0	Gets position that is unadjusted direction at the specified time.
virtual SimTime	**GetCreationTime** () const	Gets time of node creation.
virtual SimTime	**GetDeletionTime** () const	Gets time of node deletion.
void	**SetRelativeAttitudeAzimuth** (const SimTime &currentTime, const double &azimuthDegrees)	Sets relative attitude azimuth in degrees.
double	**GetRelativeAttitudeAzimuth** () const	Gets relative attitude azimuth in degrees.