



SCENARGIE®

Scenargie® 2.1 プログラマーズガイド

Space-Time Engineering, LLC

2016 年 9 月

目次

はじめに.....	1
1. Scenargie 概要.....	2
1.1. アーキテクチャ.....	2
1.2. システム構成.....	3
1.3. クラス構成.....	4
1.3.1. シミュレーションエンジンに関するクラス構成.....	4
1.3.2. システムモデルに関するクラス構成.....	4
1.3.3. インスタンス化.....	5
1.4. レイヤ間インタフェース.....	7
1.5. シナリオファイル.....	9
2. シミュレーション共通機能.....	10
2.1. ノード ID.....	10
2.2. シミュレーション時間.....	12
2.2.1. TimeType 型.....	12
2.2.2. シミュレーション時刻の取得.....	12
2.3. シミュレーションイベント.....	13
2.3.1. シミュレーションイベントの実行方法.....	13
2.3.2. シミュレーションイベントのリスケジュール、キャンセル方法.....	15
2.3.3. シミュレーションイベント実行時のエラー.....	16
2.4. パケット.....	17
2.4.1. パケット作成方法.....	17
2.4.2. ヘッダの追加方法.....	19
2.4.3. ヘッダの読み込みおよび削除方法.....	21
2.4.4. ペイロードの読み込み方法.....	22
2.4.5. パケットへの外部情報付与方法.....	23
2.4.6. パケット ID.....	25
2.5. 乱数生成器.....	26
2.6. パラメータ.....	30
2.7. 統計値出力.....	32
2.8. トレース出力.....	32
3. システムモデル.....	33
3.1. ネットワークシミュレータおよび通信ノード.....	33
3.2. アプリケーションレイヤ.....	37
3.2.1. 概要.....	37
3.2.2. アプリケーションの作成.....	38

3.2.3.	アプリケーションの追加	47
3.4.	ネットワークレイヤ	59
3.5.	MAC/PHY レイヤ	63
3.6.	電波伝搬	67
3.6.1.	パスロスモデル	67
3.6.2.	アンテナモデル	70
3.7.	モビリティモデル	73
3.8.	GIS データアクセス	76
4.	API リスト	78
4.1.	シミュレーションエンジン関連	78
4.1.1.	SimulationEvent	78
4.1.2.	EventRescheduleTicket	78
4.1.3.	SimulationEngineInterface	79
4.1.4.	SimulationEngine	82
4.2.	パケット関連	84
4.2.1.	Packet	84
4.2.2.	ExtrinsicPacketInformation	88
4.2.3.	PacketIdType	89
4.3.	乱数関連	90
4.3.1.	RandomNumberGenerator	90
4.3.2.	HighQualityRandomNumberGenerator	90
4.3.3.	ユーティリティ関数	91
4.4.	パラメータ関連	93
4.4.1.	ParameterDatabaseReader	93
4.5.	BER 関連	99
4.5.1.	BitOrBlockErrorRateCurveDatabase	99
4.5.2.	BitErrorRateCurve	99
4.5.3.	BlockErrorRateCurve	100
4.6.	統計値関連	101
4.6.1.	CounterStatistic	101
4.6.2.	RealStatistic	101
4.7.	ユーティリティ関連	102
4.7.1.	ユーティリティ関数	102
4.8.	ネットワークシミュレータ関連	105
4.8.1.	NetworkSimulator	105
4.9.	ネットワークノード関連	109

4.9.1.	NetworkNode	109
4.10.	アプリケーションレイヤ関連	112
4.10.1.	ApplicationLayer	112
4.10.2.	Application	113
4.11.	トランスポートレイヤ関連	115
4.11.1.	ProtocolPacketHandler	115
4.11.2.	TransportLayer	115
4.11.3.	UdpProtocol	116
4.11.4.	UdpProtocol::PacketForAppFromTransportLayerHandler	118
4.11.5.	TcpProtocol	118
4.11.6.	ConnectionFromTcpProtocolHandler	121
4.11.7.	TcpConnection	121
4.11.8.	TcpConnection::AppTcpEventHandler	122
4.12.	ネットワークレイヤ関連	124
4.12.1.	NetworkLayer	124
4.12.2.	BasicNetworkLayer	129
4.13.	ネットワークアドレス関連	135
4.13.1.	NetworkAddress	135
4.14.	IP ヘッダ関連	138
4.14.1.	IpHeaderModel	138
4.14.2.	IpHeaderOverlayModel	139
4.15.	ルーティングテーブル関連	141
4.15.1.	RoutingTable	141
4.16.	送信キュー関連	143
4.16.1.	InterfaceOutputQueue	143
4.17.	MAC レイヤ関連	145
4.17.1.	MacLayer	145
4.17.2.	MacAddressResolver	145
4.18.	電波伝搬関連	147
4.18.1.	SimplePropagationModelForNode	147
4.18.2.	SimplePropagationModelForNode::IncomingSignal	150
4.18.3.	SimplePropagationModelForNode::SignalHandler	152
4.18.4.	SimplePropagationModel	153
4.19.	パスロス関連	163
4.19.1.	SimplePropagationLossCalculationModel	163
4.20.	アンテナ関連	168

4.20.1.	AntennaModel.....	168
4.21.	モビリティ関連.....	169
4.21.1.	ObjectMobilityPosition.....	169
4.21.2.	ObjectMobilityModel.....	171

はじめに

本書は、統合シミュレーションフレームワーク Scenargie のソースコードを改変しシステムモデル等をカスタマイズするにあたって、Scenargie の構造や API について情報を提供するものである。関連資料である「Scenargie Base Simulator ユーザガイド」、「Scenargie Base Simulator モデルリファレンス」もあわせて参照いただきたい。

関連ドキュメント

インストレーションガイド
Visual Lab ユーザガイド
Base Simulator ユーザガイド
Base Simulator モデルリファレンス
Dot Eleven Module ユーザガイド
LTE Module ユーザガイド
ITS Extension Module ユーザガイド
Multi-Agent Extension Module ユーザガイド
Multi-Agent Extension Module モデルリファレンス
Fast Urban Propagation Module ユーザガイド
High Fidelity Propagation Module ユーザガイド
Trace Analyzer ユーザガイド
Emulation Module ユーザガイド

1. Scenargie 概要

本章では、Scenargie シミュレータの概要について説明する。

1.1. アーキテクチャ

Scenargie は、イベント駆動による離散事象シミュレータである。図 1-1 に、離散事象シミュレータによるシミュレーション実行の概念図を示す。

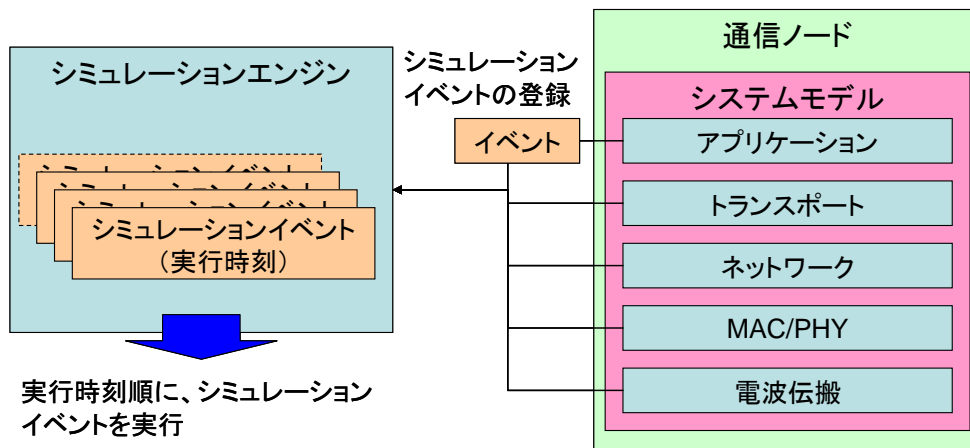


図 1-1. 離散事象シミュレータによるシミュレーションの実行

離散事象シミュレータでは、システムシミュレーションにおける一塊の処理をシミュレーションイベントとして定義し、実行時刻とともにシミュレーションエンジンに登録する。シミュレーションエンジンでは、登録されたシミュレーションイベントのうち実行時間の早いものから順次実行する。シミュレーションイベントの実行中には、システムモデルの内容に応じて、新たなシミュレーションイベントが登録されたり、登録済みのシミュレーションイベントが削除される。例えば、アプリケーションレイヤにおいてある時刻にパケットを送信するという処理が、一つのシミュレーションイベントとして登録される。

1.2. システム構成

図 1-2 は、Scenargie のシステム構成を示している。ネットワークシミュレータは、シミュレーション対象の通信ノードを複数保持するとともに、シミュレーションモデルで利用される GIS データアクセス機能、シミュレーション結果としてトレース出力、統計値出力の各機能を持つ。また、Scenargie の GUI である VisualLab とのインタフェースを保持する。

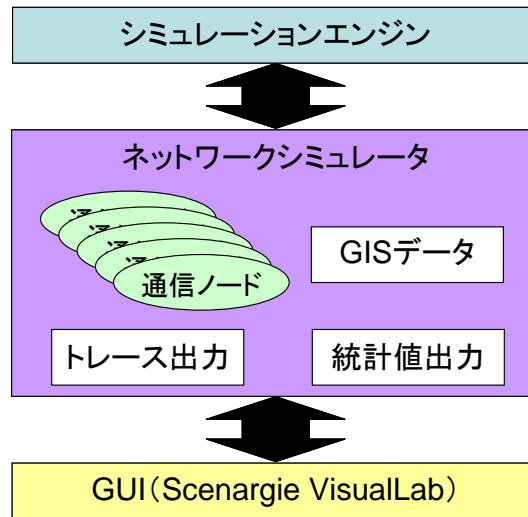


図 1-2. Scenargie のシステム構成

1.3. クラス構成

1.3.1. シミュレーションエンジンに関するクラス構成

図 1-3 は、シミュレーションエンジンに関するクラス構成を示している。シミュレーションエンジン (SimulationEngine クラス) は、プログラム内で一つだけ生成され、登録されたシミュレーションイベント (SimulationEvent クラス) の処理を行う。各ノード (NetworkNode クラス) は、シミュレーションエンジンへのインタフェース (SimulationEngineInterface クラス) を保持しており、シミュレーションイベントの登録などシミュレーションエンジンに関する操作を行う。

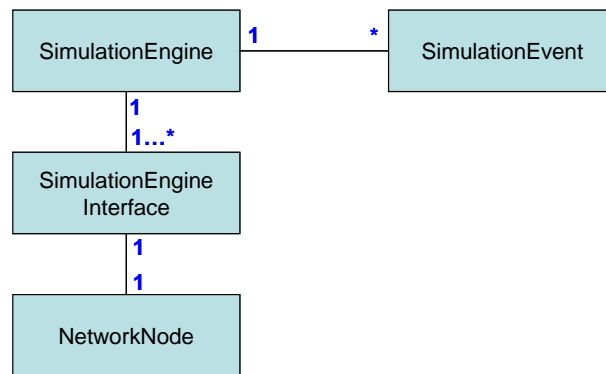


図 1-3. シミュレーションエンジンに関するクラス構成

1.3.2. システムモデルに関するクラス構成

図 1-4 は、システムモデルに関するクラス構成を示している。ネットワークシミュレータ (NetworkSimulator クラス) は、複数のノード (NetworkNode クラス) を保持しており、各ノードは、アプリケーションレイヤ (ApplicationLayer クラス)、トランスポートレイヤ (TransportLayer)、ネットワークレイヤ (NetworkLayer クラス) を保持している。また、アプリケーションレイヤは、複数のアプリケーション (Application クラス) を保持し、トランスポートレイヤは、TCP (TcpProtocol クラス)、UDP (UdpProtocol クラス) を保持している。さらに、ネットワークレイヤは、通信インタフェース毎に、MAC レイヤ (MacLayer クラス) を保持している。

無線環境のシミュレーションを行うには、電波伝搬環境 (SimplePropagationModel クラス) をネットワークシミュレータが保持しており、各ノードは、インタフェース (SimplePropagationModelForNode クラス) を経由してアクセスを行う。電波伝搬環境 (SimplePropagationModel クラス) は、システムモデルによって一つの場合や複数の場合がある。

尚、NetworkSimulator クラス、NetworkNode クラス、Application クラス、NetworkLayer クラス、MacLayer クラスは、基底クラスであり通常独自の継承クラスを作成する。

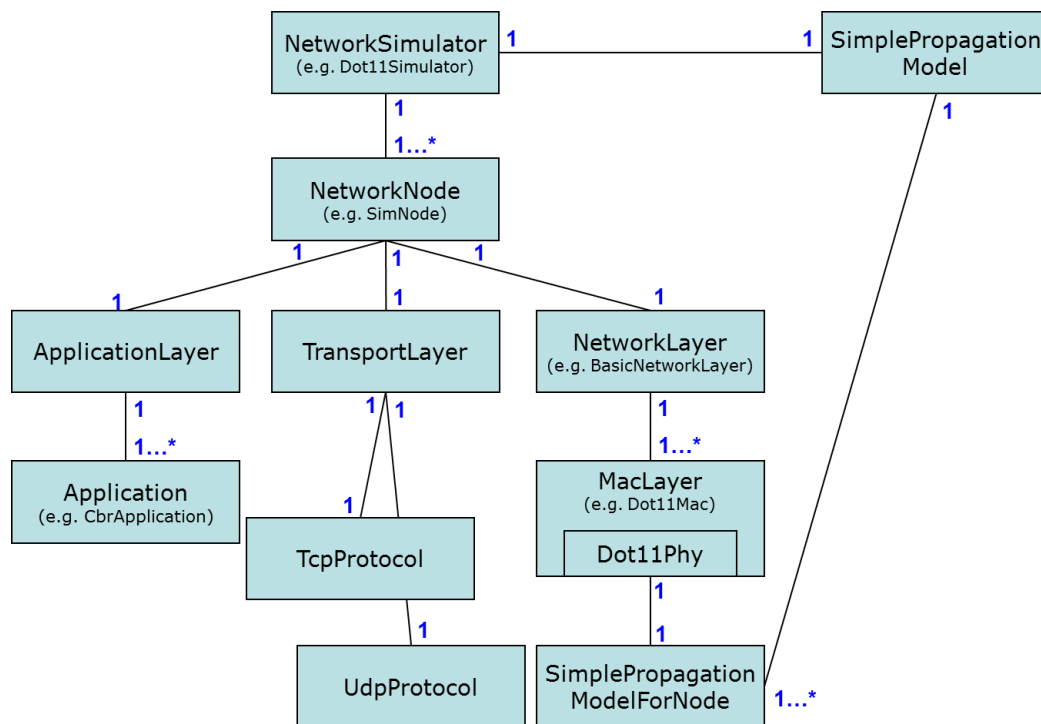


図 1-4. システムモデルに関するクラス構成

1.3.3. インスタンス化

シミュレーションエンジンやネットワークシミュレータは、main 関数内でインスタンス化が行われる。以下に、main 関数内でシミュレーションエンジンやネットワークシミュレータが生成されている記述を示す。

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. レイヤ間インタフェース

図 1-5 および表 1-1 は、パケットを送受信する際の各レイヤ間のインタフェースを示している。図 1-5 および表 1-1 に示すようにパケットの送受信の際には、予め定められた関数が呼ばれるようになっていいる。これらの関数は、抽象クラスの純粋仮想関数として定義されているため、継承クラスにおいて独自の処理を記述することが可能である。



図 1-5. レイヤ間のインタフェース

表 1-1. レイヤ間インタフェース関数

scensim_transport.h/cpp

	クラス	関数	説明
①	UdpProtocol	SendPacket ()	UDP パケットの送信
②	UdpProtocol:: PacketForAppFromTr ansportLayerHandler	ReceivePacket ()	UDP パケットの受信
③	TcpConnection	SendDataBlock ()	TCP データの送信
④	TcpConnection:: AppTcpEventHandler	ReceiveDataBlock ()	TCP データの受信
⑥	ProtocolPacketHandle r	ReceivePacketFromNetworkLaye r ()	ネットワークレイヤからのパケットの受信

scensim_network.h/cpp

	クラス	関数	説明
⑤	NetworkLayer	ReceivePacketFromUpperLayer ()	上位層(トランスポートレイヤ等)からのパケットの受信
⑧	NetworkLayer	ReceivePacketFromMac ()	MAC レイヤからのパケットの受信

scensim_mac.h

	クラス	関数	説明
⑦	MacLayer	NetworkLayerQueueChangeNotification ()	ネットワークレイヤからの送信キュー更新通知

scensim_prop.h

	クラス	関数	説明
⑨	SimplePropagationModelForNode	TransmitSignal	シグナルの送信開始
⑩	SimplePropagationModelForNode::SignalHandler	ProcessSignal	シグナルの受信開始/受信終了

1.5. シナリオファイル

「Base Simulator ユーザガイド」に記載のように、Scenargie では、テキスト形式の設定ファイルをシミュレーションシナリオとして使用している。表 1-2 に、各設定ファイルとそれらを読み込むクラスの対応関係を示す。

表 1-2. シナリオファイルとクラスの対応

設定ファイル	クラス名	ソースファイル
コンフィギュレーションファイル (.config)	ParameterDatabaseReader	scensim_paramio.h
モビリティ設定ファイル (.mob)	TraceFileMobilityModel	scensim_mobility.h
ビットエラー/ブロックエラーテーブル (.ber/.bler)	BitOrBlockErrorRateCurveDatabase	scensim_bercurves.h
統計値取得設定ファイル (.statconfig)	ReadStatConfigFile (グローバル関数)	scensim_stats.cpp
スタティックルーティング設定ファイル (.routes)	ReadStaticRoutingTableFile (グローバル関数)	scensim_network.h
アンテナパタンファイル (.ant)	AntennaPatternDatabase	scensim_proploss.h
材質定義ファイル (.material)	GisSubsystem	scensim_gis.h
シェープファイル (.shp)	GisSubsystem	scensim_gis.h

2. シミュレーション共通機能

本章では、モデルをカスタマイズするにあたって必要なる共通機能について、クラスや関数の説明を行う。Scenargie では、C++言語標準にもとづいてソースコードを記述している。また、シミュレーションエンジン、および、ファイル入出力以外のシステムモデルは、ソースコードとして提供しており、ユーザ自身でカスタマイズすることが可能である。3 章に、システムモデルの説明、4 章に、プログラミングのための API のリストを記載している。

尚、シミュレータのビルド方法等は、Scenargie BaseSimulator ユーザガイドに記載されている。

2.1. ノード ID

Scenargie におけるシミュレーションは、ノードを一つの単位として行われる。ノードには、通信オブジェクト(通信ノード)や GIS オブジェクトが含まれる。ノードは、ノード ID(NodeIdType 型)によって識別される。NodeIdType は、以下に示す定義の通り unsigned int 型であり理論上約 43 億ノードを扱うことが出来る。通信オブジェクトのノード ID は、通常 1 番から昇順で使用されるが、途中で欠番があっても構わない。GIS オブジェクトは、道路や建物など種別によって使用するノード ID の範囲が決まっている。また、ノード ID の 0 番および UINIT_MAX は、それぞれ、INVALID_NODEID、ANY_NODEID として予約されている。

scensim_nodeid.h

```
typedef unsigned int NodeIdType;

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

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


2.2. シミュレーション時間

2.2.1. TimeType 型

Scenargie におけるシミュレーション時間は、TimeType で定義されている。TimeType は、以下に示す定義の通り、long long int 型であり単位はナノ秒である。シミュレーションは、ナノ秒の精度で行うことが可能であり、十分長い時間のシミュレーションシナリオも実行可能である。尚、利便性を考慮し、秒、ミリ秒、マイクロ秒の各予約語も定義してある。

scensim_time.h

```
typedef long long int TimeType;

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

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

2.2.2. シミュレーション時刻の取得

各モデル内で、現在のシミュレーション時刻を取得する場合は、SimulationEngineInterface::CurrentTime()関数を利用して、以下のような記述で取得することが出来る。

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

2.3. シミュレーションイベント

1 章で述べたように、各ノードのシステムモデルは、処理したい内容をシミュレーションイベントとしてシミュレーションエンジンに登録し、シミュレーションエンジンによってイベントの実行が行われる。シミュレーションモデル内に、シミュレーションイベントを実行したい時刻とともに登録することで、任意の時間にシミュレーションイベントを実行することが可能である。本節では、シミュレーションイベントの実行方法、リスケジュール、キャンセル方法について説明する。

2.3.1. シミュレーションイベントの実行方法

独自のシミュレーションイベントを実行するためには、1) シミュレーションイベントクラスの定義、および、2) シミュレーションイベントの登録手続きに関するソースコードの記述が必要である。以下、それぞれについて述べる。

1) シミュレーションイベントクラスの定義

シミュレーションイベントクラスは、抽象クラスである `SimulationEvent` クラスを継承して定義する。

scensim_engine.h

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

以下に、`SimulationEvent` クラスを継承し、独自のシミュレーションイベントクラスを定義する例を示す。`ExecuteEvent()` には、実行したい処理を記述する。

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

通常は、クラス定義されたプロトコルモデル内でシミュレーションイベントを実行する場合が多い。そのため、以下のようにシミュレーションイベントのコンストラクタで自身のポインタを渡し、イベント実行時に関数を呼び出す場合が多い。以下に、プロトコル内でイベントを実行する場合の例を示す。

```

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) シミュレーションイベントの登録

シミュレーションイベントは、実行したい時刻とともに `SimulationEngineInterface` を経由してシミュレーションエンジンへの登録を行う。

例えば、現在時刻から 60 秒後に、`MyMacLayer::MyFunction()` を実行したい場合は、以下のよう記述が可能である。

```

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

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

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

```

2.3.2. シミュレーションイベントのリスケジュール、キャンセル方法

前節では、シミュレーションイベントのリスケジュールやキャンセルを行わない場合のシミュレーションイベント実行方法について述べた。本節では、シミュレーションイベントのリスケジュールおよびキャンセル方法について述べる。

リスケジュールおよびキャンセルする可能性があるイベントを登録する場合は、あらかじめイベントチケット(EventRescheduleTicket クラス)を発行しておき、イベントチケットを用いてリスケジュールおよびキャンセルを行う。尚、イベントチケットはコピーが出来ないため、vector や map などのコンテナで保持する場合は、ポインタの形で保持する必要がある。

イベントチケットを用いたシミュレーションイベントの登録

```
EventRescheduleTicket myEventTicket;

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

シミュレーションをリスケジュールまたはキャンセルする場合は、以下の記述を行う。この場合、シミュレーションイベントはまだ実行されていないことが前提となる。

既にスケジュールされているシミュレーションイベントの実行を現在時刻から 3 秒後にリスケジュールする場合

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

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

イベントをキャンセルする場合

```

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

```

2.3.3. シミュレーションイベント実行時のエラー

シミュレーションイベント実行時に、以下のような Assertion エラーが発生する場合がある。

```

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

```

これは、シミュレーションイベント作成時点において、その時刻よりも前の時刻（過去の時刻）に対してイベントをスケジューリングした場合に発生するエラーである。よって、独自のシミュレーションイベントを登録する際には、シミュレーションの現在時刻（currentTime）に対して正の相対時刻を加えた時刻（currentTime + α ）の形式でシミュレーションイベントを登録することで、本エラーの発生を防ぐことができる。

2.4. パケット

Scenargie では、Packet クラスによってパケットの型が定義されている。アプリケーションなどにおいて、パケットが作成され、各レイヤを経由し、最終的に宛先ノードに届けられる。以下に、パケットの作成、ヘッダの操作（追加、読み込み、削除）、パケットへの外部情報の付与等に関する方法を説明する。

2.4.1. パケット作成方法

パケットを作成するには、SimulationEngineInterface およびペイロードを引数として、Packet::CreatePacket() によって作成する。関数は、ペイロードの型に応じて多重定義されている。ペイロードは、構造体、vector<unsigned char>、string、unsigned char*などが使用できる。（詳細は、4 章の API リストを参照）

Scenargie では、実パケットと同様に全てのペイロード分のメモリを確保する方法と、必要最小限のメモリ確保を行い全てのペイロード分のメモリを確保しない方法（バーチャルペイロード機能）を提供している。前者は、実パケットと同等のパケットデータを生成しているため、エミュレーションなど実パケットを扱ったシミュレーションを行うことも可能である（エミュレーション機能の実行のためには、別途拡張モジュールが必要になります）。バーチャルペイロード機能は、シミュレーションのみを行う場合に、メモリ使用量を削減する場合に有効である。バーチャルペイロード機能は、アプリケーションでパケットを作成する際に、Packet::CreatePacket()クラスの引数として機能を使用するか否かを指定する。また、TCP にもバーチャルペイロードを使用するか否かを指定する API が提供されている。

以下に、CBR アプリケーションにおいて、CBR パケットを作成する場合を例として示す。以下の例では、Packet::CreatePacket()関数の第 4 引数に、バーチャルペイロード機能を使用するか否かの Bool 値が渡されている。true の場合は、バーチャルペイロードを使用し、false の場合は、バーチャルペイロードを使用しないことになる。

scensim_app_cbr.h

```

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

        CbrPayloadType(
            const unsigned int initSequenceNumber,
            const TimeType 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. ヘッダの追加方法

パケットにヘッダを追加する場合は、`Packet::AddPlainStructHeader()`、または、`Packet::AddRawHeader()` を使用する。前者は、予め定義しておいた構造体をヘッダにする場合で、後者は、任意のバイト列をヘッダとする場合である。以下に、UDP ヘッダを追加する場合と、IP ヘッダを追加する場合を例として示す。

UDP ヘッダの追加

`scensim_transport.h/cpp`

```
struct UdpHeaderType {
    UdpHeaderType(
        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(
        UdpHeaderType(
            HostToNet16(sourcePort),
            HostToNet16(destinationPort),
            HostToNet16(static_cast<unsigned short int>(
                packetPtr->LengthBytes() + sizeof(UdpHeaderType)))));
...
}

```

IP ヘッダの追加

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. ヘッダの読み込みおよび削除方法

パケットからヘッダ情報を読み込む場合は、`Packet::GetAndReinterpretPayloadData()`、または、`Packet::GetRawPayloadDat()`を使用する。前者は構造化されたデータを読み込み、後者は、バイト列を読み込む。関数の引数は、パケットの先頭から読み込み開始位置までのオフセット(バイト数)である。パケットの先頭から読み込む場合は、引数を指定する必要はない。

ヘッダを削除する場合は、`Packet::DeleteHeader()`を使用する。関数の引数は、削除したいヘッダのバイト数である。以下に、UDP ヘッダと IP ヘッダの読み込み、および、削除する場合を例として示す。

UDP ヘッダの読み込み、および、削除

scensim_transport.cpp

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

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

IP ヘッダの読み込み、および、削除

secnesim_network.cpp

```

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

```

2.4.4.ペイロードの読み込み方法

アプリケーションでのペイロードの読み込み方法は、ヘッダの読み込み方法と同じである。以下に、CBR アプリケーションにおいて、ペイロードを読み込む場合を例として示す。

scensim_app_cbr.h

```

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

```

2.4.5. パケットへの外部情報付与方法

前述したように、Scenargie におけるパケットは、実パケットと同様にアプリケーションにおけるペイロードや各レイヤにおけるヘッダに現実と同じサイズの情報が保持される。そのため、シミュレーションに必要な情報も、ペイロードやヘッダを使って情報を付与する。

しかしながら、シミュレーションによる評価のために、パケットのペイロードやヘッダとは別に外部情報を付与することも可能である。例えば、送信キューにエンキューされた時刻とデキューされた時刻を付与することなどが可能である。但し、本方法はシミュレーションにおけるパケットの処理へのオーバーヘッドが大きいため、可能な限りパケットのペイロードやヘッダを利用することをお勧めする。

パケットへ外部情報を付与するためには、(1) 情報を格納するためのコンテナの作成、(2) パケットへの情報の付与、(3) 付与した情報の取得の 3 つの処理に対する記述が必要である。以下、それぞれについて述べる。

(1) 外部情報を格納するためのコンテナの作成

外部情報を格納するためのコンテナは、ExtrinsicPacketInformation クラスを継承した独自のクラスを生成する。ExtrinsicPacketInformation::Clone() は、純粋仮想関数であり、継承クラスにおいて、自分自身のコピー(クローン)を作成しスマートポインタを返す関数を実装する必要がある。

scensim_packet.h

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

    virtual shared_ptr<ExtrinsicPacketInformation> Clone() = 0;
};
```

以下に、TimeType 型の情報を格納するためのコンテナの作成例を示す。

```

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

```

(2) パケットへの外部情報の付与

パケットに外部情報を付与するためには、Packet::AddExtrinsicPacketInformation() を使用する。尚、予め情報の識別子を ExtrinsicPacketInfoIdType 型で定義しておく。ExtrinsicPacketInfoIdType は、string 型の別名である。以下に、外部情報として現在時刻をパケットに付与する例を示す。

```

const ExtrinsicPacketInfoIdType enqueuedTimeInfoId = "enqueuedTime";
...
const TimeType currentTime =
    simulationEngineInterfacePtr->CurrentTime();

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

```

(3) 付与した外部情報の取得

パケットに付与した外部情報を取得するためには、Packet:: GetExtrinsicPacketInformation() を使用する。また、取得するためには指定する情報が付与されていることが前提となるため、取得する前にパケットに外部情報が付与されているか確認する必要がある。パケットに指定する外部情報

が付与されているかを判断するためには、Packet:: CheckExtrinsicPacketInformationExist() を使用する。以下に、予め付与しておいた外部情報を取得する例を示す。

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

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

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

2.4.6. パケット ID

Scenargie では、パケットが作られた時点(Packet::CreatePacket())が呼ばれた時点で、一意な ID としてパケット ID(PacketIdType クラス)が作成されパケット自身に付与されている。パケット ID は、パケットを作成したノード(送信元ノード)のノード ID とシーケンス番号からなる。パケットがコピーされた場合は、新たにパケット ID を設定しない限り、コピー後のパケットはコピー元と同じパケット ID を持つ。

パケットからパケット ID を取得するためには、Pakcet::GetPakcetId() を用いる。また、パケット ID から送信元ノード ID およびシーケンス番号を取得するには、PakcetIdType::GetSourceNodeId()、および、PakcetIdType::GetSourceNodeSequenceNumber () を用いる。

2.5. 乱数生成器

Scenargie では、シミュレーションシナリオにパラメータ”seed”を指定することで、乱数の種（以下、run seed）を設定する。run seed は、シミュレーション全体で共通なグローバルな乱数の種であり、run seed の値を変えない限り、毎回同じシミュレーション結果を得ることが出来る。各ノードでは、ノードレベルの乱数の種（以下、node seed）を使用して乱数を発生させる。node seed は、ノード ID に対して run seed を用いてハッシュをかけることで生成される。このような仕組みにより、node seed は、ノード ID と run seed の値により一意に定まる値であり、かつ、run seed とは相関性の極めて低い値となる。各インタフェースでは、node seed を作成した場合と同様に、インタフェースインデックスと node seed から乱数の種(interface seed)を生成する。更に、各モデル(アプリケーション、MAC など)においては、任意の数字をキーとして、node seed や interface seed を使用してハッシュを行い、乱数の種を生成する。

また、”seed”とは別にパラメータ”mobility-seed”を設定することで、モビリティモデルのみ別の乱数の種を使用することも可能である。mobility-seedの値は固定しseedの値のみを変化させることで、モビリティに乱数の変化の影響を与えない形で複数の乱数の種による通信シミュレーションを行うことが可能である。mobility-seed を指定しない場合は、seed で指定された乱数の種がモビリティモデルでも使用される。

尚、通常の擬似乱数生成および高精度擬似乱数生成には、それぞれ Boost ライブラリの boost::rand48 および boost::mt19937 を使用している。

乱数の種の型は、以下で定義されており、uint32_t 型の別名である。

randomnumbergen.h

```
typedef uint32_t RandomNumberGeneratorSeedType;
```

乱数を発生させるためには、あらかじめ乱数生成器に種(シード)を設定し初期化しておき、乱数生成の関数を呼び出すことで、一様乱数の生成する。以下、それぞれについて説明する。

1) 乱数の種の設定

乱数の種を設定するには、RandomNumberGenerator::SetSeed() を用いる。乱数の種自体は、HashInputsToMakeSeed() 関数を用いて生成する。尚、node seed は、ノード生成時に run seed を元に自動的に生成され、NetworkNode::GetNodeSeed() を用いて取得可能である。

2) 乱数の生成

一様乱数の生成は、0 以上 1 未満の double 型の乱数を発生させる RandomNumberGenerator::GenerateRandomDouble()、および、指定した整数範囲から乱数を発生させる RandomNumberGenerator::GenerateRandomInt() を用いること行われる。

乱数の生成の例として、RandomWaypointMobilityModel クラスにおける位置の算出と Dot11Mac クラスにおけるバックオフスロット数の算出方法を以下に示す。

scensim_mobility.h

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

```
RandomWaypointMobilityModel::RandomWaypointMobilityModel(
...
    const RandomNumberGeneratorSeedType nodeSeed =
        HashInputsToMakeSeed(runSeed, nodeId);
    RandomNumberGenerator aRandomNumberGenerator(
        HashInputsToMakeSeed(
            nodeSeed, interfaceId, 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(Dot11 モジュール)

```

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

```

```

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

```

```

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

```

2.6. パラメータ

コンフィグレーションファイルに記述されたシミュレーション用のパラメータは、全て `ParameterDatabaseReader` クラスを用いて利用される。尚、コンフィグレーションファイルの記述方法については、Scenargie BaseSimulator ユーザガイドに記載されている。

コンフィグレーションファイルの読み込み、および、`ParameterDatabaseReader` の作成は、通常、`main` 関数内で行われ、各モデルでは、コンストラクタで参照渡しされた `ParameterDatabaseReader` を利用する。

以下に、`BaseSimulator(base/sim.cpp)`における `ParameterDatabaseReader` の生成例を示す。

`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` クラスには、指定するパラメータ名が存在するかどうかを識別するための関数として、以下の関数が用意されている。本関数は、スコープ(グローバル、ノード、インタフェース/インスタンス)に応じて複数の関数が多重定義されている。尚、スコープについては、Scenargie BaseSimulator ユーザガイドに記載されている。

ParameterDatabaseReader::ParameterExists()

実際に、パラメータを読み込むためには、以下の関数を用いる。読み込むパラメータ値の型に応じて複数定義されているとともに、スコープに応じて複数の関数が多重定義されている。

ParameterDatabaseReader::ReadBool(): Bool 型パラメータの読み込み

ParameterDatabaseReader::ReadInt(): int 型パラメータの読み込み

ParameterDatabaseReader::ReadBigInt(): long long int 型パラメータの読み込み

ParameterDatabaseReader:: ReadNonNegativeInt (): unsigned int 型パラメータの読み込み

ParameterDatabaseReader:: ReadNonNegativeBigInt (): unsigned long long int 型パラメータの読み込み

ParameterDatabaseReader::ReadDouble(): double 型パラメータの読み込み

ParameterDatabaseReader::ReadTime(): TimeType 型パラメータの読み込み

ParameterDatabaseReader::ReadString(): string 型パラメータの読み込み

以下に、実際に使用されている例を示す。順に、グローバルパラメータ、ノードパラメータ、インタフェースパラメータを読み込んでいる場合を示している。

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", nodeId)) {
        const int hopLimitInt =
            theParameterDatabaseReader.ReadInt(
                "network-hop-limit", nodeId);
...
    }

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

```

2.7. 統計値出力

独自の統計値出力を追加する方法は、「Scenargie Base Simulator ユーザガイド」に記載されている。

2.8. トレース出力

独自のトレース出力を追加する方法は、「Scenargie Base Simulator ユーザガイド」に記載されている。

3. システムモデル

本章では、システムモデルについて、各レイヤのクラス構成およびカスタマイズ方法について説明する。はじめにネットワークシミュレータとノードについて説明した後、各レイヤ別に説明を行う。尚、各モデルにおける API リストは、4 章に記載されている。

3.1. ネットワークシミュレータおよび通信ノード

Scenargie では、図 1-4 で示したように一つのネットワークシミュレータが複数の通信ノードを保持し、各ノードは、各システムモデルのプロトコルスタックを持つ。ネットワークシミュレータは、NetworkSimulator クラスを継承して作成され、通信ノードは、NetworkNode を継承して作成される。

以下に、Dot11 モジュールにおけるネットワークシミュレータおよび通信ノードの定義例の一部を示す。これらと同様に、NetworkSimulator クラスおよび NetworkNode を継承することで、独自のネットワークシミュレータまたは通信ノードを作成することが可能である。

dot11/sim.cpp (Dot11 モジュール)

```
class Dot11Simulator : public NetworkSimulator {
public:
    Dot11Simulator(
        const shared_ptr<ParameterDatabaseReader>&
            initParameterDatabaseReaderPtr,
        const shared_ptr<SimulationEngine>& initSimulationEnginePtr,
        const RandomNumberGeneratorSeedType& initRunSeed,
        const bool initRunSequentially);
    ...
Dot11Simulator::Dot11Simulator(
    const shared_ptr<ParameterDatabaseReader>&
        initParameterDatabaseReaderPtr,
    const shared_ptr<SimulationEngine>& initSimulationEnginePtr,
    const RandomNumberGeneratorSeedType& 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 NodeIdType& nodeId,
        const RandomNumberGeneratorSeedType& runSeed,
        const shared_ptr<ObjectMobilityModel>&
            nodeMobilityModelPtr);
    ~SimNode() {}
    ...
}
...
SimNode::SimNode(
    const ParameterDatabaseReader& theParameterDatabaseReader,
    const GlobalNetworkingObjectBag& theGlobalNetworkingObjectBag,
    const
        shared_ptr<SimulationEngineInterface>&
initSimulationEngineInterfacePtr,
    const NodeIdType& initNodeId,
    const RandomNumberGeneratorSeedType& initRunSeed,
    const shared_ptr<ObjectMobilityModel>& initNodeMobilityModelPtr)
    :
    NetworkNode(
        theParameterDatabaseReader,
        theGlobalNetworkingObjectBag,
        initSimulationEngineInterfacePtr,
        initNodeMobilityModelPtr,
        initNodeId,
        initRunSeed),
    nodeMobilityModelPtr(initNodeMobilityModelPtr)
{
}
...

```


3.2. アプリケーションレイヤ

1 章で述べたように、各ノードは、ApplicationLayer を保持しており、ApplicationLayer は、シミュレーションシナリオの内容によって複数の Application を保持する形となる。以下に、既存のアプリケーションのクラス構成、および、独自のアプリケーションを追加する方法について説明する。

3.2.1. 概要

アプリケーションは、抽象クラスである Application クラスを継承して作成されている。図 3-1 は、Application クラスを継承して作成されている既存のアプリケーションの例である。CBR、VBR 等の各アプリケーションクラスがある。アプリケーションによっては、さらにこれらのクラスを継承し送信用および受信用クラスを別に設けているものもある。

アプリケーションを追加する基本的な仕組みは、Application のインスタンスを作成した後、ApplicationLayer::AddApp() 関数を用いてノードにアプリケーションを追加する。Scenargie が標準機能として提供しているアプリケーションでは、NetworkNode クラスのコンストラクタでコンフィグレーションファイル(.config)に記載されたアプリケーションの設定が自動的に読み込まれ、各ノードにアプリケーションが追加される。

各アプリケーションでは、指定する時刻にパケットを送信するイベントを登録することで、パケットの送信処理を行う。パケットの送信には、トランスポートレイヤのポインタを用いて UDP へパケットを渡したり、TCP にデータブロックを渡したりする。受信側はあらかじめ受信用のポート番号とともにパケットハンドラをトランスポートレイヤに登録しておく。トランスポートレイヤでは、ネットワークレイヤから受け取ったパケットのポート番号を元に該当するパケットハンドラを利用してアプリケーションレイヤにパケットまたはデータブロックを渡す。

次節以降、アプリケーションを作成する方法について説明する。

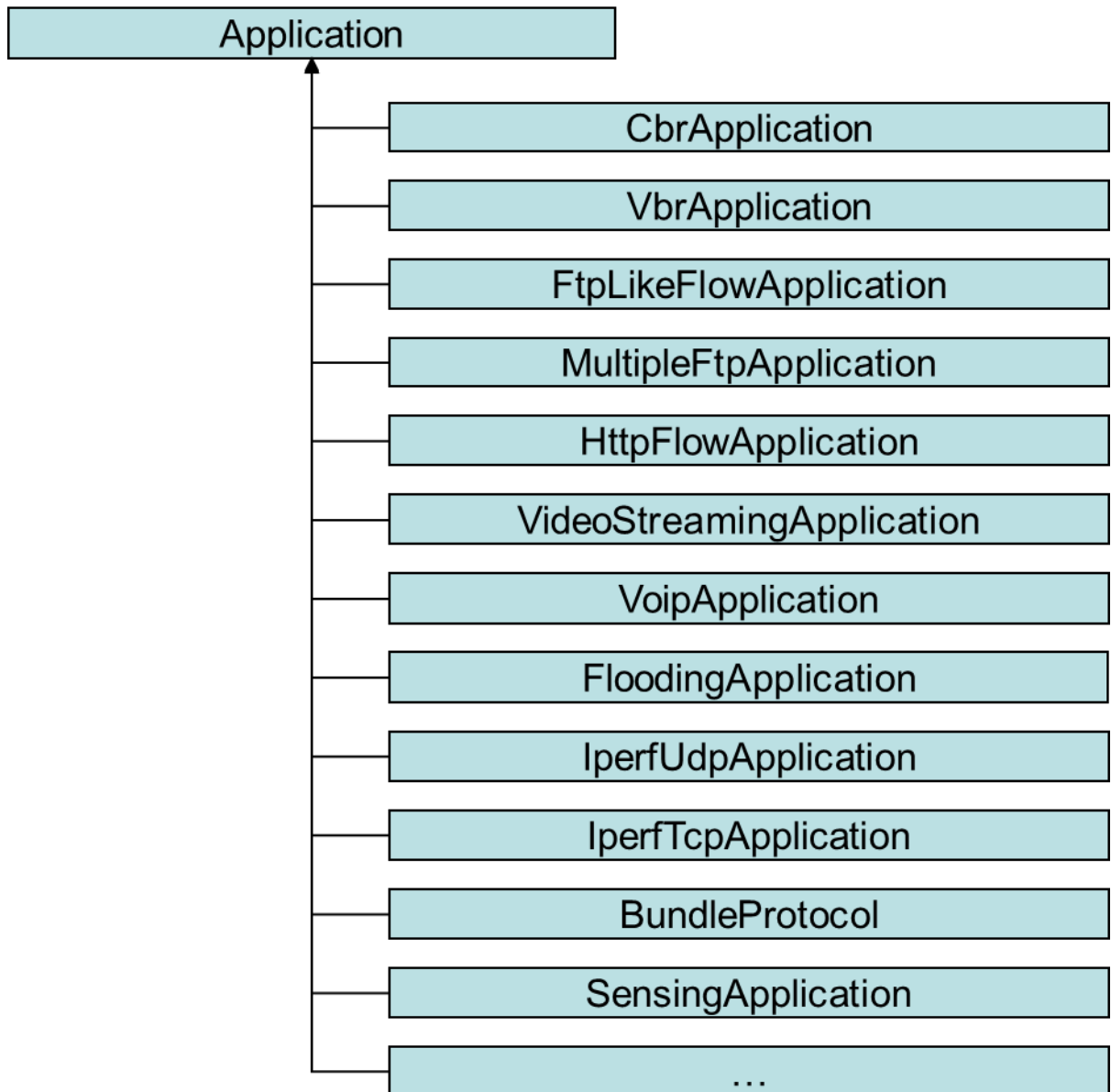


図 3-1. アプリケーションモデルのクラス構成

3.2.2. アプリケーションの作成

アプリケーションは、トランスポートレイヤに UDP を用いるもの、あるいは、TCP を用いるものなどがある。ここでは、トランスポートレイヤに UDP を用いる場合について説明する。また、アプリケーションの定義方法として、既存のアプリケーションと同様に、送信側と受信側でそれぞれ異なるアプリケーションを定義する方法と、送受信を両方を行うアプリケーションを 1 つ定義する方法が考えられる。前者は、主にユニキャスト、後者は、主にブロードキャストのアプリケーションを定義するのに向いている。既存のアプリケーションでは、CbrApplication が前者、FloodingApplication が後者の形態を取っている。

以下、UDP を用いたアプリケーションの作成方法として、既存の CbrApplication を例に説明する。CbrApplication クラスは、Application クラスを継承したクラスで、CBR アプリケーションに共通な型の定義やパラメータの読み込み等を行っている。パケットの送信および受信には、CbrApplication クラスを継承した CbrSourceApplication クラスおよび CbrSinkApplication クラスを用いる。

3.2.2.1. アプリケーション用基底クラスの作成

アプリケーション用基底クラスは、必ずしも必要ではないが、パラメータの読み込みなど送信用クラスと受信用クラスで共通の処理をまとめることが可能である。アプリケーション用基底で実装が必要となる項目を以下に示す。

- パラメータの読み込み
- ペイロードタイプの定義

以下は、CbrApplication クラスの抜粋である。CbrApplication クラスは、Application クラスを継承している。その中に、アプリケーションのペイロードとなる CbrPayloadType の定義、および、パラメータの読み込みを行っている。ペイロードは、任意の型をあらかじめ定義しておき、送受信アプリケーション間で共有しておくことでデータの送受信を行うことができる。また、定義しておいたペイロードの型のサイズに限らずパケットを作成する際にパディングを行うことでよりサイズの大きなパケットを作ることにも可能である。

scensim_app_cbr.h

```

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

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

```

```

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

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

    cbrPriority = static_cast<PacketPriorityType>(
        parameterDatabaseReader.ReadNonNegativeInt(
            parameterPrefix + "-priority", sourceNodeId, applicationId));
...

```

3.2.2.2. 送信用アプリケーションの作成

送信用アプリケーションで実装が必要となる項目を以下に示す。

- アプリケーション用基底クラスを継承した送信用アプリケーションクラスの定義
- 送信イベントの定義
- 送信イベントの登録
- 送信処理

以下は、CbrSourceApplication クラスの抜粋である。CbrSourceApplication クラスは、CbrApplicationクラスを継承している。その中に、送信イベントとしてCbrEventが定義されている。パケットの送信イベントである CbrEvent は、イベントの実行時に、CbrSourceApplication::SendPacket() 関数が呼び出されるように定義されている。

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//
...
}
```

以下に、送信イベントの登録および送信処理について、CbrSourceApplication の例を示す。アプリケーションをインスタンス化したあとは、CbrSourceApplication::CompleteInitialization() 関数を呼び出すことでアプリケーションの初期化を終わらせる。この関数では、パケットを送信する最初のイベ

ントを登録する。CbrSourceApplication の例では、パラメータとして渡された送信開始時刻にイベントが実行されるように、CbrEvent を登録している。この際、自身のポインタを渡すが、通常ポインタ (this) ではなくスマートポインタ(shared_ptr)を使用する場合は、shared_from_this() を渡す。また、そのためには、アプリケーションのクラス定義として enable_shared_from_this<ClassName> を継承する必要がある。

続いて、CbrEvent が実行されると、CbrSourceApplication::SendPacket() 関数が呼び出される。SendPaket() 関数では、ペイロードの中に必要な情報が書き込まれ、Packet::CreatePakcet() によってパケットを生成している。宛先アドレスなどを指定したあと、UDPプロトコルのポインタを通じて、UdpProtocol::SendPacket() が呼び出され、UDP にパケットが渡され、UDP での処理に移る。また、定期的にパケットを送信するために、パケットの送信後は、次の送信のための CbrEvent の登録が行われている。

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 TimeType nextPacketTime =
        simulationEngineInterfacePtr->CurrentTime() + packetInterval;

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

```

3.2.2.3. 受信用アプリケーションの作成

受信側アプリケーションで実装が必要となる項目を以下に示す。

- アプリケーション用基底クラスを継承した受信用アプリケーションクラスの定義
- パケットハンドラの定義
- パケットハンドラの登録

- 受信処理

以下に、CbrSinkApplication を例に受信側アプリケーションについて説明する。CbrSinkApplication クラスも送信側アプリケーションと同様に、CbrApplication クラスを継承して定義する。UDP アプリケーションの場合、パケットを受信するためのパケットハンドラを UdpProtocol::PacketForAppFromTransportLayerHandler クラスを継承して作成しておく。パケットを受信した際には、ハンドラの ReceivePacket() 関数が呼ばれるため、ReceivePacket() 関数の中に、パケットを受信した際の処理を記述する。CbrSinkApplication クラスの場合、CbrSinkApplication::ReceivePacket() 関数が呼び出される。

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 PacketPriorityType& priority)
        {
            cbrSinkPtr->ReceivePacket(packetPtr);
        }

    private:
        shared_ptr<CbrSinkApplication> cbrSinkPtr;

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

```

続いて、パケットハンドラの登録と受信処理について説明する。パケットハンドラの登録は、アプリケーションクラスの初期化の完了処理を行う `CbrSinkApplication::CompleteInitialization()` 内で行う。パケットハンドラを作成し、受信ポート番号に対してパケットハンドラの登録を行う。この処理により、トランスポートレイヤに当該受信ポートあてにパケットが届いた場合に、パケットハンドラ経由でアプリケーションにパケットが運ばれる。パケットがアプリケーションに届いた後は、実際のパケットの受信処理を行う。以下の例では、`CbrSinkApplication::ReceivePacket()` 関数でパケットの受信処理を行っ

ている。本アプリケーションでは、受信処理としてはエンドツーエンドの遅延時間など統計値を取得することだけを行っている。送信側でペイロードに様々な情報を付与しておくことで、受信側でその情報を元に別のアクション（例えば、送信元にパケットを送信するなど）を行うことも可能である。

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>();

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

3.2.2.4. 送受信アプリケーションの作成

1 つのクラスで送受信の処理を行うためには、前節までで説明した送信用アプリケーションおよび受信アプリケーションで必要になる機能を一つのクラスで定義すればよい。具体的には、以下の項目が必要であり、CbrSourceApplication や CbrSinkApplication に記載の内容を一つのクラスにまとめる形となる。

- Application クラスを継承した送受信用アプリケーションクラスの定義
- 送信イベントの定義
- パケットハンドラの定義
- 送信イベントの登録
- パケットハンドラの登録
- 送信処理
- 受信処理

あらかじめ本アプリケーションで使用するパケットの受信用ポートを決めておくことで、各ノードに本アプリケーションを1つインスタンス化すればよく、ブロードキャスト型のアプリケーションの定義などに適している。尚、各ノードにおいて複数のアプリケーションを使用する場合は、アプリケーション間で受信用ポートが重複しないように設定する必要がある。Scenarie 標準のアプリケーションでは、FloodingApplication などがこの形態で作成されている。

3.2.3. アプリケーションの追加

アプリケーションを各ノードに追加するには、既存の CBR アプリケーションなどと同様に送受信ノードのみにアプリケーションを追加する方法と、全てのノードを対象にしてアプリケーションを追加する方法がある。いずれの場合もコンフィグレーションファイルに記載されたパラメータを元にアプリケーションクラスをインスタンス化したのち、アプリケーションレイヤに追加 (ApplicationLayer::AddApp()) を行う。以下、それぞれについて説明する。

3.2.3.1. アプリケーションの追加方法(1)

アプリケーションの追加には、ApplicationMaker クラスを用いる。ApplicationMaker クラスは、NetworkNode クラスのコンストラクタで自動生成されており、コンフィグレーションファイルに記載のアプリケーションの設定内容が読み込まれノードにアプリケーションが追加される。

独自のアプリケーションを追加する場合に ApplicationMaker クラスで変更が必要な項目は以下の 4 項目である。

- アプリケーションタイプの定義
- アプリケーション識別用パラメータの設定
- アプリケーション用のパーサの定義、実装
- アプリケーション用のパーサの呼び出し

以下は、既存のソースコードに、独自のアプリケーションタイプを定義した例である。追記した部分を青字で示している。

scensim_application.h

```
class ApplicationMaker {  
...  
    enum ApplicationType {  
        APPLICATION_CBR,  
...  
        APPLICATION_MY_APP  
  
        //Add new app  
    };  
...  
}
```

以下は、既存のソースコードに、独自アプリケーション識別用パラメータの設定を行った例である。追記した部分を青字で示している。

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";
    ...
}
```

以下は、独自アプリケーション用のパーサの定義、および、実装を記述した例である。追記した部分を青字で示している。送信元ノード ID が自分のノード ID と一致した場合に、MySourceApplication を作成し、アプリケーションレイヤに追加を行っている。

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 ApplicationInstanceIdInfo& applicationInstanceId)
{
    const NodeIdType& sourceNodeId =
        applicationInstanceId. nodeIdWithParameter;
    const InterfaceOrInstanceIdType& instanceId =
        applicationInstanceId.instanceId;
    const NodeIdType 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();
    }
    ...
}

```

以下は、既存のソースコードに、独自アプリケーション用のパーサの呼び出しの記述を追加した例である。追記した部分を青字で示している。

scensim_application.cpp

```

void ApplicationMaker::ReadApplicationIntancesFromConfig(
{
...
    switch(applicationType) {
    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. アプリケーションの追加方法(2)

前節で述べたアプリケーションの追加方法は、設定内容に応じて送受信ノードに対してアプリケーションの追加が行われる。これは、送受信ノードが限定されているユニキャストのようなアプリケーションには適している。一方、全てのノードが共通にアプリケーションを持つような場合には、以下の方法によりアプリケーションを追加することも出来る。

以下は、送受信アプリケーションを全てのノードに追加する例を示している。具体的には、ノードのコンストラクタ(ノードクラスを定義している sim.cpp などに記載)で、アプリケーションのインスタンス化を行い、アプリケーションレイヤに追加を行っている。アプリケーションのパラメータは、アプリケーション内部で、パラメータリーダー(ParameterDatabaseReader)から直接取得することが可能である。尚、ApplicationLayer::AddApp()関数において、トランスポートレイヤのポインタ等の設定が行われるため、ハンドラの登録を行うなど各アプリケーションの CompleteInitialization() 関数は、ApplicationLayer::AddApp()関数を呼び出した後に、呼ぶ必要がある。

```
SimNode::SimNode()
{
...

    shared_ptr<MyApp> myAppPtr(
        new MyApp (
            simulationEngineInterfacePtr,
            theParameterDatabaseReader,
            ...));

    (*this).GetAppLayerPtr()->AddApp(myAppPtr);
    myAppPtr->CompleteInitialization();

...
}
```

3.3. トランスポートレイヤ

本節では、トランスポートレイヤについて説明する。Scenargie では、トランスポートプロトコルとして TCP および UDP を提供している。トランスポートレイヤは、BasicNetworkLayer クラスのコンストラクタで作成されており、Application クラスにもトランスポートレイヤへのポインタが渡されている。各アプリケーションでは、トランスポートレイヤが保持する TCP または UDP へのポインタを用いて、パケットを送信することが出来る。尚、TCP は、BSD9 の TCP のコードを、Scenargie に移植したものである。

トランスポートレイヤは、以下のように定義されている。

scensim_transport.h

```
class TransportLayer {
public:
    TransportLayer(
        const ParameterDatabaseReader& theParameterDatabaseReader,
        const shared_ptr<SimulationEngineInterface>&
            simulationEngineInterfacePtr,
        const shared_ptr<NetworkLayer>& networkLayerPtr,
        const NodeIdType& nodeId,
        const RandomNumberGeneratorSeedType& 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//
```

トランスポートレイヤのプロトコルである TCP および UDP は、ネットワークレイヤとのインタフェースを確立するために ProtocolPacketHandler クラスを継承して実装されている(図 3-2)。独自のトランスポートプロトコルを実装する場合は、UDP や TCP と同様に、ProtocolPacketHandler クラスを継承し

たクラスを定義し、BasicNetworkLayer::RegisterPacketHandlerForProtocol() によってプロトコル番号とともにネットワークレイヤに登録する。

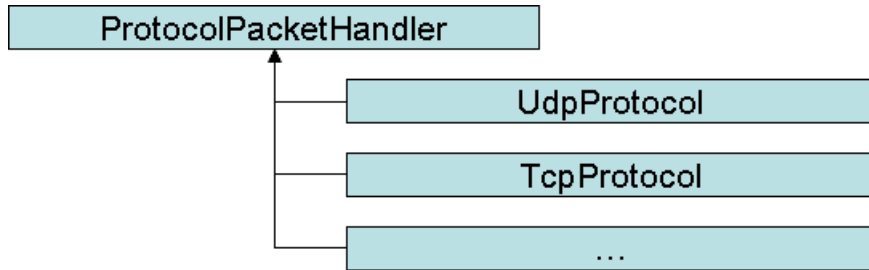


図 3-2. トランスポートレイヤのクラス構成

[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 PacketPriorityType 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. ネットワークレイヤ

本節では、ネットワークレイヤについて説明する。Scenargie では、ネットワークレイヤの protokol として IP (インターネットプロトコル) をサポートしている。インターネットプロトコルは、抽象クラスである `NetworkLayer` を継承し作成されている。独自のネットワークレイヤを作成する場合は、`BasicNetworkLayer` と同様に、`NetworkLayer` を継承し作成する(図 3-3)。

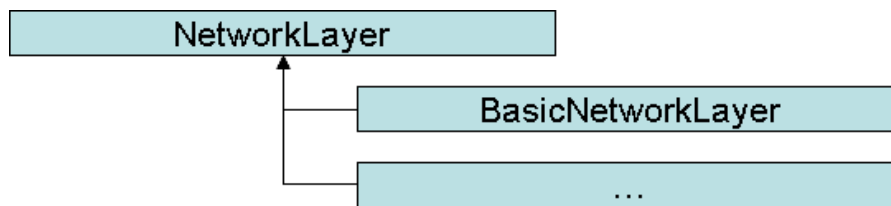


図 3-3. ネットワークレイヤのクラス構成

`NetworkLayer` は、完全抽象クラスのため、定義されている全ての関数を実装する必要がある。その中でも、上位レイヤからパケットを受信するための関数 `NetworkLayer::ReceivePacketFromUpperLayer()`、MAC レイヤからパケットを受信するための関数 `NetworkLayer::ReceivePacketFromMac()` が実装上、重要になる。また、MAC レイヤにパケットを渡す場合は、送信キューにパケットを渡した後、`MacLayer::NetworkLayerQueueChangeNotification()` を呼び出し、MAC レイヤに送信キューに変化があったことを通知する。また、パケットを上位レイヤに渡す場合は、あらかじめ登録しておいたハンドラを用いて、`ProtocolPacketHandler::ReceivePacketFromNetworkLayer()` を呼び出す。

以下に、`BasicNetworkLayer` における実装例を示す。

[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,
    PacketPriorityType 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 PacketPriorityType initialTrafficClass,  
    const EtherTypeFieldType 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 EtherTypeFieldType 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 レイヤ

本節では、MAC レイヤについて説明する。Scenargie では、PHY レイヤの抽象クラスは提供しておらず MAC レイヤの抽象クラスのみ提供している。よって、構造上 PHY レイヤは MAC レイヤに含まれると見なすことが出来る。また、電波伝搬環境を介したシグナルの送受信は、あらかじめ API が提供されており、それらを使うことで無線通信のシミュレーションを行うことが可能である。

以下に、既存の MAC レイヤの概要について説明する。

MAC レイヤの抽象クラスは、以下のように定義されている。ネットワークレイヤにおいて送信キューにパケットが挿入された場合に、MAC レイヤに通知するための純粋仮想関数である `NetworkLayerQueueChangeNotification()` 関数、および、スマートポインタで参照されている `NetworkLayer` のデストラクタを呼び出すために、`NetworkLayer` のポインタをリセットするための関数等が定義されている。図 3-4 は、MAC レイヤのクラス構成を示している。

scensim_mac.h

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

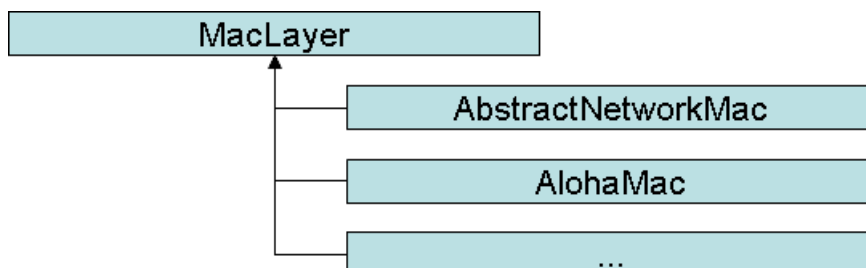


図 3-4. MAC レイヤのクラス構成

独自の MAC レイヤを作成する場合は、MacLayer を継承し、独自のクラスを作成する。ネットワークレイヤから MAC レイヤへのインタフェースは、NetworkLayerQueueChangeNotification() 関数が使用される。また、MAC レイヤからネットワークレイヤへのインタフェースは、MAC レイヤに NetworkLayer のポインタを渡しておくことで、MAC レイヤから直接 NetworkLayer の関数を呼び出しパケットを渡す。BasicNetworkLayer クラスでは、ReceivePacketFromMac() 関数が該当する。その他に、ノードの作成過程において、Mac レイヤをネットワークレイヤに登録することや、MAC レイヤで作成した送信キューをネットワークレイヤに登録する機能の実装が必要である。

以下に、MAC レイヤの例として、AlohaMac クラスを用いて説明する。AlohaMac は、UnslottedAloha や SlottedAloha によるマルチアクセスを実現した無線ネットワークを想定した MAC モデルである。以下に示すように、MacLayer を継承したクラスが定義されており、コンストラクタにおいて送信キューの生成とネットワークレイヤへの登録が行われている。これによってインタフェース毎に送信キューが作成され、MAC レイヤと 1 対 1 に対応する。ネットワークレイヤによって送信キューにパケットがエンキューされた際には、ネットワークレイヤから MAC レイヤの NetworkLayerQueueChangeNotification() 関数が呼び出され、MAC レイヤに通知される。MAC レイヤでは、ネットワークレイヤからの通知を受けパケットの送信処理が開始される。パケットの送信処理では、MAC レイヤの仕様に応じて様々な処理が行われるが最終的には送信キューからパケットがデキューされ、PHY レイヤなどに渡される。また、MAC レイヤからネットワークレイヤにパケットを渡す場合には、ネットワークレイヤの ReceivePacketFromMac() 関数によりパケットが渡される。

aloha_mac.h

```

class AlohaMac :
    public MacLayer, public enable_shared_from_this<AlohaMac> {
...
    //ネットワークレイヤからの通知
    void NetworkLayerQueueChangeNotification();
...
};

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

```

```

AlohaMac::AlohaMac(
...
    //送信キューの作成
    networkOutputQueuePtr(
        new FifoInterfaceOutputQueue(
            theParameterDatabaseReader,
            initInterfaceId,
            simulationEngineInterfacePtr)),
...
{
    //送信キューのネットワークレイヤへの登録
    networkLayerPtr->SetInterfaceOutputQueue(
        interfaceIndex, networkOutputQueuePtr);
...
}

```

```

void AlohaMac::RetrievePacketFromNetworkLayer(bool& wasRetrieved)
{
...
    //送信キューからのパケットのデキュー
    networkOutputQueuePtr->DequeuePacket(
        (*this).currentDataPacketPtr,
        nextHopAddress,
        notUsed1, notUsed2);
...
}

```

```

void AlohaMac::ProcessReceivedDataFrame(const Packet& aFrame)
{
...
    //ネットワークレイヤへのパケットの受け渡し
    networkLayerPtr->ReceivePacketFromMac(
        interfaceIndex, dataPacketPtr, lastHopAddress);
...
}

```

尚、Dot11 モジュールにおいて提供している Dot11Mac 等の MAC レイヤもネットワークレイヤとのインタフェースなどの基本構造は同じである。

3.6. 電波伝搬

本節では、電波伝搬モデルを構成するパスロスモデルおよびアンテナモデルについて説明する。

3.6.1. パスロスモデル

図 3-5 に、パスロスモデルのクラス構成を示す。

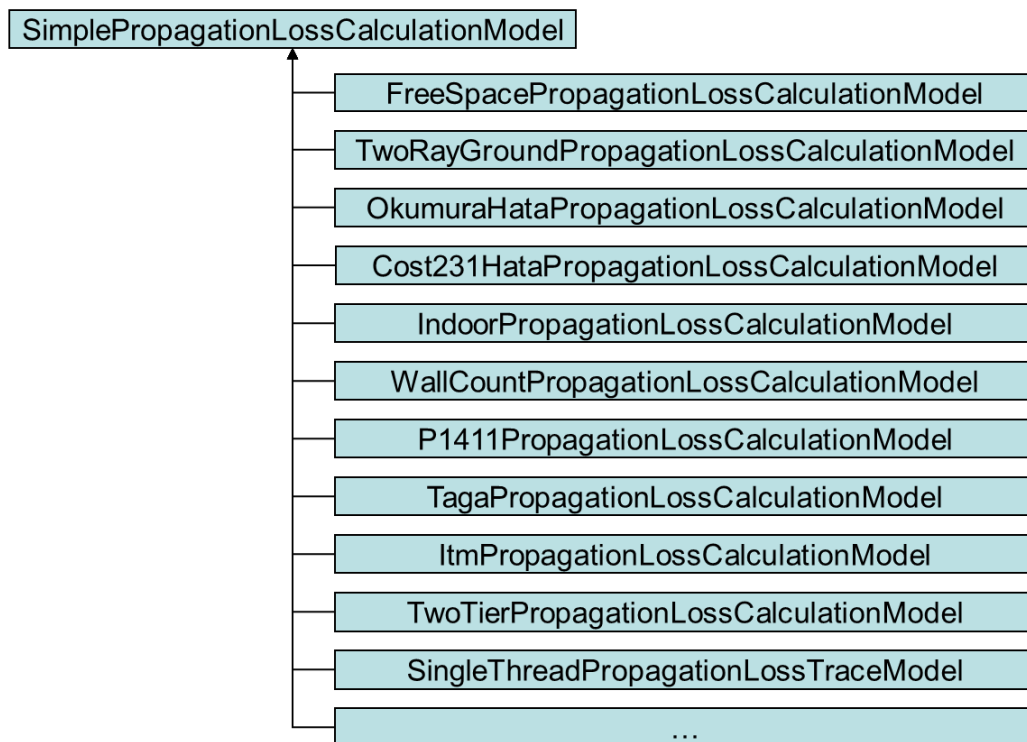


図 3-5. パスロスモデルのクラス構成

各パスロスモデルは、以下に示すように SimplePropagationLossCalculationModel クラスを継承して作成する。以下に、SimplePropagationLossCalculationModel クラスと FreeSpacePropagationLossCalculationModel クラスの一部を示す。

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
{
...
}

```

独自のパスロスモデルを追加する場合も、SimplePropagationLossCalculationModel を継承し、純粋仮想関数である SimplePropagationLossCalculationModel::CalculatePropagationLossDb() 関数を実装する。また、パスロスモデル名を定義するとともに、以下のように CreatePropagationLossCalculationModel() 関数で独自のパスロスモデルを利用するための記述を追加する。

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 InterfaceOrInstanceIdType& instanceId,
    const RandomNumberGeneratorSeedType& 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. アンテナモデル

図 3-6 に、アンテナモデルのクラス構成を示す。

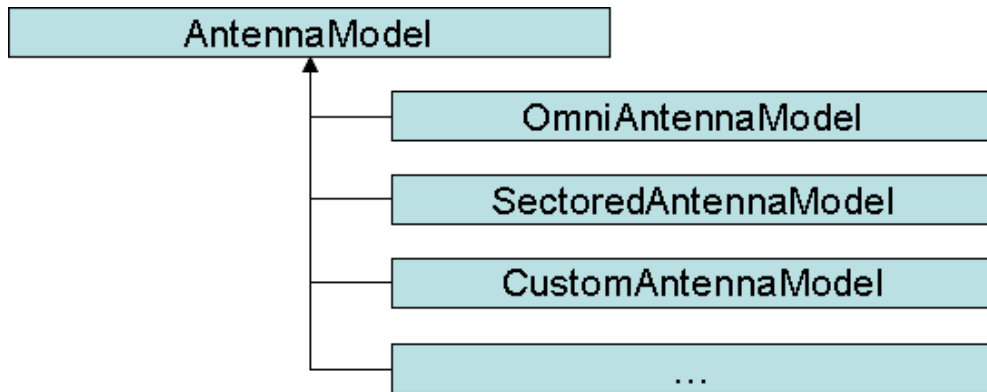


図 3-6. アンテナモデルのクラス構成

アンテナモデルは、以下に示すように AntennaModel クラスを継承し作成する。以下に、AntennaModel クラスと SectedAntennaModel クラスの一部を示す。

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  
    {  
...  
    }
```

独自のアンテナモデルを追加する場合も、AntennaModel を継承し、3 つの純粋仮想関数を実装する。また、アンテナモデル名を定義するとともに、以下のように指定するアンテナモデル名の場合に、独自のアンテナモデルを利用するための記述を追加する。

scenargiesim.cpp

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

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

尚、アンテナパタンファイルをあらかじめ用意しておくことで、カスタムアンテナモデルを利用することも出来る。カスタムアンテナモデルの設定方法は、「Scenargie Base Simulator ユーザガイド」に記載されている。

3.7. モビリティモデル

本節では、モビリティモデルについて説明する。図 3-7 に、モビリティモデルのクラス構成図を示す。

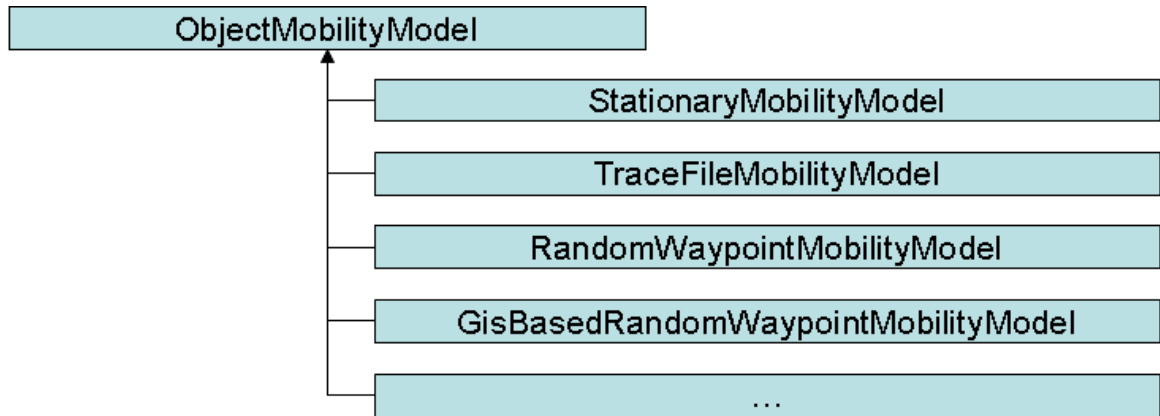


図 3-7. モビリティモデルのクラス構成

モビリティモデルは、以下に示すように ObjectMobilityModel クラスを継承し作成する。以下に、ObjectMobilityModel クラスと TraceFileMobilityModel クラスの一部を示す。

scensim_proploss.h

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

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

独自のモビリティモデルを追加する場合も、ObjectMobilityModel を継承し、純粋仮想関数である ObjectMobilityModel:: GetUnadjustedPositionForTime()を実装する。また、モビリティモデル名を定義するとともに、以下のように CreateAntennaMobilityModel()関数に独自のモビリティモデルを利用するための記述を追加する。

scensim_mobility.cpp

```

shared_ptr<ObjectMobilityModel> CreateAntennaMobilityModel(
    const ParameterDatabaseReader& theParameterDatabaseReader,
    const NodeIdType& nodeId,
    const InterfaceOrInstanceIdType& interfaceId,
    const RandomNumberGeneratorSeedType& 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 データアクセス

本節では、GIS データアクセスについて述べる。Scenargie では、ESRI シェープ形式の GIS データをサポートしており、シミュレーションモデルで使用することが可能である。標準では、GIS データにもとづいた移動を行う `GisBasedRandomWaypointMobilityModel` や GIS データに基づいたパスロス計算を行う `IndoorPropagationLossCalculationModel` や `P1411PropagationLossCalculationModel` などで利用されている。GIS データは、`GisSubsystem` クラスによってデータの読み込みや各種 API の提供が行われている。`NetworkSimulator` クラスのコンストラクタでインスタンス化が行われ、各モデルにポインタを渡すことで GIS データにアクセスが可能である。

以下に、`NetworkSimulator` クラスのコンストラクタおよび関数の一部を示す。

[scensim_netsim.h](#)

```
NetworkSimulator::NetworkSimulator(
    const shared_ptr<ParameterDatabaseReader>&
        initParameterDatabaseReaderPtr,
    const shared_ptr<SimulationEngine>& initSimulationEnginePtr,
    const RandomNumberGeneratorSeedType& 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 NodeIdType& nodeId,
    const InterfaceOrInstanceIdType& interfaceId,
    const RandomNumberGeneratorSeedType& 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,
                nodeId,
                interfaceId,
                runSeed,
                mobilityFileCache,
                mobilityGranularityMeters,
                theGisSubsystemPtr));

    }
    ...
}

```

4. API リスト

本章では、シミュレーションモデルの開発に利用できる API のリストを分野別に示す。

4.1. シミュレーションエンジン関連

ソースファイル: scensim_engine.h

4.1.1. SimulationEvent

シミュレーションイベントの抽象クラス

戻り値	関数(引数)	説明
virtual void	ExecuteEvent ()=0	シミュレーションイベントの実行 (純粋仮想関数)

4.1.2. EventRescheduleTicket

リスケジューリング用シミュレーションイベントチケット

戻り値	関数(引数)	説明
void	Clear ()	シミュレーションイベントチケットの無効化
bool	IsNull () const	シミュレーションイベントチケットが無効か否かの判断

4.1.3.SimulationEngineInterface

シミュレーションエンジンインタフェース

戻り値	関数(引数)	説明
void	ShutdownThisInterface ()	シミュレーションエンジンから本インタフェースの切断
TimeType	CurrentTime () const	現在時刻の取得
NodeIdType	GetNodeId () const	ノード ID の取得
void	ScheduleEvent (const shared_ptr< SimulationEvent > &eventPtr, const TimeType &eventTime, EventRescheduleTicket &eventTicket)	(リスケジューリング可能)シミュレーションイベントの登録 (スマートポインタ用)
void	ScheduleEvent (const shared_ptr< SimulationEvent > &eventPtr, const TimeType &eventTime)	シミュレーションイベントの登録 (スマートポインタ用)
void	ScheduleEvent (unique_ptr< SimulationEvent > &eventPtr, const TimeType &eventTime)	シミュレーションイベントの登録 (ユニークポインタ用)
void	CancelEvent (EventRescheduleTicket &eventTicket)	シミュレーションイベントのキャンセル
void	RescheduleEvent (EventRescheduleTicket &eventTicket, const TimeType &eventTime)	シミュレーションイベントのリスケジューリング
void	ScheduleExternalEventAtNode (const NodeIdType &destinationNodeId, unique_ptr< SimulationEvent > &eventPtr, const TimeType &eventTime)	指定するノードへのシミュレーションイベントの登録

void	ScheduleExternalEventAtPartition (const unsigned int destinationPartitionIndex, unique_ptr< SimulationEvent > &eventPtr, const TimeType &eventTime)	他スレッドへのシミュレーションイベントの登録
unsigned long long int	GenerateAndReturnNewLocalSequenceNumber ()	ノードレベルのシーケンス番号の生成と取得
bool	TracelsOn (const TraceTagType traceTag) const	トレース出力を行うか否かの識別
void	OutputTrace (const string &modelName, const string &modelInstanceId, const string &eventName, const string &stringToOutput) const	テキスト型トレースの出力
bool	BinaryOutputsOn () const	バイナリ型のトレース出力を行うか否かの識別
template<typename T > void	OutputTraceInBinary (const string &modelName, const string &modelInstanceId, const string &eventName, const T &data) const	バイナリ型トレースの出力
void	OutputTraceInBinary (const string &modelName, const string &modelInstanceId, const string &eventName) const	バイナリ型トレース出力(個別情報なし)
bool	ParallelismIsOn () const	並列計算モードが有効か否かの識別
shared_ptr< CounterStatistic >	CreateCounterStat (const string &statName, const bool useBigCounter=false)	カウンタ型統計値の作成
shared_ptr< RealStatistic >	CreateRealStat (const string &statName, const bool useBigReal=false)	実数型統計値の作成
shared_ptr<	CreateRealStatWithDbConversion	実数型統計値の作成

RealStatistic >	(const string &statName, const bool useBigReal=false)	(真値で記録し、出力時に dB 変換を 行う)
-----------------	--------------------------------------------------------------	----------------------------

4.1.4.SimulationEngine

シミュレーションエンジン

戻り値	関数(引数)	説明
	SimulationEngine (const ParameterDatabaseReader &theParameterDatabaseReader, const bool initIsRunningSimulationSequentially =true, const unsigned int numberPartitionThreads=1)	SimulationEngine クラスのコンストラクタ
shared_ptr < SimulationEngine Interface >	GetSimulationEngineInterface (const ParameterDatabaseReader &theParameterDatabaseReader, const NodeIdType &nodeId, const size_t startingPartitionIndex=0)	シミュレーションエンジンインタフェースの取得
void	RunSimulationSequentially (const TimeType &runUntilTime)	シミュレーションの実行
void	RunSimulationInParallel (const TimeType &stopTime)	シミュレーションの実行 (並列計算用)
void	PauseSimulation ()	シミュレーションの一時停止
void	ClearAnyImpendingPauseSimulation ()	シミュレーションの再開
unsigned int	GetNumberPartitionThreads () const	スレッド数の取得
TimeType	CurrentTime () const	現在時刻の取得
bool	SimulationIsPaused () const	シミュレーションの一時停止状態か否かの識別
bool	SimulationIsDone () const	シミュレーションの終了状態か否かの識別
void	ShutdownSimulator ()	シミュレータの停止
void	EnableTraceAtANode (const NodeIdType &nodeId, const TraceTagType traceTag)	トレースの有効化

void	DisableTraceAtANode (const NodeIdType &nodeId, const TraceTagType traceTag)	トレースの無効化
RuntimeStatistics System &	GetRuntimeStatisticsSystem ()	統計値サブシステムの取得
TraceSubsystem &	GetTraceSubsystem ()	トレースサブシステムの取得
void	SetTraceSubsystem (const shared_ptr< TraceSubsystem > &newTraceSubsystemPtr)	トレースサブシステムの設定

4.2. パケット関連

ソースファイル: [scensim_packet.h](#)

4.2.1. Packet

パケットクラス

戻り値	関数(引数)	説明
	Packet (const Packet &right)	Packet クラスのコンストラクタ
const PacketIdType	GetPacketId () const	パケット ID の取得
void	SetPacketId (const PacketIdType &newPacketId)	パケット ID の設定
void	AddRawHeader (const unsigned char rawHeader[], const unsigned int sizeBytes)	ヘッダの追加
template<typename T > void	AddPlainStructHeader (const T &header)	構造化ヘッダの追加
template<typename T > void	AddPlainStructHeaderWithTrailing AlignmentBytes (const T &header, const unsigned int numTrailingAlignmentBytes)	アラインメント用バイトが付与されてい る構造化ヘッダの追加
void	AddTrailingPadding (const unsigned int paddingLengthBytes)	パケットへのパディング
void	RemoveTrailingPadding (const unsigned int paddingLengthBytes)	パディングの削除
void	DeleteHeader (const unsigned int bytesToDelete)	指定バイト数のヘッダの削除
unsigned int	LengthBytes () const	パケット長の取得
unsigned int	ActualLengthBytes () const	実パケットサイズ(バーチャルペイロー ドサイズを除いたパケット長)の取得
const unsigned	GetRawPayloadData () const	ペイロードの取得

char *		
unsigned char *	GetRawPayloadData ()	ペイロードの取得
const unsigned char *	GetRawPayloadData (const unsigned int byteOffset, const unsigned int length) const	ペイロードの取得 (オフセット指定)
unsigned char *	GetRawPayloadData (const unsigned int byteOffset, const unsigned int length)	ペイロードの取得 (オフセット指定)
template<typename T > const T &	GetAndReinterpretPayloadData (const int byteOffset=0) const	構造化ペイロードの取得
template<typename T > T &	GetAndReinterpretPayloadData (const int byteOffset=0)	構造化ペイロードの取得
void	AddExtrinsicPacketInformation (const ExtrinsicPacketInfoldType &extrinsicPacketInfold, const shared_ptr< ExtrinsicPacketInformation > &infoPtr)	パケットへの外部情報の追加
template<typename T > T &	GetExtrinsicPacketInformation (const ExtrinsicPacketInfoldType &extrinsicInfold) const	パケットからの外部情報の取得
bool	CheckExtrinsicPacketInformation Exist (const ExtrinsicPacketInfoldType &extrinsicInfold) const	パケットに外部情報が付与されている か否かの識別
void	MakeLocalCopyOfExtrinsicPacketIn fo ()	共有されているパケットの外部情報を ローカル(自パケット)にコピー
template<typename T > static unique_ptr< Packet >	CreatePacket (SimulationEngineInterface &simEngineInterface, const T &payload)	パケットの生成 (構造化ペイロードの指定)
template<typename T > static unique_ptr<	CreatePacketWithExtraHeaderSpace (SimulationEngineInterface	拡張ヘッダ領域を持つパケットの生成 (構造化ペイロードの指定)

Packet >	&simEngineInterface, const T &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)	パケットの生成 (構造化ペイロード、パケット長、バーチャルペイロード使用の有無の指定)
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)	拡張ヘッダ領域を持つパケットの生成 (構造化ペイロード、パケット長、バーチャルペイロード使用の有無の指定)
static unique_ptr<Packet >	CreatePacket (SimulationEngineInterface &simEngineInterface, const vector< unsigned char > &payload)	パケットの生成 (バイト列の指定)
static unique_ptr<Packet >	CreatePacketWithExtraHeaderSpace (SimulationEngineInterface &simEngineInterface, const vector< unsigned char > &payload, const unsigned int extraAllocatedBytesForHeaders)	拡張ヘッダ領域を持つパケットの生成 (バイト列の指定)
static unique_ptr<Packet >	CreatePacket (SimulationEngineInterface &simEngineInterface, const vector< unsigned char > &payload, const unsigned int totalPayloadLength, const bool initUseVirtualPayload=false)	パケットの生成 (バイト列、パケット長、バーチャルペイロード使用の有無の指定)

static unique_ptr< Packet >	CreatePacketWithExtraHeaderSpace (SimulationEngineInterface &simEngineInterface, const vector< unsigned char > &payload, const unsigned int totalPayloadLength, const unsigned int extraAllocatedBytesForHeaders, const bool initUseVirtualPayload=false)	拡張ヘッダ領域を持つパケットの生成 (バイト列、パケット長、バーチャルペイロード使用の有無の指定)
static unique_ptr< Packet >	CreatePacket (SimulationEngineInterface &simEngineInterface, const string &payload)	パケットの生成 (string 型データの指定)
static unique_ptr< Packet >	CreatePacketWithExtraHeaderSpace (SimulationEngineInterface &simEngineInterface, const string &payload, const unsigned int extraAllocatedBytesForHeaders)	拡張ヘッダ領域を持つパケットの生成 (string 型データの指定)
static unique_ptr< Packet >	CreatePacket (SimulationEngineInterface &simEngineInterface, const string &payload, const unsigned int totalPayloadLength, const bool initUseVirtualPayload=false)	パケットの生成 (string 型データ、パケット長、バーチャルペイロード使用の有無の指定)
static unique_ptr< Packet >	CreatePacketWithExtraHeaderSpace (SimulationEngineInterface &simEngineInterface, const string &payload, const unsigned int totalPayloadLength, const unsigned int extraAllocatedBytesForHeaders, const bool initUseVirtualPayload=false)	拡張ヘッダ領域を持つパケットの生成 (string 型データ、パケット長、バーチャルペイロード使用の有無の指定)
static unique_ptr<	CreatePacket (パケットの生成

Packet >	SimulationEngineInterface &simEngineInterface, const unsigned char *payload, const unsigned int payloadLength)	(char 型ポインタ、パケット長の指定)
static unique_ptr< Packet >	CreatePacketWithExtraHeaderSpace (SimulationEngineInterface &simEngineInterface, const unsigned char *payload, const unsigned int payloadLength, const unsigned int extraAllocatedBytesForHeaders)	拡張ヘッダ領域を持つパケットの生成 (char 型ポインタ、パケット長の指定)
static unique_ptr< Packet >	CreatePacket (SimulationEngineInterface &simEngineInterface)	パケットの生成 (ペイロードなし)
static unique_ptr< Packet >	CreatePacketWithExtraHeaderSpace (SimulationEngineInterface &simEngineInterface, const unsigned int extraAllocatedBytesForHeaders)	拡張ヘッダ領域を持つパケットの生成 (ペイロードなし)
static unique_ptr< Packet >	CreatePacketWithoutSimInfo (const unsigned char data[], const unsigned int size)	シミュレーション情報(パケットID)を持 たないパケットの生成 (unsigned char 型)
template<typename T > static unique_ptr< Packet >	CreatePacketWithoutSimInfo (const T &payload)	シミュレーション情報(パケットID)を持 たないパケットの生成 (テンプレート型)

4.2.2. ExtrinsicPacketInformation

外部パケット情報の抽象クラス

戻り値	関数(引数)	説明
virtual shared_ptr< ExtrinsicPacketInf	Clone ()=0	情報の複製 (純粋仮想関数)

ormation >		
------------	--	--

4.2.3.PacketIdType

パケット ID を定義するクラス

戻り値	関数(引数)	説明
	PacketIdType (const NodeIdType nodeId, const unsigned long long int sequenceNumber)	PacketIdType クラスのコンストラクタ
NodeIdType	GetSourceNodeId () const	送信元ノード ID の取得
unsigned long long int	GetSourceNodeSequenceNumber () const	送信元ノードのシーケンス番号の取得
string	ConvertToString () const	パケット ID の文字列への変換

4.3. 乱数関連

ソースファイル: [randomnumbergen.h](#)

4.3.1.RandomNumberGenerator

boost::rand48 を使用した乱数生成器

戻り値	関数(引数)	説明
	RandomNumberGenerator (const RandomNumberGeneratorSeedType &seed)	RandomNumberGenerator クラスの コンストラクタ
void	SetSeed (const RandomNumberGeneratorSeedType &seed)	乱数シードの設定
int32_t	GenerateRandomInt (const int32_t lowestValue, const int32_t highestValue)	指定した範囲内の int 型擬似乱数の 生成
double	GenerateRandomDouble ()	0 以上 1 未満の double 型擬似乱数の 生成

4.3.2.HighQualityRandomNumberGenerator

boost::mt19937 を使用した乱数生成器

戻り値	関数(引数)	説明
	HighQualityRandomNumberGenera tor (const RandomNumberGeneratorSeedType &seed)	HighQualityRandomNumberGenera tor クラスのコンストラクタ
void	SetSeed (const	乱数シードの設定

	RandomNumberGeneratorSeedType &seed)	
int32_t	GenerateRandomInt (const int32_t lowestValue, const int32_t highestValue)	指定した範囲内の int 型擬似乱数の 生成
double	GenerateRandomDouble ()	0 以上 1 未満の double 型擬似乱数の 生成

4.3.3.ユーティリティ関数

乱数生成に関連するユーティリティ関数

戻り値	関数(引数)	説明
RandomNumber GeneratorSeedTy pe	HashInputsToMakeSeed (const RandomNumberGeneratorSeedType seed, const unsigned long long int hashingInput)	乱数シードの生成 (ハッシュキーが unsigned long long int 型 1 つの場合)
template<typena me T> RandomNumber GeneratorSeedTy pe	HashInputsToMakeSeed (const RandomNumberGeneratorSeedType seed, const T hashingInput)	乱数シードの生成 (ハッシュキーがテンプレート型 1 つの 場合)
template<typena me T1, typename T2> RandomNumber GeneratorSeedTy pe	HashInputsToMakeSeed (const RandomNumberGeneratorSeedType seed, const T1 hashingInput1, const T2 hashingInput2)	乱数シードの生成 (ハッシュキーがテンプレート型 2 つの 場合)
template<typena me T1, typename T2, typename T3> RandomNumber GeneratorSeedTy	HashInputsToMakeSeed (RandomNumberGeneratorSeedType Seed, const T1 hashingInput1, const T2 hashingInput2, const T3 hashingInput3)	乱数シードの生成 (ハッシュキーがテンプレート型 3 つの 場合)

pe		
template<typename T> inline RandomNumberGeneratorSeedType pe	HashInputsToMakeSeed (RandomNumberGeneratorSeedType seed, const std::string& hashingInput1, const T hashingInput2)	乱数シードの生成 (ハッシュキーが string 型 1 つおよび テンプレート型 1 つの場合)
double	ConvertToExponentialDistribution (const double& randomDouble)	一様分布[0,1)から指数分布への変換
double	ConvertToGuassianDistribution (const double& uniformRandom1, const double& uniformRandom2)	一様分布[0,1)から正規分布への変換 (1 つの乱数を生成)
void	ConvertToGaussianDistribution (const double& uniformRandom1, const double& uniformRandom2, double& gaussianRandom1, double& gaussianRandom2)	一様分布[0,1)から正規分布への変換 (2 つの乱数を生成)

4.4. パラメータ関連

ソースファイル: [scensim_parmio.h](#)

4.4.1.ParameterDatabaseReader

パラメータの取得用クラス

戻り値	関数(引数)	説明
	ParameterDatabaseReader (const string &ParameterFileName)	ParameterDatabaseReader クラスの コンストラクタ
void	DisableUnusedParameterWarning ()	使用しなかったパラメータの Warning 表示の停止
bool	ParameterExists (const string ¶meterName) const	パラメータが存在するか否かの識別 (グローバルパラメータ用)
bool	ParameterExists (const string ¶meterName, const InterfaceOrInstancelIdType &instancelId) const	パラメータが存在するか否かの識別 (インスタンスパラメータ用)
bool	ParameterExists (const string ¶meterName, const NodeIdType &nodeId) const	パラメータが存在するか否かの識別 (ノードパラメータ用)
bool	ParameterExists (const string ¶meterName, const NodeIdType &nodeId, const InterfaceOrInstancelIdType &interfacelId) const	パラメータが存在するか否かの識別 (インタフェースパラメータ用)
bool	ReadBool (const string ¶meterName) const	bool 型パラメータの取得 (グローバルパラメータ用)
bool	ReadBool (const string ¶meterName, const InterfaceOrInstancelIdType &instancelId) const	bool 型パラメータの取得 (インスタンスパラメータ用)
bool	ReadBool (const string ¶meterName, const NodeIdType &nodeId) const	bool 型パラメータの取得 (ノードパラメータ用)

bool	ReadBool (const string ¶meterName, const NodeIdType &nodeId, const InterfaceOrInstanceIdType &interfaceId) const	bool 型パラメータの取得 (インタフェースパラメータ用)
int	ReadInt (const string ¶meterName) const	int 型パラメータの取得 (グローバルパラメータ用)
int	ReadInt (const string ¶meterName, const InterfaceOrInstanceIdType &instanceId) const	int 型パラメータの取得 (インスタンスパラメータ用)
int	ReadInt (const string ¶meterName, const NodeIdType &nodeId) const	int 型パラメータの取得 (ノードパラメータ用)
int	ReadInt (const string ¶meterName, const NodeIdType &nodeId, const InterfaceOrInstanceIdType &interfaceId) const	int 型パラメータの取得 (インタフェースパラメータ用)
long long int	ReadBigInt (const string ¶meterName) const	long long int 型パラメータの取得 (グローバルパラメータ用)
long long int	ReadBigInt (const string ¶meterName, const InterfaceOrInstanceIdType &instanceId) const	long long int 型パラメータの取得 (インスタンスパラメータ用)
long long int	ReadBigInt (const string ¶meterName, const NodeIdType &nodeId) const	long long int 型パラメータの取得 (ノードパラメータ用)
long long int	ReadBigInt (const string ¶meterName, const NodeIdType &nodeId, const InterfaceOrInstanceIdType &interfaceId) const	long long int 型パラメータの取得 (インタフェースパラメータ用)
unsigned int	ReadNonNegativeInt (const string ¶meterName) const	unsigned int 型パラメータの取得 (グローバルパラメータ用)
unsigned int	ReadNonNegativeInt (unsigned int 型パラメータの取得

	const string ¶meterName, const InterfaceOrInstanceldType &instanceld) const	(インスタンスパラメータ用)
unsigned int	ReadNonNegativeInt (const string ¶meterName, const NodeldType &nodeld) const	unsigned int 型パラメータの取得 (ノードパラメータ用)
unsigned int	ReadNonNegativeInt (const string ¶meterName, const NodeldType &nodeld, const InterfaceOrInstanceldType &interfaceld) const	unsigned int 型パラメータの取得 (インタフェースパラメータ用)
unsigned long long int	ReadNonNegativeBigInt (const string ¶meterName) const	unsigned long long int 型パラメータ の取得 (グローバルパラメータ用)
unsigned long long int	ReadNonNegativeBigInt (const string ¶meterName, const InterfaceOrInstanceldType &instanceld) const	unsigned long long int 型パラメータ の取得 (インスタンスパラメータ用)
unsigned long long int	ReadNonNegativeBigInt (const string ¶meterName, const NodeldType &nodeld) const	unsigned long long int 型パラメータ の取得 (ノードパラメータ用)
unsigned long long int	ReadNonNegativeBigInt (const string ¶meterName, const NodeldType &nodeld, const InterfaceOrInstanceldType &interfaceld) const	unsigned long long int 型パラメータ の取得 (インタフェースパラメータ用)
double	ReadDouble (const string ¶meterName) const	double 型パラメータの取得 (グローバルパラメータ用)
double	ReadDouble (const string ¶meterName, const InterfaceOrInstanceldType &instanceld) const	double 型パラメータの取得 (インスタンスパラメータ用)
double	ReadDouble (const string ¶meterName, const NodeldType &nodeld) const	double 型パラメータの取得 (ノードパラメータ用)
double	ReadDouble (double 型パラメータの取得

	const string ¶meterName, const NodeIDType &nodeId, const InterfaceOrInstanceIDType &interfaceId) const	(インタフェースパラメータ用)
TimeType	ReadTime (const string ¶meterName) const	TimeType 型パラメータの取得 (グローバルパラメータ用)
TimeType	ReadTime (const string ¶meterName, const InterfaceOrInstanceIDType &instanceId) const	TimeType 型パラメータの取得 (インスタンスパラメータ用)
TimeType	ReadTime (const string ¶meterName, const NodeIDType &nodeId) const	TimeType 型パラメータの取得 (ノードパラメータ用)
TimeType	ReadTime (const string ¶meterName, const NodeIDType &nodeId, const InterfaceOrInstanceIDType &interfaceId) const	TimeType 型パラメータの取得 (インタフェースパラメータ用)
string	ReadString (const string ¶meterName) const	string 型パラメータの取得 (グローバルパラメータ用)
string	ReadString (const string ¶meterName, const InterfaceOrInstanceIDType &instanceId) const	string 型パラメータの取得 (インスタンスパラメータ用)
string	ReadString (const string ¶meterName, const NodeIDType &nodeId) const	string 型パラメータの取得 (ノードパラメータ用)
string	ReadString (const string ¶meterName, const NodeIDType &nodeId, const InterfaceOrInstanceIDType &interfaceId) const	string 型パラメータの取得 (インタフェースパラメータ用)
string	GetContainingNodeIDSetNameFor (const NodeIDType &nodeId) const	is-member-of パラメータで指定され たグループ名の取得
void	MakeSetOfAllNodeIDs (全ノード ID 集合の取得

	set< NodeIdType > &setOfNodeIds) const	
void	MakeSetOfAllCommNodeIds (set< NodeIdType > &setOfNodeIds) const	全通信ノード ID 集合の取得
bool	CommNodeIdExists (const NodeIdType &nodeId) const	指定するノード ID を持つ通信ノードが 存在するか否か
void	MakeSetOfAllNodeIdsWithParameter (const string ¶meterName, set< NodeIdType > &setOfNodeIds) const	指定するパラメータを持つノード ID 集 合の取得
void	MakeSetOfAllNodeIdsWithParameter (const string ¶meterName, const string ¶meterValue, set< NodeIdType > &setOfNodeIds) const	指定するパラメータの値を持つノード ID 集合の取得
void	MakeSetOfAllInterfaceIdsForANode (const NodeIdType &nodeId, set< InterfaceOrInstanceIdType > &setOfInterfaces) const	インタフェース ID 集合の取得(特定ノ ード用)
void	MakeSetOfAllInterfaceIdsForANode (const NodeIdType &nodeId, const string ¶meterName, set< InterfaceOrInstanceIdType > &setOfInterfaces) const	指定するパラメータを持つインタフェ ース ID 集合の取得(特定ノード用)
void	MakeSetOfAllInterfaceIds (const string ¶meterName, set< InterfaceOrInstanceIdType > &setOfInterfaces) const	指定するパラメータを持つインタフェ ース ID 集合の取得
void	MakeSetOfAllInterfaceIds (const string ¶meterName, const string ¶meterMustBeEqual, set< InterfaceOrInstanceIdType > &setOfInterfaces) const	指定するパラメータの値を持つインタ フェース ID 集合の取得

void	MakeSetOfAllInstanceIdsForANode (const NodeIdType &nodeId, const string ¶meterName, set< InterfaceOrInstanceIdType > &setOfInstances) const	インタフェース/インスタンス ID 集合の 取得(特定ノード用)
void	MakeSetOfAllInstanceIds (const string ¶meterName, set< InterfaceOrInstanceIdType > &setOfInstances) const	指定するパラメータを持つインスタ ンス ID の集合の取得

4.5. BER 関連

ソースファイル: [scensim_bervurves.h](#)

4.5.1.BitOrBlockErrorRateCurveDatabase

BER/BLER カーブデータベース

戻り値	関数(引数)	説明
	BitOrBlockErrorRateCurveDatabase (const string &berFileName)	BitOrBlockErrorRateCurveDatabase クラスのコンストラクタ
void	LoadBerCurveFile (const string &berFileName)	BER カーブファイルの読み込み
void	LoadBlockErrorRateCurveFile (const string &blerFileName)	BLER カーブファイルの読み込み
shared_ptr< BitErrorRateCurve >	GetBerCurve (const string &curveFamilyName, const string &modelName)	BER カーブファイルの取得 (BER カーブファミリー名およびモデル名の指定)
shared_ptr< BlockErrorRateCurve >	GetBlockErrorRateCurve (const string &curveFamilyName, const string &modelName) const	BLER カーブファイルの取得 (BLER カーブファミリー名およびモデル名の指定)

4.5.2.BitErrorRateCurve

BER カーブ

戻り値	関数(引数)	説明
	BitErrorRateCurve (const string &initCurveFamilyName, const string &initModeName, long long int initBitsPerSecondForMode)	BitErrorRateCurve クラスのコンストラクタ
string	GetCurveFamilyName () const	BER カーブファミリー名の取得
string	GetModeName () const	モード名の取得

void	AddDataPoint (const double &snrValue, const double &bitErrorRateForThatSnr)	BER データの追加
double	CalculateBitErrorRate (const double &signalToNoiseAndInterferenceRatio) const	BER の算出

4.5.3. BlockErrorRateCurve

BLER カーブ

戻り値	関数(引数)	説明
	BlockErrorRateCurve (const string &initCurveFamilyName, const string &initModeName)	BlockErrorRateCurve クラスのコンストラクタ
string	GetCurveFamilyName () const	BLER カーブファミリー名の取得
string	GetModeName () const	モード名の取得
void	AddDataPoint (const double &snrValue, const double &blockErrorRateForThatSnr)	BLER データの追加
double	CalcBlockErrorRate (const double &signalToNoiseAndInterferenceRatio) const	BLER の算出

4.6. 統計値関連

ソースファイル: [scensim_stats.h/cpp](#)

4.6.1. CounterStatistic

カウンタ型統計値

戻り値	関数(引数)	説明
bool	IsEnabled () const	本統計値が有効か否かの識別
void	IncrementCounter (const unsigned long long int incrementNumber=1)	統計値の加算
void	UpdateCounter (const long long int newCounterValue)	統計値の更新

4.6.2. RealStatistic

実数型統計値

戻り値	関数(引数)	説明
bool	IsEnabled () const	本統計値が有効か否かの識別
void	RecordStatValue (const double &value)	統計値の記録

4.7. ユーティリティ関連

ソースファイル: [scensim_support.h](#)

4.7.1. ユーティリティ関数

戻り値	関数(引数)	説明
unsigned short int	ConvertToUShortInt (const unsigned int value)	unsigned int 型から unsigned short int 型への変換
unsigned short int	ConvertToUShortInt (const unsigned int value, const string& failureMessage)	unsigned int 型から unsigned short int 型への変換 (失敗時のメッセージあり)
unsigned char	ConvertToUChar (const unsigned int value)	unsigned int 型から unsigned char 型への変換
unsigned char	ConvertToUChar (const unsigned int value, const string& failureMessage)	unsigned int 型から unsigned char 型への変換 (失敗時のメッセージあり)
unsigned int	RoundToUInt (const double& x)	四捨五入による unsigned int 型への変換
unsigned int	RoundUpToUInt (const double& x)	切り上げによる unsigned int 型への変換
int	RoundToInt (const double& x)	四捨五入による int 型への変換
unsigned int	DivideAndRoundUp (const unsigned int x, const unsigned int y)	unsigned int 型の除算と商の切り上げ
template<typename T> T	MinOf3 (const T& x1, const T& x2, const T& x3)	3 つの中から最小の値を選択
double	ConvertToNonDb (const double& dB)	デシベル(dB)から真数への変換
double	ConvertToDb (const double& nonDb)	真数からデシベル(dB)への変換
double	ConvertIntToDb (const unsigned int value)	整数からデシベル(dB)への変換
double	ConvertYMetresToLatitudeDegrees (Y 座標(m)から緯度への変換

	const double& latitudeOriginDegrees, const double& yMeters)	
double	ConvertXMetersToLongitudeDegrees (const double& latitudeOriginDegrees, const double& longitudeOriginDegrees, const double& xMeters)	X 座標(m)から経度への変換
double	ConvertLatitudeDegreesToYMeters (const double latitudeOriginDegrees, const double latitudeDegrees)	緯度から Y 座標(m)への変換
double	ConvertLongitudeDegreesToXMeters (const double latitudeOriginDegrees, const double longitudeOriginDegrees, const double longitudeDegrees)	経度から X 座標(m)への変換
template<typename T> T	CalcInterpolatedValue (const double& x1, const T& y1, const double& x2, const T& y2, const double x)	2 点を与えられた場合の線形補間
void	ConvertStringToLowerCase (string& aString)	文字列を小文字に変換
string	MakeLowerCaseString (const string& aString)	小文字に変換した文字列の作成
bool	StringIsAllLowerCase (const string& aString)	文字列が全て小文字か否かの識別
void	ConvertStringToUpperCase (string& aString)	文字列を大文字に変換
string	MakeUpperCaseString (const string& aString)	大文字に変換した文字列の作成
bool	StringIsAllUpperCase (const string& aString)	文字列が全て大文字か否かの識別

bool	IsEqualCaseInsensitive (const string& left, const string& right)	2 つの文字列が同じか否かの識別 (大文字小文字の区別なし)
template<typename T> string	ConvertToString (const T& aT)	string 型への変換
void	ConvertStringToInt (const string& aString, int& intValue, bool& success)	string 型を int 型に変換
void	ConvertStringToNonNegativeInt (const string& aString, unsigned int& uintValue, bool& success)	string 型を unsigned int 型に変換
void	ConvertStringToBigInt (const string& aString, long long int& intValue, bool& success)	string 型を long long int 型に変換
void	ConvertStringToDouble (const string& aString, double& doubleValue, bool& success)	string 型を double 型に変換

4.8. ネットワークシミュレータ関連

ソースファイル: [scensim_netsim.h](#)

4.8.1. NetworkSimulator

ネットワークシミュレータの基底クラス

戻り値	関数(引数)	説明
	NetworkSimulator (const shared_ptr< ParameterDatabaseReader > &initParameterDatabaseReaderPtr, const shared_ptr< SimulationEngine > &initSimulationEnginePtr, const RandomNumberGeneratorSeedType &runSeed, const bool initRunSequentially=true)	NetworkSimulator クラスのコンストラクタ
void	DeleteAllNodes ()	全ネットワークノードの削除
void	GetListOfNodeIds (vector< NodeIdType > &nodeIds)	全ネットワークノード ID の取得
virtual NetworkAddress	LookupNetworkAddress (const NodeIdType &nodeId) const	ネットワークアドレスの取得
virtual void	LookupNetworkAddress (const NodeIdType &nodeId, NetworkAddress &networkAddress, bool &success) const	ネットワークアドレスの取得(取得成否の取得)
virtual NodeIdType	LookupNodeId (const NetworkAddress &aNetworkAddress) const	ノード ID の取得
virtual void	LookupNodeId (const NetworkAddress &aNetworkAddress, NodeIdType &nodeId, bool &success) const	ノード ID の取得(取得成否の取得)

virtual unsigned int	LookupInterfaceIndex (const NodeIdType &nodeId, const InterfaceIdType &interfaceName) const	インタフェースインデックスの取得
virtual void	CreateNewNode (const ParameterDatabaseReader &theParameterDatabaseReader, const NodeIdType &nodeId, const string &nodeName="")	ノードの生成(モビリティモデルなし)
virtual void	CreateNewNode (const ParameterDatabaseReader &theParameterDatabaseReader, const NodeIdType &nodeId, const shared_ptr< ObjectMobilityModel > &nodeMobilityModelPtr, const string &nodeName="")	ノードの生成
virtual void	DeleteNode (const NodeIdType &nodeId)	ノードの消滅
void	InsertApplicationIntoANode (const NodeIdType &nodeId, const shared_ptr< Application > &appPtr)	ノードへのアプリケーションの追加
shared_ptr < MacLayerInterfac eForEmulation >	GetMacLayerInterfaceForEmulation (const NodeIdType &nodeId) const	MacLayerInterfaceForEmulation ポ インタの取得(Emulation 用)
const GlobalNetworking ObjectBag &	GetGlobalNetworkingObjectBag () const	GlobalNetworkingObjectBag の 取 得
virtual double	CalculatePathlossFromNodeToLocation (const NodeIdType &nodeId, const PropagationInformationType &informationType, const size_t interfaceIndex, const double &positionXMeters, const double &positionYMeters, const double	特定のノードから指定する位置までの パスロスの算出

	&positionZMeters, PropagationStatisticsType &propagationStatistics)	
virtual void	CalculatePathlossFromNodeToNode (const NodeIdType &txNodeId, const NodeIdType &rxNodeId, const PropagationInformationType &informationType, const unsigned int txInterfaceIndex, const unsigned int rxInterfaceIndex, PropagationStatisticsType &propagationStatistics)	特定ノード間のパスロスの算出
virtual void	OutputNodePositionsInXY (const TimeType lastOutputTime, std::ostream &nodePositionOutputStream) const	ノードの位置のストリーム出力 (GUI 用)
virtual void	OutputTraceForAllNodePositions (const TimeType &lastOutputTime) const	ノードの位置のトレース出力
void	OutputAllNodeIds (std::ostream &ostream) const	ノード ID の出力 (GUI 用)
void	OutputRecentlyAddedNodeIdsWithTypes (std::ostream &ostream)	前回取得時から今までに追加された ノード ID の出力 (GUI 用)
void	OutputRecentlyDeletedNodeIds (std::ostream &ostream)	前回取得時から今までに削除された ノード ID の出力 (GUI 用)
void	RunSimulationUntil (const TimeType &simulateUpToTime)	指定時刻までのシミュレーションの実 行
void	AddPropagationCalculationTraceIf Necessary (const InterfaceIdType &channelId, const shared_ptr< SimplePropagationLossCalculationM	パスロストレース出力の設定

	odel &propagationCalculationModelPtr)	
virtual const ObjectMobilityPos ition	GetNodePosition (const NodeIdType &nodeId)	ノード位置の取得
virtual const ObjectMobilityPos ition	GetAntennaLocation (const NodeIdType &nodeId, const unsigned int interfaceIndex)	アンテナ位置の取得
TimeType	GetTimeStepEventSynchronizatio nStep () const	タイムステップ型イベントの同期間隔 の取得
RandomNumber GeneratorSeedTy pe	GetMobilitySeed () const	モビリティ用シードの取得
void	AddNode (const shared_ptr< NetworkNode > &aNodePtr)	ノードの追加
void	RemoveNode (const NodeIdType &nodeId)	ノードの削除
protected		
virtual void	CompleteSimulatorConstruction ()	シミュレータ構築の最終処理
virtual bool	SupportMultiAgent () const	マルチエージェントシミュレーションを サポートしているか否かの識別
void	SetupStatOutputFile ()	統計値出力ファイルの設定
void	CheckTheNecessityOfMultiAgentS upport ()	マルチエージェントシミュレーション機 能が必要か否かの確認
virtual void	ExecuteTimestepBasedEvent ()	タイムステップ型イベントの実行

4.9. ネットワークノード関連

ソースファイル: [scensim_netsim.h](#)

4.9.1. NetworkNode

ノードの抽象クラス

戻り値	関数(引数)	説明
	NetworkNode (const ParameterDatabaseReader &theParameterDatabaseReader, const GlobalNetworkingObjectBag &theGlobalNetworkingObjectBag, const shared_ptr< SimulationEngineInterface > &initSimulationEngineInterfacePtr, const shared_ptr< ObjectMobilityModel > &initNodeMobilityModelPtr, const NodeIdType &theNodeId, const RandomNumberGeneratorSeedType &runSeed, const bool dontBuildStackLayers=false) 	NetworkNode クラスのコンストラクタ
NodeIdType	GetNodeId ()	ノード ID の取得
string	GetNodeTypeName () const	ノードタイプ名の取得
void	SetNodeTypeName (const string &typeName)	ノードタイプ名の設置
NetworkAddress	GetPrimaryNetworkAddress () const	プライマリネットワークアドレスの取得
RandomNumber GeneratorSeedTy pe	GetNodeSeed () const	ノードシードの取得
virtual shared_ptr< NetworkLayer >	GetNetworkLayerPtr () const	ネットワークレイヤポインタの取得

virtual const NetworkLayer &	GetNetworkLayerRef () const	ネットワークレイヤ(参照)の取得
virtual shared_ptr < TransportLayer >	GetTransportLayerPtr () const	トランスポートレイヤポインタの取得
virtual shared_ptr < ApplicationLayer >	GetAppLayerPtr () const	アプリケーションレイヤポインタの取得
virtual const ObjectMobilityPos ition	GetCurrentLocation () const	現在のノード位置の取得
virtual double	CalculatePathlossToLocation (const PropagationInformationType &informationType, const size_t interfaceIndex, const double &positionXMeters, const double &positionYMeters, const double &positionZMeters, PropagationStatisticsType &propagationStatistics) const	指定された位置までのパスロスの算 出
virtual void	CalculatePathlossToNode (const PropagationInformationType &informationType, const unsigned int interfaceIndex, const ObjectMobilityPosition &rxAntennaPosition, const AntennaModel &rxAntennaModel, PropagationStatisticsType &propagationStatistics) const	指定されたノード間とのパスロスの算 出
virtual bool	HasAntenna (const InterfaceIdType &channelId) const	指定するチャンネル ID を持つアンテナ を持っているか否かの識別
virtual shared_ptr< AntennaModel >	GetAntennaModelPtr (const unsigned int interfaceIndex) const	指定するインタフェースのアンテナモ デルポインタの取得

virtual ObjectMobilityPosition	GetAntennaLocation (const unsigned int interfaceIndex) const	指定するインタフェースのアンテナ位置の取得
virtual void	OutputTraceForNodePosition (const TimeType &lastOutputTime) const	ノード位置のトレース出力
void	OutputTraceForAddNode () const	ノードの追加イベントのトレース出力
void	OutputTraceForDeleteNode () const	ノードの削除イベントのトレース出力
virtual void	CreateDynamicApplication (const ParameterDatabaseReader &theParameterDatabaseReader, const GlobalNetworkingObjectBag &theGlobalNetworkingObjectBag, const NodeIdType &sourceNodeId, const InterfaceOrInstanceIdType &instanceId)	動的なアプリケーションの生成

4.10. アプリケーションレイヤ関連

ソースファイル: scensim_application.h

4.10.1. ApplicationLayer

アプリケーションを保持するクラス

戻り値	関数(引数)	説明
	ApplicationLayer (const shared_ptr< NetworkAddressLookupInterface > &networkAddressLookupInterfacePtr , const shared_ptr< SimulationEngineInterface > &simulationEngineInterfacePtr, const shared_ptr< TransportLayer > &transportLayerPtr, const shared_ptr< ObjectMobilityModel > &nodeMobilityModelPtr, const NodeIdType &initNodeId, const RandomNumberGeneratorSeedType &initNodeSeed)	ApplicationLayer クラスのコンストラクタ
void	AddApp (const shared_ptr< Application > &appPtr)	アプリケーションレイヤへのアプリケーションの追加
void	DisconnectFromOtherLayers ()	保持しているスマートポインタの解放
template<typename T > shared_ptr< T >	GetApplicationPtr (const ApplicationIdType &applicationId)	アプリケーションポインタの取得
unsigned short int	GetNewApplicationInstanceNumber ()	アプリケーションインスタンス番号の取得

4.10.2. Application

アプリケーションの基底クラス

戻り値	関数(引数)	説明
	Application (const shared_ptr< SimulationEngineInterface > &initSimEngineInterfacePtr, const ApplicationIdType initApplicationId)	ApplicationLayer クラスのコンストラクタ
void	DisconnectFromOtherLayers ()	保持しているスマートポインタの解放
shared_ptr < MacAndPhyInfoInterface >	GetMacAndPhyInfoInterface (const InterfaceIdType &interfaceId)	MAC/PHY 情報インタフェースの取得
型	メンバ変数	説明
ApplicationIdType	applicationId	アプリケーション ID
shared_ptr < SimulationEngineInterface >	simulationEngineInterfacePtr	シミュレーションエンジンインタフェース
shared_ptr < NetworkAddressLookupInterface >	networkAddressLookupInterfacePtr	ネットワークアドレス解決インタフェース
shared_ptr< TransportLayer >	transportLayerPtr	トランスポートレイヤのポインタ
shared_ptr< RandomNumberGenerator >	aRandomNumberGeneratorPtr	乱数生成器のポインタ
shared_ptr< ObjectMobilityModel >	nodeMobilityModelPtr	モビリティモデルのポインタ
shared_ptr< ApplicationLayer	applicationLayerPtr	アプリケーションレイヤのポインタ

>		
---	--	--

4.11. トランスポートレイヤ関連

ソースファイル: [scensim_network.h](#)

4.11.1. ProtocolPacketHandler

各種プロトコルの抽象クラス

戻り値	関数(引数)	説明
virtual void	ReceivePacketFromNetworkLayer (unique_ptr< Packet > &packetPtr, const NetworkAddress &sourceAddress, const NetworkAddress &destinationAddress, const PacketPriorityType trafficClass, const NetworkAddress &lastHopAddress, const unsigned char hopLimit, const unsigned int interfaceIndex)=0	ネットワークレイヤからのパケットの受信 (純粋仮想関数)
virtual void	GetPortNumbersFromPacket (const Packet &aPacket, const unsigned int transportHeaderOffset, bool &portNumbersWereRetrieved, unsigned short int &sourcePort, unsigned short int &destinationPort) const =0	パケットヘッダ内のポート番号の取得 (純粋仮想関数)

ソースファイル: [scensim_transport.h](#)

4.11.2. TransportLayer

トランスポートレイヤ

戻り値	関数(引数)	説明
	TransportLayer (const ParameterDatabaseReader &theParameterDatabaseReader, const shared_ptr< SimulationEngineInterface > &simulationEngineInterfacePtr, const shared_ptr< NetworkLayer > &networkLayerPtr, const NodeIdType &nodeId, const RandomNumberGeneratorSeedType &nodeSeed)	TransportLayer クラスのコンストラクタ
shared_ptr< NetworkLayer >	GetNetworkLayerPtr () const	ネットワークレイヤポインタの取得
void	DisconnectProtocolsFromOtherLayers ()	保持しているスマートポインタの解放
型	メンバ変数	説明
shared_ptr< UdpProtocol >	udpPtr	UDP プロトコルのポインタ
shared_ptr< TcpProtocol >	tcpPtr	TCP プロトコルのポインタ

4.11.3. UdpProtocol

UDP プロトコル

戻り値	関数(引数)	説明
	UdpProtocol (const shared_ptr< SimulationEngineInterface > &initSimulationEngineInterfacePtr)	UdpProtocol クラスのコンストラクタ
void	DisconnectFromOtherLayers ()	保持しているスマートポインタの解放
void	ConnectToNetworkLayer (const shared_ptr< NetworkLayer >	ネットワークレイヤへの UDP プロトコル(ハンドラ)の登録

	&networkLayerPtr)	
void	SendPacket (unique_ptr< Packet > &packetPtr, const unsigned short int sourcePort, const NetworkAddress &destinationAddress, const unsigned short int destinationPort, const PacketPriorityType &priority)	ネットワークレイヤへのパケットの送信 (送信元アドレスの指定なし)
void	SendPacket (unique_ptr< Packet > &packetPtr, const NetworkAddress &sourceAddress, const unsigned short int sourcePort, const NetworkAddress &destinationAddress, const unsigned short int destinationPort, const PacketPriorityType &priority)	ネットワークレイヤへのパケットの送信
bool	PortIsAvailable (const NetworkAddress &portAddress, const int portNumber) const	指定ポートの利用可否の識別 (アドレス指定あり)
bool	PortIsAvailable (const int portNumber) const	指定ポートの利用可否の識別
void	OpenSpecificUdpPort (const NetworkAddress &address, const unsigned short int portNumber, const shared_ptr< PacketForAppFromTransportLayerH andler > &packetHandlerPtr)	指定ポート番号に対するアプリケーションへのハンドラ (PacketForAppFromTransportLayerHandler) の登録
virtual void	ReceivePacketFromNetworkLayer (unique_ptr< Packet > &packetPtr, const NetworkAddress	ネットワークレイヤからのパケットの受信

	&sourceAddress, const NetworkAddress &destinationAddress, const PacketPriorityType 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	パケットヘッダ内のポート番号の取得

4.11.4. UdpProtocol::PacketForAppFromTransportLayerHandler

UDP プロトコルからアプリケーションへパケットを渡すためのハンドラ

戻り値	関数(引数)	説明
virtual void	ReceivePacket (unique_ptr< Packet > &packetPtr, const NetworkAddress &sourceAddress, const unsigned short int sourcePort, const NetworkAddress &destinationAddress, const PacketPriorityType &priority)=0	UDP プロトコルからのパケットの受信 (純粋仮想関数)

4.11.5. TcpProtocol

TCP プロトコル

戻り値	関数(引数)	説明
	TcpProtocol (const ParameterDatabaseReader &theParameterDatabaseReader, const shared_ptr< SimulationEngineInterface > &initSimulationEngineInterfacePtr, const NodeIdType &nodeId, const RandomNumberGeneratorSeedType &nodeSeed)	TcpProtocol クラスのコンストラクタ
void	ConnectToNetworkLayer (const shared_ptr< NetworkLayer > &networkLayerPtr)	ネットワークレイヤへの TCP プロトコ ル(ハンドラ)の登録
void	DisconnectFromOtherLayers ()	保持しているスマートポインタの解放
void	CreateOutgoingTcpConnection (const NetworkAddress &localAddress, const unsigned short int localPort, const NetworkAddress &destinationAddress, const unsigned short int destinationPort, const PacketPriorityType &priority, const shared_ptr< TcpConnection::AppTcpEventHandle r > &appEventHandlerPtr, shared_ptr< TcpConnection > &newTcpConnectionPtr)	TCP コネクションの作成
void	CreateOutgoingTcpConnection (const unsigned short int localPort, const NetworkAddress &destinationAddress, const unsigned short int destinationPort, const PacketPriorityType &priority, const shared_ptr< TcpConnection::AppTcpEventHandle r > &appEventHandlerPtr, shared_ptr< TcpConnection >	TCP コネクションの作成 (送信元アドレス指定なし)

	&newTcpConnectionPtr)	
bool	PortsAvailable (const int portNumber) const	指定ポートの利用可否の識別
void	OpenSpecificTcpPort (const NetworkAddress &address, const unsigned short int portNumber, const shared_ptr< ConnectionFromTcpProtocolHandler > &connectionHandlerPtr)	指定ポート番号へのアプリケーション (ハンドラ)の登録
void	DisconnectConnectionHandlerFor Port (const NetworkAddress &address, const unsigned short int portNumber)	TCP コネクションの切断(ポートの開 放)
virtual void	ReceivePacketFromNetworkLayer (unique_ptr< Packet > &packetPtr, const NetworkAddress &sourceAddress, const NetworkAddress &destinationAddress, const PacketPriorityType trafficClass, const NetworkAddress &lastHopAddress_notused, const unsigned char hopLimit_notused, const unsigned int interfaceIndex_notused)	ネットワークレイヤからのパケットの受 信
virtual void	GetPortNumbersFromPacket (const Packet &aPacket, const unsigned int transportHeaderOffset, bool &portNumbersWereRetrieved, unsigned short int &sourcePort, unsigned short int &destinationPort)	パケットヘッダ内のポート番号の取得

	const	
--	-------	--

4.11.6. ConnectionFromTcpProtocolHandler

TCP プロトコルからアプリケーションに TCP コネクションをはるためのハンドラ

戻り値	関数(引数)	説明
virtual void	HandleNewConnection (const shared_ptr< TcpConnection > &connectionPtr)=0	TCP コネクションの設定 (純粋仮想関数)

4.11.7. TcpConnection

TCP コネクション

戻り値	関数(引数)	説明
bool	IsConnected () const	TCP コネクションの接続状態の識別
void	EnableVirtualPayload ()	バーチャルペイロード機能の有効化
void	SetPacketPriority (const PacketPriorityType &priority)	パケット優先度の設定
void	SetAppTcpEventHandler (const shared_ptr< AppTcpEventHandler > &newAppTcpEventHandlerPtr)	アプリケーションへのハンドラ (AppTcpEventHandler)の登録
void	ClearAppTcpEventHandler ()	アプリケーションへのハンドラ (AppTcpEventHandler)の開放
void	SendDataBlock (shared_ptr< vector< unsigned char > > &dataBlockPtr, const unsigned int	TCP データの送信 (バーチャルペイロード用)

		dataLength)	
void		SendDataBlock (shared_ptr< vector< unsigned char > > &dataBlockPtr)	TCP データの送信
unsigned long long int		GetNumberOfReceivedBytes () const	受信バイト数の取得
unsigned long long int		GetNumberOfSentBytes () const	送信バイト数の取得
unsigned long long int		GetNumberOfDeliveredBytes () const	送信が完了した(ACK を受信した)バ イト数の取得
unsigned long long int		GetCurrentNumberOfUnsentBuffe redBytes () const	バッファに溜まっている未送信のバ イト数
unsigned long long int		GetCurrentNumberOfAvailableBuf ferBytes () const	バッファ可能なバイト数
NetworkAddress		GetForeignAddress () const	接続先アドレスの取得
void		Close ()	TCP コネクションの切断

4.11.8. TcpConnection::AppTcpEventHandler

TCP コネクションとアプリケーションでデータを送受信するためのハンドラ

戻り値	関数(引数)	説明
virtual void	DoTcplsReadyForMoreDataAction ()=0	TCP コネクションへのデータ送信 (純粋仮想関数)
virtual void	ReceiveDataBlock (const unsigned char dataBlock[], const unsigned int dataLength, const unsigned int actualDataLength, bool &stallIncomingDataFlow)=0	TCP コネクションからのデータの受信 (純粋仮想関数)
virtual void	DoTcpRemoteHostClosedAction ()	TCP コネクション切断時のリモートホ ストの処理
virtual void	DoTcpLocalHostClosedAction ()	TCP コネクション切断時のローカルホ ストの処理

4.12. ネットワークレイヤ関連

ソースファイル: [scensim_network.h/cpp](#)

4.12.1. NetworkLayer

ネットワークレイヤの抽象クラス

戻り値	関数(引数)	説明
virtual void	DisconnectFromOtherLayers ()=0	保持しているスマートポインタの解放 (純粋仮想関数)
virtual NodeIdType	GetNodeId () const =0	ノード ID の取得 (純粋仮想関数)
virtual shared_ptr< RoutingTable >	GetRoutingTableInterface ()=0	ルーティングテーブルポインタの取得 (純粋仮想関数)
virtual NetworkAddress	GetPrimaryNetworkAddress () const =0	プライマリネットワークアドレスの取得 (純粋仮想関数)
virtual unsigned int	NumberOfInterfaces () const =0	インタフェース数の取得 (純粋仮想関数)
virtual unsigned int	LookupInterfaceIndex (const NetworkAddress &interfaceAddress) const =0	インタフェースインデックスの取得 (純粋仮想関数)
virtual unsigned int	LookupInterfaceIndex (const InterfaceIdType &interfaceName) const =0	インタフェースインデックスの取得 (純粋仮想関数)
virtual InterfaceIdType	GetInterfaceId (const unsigned int interfaceIndex) const =0	インタフェース ID の取得 (純粋仮想関数)
virtual NetworkAddress	GetNetworkAddress (const unsigned int interfaceIndex) const =0	ネットワークアドレスの取得 (純粋仮想関数)
virtual NetworkAddress	GetSubnetAddress (const unsigned int interfaceIndex) const =0	サブネットアドレスの取得 (純粋仮想関数)

virtual NetworkAddress	GetSubnetMask (const unsigned int interfaceIndex) const =0	サブネットマスクの取得 (純粋仮想関数)
virtual unsigned int	GetSubnetMaskBitLength (const unsigned int interfaceIndex) const =0	サブネットマスクのビット数の取得 (純粋仮想関数)
virtual NetworkAddress	MakeBroadcastAddressForInterface (const unsigned int interfaceIndex) const =0	ブロードキャストアドレスの取得 (純粋仮想関数)
virtual void	SetInterfaceIpAddress (const size_t interfaceIndex, const NetworkAddress &newInterfaceAddress, const unsigned int subnetMaskLengthBits)=0	ネットワークアドレスの設定 (純粋仮想関数)
virtual void	SetInterfaceGatewayAddress (const unsigned int interfaceIndex, const NetworkAddress &newGatewayAddress)=0	ゲートウェイアドレスの設定 (純粋仮想関数)
virtual void	ClearInterfaceIpInformation (const unsigned int interfaceIndex)=0	インタフェース情報の削除 (純粋仮想関数)
virtual void	RegisterPacketHandlerForProtocol (const unsigned char protocolNum, const shared_ptr< ProtocolPacketHandler > &packetHandlerPtr)=0	トランスポートプロトコル(ハンドラ)の 登録(純粋仮想関数)
virtual void	RegisterOnDemandRoutingProtocolInterface (const shared_ptr< OnDemandRoutingProtocolInterface > interfacePtr)=0	オンデマンドルーティングプロトコルの 登録 (純粋仮想関数)
virtual void	RegisterNetworkAddressInterface (const shared_ptr< NetworkAddressInterface >	ネットワークアドレスインタフェースの 登録 (純粋仮想関数)

	<code>&interfacePtr)=0</code>	
virtual void	ReceivePacketFromUpperLayer (unique_ptr< Packet > &packetPtr, const NetworkAddress &destinationAddress, PacketPriorityType trafficClass, const unsigned char protocol)=0	上位レイヤからのパケットの受け取り (送信元アドレス指定なし) (純粋仮想関数)
virtual void	ReceivePacketFromUpperLayer (unique_ptr< Packet > &packetPtr, const NetworkAddress &sourceAddress, const NetworkAddress &destinationAddress, PacketPriorityType trafficClass, const unsigned char protocol)=0	上位レイヤからのパケットの受け取り (純粋仮想関数)
virtual void	ReceiveOutgoingBroadcastPacket (unique_ptr< Packet > &packetPtr, const unsigned int outgoingInterfaceIndex, PacketPriorityType trafficClass, const unsigned char protocol)=0	送信用ブロードキャストパケットの受け取り (純粋仮想関数)
virtual void	ReceiveOutgoingPreformedNetworkPacket (unique_ptr< Packet > &packetPtr)=0	送信用 IP ヘッダ付きパケットの受け取り (純粋仮想関数)
virtual void	ReceiveRoutedNetworkPacketFromRoutingProtocol (unique_ptr< Packet > &packetPtr, const unsigned int interfaceIndex, const NetworkAddress &nextHopAddress)=0	ルーティングプロトコルからのパケットの受け取り (純粋仮想関数)
virtual void	GetInterfaceIndexForOneHopDestination (const NetworkAddress &destinationAddress, bool &success,	宛先アドレス(ワンホップ先)に対する インタフェースインデックスの取得 (純粋仮想関数)

	unsigned int &interfaceIndex) const =0	
virtual void	GetNextHopAddressAndInterfaceIndexForDestination (const NetworkAddress &destinationAddress, bool &success, NetworkAddress &nextHopAddress, unsigned int &interfaceIndex) const =0	宛先アドレスに対するネクストホップアドレスとインタフェースインデックスの取得 (純粋仮想関数)
virtual NetworkAddress	GetSourceAddressForDestination (const NetworkAddress &destinationAddress) const =0	宛先アドレスに対する送信元ネットワークアドレスの取得 (純粋仮想関数)
virtual void	GetNextHopAddressAndInterfaceIndexForNetworkPacket (const Packet &aPacket, bool &success, NetworkAddress &nextHopAddress, unsigned int &interfaceIndex) const =0	IP パケットに対するネクストホップアドレスとインタフェースインデックスの取得 (純粋仮想関数)
virtual void	ReceivePacketFromMac (const unsigned int macIndex, unique_ptr< Packet > &packetPtr, const NetworkAddress &lastHopAddress, const EtherTypeFieldType etherType=ETHERTYPE_IS_NOT_SPECIFIED)=0	MAC レイヤからのパケットの受信 (純粋仮想関数)
virtual void	ReceiveUndeliveredPacketFromMac (const unsigned int macIndex, unique_ptr< Packet > &packetPtr, const NetworkAddress &nextHopAddress)=0	MAC レイヤで送信できなかったパケットの受け取り (純粋仮想関数)
virtual void	SetInterfaceMacLayer (MAC レイヤの設定

	const unsigned int interfaceIndex, const shared_ptr< MacLayer > &macLayerPtr)=0	(純粋仮想関数)
virtual shared_ptr< MacLayer >	GetMacLayerPtr (const unsigned int interfaceIndex) const =0	MAC レイヤポインタの取得 (純粋仮想関数)
virtual shared_ptr < NetworkInterface Manager >	GetNetworkInterfaceManagerPtr (const unsigned int interfaceIndex) const =0	ネットワークインタフェースマネージャ ポインタの取得 (純粋仮想関数)
virtual void	SetInterfaceOutputQueue (const unsigned int interfaceIndex, const shared_ptr< InterfaceOutputQueue > &outputQueuePtr)=0	送信キューの設定 (純粋仮想関数)
virtual void	ProcessLinkIsUpNotification (const unsigned int interfaceIndex)=0	MAC レイヤからのリンク接続完了の 通知 (STA) (純粋仮想関数)
virtual void	ProcessLinkIsDownNotification (const unsigned int interfaceIndex)=0	MAC レイヤからのリンク切断の通知 (STA) (純粋仮想関数)
virtual void	ProcessNewLinkToANodeNotificat ion (const unsigned int interfaceIndex, const GenericMacAddressType &macAddress)=0	MAC レイヤからのノードへのリンク接 続完了の通知 (AP) (純粋仮想関数)
virtual bool	MacSupportsQualityOfService (const unsigned int interfaceIndex) const =0	MAC レイヤが QoS をサポートしてい るか否かの識別 (純粋仮想関数)
virtual shared_ptr < MacQualityOfSer viceControllInterfa ce >	GetMacQualityOfServiceInterface (const unsigned int interfaceIndex) const =0	MAC レイヤの QoS インタフェースポ インタの取得 (純粋仮想関数)

virtual void	InsertPacketIntoAnOutputQueue (unique_ptr< Packet > &packetPtr, const unsigned int interfaceIndex, const NetworkAddress &nextHopAddress, const PacketPriorityType trafficClass, const EtherTypeFieldType etherType=ETHERTYPE_IS_NOT_ SPECIFIED)=0	パケットの送信キューへの挿入 (純粋仮想関数)
virtual void	SetupDhcpServerAndClientIfNecessary (const ParameterDatabaseReader &theParameterDatabaseReader, const shared_ptr< ApplicationLayer > &appLayerPtr)=0	DHCP サーバ/クライアントの設定 (純粋仮想関数)

4.12.2. BasicNetworkLayer

IP ネットワークレイヤ

戻り値	関数(引数)	説明
void	DisconnectFromOtherLayers ()	保持しているスマートポインタの解放
NodeIdType	GetNodeId () const	ノード ID の取得
shared_ptr< RoutingTable >	GetRoutingTableInterface ()	ルーティングテーブルポインタの取得
bool	IsANetworkAddressForThisNode (const NetworkAddress &anAddress) const	自ノードのネットワークアドレスか否かの識別
void	CheckIfNetworkAddressIsForThisNode (const NetworkAddress &anAddress, bool &addressIsForThisNode, unsigned int &interfaceIndex) const	自ノードのネットワークアドレスか否かの識別
NetworkAddress	GetPrimaryNetworkAddress () const	プライマリネットワークアドレスの取得

NetworkAddress	GetNetworkAddress (const unsigned int interfaceIndex) const	ネットワークアドレスの取得
NetworkAddress	GetSubnetAddress (const unsigned int interfaceIndex) const	サブネットアドレスの取得
NetworkAddress	GetSubnetMask (const unsigned int interfaceIndex) const	サブネットマスクの取得
unsigned int	GetSubnetMaskBitLength (const unsigned int interfaceIndex) const	サブネットマスクのビット数の取得
NetworkAddress	MakeBroadcastAddressForInterface (const unsigned int interfaceIndex) const	ブロードキャストアドレスの取得
void	SetInterfaceIpAddress (const unsigned int interfaceIndex, const NetworkAddress &newInterfaceAddress, const unsigned int subnetMaskLengthBits)	ネットワークアドレスの設定
void	SetInterfaceGatewayAddress (const unsigned int interfaceIndex, const NetworkAddress &newGatewayAddress)	ゲートウェイアドレスの設定
void	ClearInterfaceIpInformation (const unsigned int interfaceIndex)	インタフェース情報の削除
void	RegisterPacketHandlerForProtocol (const unsigned char protocolNum, const shared_ptr< ProtocolPacketHandler > &packetHandlerPtr)	トランスポートプロトコル(ハンドラ)の登録
void	RegisterOnDemandRoutingProtocolInterface (const shared_ptr< OnDemandRoutingProtocolInterface	オンデマンドルーティングプロトコルの登録

	> interfacePtr)	
void	RegisterNetworkAddressInterface (const shared_ptr< NetworkAddressInterface > &interfacePtr)	ネットワークアドレスインタフェースの登録
void	GetInterfaceIndexForOneHopDestination (const NetworkAddress &destinationAddress, bool &success, unsigned int &interfaceIndex) const	宛先アドレス(ワンホップ先)に対するインタフェースインデックスの取得
void	GetNextHopAddressAndInterfaceIndexForDestination (const NetworkAddress &destinationAddress, bool &success, NetworkAddress &nextHopAddress, unsigned int &interfaceIndex) const	宛先アドレスに対するネクストホップアドレスとインタフェースインデックスの取得
NetworkAddress	GetSourceAddressForDestination (const NetworkAddress &destinationAddress) const	宛先アドレスに対する送信元ネットワークアドレスの取得
void	GetNextHopAddressAndInterfaceIndexForNetworkPacket (const Packet &aPacket, bool &success, NetworkAddress &nextHopAddress, unsigned int &interfaceIndex) const	IP パケットに対するネクストホップアドレスとインタフェースインデックスの取得
unsigned int	LookupInterfaceIndex (const NetworkAddress &interfaceAddress) const	インタフェースインデックスの取得
unsigned int	LookupInterfaceIndex (const InterfaceIdType &interfaceName) const	インタフェースインデックスの取得
InterfaceIdType	GetInterfaceId (const unsigned int interfaceIndex) const	インタフェース ID の取得

unsigned int	NumberOfInterfaces () const	インタフェース数の取得
void	SetInterfaceMacLayer (const unsigned int interfaceIndex, const shared_ptr< MacLayer > &macLayerPtr)	MAC レイヤの設定
void	ReceivePacketFromUpperLayer (unique_ptr< Packet > &packetPtr, const NetworkAddress &destinationAddress, PacketPriorityType trafficClass, const unsigned char protocol)	上位レイヤからのパケットの受け取り (送信元アドレス指定なし)
void	ReceivePacketFromUpperLayer (unique_ptr< Packet > &packetPtr, const NetworkAddress &sourceAddress, const NetworkAddress &destinationAddress, PacketPriorityType trafficClass, const unsigned char protocol)	上位レイヤからのパケットの受け取り
void	ReceiveOutgoingBroadcastPacket (unique_ptr< Packet > &packetPtr, const unsigned int outgoingInterfaceIndex, PacketPriorityType trafficClass, const unsigned char protocol)	送信用ブロードキャストパケットの受け取り
void	ReceiveOutgoingPreformedNetworkPacket (unique_ptr< Packet > &packetPtr)	送信用 IP ヘッダ付きパケットの受け取り
void	ReceiveRoutedNetworkPacketFromRoutingProtocol (unique_ptr< Packet > &packetPtr, const unsigned int interfaceIndex, const NetworkAddress &nextHopAddress)	ルーティングプロトコルからのパケットの受け取り

void	ReceivePacketFromMac (const size_t macIndex, Packet *&packetPtr, const NetworkAddress &lastHopAddress)	MAC レイヤからのパケットの受信
void	ReceiveUndeliveredPacketFromMac (const unsigned int macIndex, unique_ptr< Packet > &packetPtr, const NetworkAddress &nextHopAddress)	MAC レイヤで送信できなかったパケットの受け取り
shared_ptr< InterfaceOutputQueue >	GetInterfaceOutputQueue (const unsigned int interfaceIndex) const	送信キューポインタの取得
shared_ptr< MacLayer >	GetMacLayerPtr (const unsigned int interfaceIndex) const	MAC レイヤポインタの取得
void	SetInterfaceOutputQueue (const unsigned int interfaceIndex, const shared_ptr< InterfaceOutputQueue > &outputQueuePtr)	送信キューの設定
void	ProcessLinkIsUpNotification (const unsigned int interfaceIndex)	MAC レイヤからのリンク接続完了の通知 (STA)
void	ProcessLinkIsDownNotification (const unsigned int interfaceIndex)	MAC レイヤからのリンク切断の通知 (STA)
void	ProcessNewLinkToANodeNotification (const unsigned int interfaceIndex, const GenericMacAddressType &newNodeMacAddress)	MAC レイヤからのノードへのリンク接続完了の通知 (AP)
bool	MacSupportsQualityOfService (const unsigned int interfaceIndex) const	MAC レイヤが QoS をサポートしているか否かの識別

shared_ptr < MacQualityOfServiceControlInterface >	GetMacQualityOfServiceInterface (const unsigned int interfaceIndex) const	MAC レイヤの QoS インタフェースポインタの取得
void	InsertPacketIntoAnOutputQueue (unique_ptr< Packet > &packetPtr, const unsigned int interfaceIndex, const NetworkAddress &nextHopAddress, const PacketPriorityType trafficClass, const EtherTypeFieldType etherType=ETHERTYPE_IS_NOT_ SPECIFIED)	パケットの送信キューへの挿入
void	SetupDhcpServerAndClientIfNecessary (const ParameterDatabaseReader &theParameterDatabaseReader, const shared_ptr< ApplicationLayer > &appLayerPtr)	DHCP サーバ/クライアントの設定
static shared_ptr < BasicNetworkLayer >	CreateNetworkLayer (const ParameterDatabaseReader &theParameterDatabaseReader, const GlobalNetworkingObjectBag &theGlobalNetworkingObjects, const shared_ptr< SimulationEngineInterface > &simEngineInterfacePtr, const NodeIdType &initNodeId, const RandomNumberGeneratorSeedType &nodeSeed)	BasicNetworkLayer の作成

4.13. ネットワークアドレス関連

ソースファイル: scensim_netaddress_ipv4.h, scensim_netaddress_ipv6.h

4.13.1. NetworkAddress

ネットワークアドレスの定義クラス

戻り値	関数(引数)	説明
	NetworkAddress (const uint32_t initIpAddress)	NetworkAddress クラスのコンストラクタ
	NetworkAddress (const uint64_t &ipAddressHighBits, const uint64_t &ipAddressLowBits)	NetworkAddress クラスのコンストラクタ (IPv6 用)
	NetworkAddress (const NetworkAddress &subnetAddress, const NetworkAddress &hostIdentifierOnlyAddress)	NetworkAddress クラスのコンストラクタ
void	SetAddressFromString (const string &stringToConvert, const NodeIdType &nodeId, bool &success)	文字列からのネットワークアドレスの設定
void	SetAddressFromString (const string &stringToConvert, bool &success)	文字列からのネットワークアドレスの設定
string	ConvertToString () const	ネットワークアドレスの文字列への変換
void	SetToTheBroadcastAddress ()	ブロードキャストアドレスの設定
void	SetToAnyAddress ()	エニアドレス(ビットが全て 0)の設定
bool	IsTheBroadcastAddress () const	ブロードキャストアドレス(ビットが全て 1)か否かの識別
bool	IsABroadcastAddress (const NetworkAddress &subnetMask) const	ブロードキャストアドレスか否かの識別
bool	IsAMulticastAddress () const	マルチキャストアドレスか否かの識別

bool	IsLinkLocalAddress () const	リンクローカルアドレスか否かの識別 (IPv6 用)
bool	IsAnyAddress () const	エニアドレス (ビットが全て 0) か否かの識別
bool	IsTheBroadcastOrAMulticastAddress () const	ブロードキャストアドレス (ビットが全て 1)、または、マルチキャストアドレスか否かの識別
bool	IsABroadcastOrAMulticastAddress (const Ipv4NetworkAddress &subnetMask) const	ブロードキャストアドレス、または、マルチキャストアドレスか否かの識別
uint64_t	GetRawAddressLowBits () const	ネットワークアドレスの下位 64 ビットの取得 (IPv6 用)
uint64_t	GetRawAddressHighBits () const	ネットワークアドレスの上位 64 ビットの取得 (IPv6 用)
uint32_t	GetRawAddressLow32Bits () const	ネットワークアドレスの下位 32 ビットの取得
unsigned int	GetMulticastGroupNumber () const	マルチキャストグループ番号の取得
void	SetWith32BitRawAddress (const uint32_t newIpAddress)	ネットワークアドレスの設定
bool	IsInSameSubnetAs (const NetworkAddress &address, const NetworkAddress &subnetMask) const	ネットワークアドレスが同一サブネットか否かの識別
NetworkAddress	MakeSubnetAddress (const NetworkAddress &subnetMask) const	サブネットアドレスの取得
NetworkAddress	MakeAddressWithZeroedSubnetBits (const NetworkAddress &subnetMask) const	サブネットビットを全て 0 にしたネットワークアドレスの取得
static NetworkAddress	ReturnTheBroadcastAddress ()	ブロードキャストアドレス (ビットが全て 1) の取得

static NetworkAddress	MakeABroadcastAddress (const NetworkAddress &subnetAddress, const NetworkAddress &subnetMask)	ブロードキャストアドレスの取得
static NetworkAddress	ReturnAnyAddress ()	エニアドレス(ビットが全て 0)の取得
static NetworkAddress	MakeSubnetMask (const unsigned int numberPrefixBits)	サブネットマスクの取得
static bool	IsIpv4StyleAddressString (const string &addressString)	IP アドレス記述が IPv4 型か否かの識別
型	メンバ変数	説明
static const unsigned int	numberBits	アドレスビット数 (IPv4 : 32、IPv6 : 128)
static const NetworkAddress	anyAddress	エニアドレス
static const NetworkAddress	broadcastAddress	ブロードキャストアドレス
static const NetworkAddress	invalidAddress	無効アドレス

4.14. IP ヘッダ関連

ソースファイル: scensim_ipv4.h、scensim_ipv6.h

4.14.1. IpHeaderModel

IP ヘッダの定義クラス

戻り値	関数(引数)	説明
	IpHeaderModel (const unsigned char trafficClass, const unsigned int payloadLengthBeforeIp, const unsigned char hopLimit, const unsigned char nextHeaderTypeCode, const NetworkAddress &sourceAddress, const NetworkAddress &destinationAddress)	IpHeaderModel クラスのコンストラクタ
const unsigned char *	GetPointerToRawBytes () const	ヘッダデータへのポインタ
unsigned int	GetNumberOfRawBytes () const	IP ヘッダ長の取得
unsigned int	GetNumberOfTrailingBytes ()	ヘッダに続くバイト数の取得
unsigned char	GetNextHeaderProtocolCode () const	次ヘッダのプロトコル番号の取得 (IPv6 用)
void	SetFinalNextHeaderProtocolCode (const unsigned char nextHeader)	最終の次ヘッダプロトコル番号の設定 (IPv6 用)
void	AddBindingUpdateExtensionHeader (const unsigned short sequenceNumber, const unsigned short lifetimein4SecUnits, const unsigned short bindingId)	バインディングアップデート拡張ヘッダ の追加 (IPv6 用)
void	AddHomeAddressDestinationOptionsHeader (const NetworkAddress	ホームアドレスオプションヘッダの追加

	&homeAddress)	(IPv6 用)
bool	HasIpssecEspOverhead () const	IPSec ESP を持っているか否かの識別
void	AddIpssecEspOverhead ()	IPSec ESP の追加 (IPv6 用)

4.14.2. IpHeaderOverlayModel

IP ヘッダオーバーレイ用のクラス

戻り値	関数(引数)	説明
	IpHeaderOverlayModel (const unsigned char *initHeaderPtr, const size_t initPacketLength)	IpHeaderOverlayModel クラスのコンストラクタ
	IpHeaderOverlayModel (unsigned char *initHeaderPtr, const size_t initPacketLength)	IpHeaderOverlayModel クラスのコンストラクタ
void	StopOverlayingHeader () const	IP ヘッダオーバーレイの停止
void	GetHeaderTotalLengthAndNextHeaderProtocolCode (unsigned int &headerLength, unsigned char &protocolCode) const	総ヘッダ長と次ヘッダのプロトコル番号の取得
unsigned int	GetLength () const	総ヘッダ長の取得
unsigned char	GetTrafficClass () const	トラフィッククラスの取得
unsigned short int	GetFlowLabel () const	フローラベルの取得 (IPv6 用)
unsigned char	GetNextHeaderProtocolCode () const	次ヘッダのプロトコル番号の取得
unsigned char	GetHopLimit () const	ホップリミットの取得
NetworkAddress	GetSourceAddress () const	送信元アドレスの取得
NetworkAddress	GetDestinationAddress () const	宛先アドレスの取得
void	SetTrafficClass (const unsigned char trafficClass)	トラフィッククラスの設定
void	SetFlowLabel (unsigned short int flowLabel)	フローラベルの設定 (IPv6 用)

void	SetHopLimit (const unsigned char hopLimit)	ホップリミットの設定
void	SetSourceAddress (const NetworkAddress &sourceAddress)	送信元アドレスの設定
void	SetDestinationAddress (const NetworkAddress &destinationAddress)	宛先アドレスの設定
bool	MobilityExtensionHeaderExists () const	モバイル IP 用拡張ヘッダが存在するか否かの識別
bool	MobileIpBindingUpdateHeaderExists () const	バインディングアップデートヘッダが存在するか否か他の識別
const MobileIpBindingUpdateExtensionHeader &	GetMobileIpBindingUpdateHeader () const	バインディングアップデートヘッダの取得 (IPv6 用)
bool	HomeAddressDestinationOptionsHeaderExists () const	ホームアドレスオプションヘッダが存在するか否かの識別
NetworkAddress	GetHomeAddressFromDestinationOptionsHeader () const	ホームアドレスオプションヘッダの取得 (IPv6 用)

4.15. ルーティングテーブル関連

ソースファイル: scensim_network.h

4.15.1. RoutingTable

ルーティングテーブル

戻り値	関数(引数)	説明
void	AddOrUpdateRoute (const NetworkAddress &destinationAddress, const NetworkAddress &nextHopAddress, const unsigned int nextHopInterfaceIndex)	ルーティングエントリの追加または更新 (サブネットマスクなし)
void	AddOrUpdateRoute (const NetworkAddress &destinationAddress, const NetworkAddress &destinationAddressSubnetMask, const NetworkAddress &nextHopAddress, const unsigned int nextHopInterfaceIndex)	ルーティングエントリの追加または更新
void	DeleteRoute (const NetworkAddress &destinationAddress)	ルーティングエントリの削除 (サブネットマスクなし)
void	DeleteRoute (const NetworkAddress &destinationAddress, const NetworkAddress &destinationAddressSubnetMask)	ルーティングエントリの削除
void	LookupRoute (const NetworkAddress &destinationAddress, bool &foundRoute, NetworkAddress &nextHopAddress, unsigned int	ルーティングテーブルの参照

	&nextHopInterfaceIndex) const	
--	-------------------------------	--

4.16. 送信キュー関連

ソースファイル: scensim_queues.h

4.16.1. InterfaceOutputQueue

送信キューの抽象クラス

戻り値	関数(引数)	説明
virtual bool	InsertWithFullPacketInformationModelsOn () const	キューへパケットを挿入する際、パケット情報を付与するモードか否かの識別
virtual void	Insert (unique_ptr< Packet > &packetPtr, const NetworkAddress &nextHopAddress, const PacketPriorityType priority, EnqueueResultType &enqueueResult, unique_ptr< Packet > &packetToDropPtr, const EtherTypeFieldType etherType=ETHERTYPE_IS_NOT_ SPECIFIED)=0	キューへのパケットの挿入 (純粋仮想関数)
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 PacketPriorityType priority, const unsigned short int ipv6FlowLabel, EnqueueResultType	キューへのパケットの挿入(パケット情報を含む) (純粋仮想関数)

	<code>&enqueueResult, unique_ptr< Packet > &packetToDropPtr)</code>	
virtual PacketPriorityType	MaxPossiblePacketPriority () const	キューで管理できるパケットの最大優先度の取得
virtual bool	IsEmpty () const =0	キューが空か否かの識別 (純粋仮想関数)
virtual void	DequeuePacket (unique_ptr< Packet > &packetPtr, NetworkAddress &nextHopAddress, PacketPriorityType &priority, EtherTypeFieldType ðerType)=0	キューからのパケットの取り出し (純粋仮想関数)

4.17. MAC レイヤ関連

ソースファイル: scensim_mac.h

4.17.1. MacLayer

MAC レイヤの抽象クラス

戻り値	関数(引数)	説明
virtual void	NetworkLayerQueueChangeNotification ()=0	送信キューに変化があった場合の処理 (純粋仮想関数)
virtual void	DisconnectFromOtherLayers ()=0	保持しているスマートポインタの解放 (純粋仮想関数)
virtual GenericMacAddr essType	GetGenericMacAddress () const	MAC アドレスの取得
virtual shared_ptr < MacQualityOfSer viceControllInterfa ce >	GetQualityOfServiceInterface () const	QoS 制御用インタフェースの取得
virtual shared_ptr < MacLayerInterfac eForEmulation >	GetMacLayerInterfaceForEmulation ()	エミュレーション用 MAC レイヤインタ フェースの取得
virtual shared_ptr < MacAndPhyInfoIn terface >	GetMacAndPhyInfoInterface ()	MAC/PHY 情報インタフェースの取得

4.17.2. MacAddressResolver

MAC アドレスリゾルバモデルの抽象クラス

戻り値	関数(引数)	説明
virtual void	GetMacAddress (const NetworkAddress &aNetworkAddress, const NetworkAddress &networkAddressMask, bool &wasFound, MacAddressType &resolvedMacAddress)=0	指定するネットワークアドレスから MAC アドレスを取得 (純粋仮想関数)
virtual void	GetNetworkAddressIfAvailable (const MacAddressType &macAddress, const NetworkAddress &subnetNetworkAddress, bool &wasFound, NetworkAddress &resolvedNetworkAddress)=0	指定する MAC アドレスからネットワー クアドレスを取得 (純粋仮想関数)

4.18. 電波伝搬関連

ソースファイル: scensim_prop.h

4.18.1. SimplePropagationModelForNode

電波伝搬モデルのインタフェースクラス

戻り値	関数(引数)	説明
void	TurnOnSignalsGetFramePtrSupport ()	フレームポインタの取得機能を有効化 (MIMO チャンネルモデル用)
bool	PropagationDelaysOn () const	伝播遅延が有効になっているか否かの識別
void	DisconnectThisInterface ()	チャンネルから本インタフェースの切断
bool	IAmDisconnected () const	本インタフェースがチャンネルに接続されているか否かの識別
TimeType	DisconnectTime () const	チャンネルから本インタフェースの切断された時刻の取得
bool	IAmNotReceivingSignals () const	シグナルを受信中か否かの識別
void	StopReceivingSignals ()	シグナルの受信停止
void	StartReceivingSignals ()	シグナルの受信開始
virtual NodeIdType	GetNodeId () const	ノード ID の取得
virtual unsigned int	GetInterfaceIndex () const	インタフェースインデックスの取得
InterfaceOrInstan celdType	GetInstanceld ()	チャンネルインスタンス ID の取得
virtual const AntennaModel &	GetAntennaModel () const	アンテナモデルの取得
virtual const shared_ptr < AntennaModel >	GetAntennaModelPtr ()	アンテナモデルポインタの取得
virtual ObjectMobilityMo del &	GetMobilityModel () const	モビリティモデルの取得

virtual shared_ptr < ObjectMobilityModel >	GetMobilityModelPtr () const	モビリティモデルポインタの取得
virtual const ObjectMobilityPosition	GetCurrentMobilityPosition () const	現在位置の取得
virtual bool	ReceivedSignalPowerIncludesMyAntennaGain () const	電力に自アンテナのゲインを含むか否かの識別
unsigned int	GetBaseChannelNumber () const	基準チャンネル番号の取得
unsigned int	GetChannelCount () const	チャンネル総数の取得
double	GetCarrierFrequencyMhz (const unsigned int channelNumber) const	指定するチャンネル番号のキャリア周波数の取得
double	GetCarrierFrequencyMhz () const	現在のチャンネルのキャリア周波数の取得
double	GetChannelBandwidthMhz (const unsigned int channelNumber) const	指定するチャンネル番号のキャリア帯域幅の取得
double	GetChannelBandwidthMhz () const	現在のチャンネルのキャリア帯域幅の取得
virtual void	TransmitSignal (const double &txPowerDbm, const TimeType &duration, FrameType *framePtr)	シグナルの送信 (シングルチャンネル用)
virtual void	TransmitSignal (const vector< unsigned int > &channelNumbers, const double &txPowerDbm, const TimeType &duration, FrameType *framePtr)	シグナルの送信
double	CalculatePathlossToLocation (const double &positionXMeters, const double &positionYMeters, const double &positionZMeters)	指定された位置までのパスロスの算出

	const	
void	CalculatePathlossToLocation (const PropagationInformationType &informationType, const double &positionXMeters, const double &positionYMeters, const double &positionZMeters, PropagationStatisticsType &propagationStatistics) const	指定された位置までのパスロスの算 出(パス情報を含む)
unsigned int	GetCurrentChannelNumber () const	現在のチャンネル番号の取得
const unsigned int	CurrentlyReceivingFramesOnBondedChannels () const	ボンディングされたチャンネルでフレー ムを受信しているか否かの識別
const vector< unsigned int > &	GetCurrentChannelNumberSet () const	現在のチャンネル番号群の取得(チャネ ルボンディング用)
bool	IsOnChannel (const unsigned int channelNum) const	指定するチャンネルを当該ノードが使用 しているか否かの識別
bool	ChannelsBeingUsed (const unsigned int channelNumber) const	指定するチャンネルが使用中か否かの 識別(エミュレーション用)
void	SwitchToChannelNumber (const unsigned int channelNumber, const bool doNotCalcInterferenceLevel=false)	チャンネルの切り替え(シングルチャネ ル用)
void	SwitchToChannelNumbers (const vector< unsigned int > channelNumberSet, const bool doNotCalcInterferenceLevel=false)	チャンネルの切り替え
void	SetRelativeAntennaAttitudeAzimuth (const double &azimuthDegrees)	相対的なアンテナ方位角の指定
TimeType	GetLastTransmissionEndTime () const	最終の送信終了時刻の取得

const FrameType *	GetCurrentlyTransmittingFramePtr () const	現在送信中のフレームのポインタの取得
double	GetLastTransmissionPowerDbm () const	最終の送信電力の取得
TimeType	GetLastChannelSwitchTime () const	最終のチャンネル変更時刻の取得
void	RegisterSignalHandler (SignalHandler *initSignalHandlerPtr)	シグナル開始用ハンドラの登録
void	RegisterSignalEndHandler (SignalHandler *aSignalHandlerPtr)	シグナル終了用ハンドラの登録
void	UnregisterSignalHandler ()	シグナル開始用ハンドラの登録解除
void	UnregisterSignalEndHandler ()	シグナル終了用ハンドラの登録解除
void	ReceiveIncomingSignal (const IncomingSignal &aSignal)	シグナルの受信開始
void	ReceiveIncomingSignalEnd (const IncomingSignal &aSignal)	シグナルの受信終了
virtual void	SetMobilityModel (const shared_ptr<ObjectMobilityModel> &newMobilityModelPtr)	モビリティモデルの設定
unsigned int	CurrentThreadPartitionIndex () const	現在のスレッドインデックスの取得
shared_ptr<SimplePropagationModel> < FrameType >	GetPropagationModel () const	電波モデルの取得
double	GetDistanceMetersTo (const NodeIdType &otherNodeId) const	指定するノードまでの距離の取得

4.18.2. SimplePropagationModelForNode::IncomingSignal

シグナル

戻り値	関数(引数)	説明
	IncomingSignal (const unsigned int channelNumber, const NodeIdType &sourceNodeId, const TimeType &startTime, const TimeType &duration, const double transmittedPowerDbm, const double receivedPowerDbm, const shared_ptr< const FrameType > &framePtr, const bool initIsANoiseFrame)	IncomingSignal クラスのコンストラクタ(シングルチャネル用) (スマートポインタ用)
	IncomingSignal (const vector< unsigned int > &channelNumbers, const NodeIdType &sourceNodeId, const TimeType &startTime, const TimeType &duration, const double transmittedPowerDbm, const double receivedPowerDbm, const shared_ptr< const FrameType > &framePtr, const bool initIsANoiseFrame)	IncomingSignal クラスのコンストラクタ (スマートポインタ用)
	IncomingSignal (const unsigned int channelNumber, const NodeIdType &sourceNodeId, const TimeType &startTime, const TimeType &duration, const double transmittedPowerDbm, const double receivedPowerDbm, const FrameType *framePtr, const bool initIsANoiseFrame)	IncomingSignal クラスのコンストラクタ(シングルチャネル用)
	IncomingSignal (const vector< unsigned int > &channelNumbers, const NodeIdType &sourceNodeId, const TimeType &startTime, const	IncomingSignal クラスのコンストラクタ

	TimeType &duration, const double transmittedPowerDbm, const double receivedPowerDbm, const FrameType *framePtr, const bool initIsANoiseFrame)	
NodeIdType	GetSourceNodeId () const	送信ノード ID の取得
unsigned int	GetChannelNumber () const	チャンネル番号の取得
unsigned int	GetNumberBondedChannels () const	ボンディングチャンネル数の取得
unsigned int	GetBondedChannelNumber (const unsigned int channelIndex)	指定するチャンネルインデックスの取得
bool	IsOnChannel (const unsigned int channelNum) const	指定するチャンネルが使用されているか否かの識別
bool	ChannelIntersectionIsEmpty (const vector< unsigned int > &receivedChannels) const	共通のチャンネルがあるか否かの識別
TimeType	GetStartTime () const	シグナルの送信開始時刻の取得
TimeType	GetDuration () const	シグナルの送信時間の取得
double	GetTransmittedPowerDbm () const	送信電力の取得
double	GetReceivedPowerDbm () const	受信電力の取得
double	GetPathlossDb () const	パスロスの取得
bool	HasACompleteFrame () const	完全なフレームを保持しているか否かの識別
bool	HasAFrame () const	フレームを保持しているか否かの識別
const FrameType &	GetFrame () const	フレームの取得
shared_ptr< const FrameType >	GetFrame () const	フレームの取得(スマートポインタ用)

4.18.3. SimplePropagationModelForNode::SignalHandler

シグナルのハンドラ

戻り値	関数(引数)	説明
virtual void	ProcessSignal (const IncomingSignal &aSignal)=0	シグナルの受信開始および受信終了 (純粋仮想関数)

4.18.4. SimplePropagationModel

電波伝搬モデル

戻り値	関数(引数)	説明
	SimplePropagationModel (const ParameterDatabaseReader &theParameterDatabaseReader, const RandomNumberGeneratorSeedType &runSeed, const shared_ptr< SimulationEngine > &simulationEnginePtr, const shared_ptr< GisSubsystem > &gisSubsystemPtr, const InterfaceOrInstanceldType &instanceld=nullInstanceld, const bool takeOwnershipOfFrames=false)	SimplePropagationModel クラスのコ ンストラクタ
void	TurnOnSignalsGetFramePtrSuppo rt ()	フレームポインタの取得機能を有効化 (MIMO チャンネルモデル用)
bool	PropagationDelaysOn () const	伝播遅延が有効になっているか否か の識別
InterfaceOrInstan celdType	GetInstanceld ()	チャンネルインスタンス ID の取得
void	DisconnectNodeInterface (const unsigned int channelNumber, const shared_ptr< SimplePropagationModelForNode< FrameType > > &interfacePtr)	指定されたノードインタフェースのチャ ネルからの切断
void	StopReceivingSignalsAtNode (const unsigned int channelNumber,	指定されたノードにおけるシグナルの 受信停止

	const shared_ptr< SimplePropagationModelForNode< FrameType > > &interfacePtr)	
void	StartReceivingSignalsAtNode (const unsigned int channelNumber, const shared_ptr< SimplePropagationModelForNode< FrameType > > &interfacePtr)	指定されたノードにおけるシグナルの受信開始
void	AddNodeToChannel (const unsigned int channelNumber, const shared_ptr< SimplePropagationModelForNode< FrameType > > &nodeInterfacePtr)	指定されたノードインタフェースのチャネルへの接続
void	InvalidateCachedInformationFor (const SimplePropagationModelForNode< FrameType > &nodeInfo)	パスロスキャッシュの無効化
void	DeleteNodeFromChannel (const unsigned int channelNumber, const shared_ptr< SimplePropagationModelForNode< FrameType > > &nodeInterfacePtr)	指定されたノードインタフェースのチャネルからの削除
virtual shared_ptr< SimplePropagationModelForNode< FrameType > >	GetNewPropagationModelInterface (const shared_ptr< SimulationEngineInterface > &simEngineInterfacePtr, const shared_ptr< AntennaModel > &antennaModelPtr, const shared_ptr< ObjectMobilityModel > &antennaMobilityModelPtr, const NodeIdType &nodeId, const InterfaceIdType &interfaceId, const unsigned int interfaceIndex)	電波伝搬モデルインタフェースの新規取得 (インタフェース指定あり)
virtual shared_ptr<	GetNewPropagationModelInterface (e (電波伝搬モデルインタフェースの新規取得

SimplePropagationModelForNode < FrameType > >	const shared_ptr< SimulationEngineInterface > &simEngineInterfacePtr, const shared_ptr< AntennaModel > &antennaModelPtr, const shared_ptr< ObjectMobilityModel > &antennaMobilityModelPtr, const NodeIdType &nodeId)	(インタフェース指定なし)
double	GetCarrierFrequencyMhz (const unsigned int channelNumber) const	キャリア周波数の取得
double	GetChannelBandwidthMhz (const unsigned int channelNumber) const	キャリア帯域幅の取得
unsigned int	GetBaseChannelNumber () const	基準チャネル番号の取得
unsigned int	GetChannelCount () const	チャネル総数の取得
bool	ChannelsBeingUsed (const unsigned int channelNumber) const	チャネルが使用中か否かの識別(エミュレーション用)
shared_ptr< SimplePropagationLossCalculationModel >	GetPropagationCalculationModel () const	パスロス計算モデルの取得
void	CalculatePathlossToNode (const PropagationInformationType &informationType, const ObjectMobilityPosition &txAntennaPosition, const ObjectMobilityModel::MobilityObjectIdType &txObjectId, const AntennaModel &txAntennaModel, const ObjectMobilityPosition &rxAntennaPosition, const	ノード間のパスロス算出

	ObjectMobilityModel::MobilityObjectIdType &rxObjectId, const AntennaModel &rxAntennaModel, const unsigned int channelNumber, PropagationStatisticsType &propagationStatistics) const	
void	CalculatePathlossToLocation (const PropagationInformationType &informationType, const TimeType ¤tTime, const SimplePropagationModelForNode<FrameType> &transmittingNodeInfo, const double &positionXMeters, const double &positionYMeters, const double &positionZMeters, const unsigned int channelNumber, PropagationStatisticsType &propagationStatistics) const	指定された位置までのパスロスの算出
void	SwitchToChannelNumber (const shared_ptr<SimplePropagationModelForNode<FrameType>> &switchingNodeInfoPtr, const unsigned int channelNumber, const bool doNotCalcInterferenceLevel)	チャンネルの切り替え(シングルチャンネル用)
void	SwitchToChannelNumbers (const shared_ptr<SimplePropagationModelForNode<FrameType>> &receivingNodeInfoPtr, const vector<unsigned int> &channelNumbers, const bool doNotCalcInterferenceLevel)	チャンネルの切り替え
protected		
void	ScheduleSignalEventAtNode (SimulationEngineInterface	シグナルの受信開始および受信終了イベントの登録(チャンネルボンディング

	<pre> &simEngineInterface, const NodeIdType &txNodeId, const vector< unsigned int > &channelNumbers, const shared_ptr< SimplePropagationModelForNode< FrameType > > &receivingNodeInfoPtr, const double &transmitPowerDbm, const double &receivedPowerDbm, const TimeType &currentTime, const TimeType &propagationDelay, const TimeType &duration, const FrameType *framePtr, const bool isANoiseFrame=false) </pre>	用)
void	<pre> ScheduleSignalEventAtNode (SimulationEngineInterface &simEngineInterface, const NodeIdType &txNodeId, const unsigned int channelNumber, const shared_ptr< SimplePropagationModelForNode< FrameType > > &receivingNodeInfoPtr, const double &transmitPowerDbm, const double &receivedPowerDbm, const TimeType &currentTime, const TimeType &propagationDelay, const TimeType &duration, const FrameType *framePtr, const bool isANoiseFrame=false) </pre>	シグナルの受信開始および受信終了 イベントの登録
bool	<pre> NodeCanHearSignal (const SimplePropagationModelForNode< FrameType > &receivingNodeInfo, const IncomingSignal &aSignal) </pre>	指定するノードが当該シグナルのチャ ネルを聞いているか否かの識別

	const	
void	SendSignalStartEventToNode (SimplePropagationModelForNode< FrameType > &receivingNodeInfo, IncomingSignal *aSignalPtr)	シグナルの受信開始イベントのノード への送信
void	SendSignalEndEventToNode (SimplePropagationModelForNode< FrameType > &receivingNodeInfo, IncomingSignal *aSignalPtr)	シグナルの受信終了イベントのノード への送信
void	DeleteIncomingSignal (SimplePropagationModelForNode< FrameType > &receivingNodeInfo, IncomingSignal *aSignalPtr)	シグナル情報の削除
void	TransmitSignalToSingleNode (SimulationEngineInterface &simEngineInterface, const NodeIdType &txNodeId, const unsigned int txInterfaceIndex, const vector< unsigned int > &channelNumbers, const ObjectMobilityPosition &txAntennaPosition, const AntennaModel &txAntennaModel, const double &transmitPowerDbm, const TimeType ¤tTime, const TimeType &duration, const FrameType *framePtr, const shared_ptr< SimplePropagationModelForNode< FrameType > > &receivingNodeInfoPtr, const bool isANoiseFrame=false)noPropagation Delay=false)	ノードへのシグナルの送信

void	TransmitSignalToSingleNode (SimulationEngineInterface &simEngineInterface, const NodeIdType &txNodeId, const unsigned int txInterfaceIndex, const unsigned int txChannelNumber, const ObjectMobilityPosition &txAntennaPosition, const AntennaModel &txAntennaModel, const double &transmitPowerDbm, const TimeType ¤tTime, const TimeType &duration, const FrameType *framePtr, const shared_ptr< SimplePropagationModelForNode< FrameType > > &receivingNodeInfoPtr, const bool isANoiseFrame=false)	ノードへのシグナルの送信 (シングル チャンネル用)
void	TransmitSignalInLocalPartition (SimulationEngineInterface &simEngineInterface, const NodeIdType &txNodeId, const unsigned int txInterfaceIndex, const unsigned int txChannelNumber, const ObjectMobilityPosition &txAntennaPosition, const AntennaModel &txAntennaModel, const double &transmitPowerDbm, const TimeType ¤tTime, const TimeType &duration, const FrameType *framePtr)	シグナルの送信 (シングルチャンネル 用)
void	TransmitSignalInLocalPartition (SimulationEngineInterface &simEngineInterface, const NodeIdType &txNodeId, const unsigned int txInterfaceIndex, const	シグナルの送信

	<pre>vector< unsigned int > &channelNumbers, const ObjectMobilityPosition &txAntennaPosition, const AntennaModel &txAntennaModel, const double &transmitPowerDbm, const TimeType &currentTime, const TimeType &duration, const FrameType *framePtr)</pre>	
void	<pre>TransmitSignalInLocalPartitionUtil izingMultipleThreads (SimulationEngineInterface &simEngineInterface, const NodeIdType &txNodeId, const unsigned int txChannelNumber, const ObjectMobilityPosition &txAntennaPosition, const AntennaModel &txAntennaModel, const double &transmitPowerDbm, const TimeType &currentTime, const TimeType &duration, const FrameType *framePtr)</pre>	<p>シグナルの送信 (シングルチャネル用)</p> <p>(並列計算用)</p>
void	<pre>TransmitSignalInLocalPartitionUtil izingMultipleThreads (SimulationEngineInterface &simEngineInterface, const NodeIdType &txNodeId, const vector< unsigned int > &channelNumbers, const ObjectMobilityPosition &txAntennaPosition, const AntennaModel &txAntennaModel, const double &transmitPowerDbm, const TimeType &currentTime, const TimeType &duration, const FrameType *framePtr)</pre>	<p>シグナルの送信 (並列計算用)</p>

void	TransmitSignal (SimulationEngineInterface &simEngineInterfaceForTxNode, SimplePropagationModelForNode< FrameType > &transmittingNodeInfo, const unsigned int channelNumber, const double &transmitPowerDbm, const TimeType &duration, FrameType *framePtr)	シグナルの送信
void	TransmitSignal (SimulationEngineInterface &simEngineInterfaceForTxNode, SimplePropagationModelForNode< FrameType > &transmittingNodeInfo, const vector< unsigned int > &channelNumbers, const double &transmitPowerDbm, const TimeType &duration, FrameType *framePtr)	シグナルの送信(チャネルボンディング用)
void	TransmitChannelInterferenceSignals (SimulationEngineInterface &simEngineInterface, const NodeIdType &txNodeId, const unsigned int txInterfaceIndex, const unsigned int txChannelNumber, const ObjectMobilityPosition &txAntennaPosition, const AntennaModel &txAntennaModel, const double &transmitPowerDbm, const TimeType ¤tTime, const TimeType &duration)	干渉波シグナルの送信(シングルチャネル用)
void	TransmitChannelInterferenceSignals (SimulationEngineInterface &simEngineInterface, const	干渉波シグナルの送信

	NodeIdType &txNodeId, const unsigned int txInterfaceIndex, const vector< unsigned int > &txChannelNumbers, const ObjectMobilityPosition &txAntennaPosition, const AntennaModel &txAntennaModel, const double &transmitPowerDbm, const TimeType ¤tTime, const TimeType &duration)	
--	-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	--

4.19. パスロス関連

ソースファイル: [scensim_proploss.h/cpp](#)

4.19.1. SimplePropagationLossCalculationModel

パスロスモデルの抽象クラス

戻り値	関数(引数)	説明
	SimplePropagationLossCalculationModel (const double &carrierFrequencyMhz, const double &maximumPropagationDistanceMeters=DBL_MAX, const bool propagationDelaysEnabled=false, const int numberDataParallelThreads=0)	SimplePropagationLossCalculationModel クラスのコンストラクタ
double	GetCarrierFrequencyMhz () const	キャリア周波数の取得
virtual double	CalculatePropagationLossDb (const ObjectMobilityPosition &txAntennaPosition, const MobilityObjectIdType &txObjectId, const ObjectMobilityPosition &rxAntennaPosition, const MobilityObjectIdType &rxObjectId, const double &xyDistanceSquaredMeters) const	2 点間のパスロスの算出
virtual double	CalculatePropagationLossDbParallelVersion (const ObjectMobilityPosition &txAntennaPosition, const ObjectMobilityModel::MobilityObjectIdType &txObjectId, const ObjectMobilityPosition &rxAntennaPosition, const	2 点間のパスロスの算出 (並列計算用)

	ObjectMobilityModel::MobilityObjectIdType &rxObjectId, const double &xyDistanceSquaredMeters, const int calculationThreadId) const	
virtual void	SetTimeTo (const TimeType &time)	トレース(事前計算済み)モデルにおける時刻の移動
virtual bool	PropagationLossIsSymmetricValue () const	パスロス値が送受信点に対して対称か否かの識別
virtual bool	SupportMultipointCalculation () const	複数地点の同時計算をサポートしているか否かの識別
virtual void	CacheMultipointPropagationLossDb (const TimeType ¤tTime, const ObjectMobilityPosition &txAntennaPosition, const vector< ObjectMobilityPosition > &rxAntennaPositions)	複数地点のパスロスのキャッシュ
bool	IsCloserThanMaxPropagationDistance (const ObjectMobilityPosition &txAntennaPosition, const ObjectMobilityPosition &rxAntennaPosition) const	2 点間の距離が最大伝搬距離より短いか否かの識別
void	CalculateOrRetrieveTotalLossDbAndPropDelay (SignalLossCache &aSignalLossCache, const TimeType ¤tTime, const unsigned int channelNumber, const NodeIdType &txNodeId, const unsigned int txInterfaceIndex, const ObjectMobilityPosition &txAntennaPosition, const AntennaModel &txAntenna, const	2 ノード間の総伝搬ロスおよび伝搬遅延の算出または検索

	NodeIdType &rxNodeId, const unsigned int rxInterfaceIndex, ObjectMobilityModel &rxMobilityModel, const AntennaModel &rxAntenna, double &totalLossDb, TimeType &propagationDelay)	
double	CalculateTotalAntennaGainDbi (const ObjectMobilityPosition &txAntennaPosition, const AntennaModel &txNodeAntenna, const ObjectMobilityPosition &rxAntennaPosition, const AntennaModel &rxNodeAntenna) const	送受信ノードの総アンテナゲインの算出
double	CalculateAntennaGainDbi (const ObjectMobilityPosition &antennaPosition, const AntennaModel &nodeAntenna, const double destX, const double destY, const double destZ) const	アンテナゲインの算出
virtual void	CalculatePropagationPathInformation (const PropagationInformationType &informationType, const ObjectMobilityPosition &txAntennaPosition, const MobilityObjectIdType &txObjectId, const AntennaModel &txAntennaModel, const ObjectMobilityPosition &rxAntennaPosition, const MobilityObjectIdType &rxObjectId, const AntennaModel &rxAntennaModel, PropagationStatisticsType	パスロスおよびレイパス情報の算出

	&propagationStatistics) const	
void	ClearParallelCalculationSet ()	並列計算用パスロス情報コンテナの 削除 (並列計算用)
void	AddToParallelCalculationSet (const ObjectMobilityPosition &txAntennaPosition, const AntennaModel *txAntennaModelPtr, const size_t &rxNodeIndex, const ObjectMobilityPosition &rxAntennaPosition, const shared_ptr< AntennaModel > &rxAntennaModelPtr)	並列計算用パスロス情報コンテナの 追加 (並列計算用)
void	CalculateLossesDbAndPropDelay sInParallel ()	2 点間の総伝搬ロスおよび伝搬遅延 の算出 (並列計算用)
unsigned int	GetNumberOfParallelCalculations () const	現在の並列計算数の取得 (並列計算用)
double	GetTotalLossDb (const size_t jobIndex) const	総伝搬ロスの取得 (並列計算用)
TimeType	GetPropagationDelay (const size_t JobIndex) const	伝搬遅延の取得 (並列計算用)
NodeIdType	GetTxNodeId (const size_t jobIndex) const	送信ノード ID の取得 (並列計算用)
ObjectMobilityPos ition &	GetTxAntennaPosition (const size_t jobIndex) const	送信アンテナ位置の取得 (並列計算用)
unsigned int	GetRxNodeIndex (const size_t jobIndex) const	受信ノードインデックスの取得 (並列計算用)
ObjectMobilityPos ition &	GetRxAntennaPosition (const size_t jobIndex) const	受信アンテナ位置の取得 (並列計算用)
virtual bool	RayPathTracelsEnabled () const	レイパストレースが有効か否かの識

		別 (HFPM 用)
virtual double	CalculateTotalLossDbWithRayPathTrace (const ObjectMobilityPosition &txAntennaPosition, const MobilityObjectIdType &txNodeId, const AntennaModel &txNodeAntenna, const ObjectMobilityPosition &rxAntennaPosition, const MobilityObjectIdType &rxNodeId, const AntennaModel &rxNodeAntenna, double &totalLossDb)	レイパストレースを用いた総伝搬ロス の算出 (HFPM 用)
protected		
virtual double	CalculatePropagationLossDb (const ObjectMobilityPosition &txAntennaPosition, const ObjectMobilityPosition &rxAntennaPosition, const double &xyDistanceSquaredMeters) const =0	2 点間のパスロスの算出 (純粹仮想関数)

4.20. アンテナ関連

ソースファイル: scensim_proploss.h

4.20.1. AntennaModel

アンテナモデルの抽象クラス

戻り値	関数(引数)	説明
virtual bool	IsOmniDirectional () const =0	アンテナがオムニ(全方位)か否かの識別 (純粋仮想関数)
virtual double	GetOmniGainDbi () const =0	オムニアンテナのゲインの取得 (純粋仮想関数)
virtual double	GainInDbForThisDirection (const double &azimuthFromBoresightClockwiseDegrees=0.0, const double &elevationFromBoresightDegrees=0.0, const double ¤tAntennaRotation=0.0) const =0	指定する方位に対するアンテナゲインの取得 (純粋仮想関数)
virtual bool	SupportsQuasiOmniMode () const	擬似オムニモードをサポートしているか否かの識別
virtual bool	IsInQuasiOmniMode () const	擬似オムニモード中か否かの識別
virtual void	SwitchToQuasiOmniMode ()	擬似オムニモードに移行
virtual void	SwitchToDirectionalMode ()	ディレクショナルモードに移行

4.21. モビリティ関連

ソースファイル: scensim_proploss.h、scensim_mobility.h

4.21.1. ObjectMobilityPosition

オブジェクトの位置の定義クラス

戻り値	関数(引数)	説明
	ObjectMobilityPosition (const TimeType &initLastMoveTime, const TimeType &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) 	ObjectMobilityPosition クラスのコンストラクタ
TimeType	LastMoveTime () const	最終の移動時刻の取得
TimeType	EarliestNextMoveTime () const	次の移動時刻の取得
double	X_PositionMeters () const	X 座標の取得
double	Y_PositionMeters () const	Y 座標の取得
double	HeightFromGroundMeters () const	高さの取得
bool	TheHeightContainsGroundHeightMeters () const	高さに地表高が含まれているか否かの取得

double	AttitudeAzimuthFromNorthClockwiseDegrees () const	方位の取得
double	AttitudeElevationFromHorizonDegrees () const	仰角の取得
double	VelocityMetersPerSecond () const	速度(速さ)の取得
double	VelocityAzimuthFromNorthClockwiseDegrees () const	速度(方位)の取得
double	VelocityElevationFromHorizonDegrees () const	速度(仰角)の取得
void	SetLastMoveTime (const TimeType &lastMoveTime)	最終の移動時刻の設定
void	SetEarliestNextMoveTime (const TimeType &nextMoveTime)	次の移動時刻の設定
void	SetX_PositionMeters (const double &newXPosition)	X 座標の設定
void	SetY_PositionMeters (const double &newYPosition)	Y 座標の設定
void	SetHeightFromGroundMeters (const double &newHeight)	高さの設定
void	SetTheHeightContainsGroundHeightMeters (const bool newTheHeightContainsGroundHeightMeters)	高さに地表高が含まれているか否かの設定
void	SetAttitudeFromNorthClockwiseDegrees (const double &newAttitude)	方位の設定
void	SetAttitudeElevationFromHorizonDegrees (const double &newElevation)	仰角の設定
void	SetVelocityMetersPerSecond (const double &newVelocity)	速度(速さ)の設定
void	SetVelocityFromNorthClockwiseDegrees (const double &newVelocityAzimuth)	速度(方位)の設定
void	SetVelocityElevationFromHorizon	速度(仰角)の設定

	Degrees (const double &newVelocityElevation)	
--	---------------------------------------------------------	--

4.21.2. ObjectMobilityModel

モビリティモデルの抽象クラス

戻り値	関数(引数)	説明
	ObjectMobilityModel (const ParameterDatabaseReader &theParameterDatabaseReader, const NodeIdType &nodeId, const InterfaceOrInstanceIdType &interfaceId)	ObjectMobilityModel クラスのコンストラクタ
void	GetPositionForTime (const TimeType &snapshotTime, ObjectMobilityPosition &position)	指定時刻のノード位置の取得
virtual void	GetUnadjustedPositionForTime (const TimeType &snapshotTime, ObjectMobilityPosition &position)=0	指定時刻のノード位置の取得(方位角の調整を含まない)
virtual TimeType	GetCreationTime () const	ノードの生成時刻の取得
virtual TimeType	GetDeletionTime () const	ノードの消滅時刻の取得
void	SetRelativeAttitudeAzimuth (const TimeType ¤tTime, const double &azimuthDegrees)	相対方位角の設定
double	GetRelativeAttitudeAzimuth () const	相対方位角の取得

