EMANE
Developer Manual

0.7.3

labs.cengen.com/emane

EMANE
Developer Manual
0.7.3

DRS CenGen, LLC
1120 Route 22 East
Building 1, Suite 7
Bridgewater, NJ 08807

February 27, 2012

EMANE is being developed by the Naval Research Laboratory (NRL) under the OSD Network Communication Capability Program (NCCP) and in cooperation with the Army Research Laboratory (ARL) High Performance Computing Mobile Network Modeling Institute (HPC MNMI) effort.

Copyright© 2011-2012 - DRS CenGen, LLC, Bridgewater, New Jersey

This work is licensed under the Creative Commons Attribution 3.0 Unported License. To view a copy of this license, visit http://creativecommons.org/licenses/by/3.0/ or send a letter to Creative Commons, 444 Castro Street, Suite 900, Mountain View, California, 94041, USA.

Contents

1 Introduction **1**
1.1 How this Manual is Organized . . . 1

I Emulation Infrastructure **3**

2 The Platform **5**
2.1 XML Configuration in Brief . . . 5
2.2 Constructing NEM Layers . . . 6

3 EMANE Components **9**
3.1 `EMANE::Component` Interface . . . 10
3.1.1 `EMANE::Component::initialize` . . . 10
3.1.2 `EMANE::Component::configure` . . . 10
3.1.3 `EMANE::Component::start` . . . 10
3.1.4 `EMANE::Component::postStart` . . . 11
3.1.5 `EMANE::Component::stop` . . . 11
3.1.6 `EMANE::Component::destroy` . . . 11
3.2 Component Configuration . . . 12
3.2.1 Implementing `ConfigurationDefinition` . . . 12

4 NEM Layer Anatomy **15**
4.1 Layer Communication . . . 16
4.1.1 `EMANE::UpstreamTransport` Interface . . . 17
4.1.1.1 `EMANE::UpstreamTransport::processUpstreamPacket` . . . 17
4.1.1.2 `EMANE::UpstreamTransport::processUpstreamControl` . . . 17
4.1.1.3 `EMANE::UpstreamTransport::sendDownstreamPacket` . . . 18
4.1.1.4 `EMANE::UpstreamTransport::sendDownstreamControl` . . . 18
4.1.2 `EMANE::DownstreamTransport` Interface . . . 18
4.1.2.1 `EMANE::DownstreamTransport::processDownstreamPacket` . . . 18
4.1.2.2 `EMANE::DownstreamTransport::processDownstreamControl` . . . 18
4.1.2.3 `EMANE::DownstreamTransport::sendUpstreamPacket` . . . 18
4.1.2.4 `EMANE::DownstreamTransport::sendUpstreamControl` . . . 19
4.1.3 `EMANE::UpstreamPacket Interface` . . . 19
4.1.3.1 `EMANE::UpstreamPacket::strip` . . . 19
4.1.3.2 `EMANE::UpstreamPacket::get` . . . 19
4.1.3.3 `EMANE::UpstreamPacket::length` . . . 19
4.1.3.4 `EMANE::UpstreamPacket::getPacketInfo` . . . 19
4.1.4 `EMANE::DownstreamPacket Interface` . . . 19
4.1.4.1 `EMANE::DownstreamPacket::prepend` . . . 20
4.1.4.2 `EMANE::DownstreamPacket::combine` . . . 20
4.1.4.3 `EMANE::DownstreamPacket::get` . . . 20

4.1.4.4 EMANE::DownstreamPacket::length . . . 20
4.1.4.5 EMANE::DownstreamPacket::getPacketInfo . . . 20
4.1.5 EMANE::ControlMessage Interface . . . 20
4.1.5.1 EMANE::ControlMessage::ControlMessage . . . 20
4.1.5.2 EMANE::ControlMessage::get . . . 21
4.1.5.3 EMANE::ControlMessage::length . . . 21
4.1.5.4 EMANE::ControlMessage::getMajorIdentifier . . . 21
4.1.5.5 EMANE::ControlMessage::getMinorIdentifier . . . 21
4.2 Event Processing . . . 21
4.2.1 EMANE::TimerServiceHandler::processTimedEvent . . . 22
4.2.2 EMANE::EventServiceHandler::processEvent . . . 22
4.3 Message Processing . . . 23

5 Platform Services **25**
5.1 EMANE::LogServiceProvider . . . 25
5.1.1 EMANE::LogServiceProvider::log . . . 26
5.1.2 EMANE::LogServiceProvider::vlog . . . 26
5.1.3 Sample Usage . . . 26
5.2 EMANE::EventServiceProvider . . . 26
5.2.1 EMANE::EventServiceProvider::sendEvent . . . 27
5.2.2 EMANE::EventServiceProvider::sendEvent . . . 27
5.2.3 Sample Usage . . . 27
5.3 EMANE::StatisticServiceProvider . . . 28
5.3.1 EMANE::StatisticServiceProvider::registerStatistic . . . 28
5.3.2 EMANE::StatisticServiceProvider::unregisterStatistic . . . 28
5.3.3 Sample Usage . . . 28
5.4 EMANE::RandomNumberServiceProvider . . . 29
5.4.1 EMANE::RandomNumberServiceProvider::getRandomNumber . . . 29
5.4.2 EMANE::RandomNumberServiceProvider::getRandomNumber . . . 29
5.4.3 EMANE::RandomNumberServiceProvider::getRandomProbability . . . 30
5.4.4 Sample Usage . . . 30
5.5 EMANE::TimerServiceProvider . . . 30
5.5.1 EMANE::TimerServiceProvider::scheduleTimedEvent . . . 30
5.5.2 EMANE::TimerServiceProvider::cancelTimedEvent . . . 31
5.5.3 Sample Usage . . . 31

II Component APIs **33**

6 MAC Layer **35**
6.1 MAC Layer Packet Processing . . . 35
6.1.1 EMANE::MACLayerImplementor::processUpstreamPacket . . . 35
6.1.2 EMANE::MACLayerImplementor::sendDownstreamPacket . . . 36
6.2 MAC Layer Common Header . . . 36
6.3 Implementing a MAC Layer . . . 36

7 PHY Layer **39**
7.1 PHY Layer Packet Processing . . . 39
7.1.1 EMANE::PHYLayerImplementor::processUpstreamPacket . . . 39
7.1.2 EMANE::PHYLayerImplementor::sendDownstreamPacket . . . 40
7.2 PHY Layer Common Header . . . 40
7.2.1 EMANE::CommonPHYHeader::CommonPHYHeader . . . 41
7.2.2 EMANE::CommonPHYHeader::getRegistrationId . . . 42
7.2.3 EMANE::CommonPHYHeader::getTxPowerdBm . . . 42

7.2.4 EMANE::CommonPHYHeader::getAntennaGaindBi . . . 42
7.2.5 EMANE::CommonPHYHeader::getTxTime . . . 42
7.2.6 EMANE::CommonPHYHeader::getDuration . . . 42
7.2.7 EMANE::CommonPHYHeader::getFrequencyKHz . . . 42
7.2.8 EMANE::CommonPHYHeader::getBandwidthKHz . . . 42
7.2.9 EMANE::CommonPHYHeader::getSequenceNumber . . . 42
7.2.10 EMANE::CommonPHYHeader::prependTo . . . 42
7.2.11 EMANE::CommonPHYHeader::addOption . . . 43
7.2.12 EMANE::CommonPHYHeader::getOptionList . . . 43
7.3 Implementing a PHY Layer . . . 43

8 Shim Layer **47**
8.1 Implementing a Shim Layer . . . 47

9 Events **51**
9.1 Implementing an Event . . . 52
9.2 Event Service . . . 53
9.2.1 Implementing an Event Generator . . . 53
9.3 Event Daemon . . . 56
9.3.1 Implementing an Event Agent . . . 56

10 Transports **59**
10.1 Implementing a Transport . . . 59

III Universal PHY Layer **63**

11 Interfacing with the Universal PHY Layer **65**
11.1 Model Features . . . 65
11.1.1 Pathloss Calculation . . . 65
11.1.2 Receive Power Calculation . . . 65
11.1.3 Directional Sector Antenna Support . . . 66
11.1.4 Noise Processing . . . 67
11.1.5 MAC-PHY Control Messaging . . . 67
11.2 Control Antenna Message . . . 68
11.3 Control Send Message . . . 68
11.4 Control Receive Message . . . 70
11.5 Antenna Direction Event . . . 70

IV EMANE Libraries **73**

12 libemaneeventservice **75**
12.1 Configuration . . . 75
12.2 libemaneeventservice API . . . 75
12.2.1 emaneEventServiceInitialize . . . 75
12.2.2 emaneEventServiceDestroy . . . 76
12.2.3 emaneEventServiceSubscribe . . . 76
12.2.4 emaneEventServiceUnsubscribe . . . 76
12.2.5 emaneEventServicePublish . . . 76
12.2.6 emaneEventServiceLoop . . . 77
12.2.7 emaneEventServiceLoopWithDefault . . . 77
12.2.8 emaneEventServiceNextEvent . . . 77
12.2.9 emaneEventServiceBreakloop . . . 77
12.2.10 emaneEventServiceError . . . 77

12.2.11 Sample Usage . . . 78
12.3 libemaneeventlocation API . . . 79
12.3.1 emaneEventLocationInitialize . . . 79
12.3.2 emaneEventLocationInitializeFromImport . . . 79
12.3.3 emaneEventLocationDestroy . . . 79
12.3.4 emaneEventLocationSetLocation . . . 80
12.3.5 emaneEventLocationGetLocation . . . 80
12.3.6 emaneEventLocationNodeCount . . . 80
12.3.7 emaneEventLocationExportSizeBytes . . . 80
12.3.8 emaneEventLocationExport . . . 81
12.3.9 emaneEventLocationError . . . 81
12.4 libemaneeventpathloss API . . . 81
12.4.1 emaneEventPathlossInitialize . . . 81
12.4.2 emaneEventPathlossInitializeFromImport . . . 81
12.4.3 emaneEventPathlossDestroy . . . 81
12.4.4 emaneEventPathlossSetPathloss . . . 82
12.4.5 emaneEventPathlossGetPathloss . . . 82
12.4.6 emaneEventPathlossNodeCount . . . 82
12.4.7 emaneEventPathlossExportSizeBytes . . . 82
12.4.8 emaneEventPathlossExport . . . 83
12.4.9 emaneEventPathlossError . . . 83
12.5 libemaneeventantennadirection API . . . 83
12.5.1 emaneEventAntennaDirectionInitialize . . . 83
12.5.2 emaneEventAntennaDirectionInitializeFromImport . . . 83
12.5.3 emaneEventAntennaDirectionDestroy . . . 83
12.5.4 emaneEventAntennaDirectionSetAntennaDirection . . . 84
12.5.5 emaneEventAntennaDirectionGetAntennaDirection . . . 84
12.5.6 emaneEventAntennaDirectionNodeCount . . . 84
12.5.7 emaneEventAntennaDirectionExportSizeBytes . . . 85
12.5.8 emaneEventAntennaDirectionExport . . . 85
12.5.9 emaneEventAntennaDirectionError . . . 85

13 libemane **87**
13.1 EMANE::Logger . . . 87
13.1.1 setLogLevel . . . 87
13.1.2 redirectLogsToSysLog . . . 87
13.1.3 redirectLogsToFile . . . 88
13.1.4 redirectLogsToRemoteLogger . . . 88
13.1.5 log . . . 88
13.1.6 Sample Usage . . . 88
13.2 EMANE::NEMBuilder . . . 88
13.2.1 buildPHYLayer . . . 88
13.2.2 buildMACLayer . . . 89
13.2.3 buildShimLayer . . . 89
13.2.4 buildNEM . . . 89
13.2.5 buildPlatform . . . 89
13.2.6 Sample Usage . . . 90
13.3 EMANE::TransportBuilder . . . 90
13.3.1 buildTransport . . . 90
13.3.2 buildTransportAdapter . . . 90
13.3.3 buildTransportManager . . . 90
13.3.4 Sample Usage . . . 91
13.4 EMANE::EventGeneratorBuilder . . . 93
13.4.1 buildEventGenerator . . . 94

13.4.2 buildEventGeneratorManager 94
13.4.3 Sample Usage 94
13.5 EMANE::EventAgentBuilder 96
13.5.1 buildEventAgent 96
13.5.2 buildEventAgentManager 96
13.5.3 Sample Usage 96

Chapter 1

Introduction

In its most basic form, EMANE is an open source[1] infrastructure used for Mobile Ad-hoc Network Emulation. EMANE provides a set of APIs to allow independent development of network emulation modules (NEMs), emulation/application boundary interfaces (transports) and emulation environmental data distribution mechanisms (events).

The key to the flexibility of EMANE is the use of XML based factories to determine which emulation components to instantiate and where they reside once deployed. There are four types of component containers that can be configured to create and manage a variable number of component plugin instances:

- The *NEM Platform Server* creates and manages network emulation modules
- The *Transport Daemon* creates and manages emulation/application boundary interfaces
- The *Event Service* creates and manages emulation environmental data generators
- The *Event Daemon* creates and manages boundary agents that bridge emulation environmental data between the emulation and application space

This manual is designed to introduce the main architectural elements of EMANE and the available APIs used to develop emulation component plugins.

1.1 How this Manual is Organized

This manual is organized in four parts:

1. Emulation Infrastructure
2. Component APIs
3. Universal PHY Layer
4. EMANE Libraries

Part I begins with Chapter 2, "The Platform," which introduces the EMANE Platform and the component construction mechanisms employed by all EMANE component container applications.

Chapter 3, "EMANE Components," details the `EMANE::Component` interface, which provides the control API used by all components.

Chapter 4, "NEM Layer Anatomy," focuses on the APIs used for inter-layer communication.

[1] EMANE is released under the BSD license.

Chapter 5, "Platform Services," provides the API details for the five services which comprise the Platform Service: Log Service, Event Service, Statistic Service, Random Number Service and Timer Service.

Part II begins with chapter 6, "MAC Layer," presents the MAC Layer API and provides the steps necessary to create a MAC Layer implementation.

Chapter 7, "PHY Layer," covers the PHY Layer API and provides the steps necessary to create a PHY Layer implementation.

Chapter 8, "Shim Layer," presents the Shim Layer API and provides the steps necessary to create a Shim Layer implementation.

Chapter 9, "Events," details the inner workings of the Event Service, how to create new event types, how to create an Event Generator and how to create an Event Agent.

Chapter 10, "Transports," presents the Transport which provides the emulation/application boundary interface. The transport API is presented along with examples.

Part III contains chapter 11, "Interfacing with the Universal PHY Layer," presents the Universal PHY Layer, which is a common PHY Layer implementation for the various MAC Layers supplied as part of the standard EMANE distribution.

Part IV "EMANE Libraries," presents the EMANE libraries available for application development. Chapter 12 covers the details of `libemaneeventservice`, a C library API for interacting with the Event Service.

Chapter 13, "`libemane`," presents the library API builder classes used by EMANE applications to build their underlying containers. Developers may use this library to configure and run EMANE application containers programmatically from their own applications.

Part I

Emulation Infrastructure

Chapter 2

The Platform

The most highly visible component of the EMANE emulation framework is the Platform Server. The Platform Server is responsible for instantiating and managing one or more NEMs. EMANE deployments may contain one or more Platform Servers depending on the desired level of distributed emulation[1]. The EMANE Platform Server application, **emane**, processes platform XML files to determine which NEMs to instantiate and what configuration items to use.

2.1 XML Configuration in Brief

EMANE uses a generic XML configuration design. All EMANE components are capable of specifying any number of configuration parameters using a generic syntax. These parameter value pairs are made accessible to their respective components via a configuration API.

Component configuration is layered to allow tailoring of lower levels of configuration to simplify deployment and promote reuse. Complex EMANE components are created by layering XML definitions. Listing 2.1 shows a simple Platform Server configuration which hosts a single Development Training NEM.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <!DOCTYPE platform SYSTEM "file:///usr/share/emane/dtd/platform.dtd">
3  <platform name="Platform 1" id="1">
4    <param name="otamanagerchannelenable" value="off"/>
5    <param name="eventservicegroup" value="224.1.2.8:45703"/>
6    <param name="eventservicedevice" value="lo"/>
7
8    <nem name="NODE-001" id="1" definition="devtrainingnem.xml">
9      <param name="platformendpoint" value="vmhost:8801"/>
10     <param name="transportendpoint" value="vmnode-1:8901"/>
11     <transport definition="transvirtual.xml">
12       <param name="device" value="emane0"/>
13       <param name="address" value="10.99.0.1"/>
14       <param name="mask" value="255.255.255.0"/>
15     </transport>
16   </nem>
17 </platform>
```

Listing 2.1: Simple Platform Server configuration containing a single development training NEM.

Line 8 of Listing 2.1 determines the type of NEM that will be created by the Platform Server. In its simplest definition, an NEM is a logical stack of three types of components: PHY Layer, MAC Layer and Shim Layer. Listing 2.2 shows an NEM composed of a single `TRAINING::DevTrainingShim01` layer. The `shim` definition

[1] There are many factors which must be considered when determining the type of EMANE deployment best suited for an emulation experiment. Deployments are covered as part of EMANE User Training.

attribute on line 4 of Listing 2.2 references the Shim Layer configuration to use.

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <!DOCTYPE nem SYSTEM "file:///usr/share/emane/dtd/nem.dtd">
3 <nem name="Devel Training NEM" type="unstructured">
4   <shim definition="devtrainingshim.xml"/>
5   <transport definition="transvirtual.xml"/>
6 </nem>
```

Listing 2.2: NEM configuration referenced in the Platform Server configuration from Listing 2.1.

The Shim definition shown in Listing 2.3 is the lowest level of this example NEM configuration hierarchy. The library attribute on line 3 of Listing 2.3 associates the XML definition with the actual plugin library to load.

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <!DOCTYPE shim SYSTEM "file:///usr/share/emane/dtd/shim.dtd">
3 <shim name="Devel Training Shim" library="devtrainingshim01"/>
```

Listing 2.3: Shim Layer configuration referenced in the NEM configuration from Listing 2.2.

2.2 Constructing NEM Layers

All component container applications use a Builder design pattern [Gamma et al., 1994, p. 152] to separate the construction of the container components from the mechanism used to express the instantiation. During the build process, component plugins are loaded and managed by library factories in order to ensure only one instance is resident in memory.

The `EMANE::NEMDirector`[2] is responsible for parsing the Platform Server configuration and determining configuration item values based on the referenced NEM and NEM Layer component configurations. The `EMANE::NEMDirector` uses the `EMANE::NEMBuilder`[3] to instantiate all the required components. Figure 2.1 shows the class interface for the `EMANE::NEMBuilder`. The `EMANE::NEMBuilder` is the only location within the infrastructure where the actual NEM component plugin type is known. All specialization information is lost and components are referenced as generic entities after an NEM Layer Stack is created. The `EMANE::NEMBuilder` is exposed in the `libemane` library, discussed in Chapter 13.

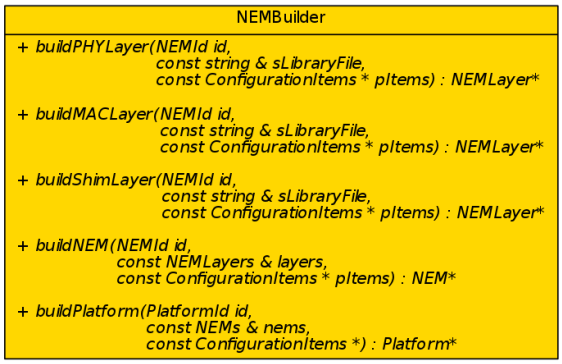

Figure 2.1: NEMBuilder Class Interface

[2]src: emane/src/emane/nemdirector.{h,cc}
[3]src: emane/include/libemane/nembuilder.h,emane/src/libemane/nembuilder.cc

Starting the Platform Server (**emane**) with the configuration from Listing 2.1 will result in the output shown in Output 1.

```
# emane platform.xml -l 4
13:43:51.801524 DEBUG SHIMI 001 TRAINING::DevTrainingShim01::initialize line 23
13:43:51.801695 DEBUG SHIMI 001 TRAINING::DevTrainingShim01::configure line 30
13:43:51.801748 DEBUG NEMLayerStack::initialize
13:43:51.801779 DEBUG ConcretePlatform::initialize
13:43:51.801782 DEBUG ConcretePlatform::configure
13:43:51.801802 DEBUG ConcretePlatform::start debugport = 47000
13:43:51.801809 DEBUG ConcretePlatform::start debugportenable = 0
13:43:51.801812 DEBUG ConcretePlatform::start eventservicedevice = lo
13:43:51.801921 DEBUG ConcretePlatform::start eventservicegroupaddr = 224.1.2.8
13:43:51.801927 DEBUG ConcretePlatform::start otamanagerchannelenable = 0
13:43:51.801929 DEBUG Not Starting Channel Device
13:43:51.803738 DEBUG EventService::processEventMessage
13:43:51.803766 DEBUG EventService::receiveEventMessage
13:43:51.832863 DEBUG SHIMI 001 TRAINING::DevTrainingShim01::start line 37
13:43:51.832899 DEBUG SHIMI 001 TRAINING::DevTrainingShim01::postStart line 43
```

Output 1: EMANE Platform Server (**emane**) running the configuration from Listing 2.1.

Chapter 3

EMANE Components

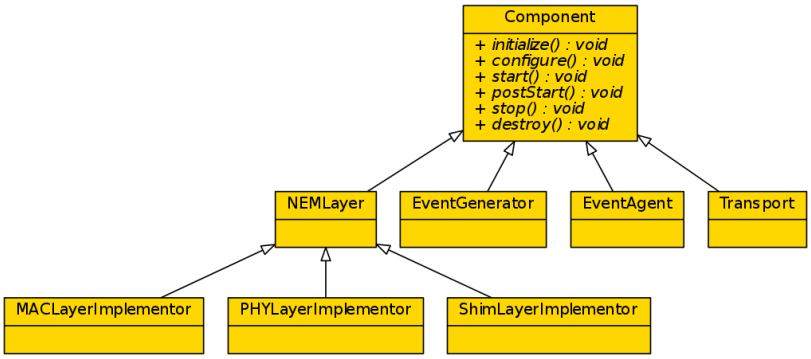

Figure 3.1: NEM Layer Plugin Component Interface Class Diagram

The `EMANE::Component`[1] interface is the most prevalent interface found in the EMANE architecture. Every component plugin type derives from `EMANE::Component`, as do many of the key framework components. After the build process has been completed, every component plugin is controlled using this interface. Listing 3.1 shows the `EMANE::Component` definition.

NEM Layer components: MAC Layer, PHY Layer and Shim Layer transition through six (6) states during their lifetime. These state transitions are driven by the dispatching of `EMANE::Component` methods. Figure 3.2 shows the NEM Layer component state diagram. The NEM Layer state machine is implemented using the State design pattern [Gamma et al., 1994, p. 305].

```
namespace EMANE
{
  class Component
  {
  public:
    virtual ~Component(){}

    virtual void initialize()
      throw(InitializeException) = 0;

    virtual void configure(const ConfigurationItems & items)
      throw(ConfigureException);

    virtual void start()
      throw(StartException) = 0;
```

[1] src: emane/include/emane/emanecomponent.{h,inl}

```

    virtual void postStart(){};

    virtual void stop()
     throw(StopException) = 0;

    virtual void destroy()
      throw() = 0;

  protected:
    Component(){}

    ConfigurationRequirements configRequirements_;
  };
}
```

Listing 3.1: `EMANE::Component` Interface

3.1 `EMANE::Component` Interface

3.1.1 `void EMANE::Component::initialize() throw(EMANE::InitializeException)`

The `initialize` method is the first method called after an EMANE component is instantiated. It provides the component developer with an opportunity to perform any initial tasks which may be required prior to component configuration. After a successful call to `initialize`, the component transitions to the `EMANE::NEMLayerStateInitialized` state. An `EMANE::InitializeException` is thrown resulting in application termination when an error is detected.

3.1.2 `void EMANE::Component::configure(const EMANE::ConfigurationItems & items) throw(EMANE::ConfigureException)`

The `configure` method is called to pass XML configuration parameters to the component. The configure method may be called any number of times but only while the component is in the `EMANE::NEMLayerStateInitialized` or `EMANE::NEMLayerStateConfigured` state. After a successful call to `configure`, the component transitions to the `EMANE::NEMLayerStateConfigured` state. An `EMANE::ConfigureException` is thrown resulting in application termination when an error is detected.

Parameter Description:

`const EMANE::ConfigurationItems & items`	List of EMANE::ConfigurationItem[a]

[a] src: emane/include/emane/emaneconfigurationitem.{h,inl}

3.1.3 `void EMANE::Component::start() throw(EMANE::StartException)`

The `start` method is called after the component has received all of its configuration. The start method may only be called when the component is in the `EMANE::NEMLayerStateConfigured` or `EMANE::NEMLayerStateStopped` state. After a successful call to `start`, the component transitions to the `EMANE::NEMLayerStateRunning` state. A component will only have its `processDownstreamPacket`, `processUpstreamPacket`, `processDownstreamControl`, `processUpstreamControl`, `processEvent` and `processTimedEvent` methods invoked while it is in the `EMANE::NEMLayerStateRunning` state. An `EMANE::StartException` is thrown resulting in application termination when an error is detected.

3.1.4 `void EMANE::Component::postStart()`

The `postStart` method is called on a running component after all NEM Layer components instantiated by the Platform Server have transitioned to the `EMANE::NEMLayerStateRunning` state. This method is only called once on each component after the entire platform has started. Prior to this call there is no guarantee that components residing together in the same container, for example an `EMANE::NEMLayerStack`, are all in fact in the `EMANE::NEMLayerStateRunning` state. The guarantee provided by the `postStart` method is particularly important for situations where cross-layer communication is required to synchronize emulation stack functionality.

3.1.5 `void EMANE::Component::stop() throw(EMANE::StopException)`

The `stop` method is called on a running component to stop the processing of control messages, packets and events. The stop method may only be called when the component is in the `EMANE::NEMLayerStateRunning` state. After a successful call to `stop`, the component transitions to the `EMANE::NEMLayerStateStopped` state. An `EMANE::StopException` is thrown resulting in application termination when an error is detected.

3.1.6 `void EMANE::Component::destroy() throw()`

The `destroy` method is the last method called prior to EMANE component deletion. It provides the component developer with an opportunity to perform any cleanup tasks. After a successful call to `destroy`, the component transitions to the `EMANE::NEMLayerStateDestroyed` state.

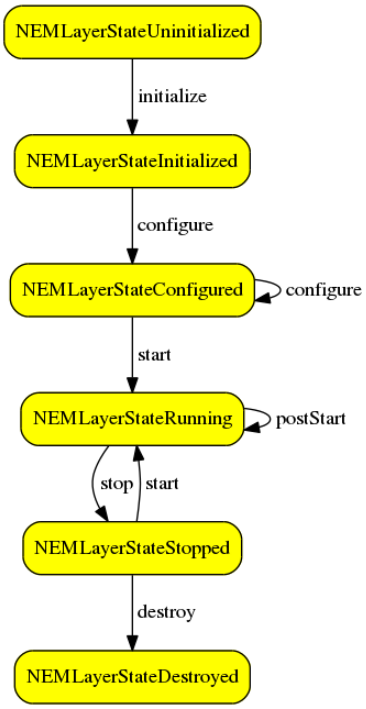

Figure 3.2: NEM Layer Component State

3.2 Component Configuration

All EMANE components receive their respective configuration generically through the `Component::configure` method. Components specify the configuration they allow and whether those items are required or optional by instantiating an `EMANE::ConfigurationDefinition`[2] array with the appropriate information. Listing 3.2 shows the definition for the `EMANE::ConfigurationDefinition` structure. Table 3.1 lists the `EMANE::ConfigurationDefinition` attribute descriptions.

```
namespace EMANE
{
  struct ConfigurationDefinition
  {
    const bool         bRequired_;
    const bool         bDefault_;
    const unsigned int uiCount_;
    const char *       pzName_;
    const char *       pzValue_;
    const char *       pzType_;
    const char *       pzDescription_;
  };
}
```

Listing 3.2: `EMANE::ConfigurationDefinition` Interface

Table 3.1: `ConfigurationDefinition` Element Description

Name	Type	Description
`bRequired_`	`bool`	true if the parameter is required
`bDefault_`	`bool`	true if the value specified should be used as the default
`pzValue_`	`char *`	Null terminated string containing the parameter name
`uiCount_`	`unsigned int`	Total number of parameter instances allowed or 0 for unlimited
`pzValue_`	`char *`	Null terminated string representation of the value
`pzType_`	`char *`	Null terminated string containing the parameter type (optional)
`pzDescription_`	`char` *	Null terminated string containing the parameter description (optional)

The `EMANE::Component` interface provides a default implementation for the `configure` method. The derived class should first invoke the `EMANE::Component:configure` method if additional functionality is required when the configure method is invoked. This default implementation loads the component's `configRequirements_`-attribute and checks to make sure that the provided configuration parameters are actually known to the component. An `EMANE::ConfigureException` is thrown if a configuration parameter is found to have not been specified in the component's `EMANE::ConfigurationDefinition`.

3.2.1 Implementing `ConfigurationDefinition`

1. Declare a `ConfigurationDefinition` array containing the desired configuration parameters, descriptions and default values. Each parameter entry must be marked as either required or optional using the `bRequired_` element.

[2] src: emane/include/emane/emaneconfigurationrequirement.{h,inl}

```
const EMANE::ConfigurationDefinition defs[] =
   {
     // req, default, count, name,            value, type, description */
     {false, true,    1,     "traceenable",   "off",  0,    "turn on packet trace"},
     {false, true,    1,     "maxstore",      "10",   0,    "max trace id amount"},
     {false, false,   0,     "ignorenode",     0,     0,    "do not trace dest node"},
     {true,  false,   1,     "floatvalue",     0,     0,    "unused value"},
     {0,0,0,0,0,0,0},
   };
```

2. Load the `ConfigurationDefinition` array into the `configRequirements_` attribute using the `loadConfigurationDefinition` function in the component constructor.

```
TRAINING::DevTrainingShim02::DevTrainingShim02(EMANE::NEMId id, EMANE::PlatformServiceProvider *
    pPlatformService) :
  ShimLayerImplementor(id, pPlatformService),
  bTraceMode_(false),
  u16MaxStore_(0)
{
  pPlatformService_->log(EMANE::DEBUG_LEVEL,
                         "SHIMI %03hu %s::%s line %d",
                         id_,
                         MODULE,
                         __func__,
                         __LINE__);

  configRequirements_ = EMANE::loadConfigurationRequirements(defs);
}
```

3. Use the `EMANE::Component::configure` implementation to parse the parameters in the component `configure` method. This step is only necessary if you are specializing the `configure` method.

```
void TRAINING::DevTrainingShim02::configure(const EMANE::ConfigurationItems & items)
  throw(EMANE::ConfigureException)
{
  pPlatformService_->log(EMANE::DEBUG_LEVEL,"SHIMI %03hu %s::%s line %d",id_, MODULE, __func__,
      __LINE__);

  Component::configure(items);
}
```

4. In the component `start` method iterate through the `configRequirements_` attribute to access the configuration parameters. Remember to throw an `EMANE::StartException` if parameters are missing or found to contain erroneous values.

```
void TRAINING::DevTrainingShim02::start()
  throw(EMANE::StartException)
{
  EMANE::ConfigurationRequirements::const_iterator iter = configRequirements_.begin();

  try
    {
      for(;iter != configRequirements_.end(); ++iter)
        {
          if(iter->second.bPresent_)
            {
              if(iter->first == "traceenable")
                {
                  bTraceMode_ =
                   EMANEUtils::ParameterConvert(iter->second.item_.getValue()).toBool();
                }
          else if(iter->first == "ignorenode")
                {
                  ACE_UINT16 u16Node =
```

```
                    EMANEUtils::ParameterConvert(iter->second.item_.getValue()).toUINT16();

                  ignoreList_.push_back(u16Node);
                }
                ....
            }
          else if(iter->second.bRequired_)
            {
              std::stringstream sstream;
              ...

              throw EMANE::StartException(sstream.str());
            }
        }
        ...
```

5. Make sure the component XML contains the appropriate configuration parameters

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE shim SYSTEM "file:///usr/share/emane/dtd/shim.dtd">
<shim name="Devel Training Shim" library="devtrainingshim02">
  <param name="floatvalue" value="2.7182818284"/>
</shim>
```

Listing 3.3: Configuration for `DevTrainingShim02` with required configuration item.

```
# emane platform.xml -l 4
13:53:57.717664 DEBUG SHIMI 001 TRAINING::DevTrainingShim02::initialize line 42
13:53:57.717809 DEBUG SHIMI 001 TRAINING::DevTrainingShim02::configure line 49
13:53:57.717856 DEBUG NEMLayerStack::initialize
13:53:57.717885 DEBUG ConcretePlatform::initialize
13:53:57.717888 DEBUG ConcretePlatform::configure
13:53:57.717907 DEBUG ConcretePlatform::start debugport = 47000
13:53:57.717915 DEBUG ConcretePlatform::start debugportenable = 0
13:53:57.717918 DEBUG ConcretePlatform::start eventservicedevice = lo
13:53:57.718028 DEBUG ConcretePlatform::start eventservicegroupaddr = 224.1.2.8
13:53:57.718033 DEBUG ConcretePlatform::start otamanagerchannelenable = 0
13:53:57.718036 DEBUG Not Starting Channel Device
13:53:57.719842 DEBUG EventService::processEventMessage
13:53:57.719864 DEBUG EventService::receiveEventMessage
13:53:57.782335 DEBUG SHIMI 001 TRAINING::DevTrainingShim02::start line 57
13:53:57.782420 DEBUG MACI 001 TRAINING::DevTrainingShim02::start floatvalue = 2.718282
13:53:57.782447 DEBUG MACI 001 TRAINING::DevTrainingShim02::start maxstore = 10
13:53:57.782458 DEBUG MACI 001 TRAINING::DevTrainingShim02::start traceenable = off
13:53:57.782482 DEBUG SHIMI 001 TRAINING::DevTrainingShim02::postStart line 155
```

Output 2: EMANE Platform Server (**emane**) running the configuration from Listing 3.3.

Chapter 4

NEM Layer Anatomy

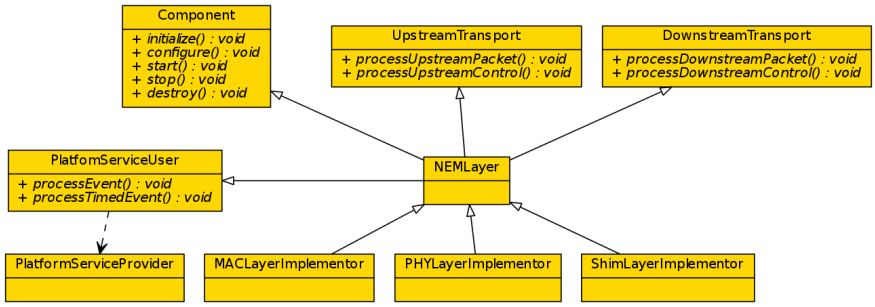

Figure 4.1: `NEMLayer` Class Diagram

EMANE supports two types of Network Emulation Modules, *structured* and *unstructured*. A structured NEM is a component stack composed of a PHY Layer implementation, a MAC Layer implementation and zero or more Shim Layer implementations. Listing 4.1 shows an example structured NEM definition. An unstructured NEM is a component stack composed of zero or one PHY Layer implementation, zero or one MAC Layer implementation and zero or more Shim Layer implementations. Listing 4.2 shows an example unstructured NEM definition. The main difference between the two NEM Layer types, besides the number of layers, is that internal NEM definition verification checks are relaxed when unstructured NEMs are built.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE nem SYSTEM "file:///usr/share/emane/dtd/nem.dtd">
<nem name="Devel Training NEM" type="structured">
  <mac definition="devtrainingmac.xml"/>
  <phy definition="devtrainingphy.xml"/>
  <transport definition="transvirtual.xml"/>
</nem>
```

Listing 4.1: Structured NEM definition example

```
  <?xml version="1.0" encoding="UTF-8"?>
    <!DOCTYPE nem SYSTEM "file:///usr/share/emane/dtd/nem.dtd">
    <nem name="Devel Training NEM" type="unstructured">
    <shim definition="devtrainingshim.xml"/>
    <transport definition="transvirtual.xml"/>
    </nem>
```

Listing 4.2: Unstructured NEM definition example

Once instantiated, each NEM Layer physically resides within an NEM's `EMANE::NEMLayerStack`[1] member. Prior to insertion into an NEM's `EMANE::NEMLayerStack`, all MAC, PHY and Shim implementations are wrapped within an `EMANE::MACLayer`[2], `EMANE::PHYLayer`[3] and `EMANE::ShimLayer`[4] object, respectively. These classes are sometimes referred to as the *concrete* MAC, PHY and Shim. This wrapping occurs as part of a Bridge design pattern implementation in order to allow the possibility for divergence between the infrastructure's layer interface and that presented to the component developer [Gamma et al., 1994, p. 151].

4.1 Layer Communication

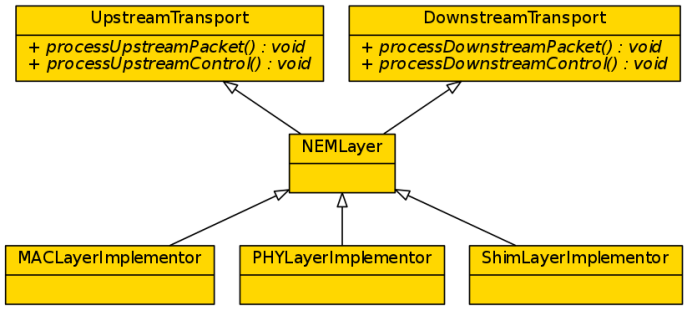

Figure 4.2: `UpstreamTransport` `DownstreamTransport` Class Diagram

NEM Layer components communicate generically using the `EMANE::UpstreamTransport`[5] and `EMANE::DownstreamTransport`[6] interfaces. The two interfaces provide similar functionality but for different communication directions. The term *upstream* is used in EMANE to describe the path from Over-The-Air (OTA) transmission up the NEM Layer stack to application space. The term *downstream* is used to describe the path from application space down the NEM Layer stack towards OTA transmission. Listing 4.3 shows the `EMANE::UpstreamTransport` interface. Listing 4.4 shows the `EMANE::DownstreamTransport` interface.

The `EMANE::NEMBuilder` connects each contiguous layer in an NEM Layer stack by providing a reference to the layer above and the layer below. NEM Layer components are unaware where they reside in the layer stack and to what layer types they are connected. A generic packet processing and control message API makes it possible to insert special purpose layers, known as Shims, in between the layers of a more traditional MAC-PHY stack without the need to modify the MAC or PHY layer.

```
namespace EMANE
{
  class UpstreamTransport
  {
  public:
    virtual ~UpstreamTransport(){}

    virtual
    void processUpstreamPacket(UpstreamPacket & pkt,
                               const ControlMessage & msg) = 0;

    virtual
    void processUpstreamControl(const ControlMessage & msg) = 0;


    virtual
```

[1]src: emane/src/libemane/nemlayerstack.{h,cc}
[2]src: emane/src/libemane/maclayer.{h,cc}
[3]src: emane/src/libemane/phylayer.{h,cc}
[4]src: emane/src/libemane/shimlayer.{h,cc}
[5]src: emane/include/emane/emaneupstreamtransport.{h,inl}
[6]src: emane/include/emane/emanedownstreamtransport.{h,inl}

```
    void setDownstreamTransport(DownstreamTransport * pDownstreamTransport);

    void sendDownstreamPacket(DownstreamPacket & pkt,
                              const ControlMessage & msg);

    void sendDownstreamControl(const ControlMessage & msg);

  protected:
    UpstreamTransport();
  };
}
```

Listing 4.3: `EMANE::UpdatreamTransport` Interface

```
namespace EMANE
{
  class DownstreamTransport
  {
  public:
    virtual
    ~DownstreamTransport(){}

    virtual
    void processDownstreamPacket(DownstreamPacket & pkt,
                                 const ControlMessage & msg) = 0;

    virtual
    void processDownstreamControl(const ControlMessage & msg) = 0;

    void sendUpstreamPacket(UpstreamPacket & pkt,
                            const ControlMessage & msg);

    void sendUpstreamControl(const ControlMessage & msg);

  protected:
    DownstreamTransport();
  };
}
```

Listing 4.4: `EMANE::DownstreamTransport` Interface

4.1.1 `EMANE::UpstreamTransport` Interface

4.1.1.1 `void EMANE::UpstreamTransport::processUpstreamPacket(UpstreamPacket & pkt, const ControlMessage & msg)`

The `processUpstreamPacket` method is called on an NEM Layer to process an upstream packet sent from the layer immediately below the recipient in the NEM Layer stack. Each component should treat the contents of a received packet as opaque. Cross-layer communication can be achieved using an optional control message that may accompany the packet.

Parameter Description:

`UpstreamPacket &`	`pkt`	Upstream packet to process
`const ControlMessage &`	`msg`	Control message to process or `EMPTY_CONTROL_MESSAGE`

4.1.1.2 `void EMANE::UpstreamTransport::processUpstreamControl(const ControlMessage & msg)`

The `processUpstreamControl` method is called on an NEM Layer to process an upstream control message sent from the layer immediately below the recipient in the NEM Layer stack.

Parameter Description:

`const ControlMessage &`	`msg`	Control message to process

4.1.1.3 `void EMANE::UpstreamTransport::sendDownstreamPacket(DownstreamPacket & pkt, const ControlMessage & msg)`

The `sendDownstreamPacket` method is called by an NEM Layer to send a downstream packet to the layer immediately below the sender in the NEM Layer stack. Cross-layer communication can be achieved using an optional control message that may accompany the packet.

Parameter Description:

`DownstreamPacket &`	`pkt`	Downstream packet to send
`const ControlMessage &`	`msg`	Control message to send or `EMPTY_CONTROL_MESSAGE`

4.1.1.4 `void EMANE::UpstreamTransport:::sendDownstreamControl(const ControlMessage & msg)`

The `sendDownstreamControl` method is called by an NEM Layer to send a downstream control message to the layer immediately below the sender in the NEM Layer stack.

Parameter Description:

`const ControlMessage &`	`msg`	Control message to send

4.1.2 `EMANE::DownstreamTransport` Interface

4.1.2.1 `void EMANE::DownstreamTransport::processDownstreamPacket(DownstreamPacket & pkt, const ControlMessage & msg)`

The `processDownstreamPacket` method is called on an NEM Layer to process a downstream packet sent from the layer immediately above the recipient in the NEM Layer stack. Each component should treat the contents of a received packet as opaque. Cross-layer communication can be achieved using an optional control message that may accompany the packet.

Parameter Description:

`DownstreamPacket &`	`pkt`	Downstream packet to process
`const ControlMessag &`	`msg`	Control message to process or `EMPTY_CONTROL_MESSAGE`

4.1.2.2 `void EMANE::DownstreamTransport::processDownstreamControl(const ControlMessage & msg)`

The `processDownstreamControl` method is called on an NEM Layer to process a downstream control message sent from the layer immediately above the recipient in the NEM Layer stack.

Parameter Description:

`const ControlMessage &`	`msg`	Control message to process

4.1.2.3 `void EMANE::DownstreamTransport::sendUpstreamPacket(UpstreamPacket & pkt, const ControlMessage & msg)`

The `sendUpstreamPacket` method is called by an NEM Layer to send an upstream packet to the layer immediately above the sender in the NEM Layer stack. Cross-layer communication can be achieved using an optional control message that may accompany the packet.

Parameter Description:

`UpstreamPacket &`	`pkt`	Upstream packet to send
`const ControlMessage &`	`msg`	Control message to send or `EMPTY_CONTROL_MESSAGE`

4.1.2.4 `void EMANE::DownstreamTransport:::sendUpstreamControl(const ControlMessage & msg)`

The `sendUpstreamControl` method is called by an NEM Layer to send an upstream control message to the layer immediately above the sender in the NEM Layer stack.

Parameter Description:

`const ControlMessage &`	`msg`	Control message to send or `EMPTY_CONTROL_MESSAGE`

4.1.3 `EMANE::UpstreamPacket Interface`

4.1.3.1 `size_t EMANE::UpstreamPacket::strip(size_t size) const`

The `strip` method removes `size` bytes from the beginning of the packet data and returns the number of bytes removed.

Parameter Description:

`size_t`	`size`	Number of bytes to strip

4.1.3.2 `const void * EMANE::UpstreamPacket::get() const`

The `get` method returns a constant pointer to the packet data or `NULL` if there is no data.

4.1.3.3 `size_t EMANE::UpstreamPacket::length() const`

The `length` method returns the packet data length in bytes.

4.1.3.4 `const PacketInfo & EMANE::UpstreamPacket::getPacketInfo() const`

The `getPacketinfo` method returns a constant `EMANE::PacketInfo`[7] reference containing the NEM Source Id, NEM Destination Id and Differentiated Service Code Point associated with the packet.

4.1.4 `EMANE::DownstreamPacket Interface`

The `EMANE::UpstreamTransport` and `EMANE::DownstreamTransport` interfaces use `EMANE::UpstreamPacket`[8] and `EMANE::DownstreamPacket`[9] instances to transmit packet data between layers, respectively.

[7] src: emane/include/emane/emanepacketinfo.h
[8] src: emane/include/emane/emaneupstreampacket.{h,inl}
[9] src: emane/include/emane/emanedownstreampacket.{h,inl}

4.1.4.1 `void EMANE::DownstreamPacket::prepend(const void * buf, size_t size)`

The `prepend` method is used to prepend `size` bytes from `buf` to the beginning of the packet. Prepending layer specific information to a packet allows a layer to communicate with its corresponding instances residing within other instantiations of the same NEM.

Parameter Description:

`const void *`	`buf`	Buffer containing data to prepend
`size_t`	`size`	Number of bytes to prepend

4.1.4.2 `void EMANE::DownstreamPacket::combine()`

The `combine` method combines any internal buffers used to hold packet data. This method call **must** proceed any calls to `get` after a `prepend` has occurred. As an optimization, prepended packet data is held in a pseudo scatter-gather I/O structure. The packet data must be combined in order to access it as a single buffer.

4.1.4.3 `const void * EMANE::DownstreamPacket::get() const`

The `get` method returns a constant pointer to the packet data. If there is no data, or if the internal packet data structure has not been combined prior to the call, a `NULL` value will be returned. One way to determine if a packet has not been combined is to examine the packet length. If the `length` method returns a length greater than zero and the `get` method returns `NULL`, the packet needs to be combined.

4.1.4.4 `size_t EMANE::DownstreamPacket::length() const`

The `length` method returns the packet data in bytes.

4.1.4.5 `const PacketInfo & EMANE::DownstreamPacket::getPacketInfo() const`

The `getPacketinfo` method returns a constant `EMANE::PacketInfo` reference containing the NEM Source Id, NEM Destination Id and Differentiated Service Code Point associated with the packet.

4.1.5 `EMANE::ControlMessage` Interface

An `EMANE::ControlMessage` object is a wrapper around a cross-layer message sent between contiguous layers of an NEM Layer stack. The message is delivered generically by the EMANE framework and is layer specific in terms of which layers can process which type of control message, if any. Component developers should consult the documentation accompanying an NEM Layer component to determine the supported control messages.

4.1.5.1 `EMANE::ControlMessage::ControlMessage(EMANE::INT32 iMajorIdentifier, EMANE::INT32 iMinorIdentifier,const void * buf,size_t len)`

Construct a control message from an opaque buffer. The `iMajorIdentifier` and `iMinorIdentifier` are used by the receiver to determine control message type.

Parameter Description:

`EMANE::INT32`	`iMajorIdentifier`	Control message major identifier
`EMANE::INT32`	`iMinorIdentifier`	Control message minor identifier
`const void *`	`buf`	Control message data
`size_t`	`len`	Length of the control message data in bytes

4.1.5.2 `const void * EMANE::ControlMessage::get() const`

The `get` method returns a constant pointer to the internal control data.

4.1.5.3 `size_t EMANE::ControlMessage::length()`

The `length` method returns the length in bytes of the internal control data.

4.1.5.4 `EMANE::INT32 EMANE::ControlMessage::getMajorIdentifier() const`

The `getMajorId` method returns the control message major identifier.

4.1.5.5 `EMANE::INT32 EMANE::ControlMessage::getMinorIdentifier() const`

The `getMinorId` method returns the control message minor identifier.

4.2 Event Processing

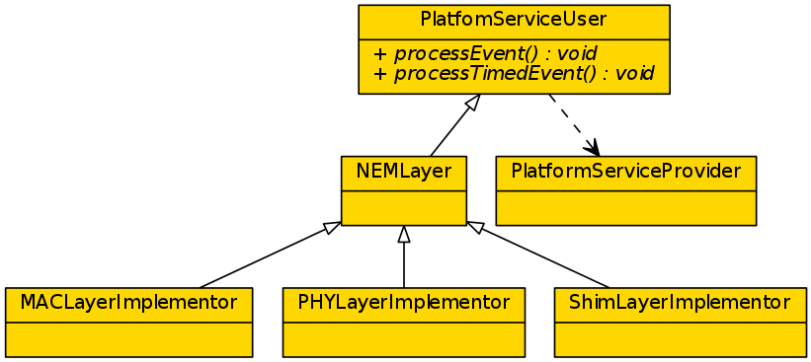

Figure 4.3: `PlatformServiceUser` Class Diagram

All components derive from `EMANE::PlatfomServiceUser`[10] which is both an `EMANE::TimerServiceHandler`[11] and an `EMANE::EventServiceHandler`[12]. The `EMANE::EventServiceHandler` and `EMANE::TimerServiceHandler` interfaces expose two virtual methods that are invoked when EMANE events and timer events are received, respectively. Listing 4.5 shows the `TimerServiceHandler` interface and Listing 4.6 shows the `EventServiceHandler` interface.

[10] src: emane/include/emane/emaneplatformserviceuser.h
[11] src: emane/include/emane/emanetimerservicehandler.h
[12] src: emane/include/emane/emaneeventservicehandler.h

```
namespace EMANE
{
  class TimerServiceHandler
  {
  public:
    virtual ~TimerServiceHandler(){}

    virtual void processTimedEvent(ACE_UINT32 taskType,
                                   long eventId,
                                   const ACE_Time_Value & tv,
                                   const void *arg) = 0;

  protected:
    TimerServiceHandler(){}
  };
}
```

Listing 4.5: `EMANE::TimerServiceHandler` Interface

```
namespace EMANE
{
  class EventServiceHandler
  {
  public:
    virtual ~EventServiceHandler(){}

    virtual void processEvent(const EventId & eventId,
                              const EventObjectState & state) = 0;

  protected:
    EventServiceHandler(){}
  };
```

Listing 4.6: `EMANE::EventServiceHandler` Interface

4.2.1 `EMANE::TimerServiceHandler::processTimedEvent(ACE_UINT32 taskType,long eventId, const ACE_Time_Value & tv,const void *arg)`

The `processTimedEvent` method is called on a component to process a scheduled timed event. The `taskType` argument is supplied by the component when the task is scheduled and can be used to identify different categories of timers.

Parameter Description:

`ACE_UINT32`	`taskType`	Task id type
`const void *`	`arg`	Opaque task data
`const ACE_Time_Value &`	`tvTimeOut`	Task schedule time (absolute time)
`const ACE_Time_Value &`	`tvInterval`	Task reschedule time

4.2.2 `EMANE::EventServiceHandler::processEvent(const EventId & eventId, const EventObjectState & state)`

The `processEvent` method is called on a component to process a received event. Events are addressed in a 3-tuple: Platfrom Id, NEM Id and Component Type. A zero element is interpreted as *all* components in that category. A component is responsible for determining which events are of interest. Event data is delivered to a component generically using an `EventObjectState` opaque wrapper.

Parameter Description:

`const EventId &`	`eventId`	Event id
`const EventObjectState &`	`state`	Opaque event data wrapper

4.3 Message Processing

The `EMANE::NEMQueuedLayer`[13] class provides a message queuing mechanism to decouple inter-layer communication and to ensure that all calls to the `EMANE::MACLayerImplementor`[14], `EMANE::PHYLayerImplementor`[15] and `EMANE::ShimLayerImplementor`[16] interfaces happen serially within their own layer on a dedicated message processing thread.

The `EMANE::NEMQueuedLayer` class enqueues messages received through the `EMANE::UpstreamTransport`, `EMANE::DownstreamTransport`, `EMANE::TimerServiceHandler` and `EMANE::EventServiceHandler` interfaces. Listing 4.7 shows the `EMANE::NEMQueuedLayer::processDownstreamPacket` method implementation.

```
void EMANE::NEMQueuedLayer::processDownstreamPacket(DownstreamPacket & pkt,
                                                    const ControlMessage & ctrl)
{
  ACE_Guard<ACE_Thread_Mutex> m(mutex_);

  queue_.push(EMANEUtils::makeFunctor(*this,
                                      &NEMQueuedLayer::handleProcessDownstreamPacket,
                                      pkt,
                                      ctrl));

  cond_.signal();
}
```

Listing 4.7: `EMANE::NEMQueuedLayer::processDownstreamPacket` method implementation

The `EMANE::NEMQueueLayer` class uses a dedicated message processing thread to process the enqueued messages in the order they were received. Listing 4.8 shows the `EMANE::NEMQueuedLayer::processWorkQueue` method implementation.

```
ACE_THR_FUNC_RETURN EMANE::NEMQueuedLayer::processWorkQueue()
{
  while(1)
    {
      mutex_.acquire();

      while(queue_.empty() && !bCancel_)
        {
          cond_.wait();
        }

      if(bCancel_)
        {
          mutex_.release();
          break;
        }

      // retrieve the next funtion to execute
      EMANEUtils::Functor<void> * f = queue_.front();

      // remove the functor from the top of the queue
      queue_.pop();

      // release the queue syncronization object
      mutex_.release();


      try
        {
          // execute the funtor
          (*f)();
        }
```

[13]src: emane/src/libemane/nemqueuedlayer.{h,cc}
[14]src: emane/include/emane/emanemaclayerimpl.{h,inl}
[15]src: emane/include/emane/emanephylayerimpl.{h,inl}
[16]src: emane/include/emane/emaneshimlayerimpl.{h,inl}

```
      catch(...)
        {
          LogServiceSingleton::instance()->log(ERROR_LEVEL,
                                               "NEMQueuedLayer::processWorkQueue %hu Excepetion caught",
                                               id_);
        }

      // clean up
      delete f;
    }

  return 0;
}
```

Listing 4.8: `EMANE::NEMQueuedLayer::processWorkQueue` method implementation

Chapter 5

Platform Services

The EMANE Platform Service provides a single interface for a set of services that are available to every NEM Layer, Event Generator, Event Agent and Transport. Figure 5.1 shows the interface hierarchy that comprises the Platform Service.

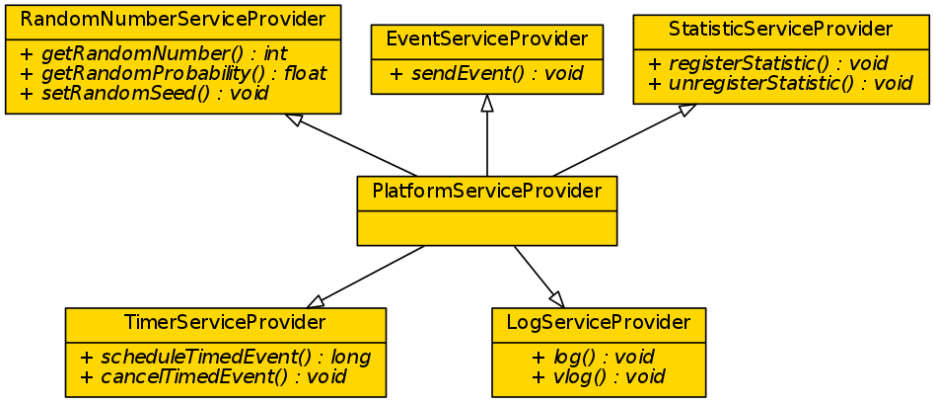

Figure 5.1: Platform Service Class Diagram

5.1 `EMANE::LogServiceProvider`

The `EMANE::LogServiceProvider`[1] interface provides a generic capability to log messages. Depending on how the container application was invoked, log messages can either be written to file, syslog, or stderr.

There are five different types of log levels. Table 5.1 lists the available log levels with descriptions. Log levels are inclusive and listed in order of inclusion. Levels at the bottom of the table will include output from any of the above levels.

[1] src: emane/include/emane/emanelogserviceprovider.h

Table 5.1: `EMANE::LogLevel` Description

Name	Description
`NOLOG_LEVEL`	No Logging
`ABORT_LEVEL`	Unrecoverable application failure
`ERROR_LEVEL`	Recoverable application failure
`STATS_LEVEL`	Statistic/Performance information
`DEBUG_LEVEL`	General application verbose debugging

5.1.1 `void EMANE::LogServiceProvider::log(LogLevel level, const char * fmt,...)`

The `log` method outputs a `printf` style log message.

Parameter Description:

`LogLevel`	`level`	Log level of message
`const char *`	`fmt`	Format string (see `printf` usage)
`...`		Variable data (see `printf` usage)

5.1.2 `void EMANE::LogServiceProvider::vlog(LogLevel level, const char * fmt,va_list ap)`

The `vlog` method outputs a `vprintf` style log message.

Parameter Description:

`LogLevel`	`level`	Log level of message
`const char *`	`fmt`	Format string (see `vprintf` usage)
`va_list`	`ap`	Variable argument list (see `vprintf` usage)

5.1.3 Sample Usage

Listing 5.1 shows sample log usage.

```
void
RFPIPEMAC::MacLayer::processUpstreamControl(const EMANE::ControlMessage & msg)
{
  pPlatformService_->log(EMANE::DEBUG_LEVEL,"MACI %03hu %s::%s",id_,pzLayerName,__func__);

  sendUpstreamControl(msg);
}
```

Listing 5.1: Logging via the `EMANE::PlatformServiceProvider` Interface

5.2 `EMANE::EventServiceProvider`

The `EMANE::EventServiceProvider`[2] interface provides a generic capability to send an event.

[2]src: emane/include/emane/emaneeventserviceprovider.h

5.2.1 `void EMANE::EventServiceProvider::sendEvent(PlatformId platformId, NEMId nemId, ComponentType type, const Event & event)`

The `sendEvent` method is used to send an event using an `EMANE::Event` reference. Event endpoints are addressable using a 3-tuple: `PlatformId`, `NEMId` and `ComponentType`.

Parameter Description:

`PlatformId`	`platformId`	Platform id of destination platform or 0 for all platforms
`NEMId`	`nemId`	NEM id of destination NEM or 0 for all NEMs in a platform
`ComponentType`	`type`	Component target type of destination
`Event`	`event`	Event object reference

5.2.2 `void EMANE::EventServiceProvider::sendEvent(PlatformId platformId, NEMId nemId, ComponentType type, EventId eventId,const EventObjectState & state)`

The `sendEvent` method is used to send an event using an `EMANE::EventObjectState` reference. Event endpoints are addressable using a 3-tuple: `PlatformId`, `NEMId` and `ComponentType`.

Parameter Description:

`PlatformId`	`platformId`	Platform id of destination platform or 0 for all platforms
`NEMId`	`nemId`	NEM id of destination NEM or 0 for all NEMs in a platform
`ComponentType`	`type`	Component target type of destination
`EventId`	`eventId`	Event Id
`const EventObjectState &`	`state`	Event object state reference

5.2.3 Sample Usage

Listing 5.3 shows send event usage.

```
    if(bPublishPathlossEvents_)
      {
        for(int i = 0; i < u16TotalNodes_ ; ++i)
          {
            . . .

            PathlossEvent pathlossEvent(entries,u16TotalNodes_ - 1);

            if(i < u16MaxNEMPresent_)
              {
                pPlatformService_->sendEvent(0,                          // all platform(s)
                                             i+1,                        // nem id
                                             EMANE::COMPONENT_PHYILAYER,
                                             pathlossEvent);
              }
          }
      }
```

Listing 5.2: Sending an event via the `EMANE::PlatformServiceProvider` Interface

5.3 EMANE::StatisticServiceProvider

The EMANE::StatisticServiceProvider[3] provides a generic way for a component to create statistics for use in debugging and performance evaluation.

5.3.1 void EMANE::StatisticServiceProvider::registerStatistic(std::string name, EMANE::Statistic *value)

The registerStatistic method registers a statistic with the Platform Statistic Manager. Statistics must be registered in order to retrieve or clear their value via the Platform Debug Port.

Parameter Description:

std::string	name	Statistic name
EMANE::Statistic *	value	Pointer to statistic to register

5.3.2 void EMANE::StatisticServiceProvider::unregisterStatistic(std::string name)

The unregisterStatistic method unregisters a statistic with the Platform Statistic Manager.

Parameter Description:

std::string	name	Statistic name

5.3.3 Sample Usage

Listing 5.3 shows sample statistic usage.

```
#include "emane/emanestatisticunsignedinteger32.h"

namespace RFPIPEMAC {
 class DownstreamQueue
 {
  ...

 private:

  EMANE::StatisticUnsignedInteger32 numPacketsEnqued_;
  EMANE::StatisticUnsignedInteger32 numBytesEnqued_;
  EMANE::StatisticUnsignedInteger32 numPacketsDequed_;
  EMANE::StatisticUnsignedInteger32 numPacketsOverFlow_;
  EMANE::StatisticUnsignedInteger32 numHighWaterMark_;

  ...
 };
}

RFPIPEMAC::DownstreamQueue::DownstreamQueue(EMANE::PlatformServiceProvider * pPlatformService)
  : pPlatformService_(pPlatformService),
    bCancel_(false),
    maxQueueSize_(QUEUE_SIZE_DEFAULT),
    cond_(mutex_)
{
```

[3] src: emane/include/emane/emanestatisticserviceprovider.h

```
  pPlatformService_->registerStatistic("numPacketsEnqued",&numPacketsEnqued_);
  pPlatformService_->registerStatistic("numBytesEnqued",&numBytesEnqued_);
  pPlatformService_->registerStatistic("numPacketsDequed",&numPacketsDequed_);
  pPlatformService_->registerStatistic("numPacketsOverflow",&numPacketsOverFlow_);
  pPlatformService_->registerStatistic("numHighWaterMark",&numHighWaterMark_);
}


RFPIPEMAC::DownstreamQueue::~DownstreamQueue()
{
  // unregister statistics container
  pPlatformService_->unregisterStatistic("numPacketsEnqued");
  pPlatformService_->unregisterStatistic("numBytesEnqued");
  pPlatformService_->unregisterStatistic("numPacketsDequed");
  pPlatformService_->unregisterStatistic("numPacketsOverflow");
  pPlatformService_->unregisterStatistic("numHighWaterMark");
}


RFPIPEMAC::DownstreamQueueEntry
RFPIPEMAC::DownstreamQueue::dequeue()
{
   ...

   // bump packets dequeued
   ++numPacketsDequed_;

   ...
}
```

Listing 5.3: Using statistics via the `EMANE::PlatformServiceProvider` Interface

5.4 EMANE::RandomNumberServiceProvider

The `EMANE::RandomNumberServiceProvider`[4] provides a way for a component to get random numbers in a thread-safe manner. Each NEM Layer has a dedicated function queue thread that processes messages in the order in which they are received. The Random Number Service stores a per component instance seed value to provide thread safety. It is the plugin's responsibility to synchronize access to the Random Number Service across any threads the component directly creates.

5.4.1 `int EMANE::RandomNumberServiceProvider::getRandomNumber(int iMinLimit, int iMaxLimit)`

The `getRandomNumber` method returns a random number between `iMinLimit` and `iMaxLimit`, inclusive.

Parameter Description:

`int`	`iMinLimit`	Minimum value
`int`	`iMaxLimit`	Maximum value

5.4.2 `int EMANE::RandomNumberServiceProvider::getRandomNumber()`

The `getRandomNumber` method returns a random number between `0` and `RAND_MAX`, inclusive.

[4] src: emane/include/emane/emanerandomnumberserviceprovider.h

5.4.3 `float EMANE::RandomNumberServiceProvider::getRandomProbability()`

The `getRandomProbability` method returns a random probability with uniform distribution between `0` and `1`, inclusive.

5.4.4 Sample Usage

Listing 5.4 shows sample random number service usage.

```
ACE_Time_Value
RFPIPEMAC::MacLayer::randomize(const ACE_Time_Value & rInterval)
{
  // the result is +/- the interval
  ACE_Time_Value tvResult = ACE_Time_Value::zero;

  // if interval is greater then zero
  if(rInterval > ACE_Time_Value::zero)
   {
     // scale up and roll the dice
     const ACE_UINT64 u64usec =
       pPlatformService_->getRandomNumber() % (2 * (rInterval.sec() *
                                                 1000000 + rInterval.usec()));

     // set the time
     tvResult.set(static_cast<double>(u64usec / 1000000.0));

     // scale back
     tvResult -= rInterval;
   }

  // return result
  return tvResult;
}
```

Listing 5.4: Getting a random number via the `EMANE::PlatformServiceProvider` Interface

5.5 `EMANE::TimerServiceProvider`

The `EMANE::TimerServiceProvider`[5] provides a means for a component to schedule timer events that are processed by the respective component's dedicated function queue thread.

5.5.1 `long EMANE::TimerServiceProvider::scheduleTimedEvent(ACE_UINT32 taskType, const void *arg, const ACE_Time_Value & tvTimeOut, const ACE_Time_Value & tvInterval)`

The `scheduleTimedEvent` method is used to schedule a timed event. A timer id is returned which can be used to cancel the pending timed event prior to expiration.

Parameter Description:

`ACE_UINT32`	`taskType`	Task id type
`const void *`	`arg`	Opaque task data
`const ACE_Time_Value &`	`tvTimeOut`	Task schedule time (absolute time)
`const ACE_Time_Value &`	`tvInterval`	Task reschedule time

[5] src: emane/include/emane/emanetimerserviceprovider.h

5.5.2 void EMANE::TimerServiceProvider::cancelTimedEvent(long eventId)

The `cancelTimedEvent` method is used to cancel a pending timed event.

Parameter Description:

`long`	`eventId`	Timed event id to cancel

5.5.3 Sample Usage

Listing 5.5 shows sample timer service usage.

```
void
RFPIPEMAC::MacLayer::processUpstreamPacket(const EMANE::CommonMACHeader & hdr,
                                           EMANE::UpstreamPacket & pkt,
                                           const EMANE::ControlMessage & msg)
{
        ...

        // if any delay is needed
        if(tvTimeOut > ACE_Time_Value::zero)
         {
           // create a copy of the pkt, the callback will take ownership of the pkt and itself
           const EMANE::UpstreamPacket *p = new EMANE::UpstreamPacket (pkt);

           // schedule the event id, pkt data, timeout (absolute time), one shot
           const long eventId =
             pPlatformService_->scheduleTimedEvent(0, p, tvTimeOut + tvCurrentTime);

            // event was scheduled
           if(eventId >= 0)
            {
              pPlatformService_->log(EMANE::DEBUG_LEVEL,
                  "MACI %03hu %s::%s: origin %hu, dst %hu, timed event %ld",
                   id_, pzLayerName, __func__,
                   pinfo.source_,
                   pinfo.destination_,
                   eventId);

        ...
```

Listing 5.5: Scheduling a timed event using the `EMANE::PlatformServiceProvider` Interface

Part II

Component APIs

Chapter 6

MAC Layer API

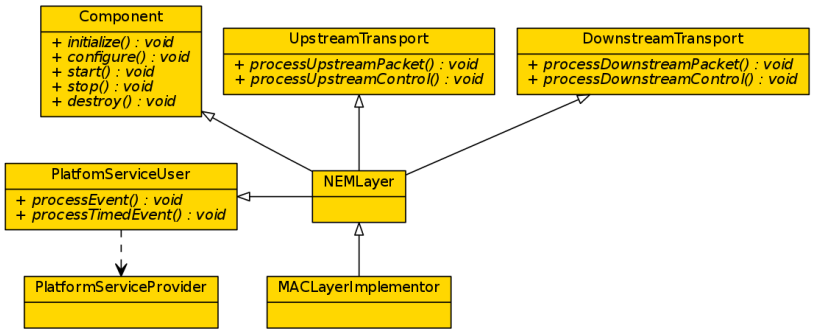

Figure 6.1: `MACLayerImplementor` Class Diagram

A MAC Layer component implementation is a realization of the `EMANE::MACLayerImplementor` class.

6.1 MAC Layer Packet Processing

The `EMANE::MACLayerImplementor` further specializes the `processUpstreamPacket` and the `sendDownstreamPacket` methods by introducing a mandatory `EMANE::CommonMACHeader` parameter. The presence of a common MAC header protects against potential error conditions that could be introduced by misconfigured NEM stacks using the Universal PHY Layer.

6.1.1 `EMANE::MACLayerImplementor::processUpstreamPacket(const EMANE::CommonMACHeader & hdr, EMANE::UpstreamPacket & pkt,const EMANE::ControlMessage & msg)`

The `processUpstreamPacket` method is called on an MAC Layer to process an upstream packet sent from the layer immediately below the recipient in the NEM Layer stack. Each component should treat the contents of a received packet as opaque. Cross-layer communication can be achieved using an optional control message that may accompany the packet.

Parameter Description:

`const EMANE::CommonMACHeader &`	`hdr`	Common MAC Layer header reference[a]
`EMANE::UpstreamPacket &`	`pkt`	Upstream packet to process
`const EMANE::ControlMessage &`	`msg`	Control message to process or `EMPTY_CONTROL_MESSAGE`

[a] src: emane/include/emane/emanecommonmacheader.{h,inl}

6.1.2 `void EMANE::MACLayerImplementor::sendDownstreamPacket(const CommonMACHeader & hdr, DownstreamPacket & pkt, const ControlMessage & msg)`

The `sendDownstreamPacket` method is called by a MAC Layer to send a downstream packet to the layer immediately below the sender in the NEM Layer stack. Cross-layer communication can be achieved using an optional control message that may accompany the packet.

Parameter Description:

`const EMANE::CommonMACHeader &`	`hdr`	Common MAC Layer header reference
`DownstreamPacket &`	`pkt`	Downstream packet to send
`const ControlMessage &`	`msg`	Control message to send or `EMPTY_CONTROL_MESSAGE`

6.2 MAC Layer Common Header

The `EMANE::CommonMACHeader` contains the registration Id of the sending MAC Layer. This allows a MAC Layer to verify that the `EMANE::UpstreamPacket` it received via the `processUpstreamPacket` method was generated by a different instance of the same MAC Layer.

```
namespace EMANE
{
  class CommonMACHeader
  {
  public:
    CommonMACHeader(UpstreamPacket & pkt)
      throw(CommonMACHeaderException);

    CommonMACHeader(EMANE::RegistrationId   registrationId);

    EMANE::RegistrationId getRegistrationId() const;

    bool verifyCheckSum() const;

    bool checkVersion() const;

    void prependTo(DownstreamPacket & pkt) const;
  }
}
```

Listing 6.1: `EMANE::EventCommonMACHeader` Interface

6.3 Implementing a MAC Layer

1. Create a class derived from `EMANE::MACLayerImplementor`.

```
4: #include "emane/emanemaclayerimpl.h"
5:
6: namespace TRAINING
7: {
8:   class DevTrainingMAC03 : public EMANE::MACLayerImplementor
```

```
9:   {
10:   public:
11:     DevTrainingMAC03(EMANE::NEMId id,
12:                      EMANE::PlatformServiceProvider *pPlatformService)
      ...
```

2. Fill in the implementation for all virtual methods.

```
49: void TRAINING::DevTrainingMAC03::initialize()
50:   throw(EMANE::InitializeException)

56: void TRAINING::DevTrainingMAC03::configure(const EMANE::ConfigurationItems & items)
57:   throw(EMANE::ConfigureException)

64: void TRAINING::DevTrainingMAC03::start()
65:   throw(EMANE::StartException)

160: void TRAINING::DevTrainingMAC03::postStart()
161:   throw(EMANE::StartException)

166: void TRAINING::DevTrainingMAC03::stop()
167:   throw(EMANE::StopException)

173: void TRAINING::DevTrainingMAC03::destroy()
174:   throw()

180: void TRAINING::DevTrainingMAC03::processUpstreamControl(const EMANE::ControlMessage &)

186: void TRAINING::DevTrainingMAC03::processDownstreamControl(const EMANE::ControlMessage &)

191: void TRAINING::DevTrainingMAC03::processUpstreamPacket(const EMANE::CommonMACHeader & hdr,
192:                                                        EMANE::UpstreamPacket & pkt,
193:                                                        const EMANE::ControlMessage &)

203: void TRAINING::DevTrainingMAC03::processDownstreamPacket(EMANE::DownstreamPacket & pkt,
204:                                                          const EMANE::ControlMessage &)

212: void TRAINING::DevTrainingMAC03::processEvent(const EMANE::EventId &,
213:                                               const EMANE::EventObjectState &)

220: void TRAINING::DevTrainingMAC03::processTimedEvent(ACE_UINT32,
221:                                                    long,
222:                                                    const ACE_Time_Value &,
223:                                                    const void *)
```

3. Define and load the `EMANE::Component` configuration requirements for the component.

```
9: namespace
10: {
      ...
15:   const EMANE::ConfigurationDefinition defs[] =
16:     {
17:       // req, default, count, name,          value, type, description */
18:       {true, false,    1,     "boolvalue",   "off",  0,    "bool value"},
19:       {true, false,    1,     "u16value",     0,     0,    "unsigned 16 bit value"},
20:       {true, false,    1,     "u32value",     0,     0,    "unsigned 32 bit value"},
21:       {true, false,    1,     "stringvalue",  0,     0,    "string value"},
22:       {0,0,0,0,0,0,0},
23:     };
24: }
25:
26: TRAINING::DevTrainingMAC03::DevTrainingMAC03(EMANE::NEMId id,
27:                                              EMANE::PlatformServiceProvider *pPlatformService)
    :
```

```
28:   MACLayerImplementor(id, pPlatformService),
29:   bBoolValue_(false),
30:   u16Value_(0),
31:   u32Value_(0)
32: {
      ...
40:   configRequirements_ = EMANE::loadConfigurationRequirements(defs);
41: }
```

4. Expose the new MAC Layer to the EMANE infrastructure using the `DECLARE_MAC_LAYER` macro.

```
229: DECLARE_MAC_LAYER(TRAINING::DevTrainingMAC03);
```

5. Create a MAC definition XML file containing the configuration parameters and library name for the new MAC Layer implementation.

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <!DOCTYPE mac SYSTEM "file:///usr/share/emane/dtd/mac.dtd">
3 <mac name="Devel Training MAC" library="devtrainingmac03">
4   <param name="boolvalue" value="on"/>
5   <param name="u16value" value="65521"/>
6   <param name="u32value" value="6371000"/>
7   <param name="stringvalue" value="emane"/>
8 </mac>
```

6. Create an NEM XML definition file using the new MAC definition.

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <!DOCTYPE nem SYSTEM "file:///usr/share/emane/dtd/nem.dtd">
3 <nem name="Devel Training NEM">
4   <mac definition="devtrainingmac.xml"/>
5   <phy definition="devtrainingphy.xml"/>
6   <transport definition="transvirtual.xml"/>
7 </nem>
```

```
# emane platform.xml -l 4
13:58:12.883695 DEBUG MACI 001 TRAINING::DevTrainingMAC03::initialize line 52
13:58:12.883846 DEBUG MACI 001 TRAINING::DevTrainingMAC03::configure line 59
13:58:12.883894 DEBUG NEMLayerStack::initialize
13:58:12.883924 DEBUG ConcretePlatform::initialize
13:58:12.883927 DEBUG ConcretePlatform::configure
13:58:12.883948 DEBUG ConcretePlatform::start debugport = 47000
13:58:12.883956 DEBUG ConcretePlatform::start debugportenable = 0
13:58:12.883959 DEBUG ConcretePlatform::start eventservicedevice = lo
13:58:12.884072 DEBUG ConcretePlatform::start eventservicegroupaddr = 224.1.2.8
13:58:12.884078 DEBUG ConcretePlatform::start otamanagerchannelenable = 0
13:58:12.884080 DEBUG Not Starting Channel Device
13:58:12.886121 DEBUG EventService::processEventMessage
13:58:12.886134 DEBUG EventService::receiveEventMessage
13:58:12.919015 DEBUG MACI 001 TRAINING::DevTrainingMAC03::start line 67
13:58:12.919065 DEBUG MACI 001 TRAINING::DevTrainingMAC03::start boolvalue = on
13:58:12.919075 DEBUG MACI 001 TRAINING::DevTrainingMAC03::start stringvalue = emane
13:58:12.919081 DEBUG MACI 001 TRAINING::DevTrainingMAC03::start u16value = 65521
13:58:12.919087 DEBUG MACI 001 TRAINING::DevTrainingMAC03::start u32value = 6371000
13:58:12.919109 DEBUG MACI 001 TRAINING::DevTrainingMAC03::postStart line 163
```

Output 3: EMANE Platform Server (**emane**) running the configuration and MAC Implementation from Section 6.3

Chapter 7

PHY Layer API

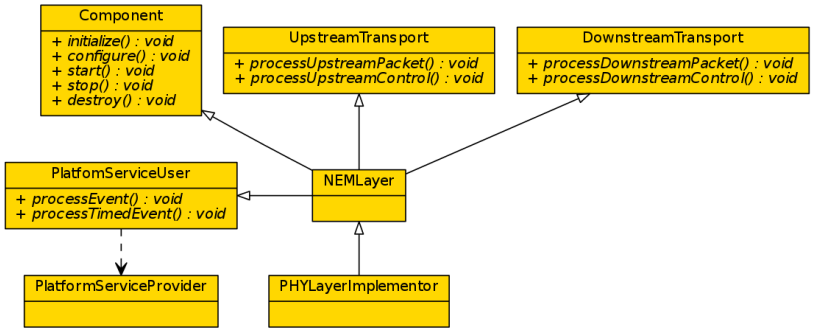

Figure 7.1: `PHYLayerImplementor` Class Diagram

A PHY Layer component implementation is a realization of the `EMANE::PHYLayerImplementor` class.

7.1 PHY Layer Packet Processing

The `EMANE::PHYLayerImplementor` further specializes the `processUpstreamPacket` and the `sendDownstreamPacket` methods by introducing a mandatory `EMANE::CommonPHYHeader`[1] parameter. The presence of a common PHY header allows for emulation of heterogeneous networks, directional antenna, frequency hopping, interference mitigation techniques and dynamic spectral allocation algorithms.

7.1.1 `EMANE::PHYLayerImplementor::processUpstreamPacket(const EMANE::CommonPHYHeader & hdr, EMANE::UpstreamPacket & pkt,const EMANE::ControlMessage & msg`

The `processUpstreamPacket` method is called on an PHY Layer to process an upstream packet sent from the layer immediately below the recipient in the NEM Layer stack. Each component should treat the contents of a received packet as opaque. Cross-layer communication can be achieved using an optional control message that may accompany the packet.

[1] src: emane/include/emane/emanecommonphyheader.{h,inl}

Parameter Description:

`const EMANE::CommonPHYHeader &`	`hdr`	Common PHY Layer header reference
`EMANE::UpstreamPacket &`	`pkt`	Upstream packet to process
`const EMANE::ControlMessage &`	`msg`	Control message to process or `EMPTY_CONTROL_MESSAGE`

7.1.2 `void EMANE::PHYLayerImplementor::sendDownstreamPacket(const CommonPHYHeader & hdr, DownstreamPacket & pkt, const ControlMessage & msg)`

The `sendDownstreamPacket` method is called by a PHY Layer to send a downstream packet to the layer immediately below the sender in the NEM Layer stack. Cross-layer communication can be achieved using an optional control message that may accompany the packet (Note: There is no use case where a PHY Layer would send a downstream control message).

Parameter Description:

`const EMANE::CommonPHYHeader &`	`hdr`	Common PHY Layer header reference
`DownstreamPacket &`	`pkt`	Downstream packet to send
`const ControlMessage &`	`msg`	Control message to send or `EMPTY_CONTROL_MESSAGE`

7.2 PHY Layer Common Header

The EMANE Common PHY Header is a mandatory PHY Layer implementation header. A PHY Layer implementation must use the `EMANE::CommonPHYHeader`[2] to allow for support of heterogeneous NEM deployments. This header allows different physical layer implementations to process the potential spectrum impact of packets generated by other waveforms. A PHY layer implementation, just like all NEM Layers, is not limited on the number of headers it can add to an `EMANE::DownstreamPacket`. Listing 7.1 contains the `EMANE::CommonPHYHeader` definition. Listing 7.2 shows sample `EMANE::CommonPHYHeader` usage.

```
namespace EMANE
{
  class CommonPHYHeader
  {
  public:

    CommonPHYHeader(UpstreamPacket & pkt)
      throw(CommonPHYHeaderException);

    CommonPHYHeader(EMANE::RegistrationId    registrationId,
                    float                    fTxPowerdBm,
                    float                    fAntennaGaindBm,
                    const ACE_Time_Value &   tvTxTime,
                    const ACE_Time_Value &   tvDuration,
                    ACE_UINT32               u32CenterFrequencyKHz,
                    ACE_UINT16               u16BandwidthKHz,
                    ACE_UINT16               u16Sequence);

    EMANE::RegistrationId getRegistrationId() const;

    float getTxPowerdBm() const;

    float getAntennaGaindBi() const;

    ACE_Time_Value getTxTime() const;

    ACE_Time_Value getDuration() const;

    ACE_UINT32 getFrequencyKHz() const;

    ACE_UINT16 getBandWidthKHz() const;
```

[2] src: emane/include/emane/emanecommonphyheader.{h,inl}

```

    ACE_UINT16 getSequenceNumber() const;

    bool verifyCheckSum() const;

    bool checkVersion() const;

    void prependTo(DownstreamPacket & pkt) const;

    void addOption(const CommonPHYHeaderOption & option);

    CommonPHYHeaderOptionObjectStateList getOptionList() const;
  };
}
```

Listing 7.1: `EMANE::EventCommonPHYHeader` Interface

```
// phy header
EMANE::CommonPHYHeader commonPHYHeader (u16RegistrationId_,
                                        txControlMessage.getTxPowerdBm(),
                                        fAntennaGaindBi_,
                                        tvCurrentTime,
                                        txControlMessage.getDuration(),
                                        txControlMessage.getFrequencyKHz(),
                                        u16BandWidthKHz_,
                                        u16TxSequenceNumber_++);

// add header for antenna direction
if(antennaType_ == UniversalPHY::UniversalPHYLayer::ANTENNA_TYPE_UNI)
 {
   EMANE::CommonPHYHeaderAntennaOption opt(txControlMessage.getAntennaAzBeamWidthDegrees(),
                                           txControlMessage.getAntennaElBeamWidthDegrees(),
                                           txControlMessage.getAntennaAzimuthDegrees(),
                                           txControlMessage.getAntennaElevationDegrees());

   // add optional header
   commonPHYHeader.addOption(opt);
 }

// send header and packet
sendDownstreamPacket(commonPHYHeader, rPacket);
```

Listing 7.2: `EMANE::EventCommonPHYHeader` sample usage

7.2.1 `EMANE::CommonPHYHeader::CommonPHYHeader((EMANE::RegistrationId, float fTxPowerdBm, float fAntennaGaindBi, const ACE_Time_Value & tvTxTime, const ACE_Time_Value & tvDuration, ACE_UINT32 u32CenterFrequencyKHz, ACE_UINT16 u16BandwidthKHz, ACE_UINT16 u16Sequence)`

Construct an `EMANE::CommonPHYHeader` instance.

Parameter Description:

`EMANE::RegistrationId &`	`registrationId`	Registration Id of PHY Layer
`float`	`fTxPowerdBm`	Transmit Power in dBm
`float`	`fAntennaGaindBi`	Antenna Gain in dBi
`const ACE_Time_Value &`	`tvTxTime`	Transmit timestamp
`const ACE_Time_value &`	`tvDuration`	Packet transmit duration
`ACE_UINT32`	`u32CenterFrequencyKHz`	Center Frequency in KHz
`ACE_UINT16`	`u16BandwidthKHz`	Bandwidth in KHz
`ACE_UINT16`	`u16Sequence`	Sequence number

7.2.2 `EMANE::RegistrationId EMANE::CommonPHYHeader::getRegistrationId() const`

The `getRegistrationId` method returns PHY Layer registration id of the transmitter. The registration id is used to identify PHY Layers of a different type (waveform).

7.2.3 `float EMANE::CommonPHYHeader::getTxPowerdBm() const`

The `getTxPowerdBm` method returns the transmit power in dBm of the transmitted packet.

7.2.4 `float EMANE::CommonPHYHeader::getAntennaGaindBi() const`

The `getAntennaGaindBi` method returns the antenna gain in dBi of the transmitter.

7.2.5 `ACE_Time_Value EMANE::CommonPHYHeader::getTxTime() const`

The `getTxTime` method returns the packet transmit timestamp.

7.2.6 `ACE_Time_Value EMANE::CommonPHYHeader::getDuration() const`

The `getDuration` method returns the transmit duration for the packet.

7.2.7 `ACE_UINT32 EMANE::CommonPHYHeader::getFrequencyKHz() const`

The `getFrequencyKHz` method returns the center frequency of the transmitted packet in KHz.

7.2.8 `ACE_UINT16 EMANE::CommonPHYHeader::getBandWidthKHz() const`

The `getBandwidthKHz` method returns the bandwidth of the transmitted packet in KHz.

7.2.9 `ACE_UINT16 EMANE::CommonPHYHeader::getSequenceNumber() const`

The `getSequenceNumber` method returns the packet sequence number.

7.2.10 `void EMANE::CommonPHYHeader::prependTo(EMANE::Downstream & pkt) const`

The `prependTo` method prepends the `EMANE::CommonPHYHeader` to an `EMANE::DownstreamPacket`.

Parameter Description:

`EMANE::Downstream &`	`pkt`	Downstream packet to prepend header

7.2.11 `void EMANE::CommonPHYHeader::addOption(const CommonPHYHeaderOption & option)`

The `addOption` method is used to add an `EMANE::CommonPHYHeaderOption`[3] to the `EMANE::CommonPHYHeader`. Optional header support allows for additional common PHY information for use by higher fidelity models without requiring functional changes to existing models.

Parameter Description:

`const CommonPHYHeaderOption &`	`option`	Option to add to header

7.2.12 `EMANE::CommonPHYHeaderOptionObjectStateList EMANE::CommonPHYHeader::getOptionList() const`

The `getOptionList` method returns a list of `EMANE::CommonPHYHeaderOptionObjectState`[4] objects corresponding to the optional header items contained in the `EMANE::CommonPHYHeader`.

7.3 Implementing a PHY Layer

1. Create a class derived from `EMANE::PHYLayerImplementor`.

```
4: #include "emane/emanephylayerimpl.h"
5:
6: namespace TRAINING
7: {
8:   class DevTrainingPHY03 : public EMANE::PHYLayerImplementor
9:   {
10:   public:
11:     DevTrainingPHY03(EMANE::NEMId id,
12:                      EMANE::PlatformServiceProvider *pPlatformService)
      ...
```

2. Fill in the implementation for all virtual methods.

```
52: void TRAINING::DevTrainingPHY04::initialize()
53:   throw(EMANE::InitializeException)

59: void TRAINING::DevTrainingPHY04::configure(const EMANE::ConfigurationItems & items)
60:   throw(EMANE::ConfigureException)

67: void TRAINING::DevTrainingPHY04::start()
68:   throw(EMANE::StartException)

163: void TRAINING::DevTrainingPHY04::postStart()
164:   throw(EMANE::StartException)

169: void TRAINING::DevTrainingPHY04::stop()
170:   throw(EMANE::StopException)

176: void TRAINING::DevTrainingPHY04::destroy()
177:   throw()

183: void TRAINING::DevTrainingPHY04::processUpstreamControl(const EMANE::ControlMessage &)

189: void TRAINING::DevTrainingPHY04::processDownstreamControl(const EMANE::ControlMessage &)

194: void TRAINING::DevTrainingPHY04::processUpstreamPacket(const EMANE::CommonPHYHeader & hdr,
195:                                                        EMANE::UpstreamPacket & pkt,
```

[3] src: emane/include/emane/emanecommonphyheaderoption.{h,inl}
[4] src: emane/include/emane/emanecommonphyheaderoptionobjectstate.{h,inl}

```
                                                        const EMANE::ControlMessage &)
```

```
void TRAINING::DevTrainingPHY04::processDownstreamPacket(EMANE::DownstreamPacket & pkt,
                                                         const EMANE::ControlMessage &)
```

```
void TRAINING::DevTrainingPHY04::processEvent(const EMANE::EventId &,
                                              const EMANE::EventObjectState &)
```

```
void TRAINING::DevTrainingPHY04::processTimedEvent(ACE_UINT32,
                                                   long,
                                                   const ACE_Time_Value &,
                                                   const void *)
```

3. Define and load the `EMANE::Component` configuration requirements for the component.

```
namespace
{
    ...

  const EMANE::ConfigurationDefinition defs[] =
    {
      // req, default, count, name,          value, type, description */
      {true, false,    1,     "boolvalue",   "off",  0,   "bool value"},
      {true, false,    1,     "u16value",     0,     0,   "unsigned 16 bit value"},
      {true, false,    1,     "u32value",     0,     0,   "unsigned 32 bit value"},
      {true, false,    1,     "stringvalue",  0,     0,   "string value"},
      {0,0,0,0,0,0,0},
    };
}

TRAINING::DevTrainingPHY04::DevTrainingPHY04(EMANE::NEMId id,
                                             EMANE::PlatformServiceProvider *pPlatformService)
  :
  PHYLayerImplementor(id, pPlatformService),
  bBoolValue_(false),
  u16Value_(0),
  u32Value_(0),
  u16TxSequenceNumber_(0)
{
    ...
  configRequirements_ = EMANE::loadConfigurationRequirements(defs);
}
```

4. Expose the new PHY Layer to the EMANE infrastructure using the `DECLARE_PHY_LAYER` macro.

```
DECLARE_PHY_LAYER(TRAINING::DevTrainingPHY04);
```

5. Create a PHY definition XML file containing the configuration parameters and library name for the new PHY Layer implementation.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE phy SYSTEM "file:///usr/share/emane/dtd/phy.dtd">
<phy name="Devel Training PHY" library="devtrainingphy04">
  <param name="boolvalue" value="on"/>
  <param name="u16value" value="65521"/>
  <param name="u32value" value="6371000"/>
  <param name="stringvalue" value="emane"/>
</phy>
```

6. Create an NEM XML definition file using the new PHY definition.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE nem SYSTEM "file:///usr/share/emane/dtd/nem.dtd">
<nem name="Devel Training NEM">
  <mac definition="devtrainingmac.xml"/>
  <phy definition="devtrainingphy.xml"/>
  <transport definition="transvirtual.xml"/>
</nem>
```

```
# emane platform.xml -l 4
14:00:04.307020 DEBUG MACI 001 TRAINING::DevTrainingMAC03::initialize line 52
14:00:04.307174 DEBUG MACI 001 TRAINING::DevTrainingMAC03::configure line 59
14:00:04.307605 DEBUG PHYI 001 TRAINING::DevTrainingPHY04::initialize line 55
14:00:04.307619 DEBUG PHYI 001 TRAINING::DevTrainingPHY04::configure line 62
14:00:04.307665 DEBUG NEMLayerStack::initialize
14:00:04.307693 DEBUG ConcretePlatform::initialize
14:00:04.307696 DEBUG ConcretePlatform::configure
14:00:04.307716 DEBUG ConcretePlatform::start debugport = 47000
14:00:04.307727 DEBUG ConcretePlatform::start debugportenable = 0
14:00:04.307730 DEBUG ConcretePlatform::start eventservicedevice = lo
14:00:04.307840 DEBUG ConcretePlatform::start eventservicegroupaddr = 224.1.2.8
14:00:04.307846 DEBUG ConcretePlatform::start otamanagerchannelenable = 0
14:00:04.307849 DEBUG Not Starting Channel Device
14:00:04.309615 DEBUG EventService::receiveEventMessage
14:00:04.309626 DEBUG EventService::processEventMessage
14:00:04.348323 DEBUG MACI 001 TRAINING::DevTrainingMAC03::start line 67
14:00:04.348341 DEBUG MACI 001 TRAINING::DevTrainingMAC03::start boolvalue = on
14:00:04.348348 DEBUG MACI 001 TRAINING::DevTrainingMAC03::start stringvalue = emane
14:00:04.348355 DEBUG MACI 001 TRAINING::DevTrainingMAC03::start u16value = 65521
14:00:04.348361 DEBUG MACI 001 TRAINING::DevTrainingMAC03::start u32value = 6371000
14:00:04.348455 DEBUG PHYI 001 TRAINING::DevTrainingPHY04::start line 70
14:00:04.348491 DEBUG PHYI 001 TRAINING::DevTrainingPHY04::start boolvalue = on
14:00:04.348500 DEBUG PHYI 001 TRAINING::DevTrainingPHY04::start stringvalue = emane
14:00:04.348507 DEBUG PHYI 001 TRAINING::DevTrainingPHY04::start u16value = 65521
14:00:04.348511 DEBUG PHYI 001 TRAINING::DevTrainingPHY04::start u32value = 6371000
14:00:04.348528 DEBUG MACI 001 TRAINING::DevTrainingMAC03::postStart line 163
14:00:04.348532 DEBUG PHYI 001 TRAINING::DevTrainingPHY04::postStart line 166
```

Output 4: EMANE Platform Server (**emane**) running the configuration and PHY Implementation from Section 7.3 and MAC Implementation from Section 6.3

Chapter 8

Shim Layer API

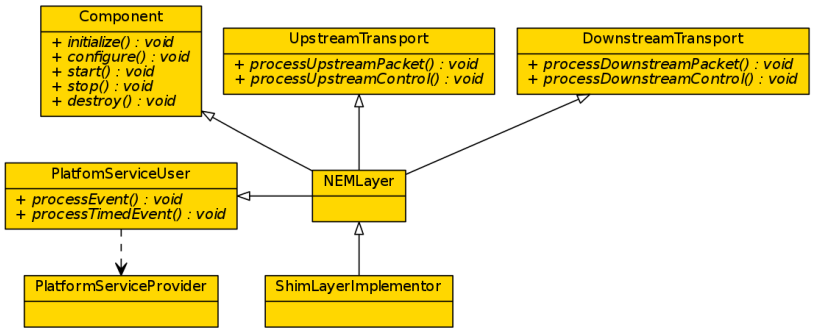

Figure 8.1: `ShimLayerImplementor` Class Diagram

A Shim Layer may be inserted between component layers in an NEM Layer stack without requiring those components to have knowledge of its presence. A Shim Layer component implementation is a realization of the `EMANE::ShimLayerImplementor` class.

8.1 Implementing a Shim Layer

1. Create a class derived from `EMANE::ShimLayerImplementor`.

```
4: #include "emane/emaneshimlayerimpl.h"

8: namespace TRAINING
9: {
10:   class DevTrainingShim02 : public EMANE::ShimLayerImplementor
11:   {
12:   public:
13:     DevTrainingShim02(EMANE::NEMId id, EMANE::PlatformServiceProvider *pPlatformService);
      ...
```

2. Fill in the implementation for all virtual methods.

```
39: void TRAINING::DevTrainingShim02::initialize()
40:   throw(EMANE::InitializeException)

46: void TRAINING::DevTrainingShim02::configure(const EMANE::ConfigurationItems & items)
```

```
  throw(EMANE::ConfigureException)

void TRAINING::DevTrainingShim02::start()
  throw(EMANE::StartException)

void TRAINING::DevTrainingShim02::postStart()
  throw(EMANE::StartException)

void TRAINING::DevTrainingShim02::stop()
  throw(EMANE::StopException)

void TRAINING::DevTrainingShim02::destroy()
  throw()

void TRAINING::DevTrainingShim02::processUpstreamControl(const EMANE::ControlMessage & msg)

void TRAINING::DevTrainingShim02::processDownstreamControl(const EMANE::ControlMessage & msg)

void TRAINING::DevTrainingShim02::processUpstreamPacket(EMANE::UpstreamPacket & pkt,
                                                        const EMANE::ControlMessage & msg)

void TRAINING::DevTrainingShim02::processDownstreamPacket(EMANE::DownstreamPacket & pkt,
                                                          const EMANE::ControlMessage & msg)

void TRAINING::DevTrainingShim02::processEvent(const EMANE::EventId &,
                                               const EMANE::EventObjectState &)

void TRAINING::DevTrainingShim02::processTimedEvent(ACE_UINT32,
                                                    long,
                                                    const ACE_Time_Value &,
                                                    const void *)
```

3. Define and load the `EMANE::Component` configuration requirements for the component.

```
namespace
{
    ...
 const EMANE::ConfigurationDefinition defs[] =
    {
      // req, default, count, name,             value, type, description */
      {false, true,    1,     "traceenable",   "off",  0,    "turn on packet trace"},
      {false, true,    1,     "maxstore",      "10",   0,    "max trace id amount"},
      {false, false,   0,     "ignorenode",     0,     0,    "do not trace dest node"},
      {true,  false,   1,     "floatvalue",     0,     0,    "unused value"},
      {0,0,0,0,0,0,0},
    };
}

TRAINING::DevTrainingShim02::DevTrainingShim02(EMANE::NEMId id, EMANE::PlatformServiceProvider
    *pPlatformService) :
  ShimLayerImplementor(id, pPlatformService),
  bTraceMode_(false),
  u16MaxStore_(0)
{
    ...
  configRequirements_ = EMANE::loadConfigurationRequirements(defs);
}
```

4. Expose the new Shim Layer to the EMANE infrastructure using the `DECLARE_SHIM_LAYER` macro.

```
DECLARE_SHIM_LAYER(TRAINING::DevTrainingShim02);
```

5. Create a SHIM definition XML file containing the configuration parameters and library name for the new Shim Layer implementation.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE shim SYSTEM "file:///usr/share/emane/dtd/shim.dtd">
<shim name="Devel Training Shim" library="devtrainingshim02">
  <param name="floatvalue" value="2.7182818284"/>
</shim>
```

6. Create an NEM XML definition file using the new Shim definition.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE nem SYSTEM "file:///usr/share/emane/dtd/nem.dtd">
<nem name="Devel Training NEM">
  <mac definition="devtrainingmac.xml"/>
  <shim definition="devtrainingshim.xml"/>
  <phy definition="devtrainingphy.xml"/>
  <transport definition="transvirtual.xml"/>
</nem>
```

```
# emane platform.xml -l 4
14:03:05.283907 DEBUG MACI 001 TRAINING::DevTrainingMAC03::initialize line 52
14:03:05.283987 DEBUG MACI 001 TRAINING::DevTrainingMAC03::configure line 59
14:03:05.284168 DEBUG SHIMI 001 TRAINING::DevTrainingShim02::initialize line 42
14:03:05.284174 DEBUG SHIMI 001 TRAINING::DevTrainingShim02::configure line 49
14:03:05.284363 DEBUG PHYI 001 TRAINING::DevTrainingPHY04::initialize line 55
14:03:05.284367 DEBUG PHYI 001 TRAINING::DevTrainingPHY04::configure line 62
14:03:05.284388 DEBUG NEMLayerStack::initialize
14:03:05.284401 DEBUG ConcretePlatform::initialize
14:03:05.284403 DEBUG ConcretePlatform::configure
14:03:05.284412 DEBUG ConcretePlatform::start debugport = 47000
14:03:05.284417 DEBUG ConcretePlatform::start debugportenable = 0
14:03:05.284418 DEBUG ConcretePlatform::start eventservicedevice = lo
14:03:05.284489 DEBUG ConcretePlatform::start eventservicegroupaddr = 224.1.2.8
14:03:05.284492 DEBUG ConcretePlatform::start otamanagerchannelenable = 0
14:03:05.284493 DEBUG Not Starting Channel Device
14:03:05.285407 DEBUG EventService::receiveEventMessage
14:03:05.285522 DEBUG EventService::processEventMessage
14:03:05.317700 DEBUG MACI 001 TRAINING::DevTrainingMAC03::start line 67
14:03:05.317721 DEBUG MACI 001 TRAINING::DevTrainingMAC03::start boolvalue = on
14:03:05.317729 DEBUG MACI 001 TRAINING::DevTrainingMAC03::start stringvalue = emane
14:03:05.317736 DEBUG MACI 001 TRAINING::DevTrainingMAC03::start u16value = 65521
14:03:05.317743 DEBUG MACI 001 TRAINING::DevTrainingMAC03::start u32value = 6371000
14:03:05.317909 DEBUG SHIMI 001 TRAINING::DevTrainingShim02::start line 57
14:03:05.317953 DEBUG MACI 001 TRAINING::DevTrainingShim02::start floatvalue = 2.718282
14:03:05.317974 DEBUG MACI 001 TRAINING::DevTrainingShim02::start maxstore = 10
14:03:05.317982 DEBUG MACI 001 TRAINING::DevTrainingShim02::start traceenable = off
14:03:05.318083 DEBUG PHYI 001 TRAINING::DevTrainingPHY04::start line 70
14:03:05.318098 DEBUG PHYI 001 TRAINING::DevTrainingPHY04::start boolvalue = on
14:03:05.318103 DEBUG PHYI 001 TRAINING::DevTrainingPHY04::start stringvalue = emane
14:03:05.318108 DEBUG PHYI 001 TRAINING::DevTrainingPHY04::start u16value = 65521
14:03:05.318112 DEBUG PHYI 001 TRAINING::DevTrainingPHY04::start u32value = 6371000
14:03:05.318126 DEBUG MACI 001 TRAINING::DevTrainingMAC03::postStart line 163
14:03:05.318130 DEBUG SHIMI 001 TRAINING::DevTrainingShim02::postStart line 155
14:03:05.318132 DEBUG PHYI 001 TRAINING::DevTrainingPHY04::postStart line 166
```

Output 5: EMANE Platform Server (**emane**) running the configuration and Shim implementation from Section 8.1, the MAC implementation from Section 6.3, and the PHY implementation from Section 7.3

Chapter 9

Events

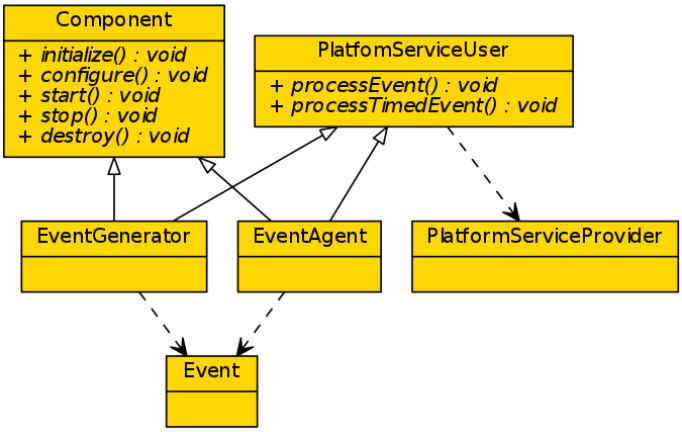

Figure 9.1: `Event` Class Diagram

EMANE Events are messages sent to EMANE components containing control information relevant to the functional operation of the component. EMANE provides the infrastructure necessary to generically transmit and receive events using the Event Service, Event Generators and Event Agents.

An Event implementation is a realization of the `EMANE::Event` interface. The `EMANE::Event` interface provides a method to extract event object data as an opaque octet buffer for transmission to targeted EMANE components.

```
namespace EMANE
{
  class Event
  {
  public:
    virtual ~Event(){}

    EventId getEventId() const;

    const std::string & getEventDescription() const;

    virtual EventObjectState getObjectState() const = 0;
  };
}
```

The `EventObjectState` class is a wrapper around an octet buffer containing the event data necessary to reconstruct the derived event object once it is received by the targeted EMANE component.

```
namespace EMANE
{
  class EventObjectStateException{};

  class EventObjectState
  {
  public:
    EventObjectState(const void * buf, size_t len);

    ~EventObjectState(){};

    const void * get() const;

    size_t length() const;
  };
}
```

9.1 Implementing an Event

1. Create a class derived from `EMANE::Event`.

```
namespace TRAINING
{
  class DevTrainingEvent : public EMANE::Event
  {
  public:
    ...
```

2. Provide a constructor that takes an `EMANE::EventObjectState` constant reference argument and use the object state to reconstruct the transmitted event object. The `EMANE::Event` base class takes an event id and an event name string as constructor arguments. Throw an `EMANE::EventObjectStateExcpetion` if an error is detected with the object state data. **Event data contained in `EMANE::EventObjectState` objects are in Network Byte Order.** Event ids are 16 bit values. Ids with the most significant bit set are local ids.

```
TRAINING::DevTrainingEvent::DevTrainingEvent(const EMANE::EventObjectState & state)
  throw(EMANE::EventObjectStateException):
  Event(EVENT_ID,"Devel Training Loss Event")
{
  const DevTrainingEventState * p = reinterpret_cast<const DevTrainingEventState *>(state.get()
);

  ACE_UINT16 u16NumberOfEntries = ACE_NTOHS(p->u16NumberEntries_);

  for(ACE_UINT16 i = 0; i < u16NumberOfEntries; ++i)
    {
      lossMap_.insert(std::make_pair(ACE_NTOHS(p->entries_[i].u16Node_),
                                     ACE_NTOHS(p->entries_[i].u16LossPercentage_)));
    }
}
```

3. Provide an implementation for the `getObjectState` method. This method returns an `EMANE::EventObjectState` object containing all the data necessary to reconstruct this object when received by the targeted EMANE components. **Event data contained in `EMANE::EventObjectState` objects are in Network Byte Order.**

```
43: EMANE::EventObjectState TRAINING::DevTrainingEvent::getObjectState() const
44: {
45:   ACE_UINT8 buf[sizeof(DevTrainingEventState) +
46:                 lossMap_.size() * sizeof(DevTrainingEventEntry)];
47:
48:   DevTrainingEventState * p = reinterpret_cast<DevTrainingEventState *>(buf);
49:
50:   p->u16NumberEntries_ = ACE_HTONS(lossMap_.size());
51:
52:   LossMap::const_iterator iter = lossMap_.begin();
53:
54:   for(ACE_UINT16 i = 0; iter != lossMap_.end(); ++i,++iter)
55:     {
56:       p->entries_[i].u16Node_           = ACE_HTONS(iter->first);
57:       p->entries_[i].u16LossPercentage_ = ACE_HTONS(iter->second);
58:     }
59:
60:   return EMANE::EventObjectState(buf,sizeof(buf));
61: }
```

9.2 Event Service

The Event Service provides an application framework for the generic loading, configuring and controlling of Event Generators, which may themselves have varying degrees of implementation complexity. The Event Service allows a decoupling of the creation of emulation events from their distribution. The `EMANE::EventGenerator` interface provides a standard way to communicate with event generation functionality without imposing restrictions on how events are created.

Simple Event Generators may parse time based scenario files or contain algorithms to generate event content and then use the API to send those events to all or some of the NEMs contained in the deployment. More complex generators may communicate with backend database servers or other subsystems to generate the appropriate event content.

9.2.1 Implementing an Event Generator

1. Create a class derived from `EMANE::EventGenerator`.

```
11: namespace TRAINING
12: {
13:   class DevTrainingGen06 : public EMANE::EventGenerator
14:   {
15:   public:
16:     DevTrainingGen06(EMANE::PlatformServiceProvider *pPlatformService);
      ...
```

2. Fill in the implementation for all virtual methods.

```
52: void TRAINING::DevTrainingGen06::initialize()
53:   throw(EMANE::InitializeException)
```

```
59: void TRAINING::DevTrainingGen06::configure(const EMANE::ConfigurationItems & items)
60:   throw(EMANE::ConfigureException)
```

```
67: void TRAINING::DevTrainingGen06::start()
68:   throw(EMANE::StartException)
```

```
171: void TRAINING::DevTrainingGen06::postStart()
```

```
  throw(EMANE::StartException)

void TRAINING::DevTrainingGen06::stop()
  throw(EMANE::StopException)

void TRAINING::DevTrainingGen06::destroy()
  throw()

void TRAINING::DevTrainingGen06::processEvent(const EMANE::EventId &,
                                              const EMANE::EventObjectState &)

void TRAINING::DevTrainingGen06::processTimedEvent(ACE_UINT32,
                                                   long,
                                                   const ACE_Time_Value &,
                                                   const void *)
```

3. Define and load the `EMANE::Component` configuration requirements for the component.

```
namespace
{
    ...
  const EMANE::ConfigurationDefinition defs[] =
    {
      // req, default, count, name,           value, type, description */
      {true, false,    1,     "boolvalue",   "off",  0,   "bool value"},
      {true, false,    1,     "u16value",     0,     0,   "unsigned 16 bit value"},
      {true, false,    1,     "u32value",     0,     0,   "unsigned 32 bit value"},
      {true, false,    1,     "stringvalue",  0,     0,   "string value"},
      {true, false,    1,     "modelfilename",0,     0,   "model file name"},
      {0,0,0,0,0,0,0},
    };
}

TRAINING::DevTrainingGen06::DevTrainingGen06(EMANE::PlatformServiceProvider *pPlatformService)
  :
  EventGenerator(MODULE,pPlatformService),
  bBoolValue_(false),
  u16Value_(0),
  u32Value_(0),
  bCancel_(false),
  thread_(0)
{
    ...
  configRequirements_ = EMANE::loadConfigurationRequirements(defs);
}
```

4. Register any events that the Event Generator will produce using the `EMANE::EventGenerator::addEventId` method.

```
TRAINING::DevTrainingGen06::DevTrainingGen06(EMANE::PlatformServiceProvider *pPlatformService)
  :
  EventGenerator(MODULE,pPlatformService),
  bBoolValue_(false),
  u16Value_(0),
  u32Value_(0),
  bCancel_(false),
  thread_(0)
{
  addEventId(DevTrainingEvent::getEventId());

  pPlatformService_->log(EMANE::DEBUG_LEVEL,
                         "%s::%s line %d",

                         MODULE,
                         __func__,
                         __LINE__);
```

```

  configRequirements_ = EMANE::loadConfigurationRequirements(defs);
}
```

5. Create a new thread to execute the backend code necessary to generate events. This might be as simple as parsing input files, building events and then sending them using the Platform Service.

```
  EMANEUtils::spawn(*this,&TRAINING::DevTrainingGen06::generate,&thread_);
...
ACE_THR_FUNC_RETURN TRAINING::DevTrainingGen06::generate()
{
  while(!bCancel_)
    {
          ...
          DevTrainingEvent event;
          for(; iterLossPair != iterLoss->second.end(); ++iterLossPair)
            {
              event.addEntry(iterLossPair->first,iterLossPair->second);
            }

          //send the event
          pPlatformService_->sendEvent(0,                // all platform(s)
                                       0,                // all nems
                                       EMANE::COMPONENT_PHYILAYER,
                                       event);

          ...
    }

  return 0;
}
```

6. Expose the new Event Generator to the EMANE Event Service using the `DECLARE_EVENT_GENERATOR` macro.

```
DECLARE_EVENT_GENERATOR(TRAINING::DevTrainingGen06);
```

7. Create an Event Generator definition XML file containing the configuration parameters and library name for the new Event Generator implementation.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE eventgenerator SYSTEM "file:///usr/share/emane/dtd/eventgenerator.dtd">
<eventgenerator name="Devel Training Generator" library="devtraininggen06">
  <param name="boolvalue" value="on"/>
  <param name="u16value" value="65521"/>
  <param name="u32value" value="6371000"/>
  <param name="stringvalue" value="emane"/>
  <param name="modelfilename" value="model.txt"/>
</eventgenerator>
```

8. Add the Event Generator to the EMANE Event Service XML.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE eventservice SYSTEM "file:///usr/share/emane/dtd/eventservice.dtd">
<eventservice name="Sample Event Service" deployment="deployment.xml">
  <param name="eventservicegroup" value="224.1.2.8:45703"/>
  <param name="eventservicedevice" value="lo"/>
  <generator name="Devel Training Generator" definition="devtraininggen.xml"/>
</eventservice>
```

```
# emaneeventservice eventservice.xml -l 4
14:06:24.319148 DEBUG TRAINING::DevTrainingGen06::DevTrainingGen06 line 43
14:06:24.319312 DEBUG TRAINING::DevTrainingGen06::initialize line 57
14:06:24.319317 DEBUG TRAINING::DevTrainingGen06::configure line 64
14:06:24.319343 DEBUG ConcreteEventGeneratorManager::configure
14:06:24.319357 DEBUG ConcreteEventGeneratorManager::start Event Service Device: lo
14:06:24.319470 DEBUG ConcreteEventGeneratorManager::start Event Service Group: 224.1.2.8
14:06:24.320899 DEBUG TRAINING::DevTrainingGen06::start line 72
14:06:24.320920 DEBUG TRAINING::DevTrainingGen06::start boolvalue = on
14:06:24.320924 DEBUG TRAINING::DevTrainingGen06::start modelfilename = model.txt
14:06:24.321051 DEBUG TRAINING::DevTrainingGen06::start stringvalue = emane
14:06:24.321057 DEBUG TRAINING::DevTrainingGen06::start u16value = 65521
14:06:24.321061 DEBUG TRAINING::DevTrainingGen06::start u32value = 6371000
14:06:24.321342 DEBUG Event 32770 EMANE::EventService::sendEvent
14:06:24.321618 DEBUG EventService::processEventMessage
14:06:24.321627 DEBUG TRAINING::DevTrainingGen06::postStart line 176
14:06:24.321618 DEBUG EventService::receiveEventMessage
14:06:24.321652 DEBUG EventService Packet Received len: 36
14:06:24.321689 DEBUG ConcreteEventGeneratorManager::svc
14:06:24.321710 DEBUG ConcreteEventGeneratorManager Packet Received len: 36
14:06:25.321620 DEBUG Event 32770 EMANE::EventService::sendEvent
14:06:25.321721 DEBUG ConcreteEventGeneratorManager Packet Received len: 36
14:06:25.321749 DEBUG EventService Packet Received len: 36
14:06:26.321839 DEBUG Event 32770 EMANE::EventService::sendEvent
14:06:26.321940 DEBUG ConcreteEventGeneratorManager Packet Received len: 36
```

Output 6: EMANE Event Service (**emaneeventservice**) running the configuration and Generator implementation from Section 9.2

9.3 Event Daemon

The EMANE Event Daemon provides a mechanism to transport EMANE Event data from the emulation domain to other application domains. Agent implementations use the `EMANE::EventAgent` interface to register to receive EMANE Events in order to communicate the data with other non EMANE aware applications.

The **emaneeventd** service must run on any test node that requires agent provided event translation. The Event Multicast Channel must be accessible by the test node running the daemon.

9.3.1 Implementing an Event Agent

1. Create a class derived from `EMANE::EventAgent`.

```
4: #include <emane/emaneeventagent.h>
5:
6: namespace TRAINING
7: {
8:   class DevTrainingAgent08 : public EMANE::EventAgent
9:   {
10:   public:
11:     DevTrainingAgent08(EMANE::NEMId id,
12:                        EMANE::PlatformServiceProvider *pPlatformService);
13:
       ...
```

2. Fill in the implementation for all virtual methods.

```
49: void TRAINING::DevTrainingAgent08::initialize()
50:   throw(EMANE::InitializeException)

56: void TRAINING::DevTrainingAgent08::configure(const EMANE::ConfigurationItems & items)
```

```
  throw(EMANE::ConfigureException)
```

```
void TRAINING::DevTrainingAgent08::start()
  throw(EMANE::StartException)
```

```
void TRAINING::DevTrainingAgent08::postStart()
  throw(EMANE::StartException)
```

```
void TRAINING::DevTrainingAgent08::stop()
  throw(EMANE::StopException)
```

```
void TRAINING::DevTrainingAgent08::destroy()
  throw()
```

```
void TRAINING::DevTrainingAgent08::processEvent(const EMANE::EventId & eventId,
                                                const EMANE::EventObjectState & state)
```

```
void TRAINING::DevTrainingAgent08::processTimedEvent(ACE_UINT32,
                                                     long,
                                                     const ACE_Time_Value &,
                                                     const void *)
```

```
EMANE::EventRegistrationList TRAINING::DevTrainingAgent08::getEventRegistrationList()
```

3. Define and load the `EMANE::Component` configuration requirements for the agent.

```
namespace
{
    ...
  const EMANE::ConfigurationDefinition defs[] =
    {
      // req, default, count, name,          value, type, description */
      {true, false,    1,     "boolvalue",   "off",  0,   "bool value"},
      {true, false,    1,     "u16value",     0,     0,   "unsigned 16 bit value"},
      {true, false,    1,     "u32value",     0,     0,   "unsigned 32 bit value"},
      {true, false,    1,     "stringvalue",  0,     0,   "string value"},
      {0,0,0,0,0,0,0},
    };
}

TRAINING::DevTrainingAgent08::DevTrainingAgent08(EMANE::NEMId nemId,
                                                 EMANE::PlatformServiceProvider *
 pPlatformService) :
  EventAgent(nemId,pPlatformService),
  bBoolValue_(false),
  u16Value_(0),
  u32Value_(0)
{
    ...
  configRequirements_ = EMANE::loadConfigurationRequirements(defs);
}
```

4. Expose the new Event Agent to the Event Daemon using the `DECLARE_EVENT_AGENT` macro.

```
DECLARE_EVENT_AGENT(TRAINING::DevTrainingAgent08)
```

5. Create an Event Agent definition XML file containing the configuration parameters and library name for the new Event Agent implementation.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE eventagent SYSTEM "file:///usr/share/emane/dtd/eventagent.dtd">
<eventagent name="Devel Training Agent" library="devtrainingagent08">
  <param name="boolvalue" value="on"/>
  <param name="u16value" value="65521"/>
```

```
  <param name="u32value" value="6371000"/>
  <param name="stringvalue" value="emane"/>
</eventagent>
```

6. Add the Event Agent to the EMANE Event Daemon XML.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE eventdaemon SYSTEM "file:///usr/share/emane/dtd/eventdaemon.dtd">
<eventdaemon name="EMANE Event Daemon" nemid="1">
  <param name="eventservicegroup" value="224.1.2.8:45703"/>
  <param name="eventservicedevice" value="lo"/>
  <agent definition="devtrainingagent.xml"/>
</eventdaemon>
```

```
# emaneeventd eventdaemon.xml -l 4
14:11:58.588725 DEBUG TRAINING::DevTrainingAgent08::DevTrainingAgent08 line 38
14:11:58.588876 DEBUG TRAINING::DevTrainingAgent08::initialize line 52
14:11:58.588881 DEBUG TRAINING::DevTrainingAgent08::configure line 59
14:11:58.590781 DEBUG TRAINING::DevTrainingAgent08::start line 67
14:11:58.590814 DEBUG TRAINING::DevTrainingAgent08::start boolvalue = on
14:11:58.590822 DEBUG TRAINING::DevTrainingAgent08::start stringvalue = emane
14:11:58.590828 DEBUG TRAINING::DevTrainingAgent08::start u16value = 65521
14:11:58.590937 DEBUG TRAINING::DevTrainingAgent08::start u32value = 6371000
14:11:58.590985 DEBUG EventService::processEventMessage
14:11:58.591013 DEBUG TRAINING::DevTrainingAgent08::postStart line 156
14:11:58.591051 DEBUG EventService::receiveEventMessage
14:12:00.541298 DEBUG EventService Packet Received len: 36
14:12:00.541301 DEBUG ConcreteEventAgentManager Packet Received len: 36
14:12:00.541359 DEBUG TRAINING::DevTrainingAgent08::processEvent line 175
14:12:00.541381 DEBUG TRAINING::DevTrainingAgent08::processEvent 1 0% loss
14:12:00.541389 DEBUG TRAINING::DevTrainingAgent08::processEvent 2 0% loss
14:12:00.541394 DEBUG TRAINING::DevTrainingAgent08::processEvent 3 0% loss
14:12:00.541398 DEBUG TRAINING::DevTrainingAgent08::processEvent 4 0% loss
14:12:00.541402 DEBUG TRAINING::DevTrainingAgent08::processEvent 5 0% loss
14:12:01.541519 DEBUG EventService Packet Received len: 36
14:12:01.541578 DEBUG ConcreteEventAgentManager Packet Received len: 36
14:12:01.541596 DEBUG TRAINING::DevTrainingAgent08::processEvent line 175
14:12:01.541602 DEBUG TRAINING::DevTrainingAgent08::processEvent 1 10% loss
14:12:01.541604 DEBUG TRAINING::DevTrainingAgent08::processEvent 2 20% loss
14:12:01.541606 DEBUG TRAINING::DevTrainingAgent08::processEvent 3 30% loss
14:12:01.541607 DEBUG TRAINING::DevTrainingAgent08::processEvent 4 40% loss
14:12:01.541608 DEBUG TRAINING::DevTrainingAgent08::processEvent 5 50% loss
14:12:02.541739 DEBUG EventService Packet Received len: 36
14:12:02.541773 DEBUG ConcreteEventAgentManager Packet Received len: 36
14:12:02.541791 DEBUG TRAINING::DevTrainingAgent08::processEvent line 175
14:12:02.541801 DEBUG TRAINING::DevTrainingAgent08::processEvent 1 50% loss
14:12:02.541806 DEBUG TRAINING::DevTrainingAgent08::processEvent 2 20% loss
14:12:02.541808 DEBUG TRAINING::DevTrainingAgent08::processEvent 3 30% loss
14:12:02.541811 DEBUG TRAINING::DevTrainingAgent08::processEvent 4 20% loss
14:12:02.541813 DEBUG TRAINING::DevTrainingAgent08::processEvent 5 10% loss
14:12:03.541943 DEBUG EventService Packet Received len: 36
14:12:03.541997 DEBUG ConcreteEventAgentManager Packet Received len: 36
14:12:03.542015 DEBUG TRAINING::DevTrainingAgent08::processEvent line 175
14:12:03.542045 DEBUG TRAINING::DevTrainingAgent08::processEvent 1 0% loss
14:12:03.542053 DEBUG TRAINING::DevTrainingAgent08::processEvent 2 0% loss
14:12:03.542055 DEBUG TRAINING::DevTrainingAgent08::processEvent 3 0% loss
14:12:03.542058 DEBUG TRAINING::DevTrainingAgent08::processEvent 4 0% loss
14:12:03.542060 DEBUG TRAINING::DevTrainingAgent08::processEvent 5 0% loss
```

Output 7: EMANE Event Daemon (**emaneeventd**) running the configuration and Agent implementation from Section 9.3.1

Chapter 10

Transport API

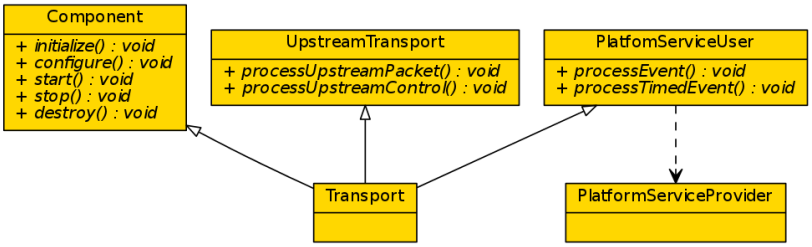

Figure 10.1: `Transport` Class Diagram

Transports are the emulation boundary interfaces that provide the entry and exit points for all data routed through the emulation. A Transport component implementation is a realization of the `EMANE::Transport` class.

10.1 Implementing a Transport

1. Create a class derived from `EMANE::Transport`.

```
4: #include "emane/emanetransport.h"
5:
6: namespace TRAINING
7: {
8:   class DevTrainingTrans07 : public EMANE::Transport
9:   {
10:   public:
11:     DevTrainingTrans07(EMANE::NEMId id,
12:                        EMANE::PlatformServiceProvider *pPlatformService);
        ...
```

2. Fill in the implementation for all virtual methods.

```
47: void TRAINING::DevTrainingTrans07::initialize()
48:   throw(EMANE::InitializeException)

54: void TRAINING::DevTrainingTrans07::configure(const EMANE::ConfigurationItems & items)
55:   throw(EMANE::ConfigureException)

62: void TRAINING::DevTrainingTrans07::start()
```

```
  throw(EMANE::StartException)

void TRAINING::DevTrainingTrans07::postStart()
  throw(EMANE::StartException)

void TRAINING::DevTrainingTrans07::stop()
  throw(EMANE::StopException)

void TRAINING::DevTrainingTrans07::destroy()
  throw()

void TRAINING::DevTrainingTrans07::processUpstreamControl(const EMANE::ControlMessage &)

void TRAINING::DevTrainingTrans07::processUpstreamPacket(EMANE::UpstreamPacket &,
                                                         const EMANE::ControlMessage &)

void TRAINING::DevTrainingTrans07::processEvent(const EMANE::EventId &,
                                                const EMANE::EventObjectState &)

void TRAINING::DevTrainingTrans07::processTimedEvent(ACE_UINT32,
                                                     long,
                                                     const ACE_Time_Value &,
                                                     const void *)
```

3. Define and load the `EMANE::Component` configuration requirements for the component.

```
namespace
{
  const char * MODULE = "TRAINING::DevTrainingTrans07";

  const EMANE::ConfigurationDefinition defs[] =
    {
      // req, default, count, name,          value, type, description */
      {true, false,    1,     "boolvalue",   "off",  0,   "bool value"},
      {true, false,    1,     "u16value",     0,     0,   "unsigned 16 bit value"},
      {true, false,    1,     "u32value",     0,     0,   "unsigned 32 bit value"},
      {true, false,    1,     "stringvalue",  0,     0,   "string value"},
      {0,0,0,0,0,0,0},
    };
}

TRAINING::DevTrainingTrans07::DevTrainingTrans07(EMANE::NEMId id,
                                                 EMANE::PlatformServiceProvider *
    pPlatformService) :
  Transport(id, pPlatformService),
  bBoolValue_(false),
  u16Value_(0),
  u32Value_(0)
{
      ...
  configRequirements_ = EMANE::loadConfigurationRequirements(defs);
}
```

4. Create a new thread to execute the backend code necessary to receive and process the input appropriate for the transport implementation. Use the `sendDownstreamPacket` and `sendDownstreamControl` methods to send packets and control messages to the transport's respective NEM[1].

```
  // start pcap read thread
  EMANEUtils::spawn(*this, &RawTransport::readDevice, &threadRead_);
  ...
ACE_THR_FUNC_RETURN RawTransport::readDevice()
{
```

[1]src: emane/transports/raw/rawtransport.cc

```
506:   const ACE_UINT8* buf;
507:
508:   struct pcap_pkthdr *pcap_hdr;
509:
510:   int iPcapResult;
511:
512:   while(1)
513:     {
514:       // get frame, blocks here
515:       iPcapResult = pcap_next_ex(pPcapHandle_, &pcap_hdr, &buf);
516:
           ...
567:                   // create downstream packet with packet info
568:                   EMANE::DownstreamPacket pkt(EMANE::PacketInfo (id_, nemDestination, dscp),
     buf, pcap_hdr->caplen);
569:
570:                   // send to downstream transport
571:                   sendDownstreamPacket(pkt);
572:
573:                   // drain the bit pool
574:                   const size_t sizePending = pBitPool_->get(pcap_hdr->caplen * 8);
575:
           ...
585:      }
586:
587:      return (ACE_THR_FUNC_RETURN) 0;
588: }
```

5. Expose the new transport to the EMANE infrastructure using the `DECLARE_TRANSPORT` macro.

```
207: DECLARE_TRANSPORT(TRAINING::DevTrainingTrans07);
```

6. Create a transport definition XML file containing the configuration parameters and library name for the new Transport Layer implementation.

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <!DOCTYPE transport SYSTEM "file:///usr/share/emane/dtd/transport.dtd">
3 <transport name="Devel Training Transport" library="devtrainingtrans07">
4   <param name="boolvalue" value="on"/>
5   <param name="u16value" value="65521"/>
6   <param name="u32value" value="6371000"/>
7   <param name="stringvalue" value="emane"/>
8 </transport>
```

7. Create an NEM XML definition file using the new transport definition.

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <!DOCTYPE nem SYSTEM "file:///usr/share/emane/dtd/nem.dtd">
3 <nem name="Devel Training NEM">
4   <mac definition="devtrainingmac.xml"/>
5   <shim definition="devtrainingshim.xml"/>
6   <phy definition="devtrainingphy.xml"/>
7   <transport definition="devtrainingtrans.xml"/>
8 </nem>
```

```
# emanetransportd transportdaemon1.xml -l 4
14:16:43.961589 DEBUG PHYI 001 TRAINING::DevTrainingTrans07::DevTrainingTrans07 line 36
14:16:43.961742 DEBUG PHYI 001 TRAINING::DevTrainingTrans07::initialize line 50
14:16:43.961748 DEBUG PHYI 001 TRAINING::DevTrainingTrans07::configure line 57
14:16:44.009516 DEBUG NEM 001 ConcreteTransportAdapter::start platformendpoint: 127.0.0.1
14:16:44.052168 DEBUG NEM 001: ConcreteTransportAdapter::start transportendpoint: 127.0.0.1
14:16:44.052216 DEBUG PHYI 001 TRAINING::DevTrainingTrans07::start line 65
14:16:44.052230 DEBUG PHYI 001 TRAINING::DevTrainingTrans07::start boolvalue = on
14:16:44.052235 DEBUG PHYI 001 TRAINING::DevTrainingTrans07::start stringvalue = emane
14:16:44.052241 DEBUG PHYI 001 TRAINING::DevTrainingTrans07::start u16value = 65521
14:16:44.052245 DEBUG PHYI 001 TRAINING::DevTrainingTrans07::start u32value = 6371000
14:16:44.052331 DEBUG PHYI 001 TRAINING::DevTrainingTrans07::postStart line 161
14:16:44.052464 DEBUG NEM 001 ConcreteTransportAdapter::processNetworkMessage
```

Output 8: EMANE Transport Daemon (**emanetransportd**) running the configuration and Transport implementation from Section 10.1

Part III

Universal PHY Layer

Chapter 11

Interfacing with the Universal PHY Layer

The Universal PHY Layer provides a common PHY implementation for the various MAC Layers supplied as part of the standard EMANE distribution. Its use is not mandatory but is encouraged for authors of other proprietary and non-proprietary MAC implementations as it provides a set of core functionality required by most wireless Network Emulation Modules. The key functionality includes the following:

- Pathloss Calculation
- Receive Power Calculation
- Directional Sector Antenna Support
- Noise Processing
- MAC-PHY Control Messaging

11.1 Model Features

11.1.1 Pathloss Calculation

Pathloss within the Universal PHY Layer is based on location or pathloss events. Pathloss is dynamically calculated based on location events when the *pathlossmode* configuration parameter is set to either *2ray* or *freespace*, which selects between the 2-ray flat earth or freespace propagation models, respectively. Pathloss can be provided in realtime based on external propagation calculations using pathloss events. The *pathlossmode* configuration parameter should be set to *pathloss* in order to process inbound pathloss events.

11.1.2 Receive Power Calculation

For each received packet, the Universal PHY Layer computes the receiver power associated with that packet using the following calculation:

$$rxPower = txPower + txAntennaGain + rxAntennaGain - pathloss$$

Where,

txPower	Packet Common PHY Header transmitter power (See Section 7.2)
txAntennaGain	Packet Common PHY Header transmitter antenna gain (See Section 7.2)
rxAntennaGain	Configuration parameter *antennagain*
pathloss	Pathloss between transmitter and receiver determined based on *pathlossmode* configuration parameter

If the *rxPower* is less than the *rxSensitivity*, the packet is silently discarded.

$$rxSensitivity = -174 + noiseFigure + 10\log(bandWidth)$$

Where,

bandWidth	Configuration parameter *bandwidth*
noiseFigure	Configuration parameter *noisefigure*

11.1.3 Directional Sector Antenna Support

The Universal PHY Layer provides support for directional antenna, if required. This support includes the ability to statically configure the directional antenna parameters (pointing and profile) as well as the ability to accept parameters from the MAC Layer via a control message on a per packet basis and via antenna pointing events. The Universal PHY Layer utilizes location events and Tx and Rx antenna information to determine if two nodes are visible. Current directional antenna support is based on sector antennas, where a sector is defined by antenna azimuth and elevation beam width. Any intersection between the transmitting and receiving antenna will apply full gain. Figure 11.1 illustrates a transmitter and receiver with and without beam overlap.

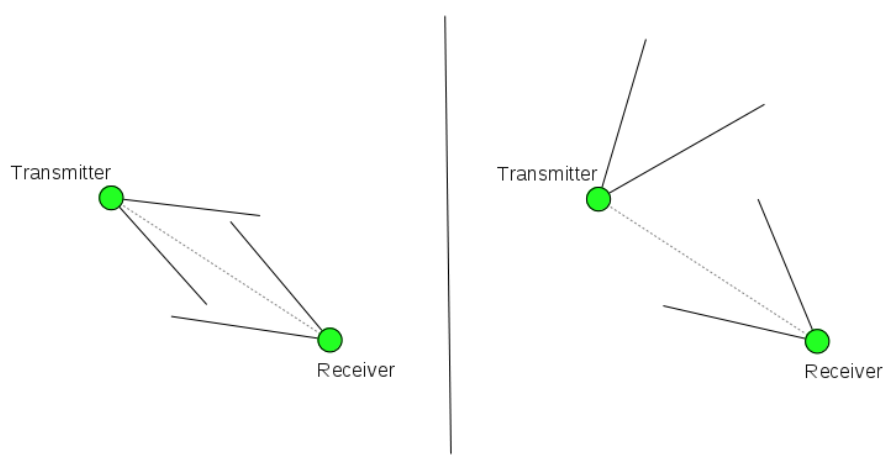

Figure 11.1: Transmitter and receiver antenna beam overlap *(Left)*. Transmitter and receiver no antenna beam overlap *(Right)*.

Figure 11.2 visualizes the four configuration items that are necessary to define the pointing and profile characteristics of a directional antenna: *antennaazimuth*, *antennaelevation*, *antennaazimuthbeamwidth* and *antennaelevationbeamwidth*.

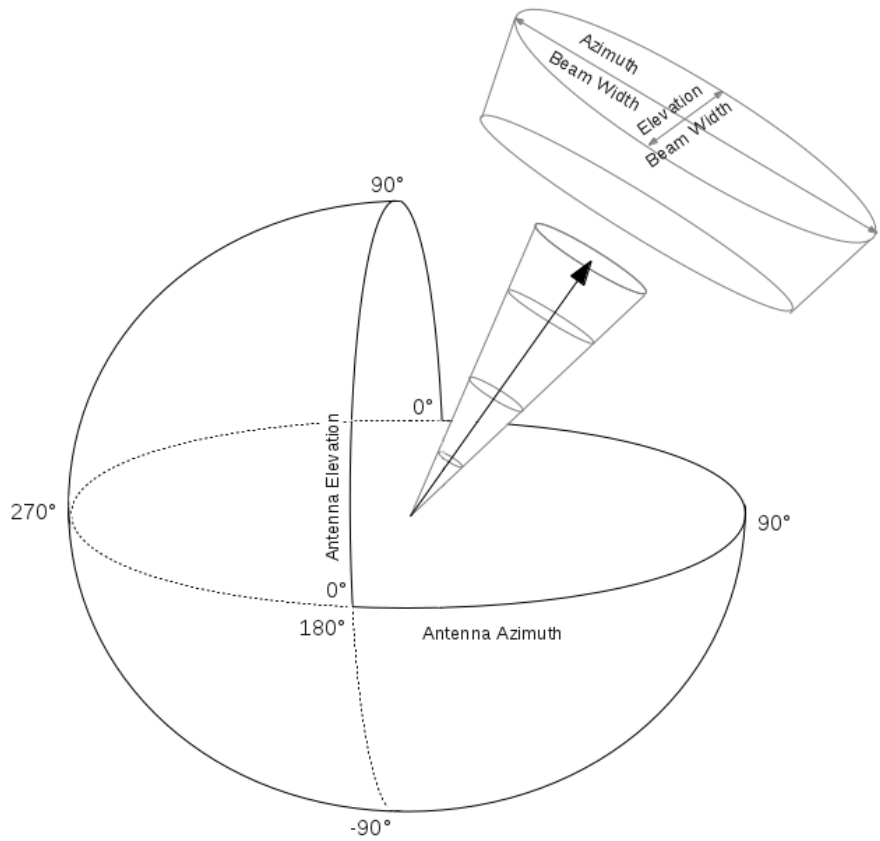

Figure 11.2: Universal PHY Layer Directional Antenna configuration.

11.1.4 Noise Processing

The Universal PHY Layer provides the ability to assess the impact of intentional and unintentional noise sources within the emulation by adjusting the noise floor. This is achieved by summing the energy of interferers within the appropriate frequency of interest over a given time interval and adjusting the noise floor accordingly when a valid packet is received. The Universal PHY Layer only computes interference for out-of-band packets. An out-of-band packet is one which is not from the same emulated waveform. The Universal PHY Layer determines waveform type by comparing the PHY Registration Id, center frequency, and Universal PHY Layer subid of each packet. It is the responsibility of the MAC Layer implementation to account for in-band interference.

11.1.5 MAC-PHY Control Messaging

The Universal PHY Layer provides a control API on a per packet basis for every transmit (Tx) packet received from the MAC Layer for over-the-air transmission and every received (Rx) over-the-air packet sent to the MAC Layer for processing. The Tx Control API provides the MAC Layer with the ability to override default PHY Layer configuration for transmit power, transmit frequency and antenna pointing as required. The Universal PHY Layer utilizes the data from the Tx Control message to populate the Common PHY Header (See Section 7.2). The Rx Control API provides the MAC Layer with the appropriate receive information (receive power, noise floor, message duration and receive frequency) to perform functions such as SINR based packet completion calculation, in-band collision detection and channel access protocol.

11.2 Control Antenna Message

The `UniversalPhyControlAntennaMessage` is used to set the Universal PHY Layer's antenna direction.

Message constraints:

- Control message only valid when sent from MAC Layer to Universal PHY Layer.
- Control message only valid when sent via the MAC Layer's `sendDownstreamControl()` method.
- Control message only processed when the Universal PHY Layer *antennatype* configuration parameter is set to *unidirectional*.
- Antenna direction settings persist until updated by another control message or event.

Listing 11.1 shows the `UniversalPhyControlAntennaMessage` interface.

```
namespace EMANE
{
  class UniversalPhyControlAntennaMessage
  {
    UniversalPhyControlAntennaMessage();

    UniversalPhyControlAntennaMessage(float fAntennaAzBeamWidthDegrees,
                                      float fAntennaElBeamWidthDegrees,
                                      float fElevationDegrees,
                                      float fAzimuthDegrees,
                                      float fAntennaGaindBi);

    float getAntennaAzBeamWidthDegrees() const;

    void setAntennaAzBeamWidthDegrees(float fAntennaAzBeamWidthDegrees);

    float getAntennaElBeamWidthDegrees() const;

    void setAntennaElBeamWidthDegrees(float fAntennaElBeamWidthDegrees);

    float getAntennaElevationDegrees() const ;

    void setAntennaElevationDegrees(float fAntennaElevationDegrees);

    float getAntennaAzimuthDegrees() const;

    void setAntennaAzimuthDegrees(float fAntennaAzimuthDegrees);

    float getAntennaGaindBi() const;

    void setAntennaGaindBi(float fAntennaGaindBi);

    void toNetworkByteOrder();

    void toHostByteOrder();
  };
}
```

Listing 11.1: `UniversalPhyControlAntennaMessage` Interface

11.3 Control Send Message

The `UniversalPhyControlSendMessage` accompanies all MAC Layer transmitted downstream packets. This control message allows the MAC Layer to set power, frequency and antenna direction information on a per packet basis.

Message constraints:

- Control message only valid when sent from MAC Layer to Universal PHY Layer.

- Control message only valid when sent via the MAC Layer's `sendDownstreamPacket()` method.
- If a control message does not accompany the downstream packet the Universal PHY Layer will use values from its configuration for the respective transmit settings.
- Any item not set in the control message will result in the Universal PHY Layer using values from its configuration for that particular transmit setting(s).
- Antenna direction settings only processed when the Universal PHY Layer *antennatype* configuration parameter is set to *unidirectional*.
- Antenna direction settings persist until updated by another control message or event.

Listing 11.2 shows the `UniversalPhyControlSendMessage` interface.

```
namespace EMANE
{
  class UniversalPhyControlSendMessage
  {
    public:

      UniversalPhyControlSendMessage();

      UniversalPhyControlSendMessage(float fTxPowerdBm,
                                     const ACE_Time_Value & tvDuration,
                                     ACE_UINT32 u32FrequencyKHz,
                                     float fAntennaAzBeamWidthDegrees,
                                     float fAntennaElBeamWidthDegrees,
                                     float fAzimuthDegrees,
                                     float fElevationDegrees);


      float getTxPowerdBm() const;

      void setTxPowerdBm(float dTxPowerdBm);

      ACE_Time_Value getDuration() const;

      void setDuration(const ACE_Time_Value & tvDuration);

      ACE_UINT32 getFrequencyKHz() const;

      void setFrequencyKHz(ACE_UINT32 u32FrequencyKHz);

      float getAntennaAzBeamWidthDegrees() const;

      void setAntennaAzBeamWidthDegrees(float fAntennaAzBeamWidthDegrees);

      float getAntennaElBeamWidthDegrees() const;

      void setAntennaElBeamWidthDegrees(float fAntennaElBeamWidthDegrees);

      float getAntennaElevationDegrees() const;

      void setAntennaElevationDegrees(float fAntennaElevationDegrees);

      float getAntennaAzimuthDegrees() const;

      void setAntennaAzimuthDegrees(float fAntennaAzimuthDegrees);

      void toNetworkByteOrder();

      void toHostByteOrder();
  };
}
```

Listing 11.2: `EMANE::UniversalPhyControlSendMessage` Interface

11.4 Control Receive Message

The `UniversalPhyControlReceiveMessage` accompanies all upstream packets sent by the Universal PHY Layer. This control message contains the receive power, noise floor, frequency, propagation delay and timing information on a per packet basis.

Message constraints:

- Control message only valid when sent from Universal PHY Layer to the MAC Layer.
- Control message only valid when received during the MAC Layer's `processUpstreamPacket()` method.

Listing 11.3 shows the `UniversalPhyControlReceiveMessage` interface.

```
namespace EMANE
{
  class UniversalPhyControlRecvMessage
  {
    public:
      UniversalPhyControlRecvMessage();

      UniversalPhyControlRecvMessage(float fRxPowerdBm,
                                     float fNoiseFloordBm,
                                     const ACE_Time_Value &tvDuration,
                                     const ACE_Time_Value &tvTxTime,
                                     const ACE_Time_Value &tvPropDelay,
                                     ACE_UINT32 u32FrequencyKHz);

      ACE_UINT16 getFromNEM() const;

      float getRxPowerdBm() const;

      float getNoiseFloordBm() const;

      ACE_Time_Value getDuration() const;

      ACE_Time_Value getTxTime() const;

      ACE_UINT32 getFrequencyKHz() const;

      ACE_Time_Value getPropagationDelay() const;

     void toNetworkByteOrder();

     void toHostByteOrder();
    };
}
```

Listing 11.3: `EMANE::UniversalPhyControlReceiveMessage` Interface

11.5 Antenna Direction Event

The Universal PHY Layer uses the `AntennaDirectionEvent` as an additional mechanism to update antenna direction.

Event constraints:

- Event is only processed when the Universal PHY Layer *antennatype* configuration parameter is set to *unidirectional*.
- Antenna direction settings persist until updated by another control message or event.

Listing 11.4 shows the `AntennaDirectionEvent` interface.

```
class AntennaDirectionEvent : public EMANE::Event
{
public:

  struct AntennaDirectionEntry
    {
      ACE_UINT16 u16Node_;
      float  fElevationDegrees_;
      float  fAzimuthDegrees_;
      float  fBeamwidthElevationDegrees_;
      float  fBeamwidthAzimuthDegrees_;
    } __attribute__((packed));

  AntennaDirectionEvent(AntennaDirectionEntry * pEntries, ACE_UINT16 u16NumberOfEntries);

  AntennaDirectionEvent(const EMANE::EventObjectState)
    throw(EMANE::EventObjectStateException);

  ~AntennaDirectionEvent();

  EMANE::EventObjectState getObjectState() const;

  const AntennaDirectionEntry * getEntries() const;

  bool findEntry(EMANE::NEMId id, AntennaDirectionEntry & entry) const;

  ACE_UINT16 getNumberOfEntries() const;
};
```

Listing 11.4: `AntennaDirectionEvent` Interface

Part IV

EMANE Libraries

Chapter 12

libemaneeventservice

`libemaneeventservice` allows publication and subscription of EMANE events using a C library API.

12.1 Configuration

Configuration for the Event Service is searched for in three locations prior to using default configuration in the following order:

1. If the environment variable `LIBEMANEEVENTSERVICECONFIG` exists, it will be used. This variable may be set to a configuration file name.
2. If `$HOME/.libemaneeventservice.xml` exists, it will be used.
3. If `/etc/libemaneeventservice.xml` exists, it will be used.
4. The default values are: group 224.1.2.8, port 45703, multicast loop enabled (1), TTL 32. No default multicast device is specified. The kernel routing table is used.

```
<?xml version='1.0' standalone='yes'?>
<emaneeventmsgsvc>
  <group>224.1.2.8</group>
  <port>45703</port>
  <device>lo</device>
  <mcloop>1</mcloop>
  <ttl>32</ttl>
</emaneeventmsgsvc>
```

Listing 12.1: `libemaneeventservice` Configuration File

12.2 libemaneeventservice API

12.2.1 `int emaneEventServiceInitialize(emaneEventService * pService)`

The `emaneEventServiceInitialize` function initializes an `emaneEventService` instance. Returns `0` on success or `-1` on error.

Parameter Description:

`emaneEventService *`	`pService`	Event Service pointer

12.2.2 `int emaneEventServiceDestroy(emaneEventService * pService)`

The `emaneEventServiceDestroy` function destroys an `emaneEventService` instance. Returns `0` on success or `-1` on error.

Parameter Description:

`emaneEventService *`	`pService`	Event Service pointer

12.2.3 `int emaneEventServiceSubscribe(emaneEventService * pService, emaneEventId id, emaneEventServiceHandler handler, void * arg)`

The `emaneEventServiceSubscribe` function subscribes to EMANE event id. Returns `0` on success or `-1` on error.

Parameter Description:

`emaneEventService *`	`pService`	Event Service pointer
`emaneEventId *`	`id`	Event id
`emaneEventServiceHandler`	`handler`	Event callback function
`void *`	`arg`	Event callback argument

12.2.4 `int emaneEventServiceUnsubscribe(emaneEventService * pService, emaneEventId id, void ** p)`

The `emaneEventServiceUnsubscribe` function unsubscribes the event id from callback processing. Returns `0` on success or `-1` on error.

Parameter Description:

`emaneEventService *`	`pService`	Event Service pointer
`emaneEventId *`	`id`	Event id
`void **`	`p`	Address of pointer to hold the `arg` reference or `NULL`

12.2.5 `int emaneEventServicePublish(emaneEventService * pService, emaneEventId id, emanePlatformId platform, emaneNEMId nem, emaneComponentId component, const void * buf, size_t count)`

The `emaneEventServicePublish` function publishes events for a specified platform nem component. Returns `0` on success or `-1` on error.

Parameter Description:

`emaneEventService *`	`pService`	Event Service pointer
`emaneEventId *`	`id`	Event id
`emanePlatformId`	`platform`	Platform id of destination or `EMANE_PLATFORMID_ANY` for all
`emaneNEMId`	`nem`	NEM id of destination or `EMANE_NEM_ANY` for all
`emaneComponentId`	`component`	Component id of destination or `EMANE_COMPONENT_ANY` for all
`const void*`	`buf`	Buffer containing event data
`size_t`	`count`	Size of the event data

12.2.6 `int emaneEventServiceLoop(emaneEventService * pService)`

The `emaneEventServiceLoop` function processes incoming events, calling the appropriate handlers when subscribed events are received. Returns `0` on success or `-1` on error.

Parameter Description:

`emaneEventService *`	`pService`	Event Service pointer

12.2.7 `int emaneEventServiceLoopWithDefault(emaneEventService * pService, emaneEventServiceHandler handler, void * arg)`

The `emaneEventServiceLoopWithDefault` function processes incoming events, calling the appropriate handlers when subscribed events are received. A default callback handler will be invoked when an unsubscribed event is received. Returns `0` on success or `-1` on error.

Parameter Description:

`emaneEventService *`	`pService`	Event Service pointer
`emaneEventServiceHandler`	`handler`	Default event callback function
`void *`	`arg`	Event callback argument

12.2.8 `int emaneEventServiceNextEvent(emaneEventService * pService, emaneEventId * pEvent, emanePlatformId * pPlatform, emaneNEMId * pNEM, emaneComponentId * pComponent, void * pBuf, size_t length)`

The `emaneEventServiceNextEvent` function blocks waiting to return the next received event. Returns the number of bytes written or `-1` on error.

Parameter Description:

`emaneEventService *`	`pService`	Event Service pointer
`emaneEventId *`	`pEvent`	Pointer to set received event id
`emanePlatformId *`	`pPlatform`	Pointer to set received Platform id
`emaneNEMIdId *`	`pNEM`	Pointer to set received NEM id
`emaneComponentId *`	`pComponent`	Pointer to set received Component id
`void *`	`pBuf`	Buffer to hold event data
`size_t`	`length`	Size of `pBuf`

12.2.9 `int emaneEventServiceBreakloop(emaneEventService * pService)`

The `emaneEventServiceBreakloop` function will cause the `emaneEventServiceLoop`, `emaneEventServiceLoopWithDefault` and `emaneEventServiceNextEvent` functions to exit. Returns `0` on success or `-1` on error.

12.2.10 `const char * emaneEventServiceError(emaneEventService * pService)`

The `emaneEventServiceError` function returns a string description of the current `emaneEventService` error.

Parameter Description:

`emaneEventService *`	`pService`	Event Service pointer

12.2.11 Sample Usage

Listing 12.2 shows sample `libemaneeventservice` usage.

```
#include <libemaneeventservice/emaneeventservice.h>
#include <libemaneeventservice/emaneeventpathloss.h>

int handlePathlossEvent(emaneEventId event,
   emanePlatformId platform,
   emaneNEMId nem,
   emaneComponentId component,
   const void * buf,
   size_t length,
   void * p);

...

emaneEventService message_service;


if(emaneEventServiceInitialize(&message_service) < 0)
  {
    fprintf(stderr,
     "emaneEventServiceInitialize: %s\n",
     emaneEventServiceError(&message_service));

    return EXIT_FAILURE;
  }

if(emaneEventServiceSubscribe(&message_service,
         EMANE_EVENT_PATHLOSS,
         &handlePathlossEvent,
         &pathlossStat) < 0)
  {
    fprintf(stderr,
     "emaneEventServiceSubscribe: %s\n",
     emaneEventServiceError(&message_service));

    return EXIT_FAILURE;
  }


if(emaneEventServiceLoop(&message_service) < 0)
  {
    fprintf(stderr,
     "emaneEventServiceLoop: %s\n",
     emaneEventServiceError(&message_service));

    return EXIT_FAILURE;
  }

if(emaneEventServiceUnsubscribe(&message_service,
    EMANE_EVENT_PATHLOSS,0) < 0)
  {
    fprintf(stderr,
     "emaneEventServiceUnsubscribe: %s [%hu]\n",
     emaneEventServiceError(&message_service),
     EMANE_EVENT_PATHLOSS);

    return EXIT_FAILURE;
  }


if(emaneEventServiceDestroy(&message_service) < 0)
  {
    fprintf(stderr,
     "emaneEventServiceDestroy: %s\n",
     emaneEventServiceError(&message_service));

    return EXIT_FAILURE;
  }

...

if(emaneEventServicePublish(&eventService,
       EMANE_EVENT_PATHLOSS,
```

```
        EMANE_PLATFORMID_ANY,
        1,
        EMANE_COMPONENTID_PHYI,
        emaneEventPathlossExport(&ePathloss),
        emaneEventPathlossExportSizeBytes(&ePathloss)) < 0)
    {
      fprintf(stderr,
       "emaneEventServicePublish: %s\n",
       emaneEventServiceError(&eventService));

      return EXIT_FAILURE;
    }

  ...
```

Listing 12.2: `libemaneeventservice` Sample Usage

12.3 libemaneeventlocation API

libemaneeventlocation creates and parses EMANE Location events. Location events contain the GPS location of one or more NEMs contained in an emulation deployment.

12.3.1 `int emaneEventLocationInitialize(emaneEventLocation * pEvent, int iNodeCount)`

The `emaneEventLocationInitialize` function initializes an emaneEventLocation instance. Returns `0` on success or `-1` on error.

Parameter Description:

`emaneEventLocation *`	`pEvent`	Event instance pointer
`int`	`iNodeCount`	Number of node entries contained in event

12.3.2 `int emaneEventLocationInitializeFromImport(emaneEventLocation * pEvent, const void * buf, size_t length)`

The `emaneEventLocationInitializeFromImport` function initializes an `emaneEventLocation` instance from event data. Returns `0` on success or `-1` on error.

Parameter Description:

`emaneEventLocation *`	`pEvent`	Event instance pointer
`const void *`	`buf`	Buffer containing event data
`size_t`	`length`	Length of event data in bytes

12.3.3 `int emaneEventLocationDestroy(emaneEventLocation * pEvent)`

The `emaneEventLocationDestroy` function destroys an `emaneEventLocation` instance. Returns `0` on success or `-1` on error.

Parameter Description:

`emaneEventLocation *`	`pEvent`	Event instance pointer

12.3.4 `int emaneEventLocationSetLocation(emaneEventLocation * pEvent, int iIndex, u_int16_t u16NEMId, double dLatitude, double dLongitude, int32_t i32AltitudeMeters)`

The `emaneEventLocationSetLocation` function sets the location information for a NEM located at the `iIndex` position of the location entry data. Returns `0` on success or `-1` on error.

Parameter Description:

`emaneEventLocation *`	`pEvent`	Event instance pointer
`int`	`iIndex`	Entry index
`u_int16_t`	`u16NEMId`	NEM id
`double`	`dLatitude`	Latitude in degrees
`double`	`dLongitude`	Longitude in degrees
`int_32_t`	`i32AltitudeMeters`	Altitude in meters

12.3.5 `int emaneEventLocationGetLocation(emaneEventLocation * pEvent, int iIndex, u_int16_t * pu16NEMId, double * pdLatitude, double * dLongitude, int32_t * pi32AltitudeMeters)`

The `emaneEventLocationGetLocation` function gets the location information for a NEM located at the `iIndex` position of the location entry data. Returns `0` on success or `-1` on error.

Parameter Description:

`emaneEventLocation *`	`pEvent`	Event instance pointer
`int`	`iIndex`	Entry index
`u_int16_t *`	`pu16NEMId`	Pointer to store NEM id
`double *`	`pdLatitude`	Pointer to store Latitude in degrees
`double *`	`pdLongitude`	Pointer to store Longitude in degrees
`int32_t *`	`pi32AltitudeMeters`	Pointer to store altitude

12.3.6 `int emaneEventLocationNodeCount(emaneEventLocation * pEvent, int * piNodeCount)`

The `emaneEventLocationNodeCount` function gets the total number of NEM entries contained in the location message. Returns `0` on success or `-1` on error.

Parameter Description:

`emaneEventLocation *`	`pEvent`	Event instance pointer
`int *`	`piNodeCount`	Pointer to store node count

12.3.7 `int emaneEventLocationExportSizeBytes(emaneEventLocation * pEvent)`

The `emaneEventLocationExportSizeBytes` returns the length in bytes of the data returned by `emaneEventLocationExport` or `-1` on error.

Parameter Description:

`emaneEventLocation *`	`pEvent`	Event instance pointer

12.3.8 `const void * emaneEventLocationExport(emaneEventLocation * pEvent)`

The `emaneEventLocationExport` function returns the location event data in a format suitable for publication. Returns `0` on success or `NULL` on error.

Parameter Description:

`emaneEventLocation *`	`pEvent`	Event instance pointer

12.3.9 `const char * emaneEventLocationError(emaneEventLocation * pEvent)`

The `emaneEventLocationError` function returns a string description of the current `emaneEventLocation` error.

Parameter Description:

`emaneEventLocation *`	`pEvent`	Event instance pointer

12.4 libemaneeventpathloss API

libemaneeventpathloss creates and parses EMANE Pathloss events. Pathloss events contain RF Pathloss in dB from one or more NEMs to a single destination NEM.

12.4.1 `int emaneEventPathlossInitialize(emaneEventPathloss * pEvent, int iNodeCount)`

The `emaneEventPathlossInitialize` function initializes an emaneEventPathloss instance. Returns `0` on success or `-1` on error.

Parameter Description:

`emaneEventPathloss *`	`pEvent`	Event instance pointer
`int`	`iNodeCount`	Number of node entries contained in event

12.4.2 `int emaneEventPathlossInitializeFromImport(emaneEventPathloss * pEvent, const void * buf, size_t length)`

The `emaneEventPathlossInitializeFromImport` function initializes an `emaneEventPathloss` instance from event data. Returns `0` on success or `-1` on error.

Parameter Description:

`emaneEventPathloss *`	`pEvent`	Event instance pointer
`const void *`	`buf`	Buffer containing event data
`size_t`	`length`	Length of event data in bytes

12.4.3 `int emaneEventPathlossDestroy(emaneEventPathloss * pEvent)`

The `emaneEventPathlossDestroy` function destroys an `emaneEventPathloss` instance. Returns `0` on success or `-1` on error.

Parameter Description:

`emaneEventPathloss *`	`pEvent`	Event instance pointer

12.4.4 `int emaneEventPathlossSetPathloss(emaneEventPathloss * pEvent, int iIndex, u_int16_t u16TxNEMId,float fPathloss,float fRevPathloss)`

The `emaneEventPathlossSetPathloss` function sets the pathloss information for the transmitting NEM located at the `iIndex` position of the pathloss entry data. Returns `0` on success or `-1` on error.

Parameter Description:

`emaneEventPathloss *`	`pEvent`	Event instance pointer
`int`	`iIndex`	Entry index
`u_int16_t`	`u16TxNEMId`	NEM id of transmitter
`float`	`fPathloss`	Pathloss in dB
`float`	`fRevPathloss`	Reverse pathloss in dB

12.4.5 `int emaneEventPathlossGetPathloss(emaneEventPathloss * pEvent, int iIndex, u_int16_t * pu16TxNEMId, float * pfPathloss, float * pfRevPathloss)`

The `emaneEventPathlossGetPathloss` function gets the pathloss information for the transmitting NEM located at the `iIndex` position of the pathloss entry data. Returns `0` on success or `-1` on error.

Parameter Description:

`emaneEventPathloss *`	`pEvent`	Event instance pointer
`int`	`iIndex`	Entry index
`u_int16_t *`	`pu16TxNEMId`	Pointer to store NEM id of transmitter
`float *`	`pfPathloss`	Pointer to store pathloss in dB
`float*`	`pfRevPathloss`	Pointer to store reverse pathloss in dB

12.4.6 `int emaneEventPathlossNodeCount(emaneEventPathloss * pEvent, int * piNodeCount)`

The `emaneEventPathlossNodeCount` function gets the total number of NEM entries contained in the pathloss message. Returns `0` on success or `-1` on error.

Parameter Description:

`emaneEventPathloss *`	`pEvent`	Event instance pointer
`int *`	`piNodeCount`	Pointer to store node count

12.4.7 `int emaneEventPathlossExportSizeBytes(emaneEventPathloss * pEvent)`

The `emaneEventPathlossExportSizeBytes` returns the length in bytes of the data returned by `emaneEventPathlossExport` or `-1` on error.

Parameter Description:

`emaneEventPathloss *`	`pEvent`	Event instance pointer

12.4.8 `const void * emaneEventPathlossExport(emaneEventPathloss * pEvent)`

The `emaneEventPathlossExport` function returns the pathloss event data in a format suitable for publication. Returns `0` on success or `NULL` on error.

Parameter Description:

`emaneEventPathloss *`	`pEvent`	Event instance pointer

12.4.9 `const char * emaneEventPathlossError(emaneEventPathloss * pEvent)`

The `emaneEventPathlossError` function returns a string description of the current `emaneEventPathloss` error.

Parameter Description:

`emaneEventPathloss *`	`pEvent`	Event instance pointer

12.5 libemaneeventantennadirection API

libemaneeventantennadirection creates and parses EMANE Antenna Direction events. Antenna direction events contain the antenna direction of one or more NEMs in an emulation deployment.

12.5.1 `int emaneEventAntennaDirectionInitialize(emaneEventAntennaDirection * pEvent, int iNodeCount)`

The `emaneEventAntennaDirectionInitialize` function initializes an emaneEventAntennaDirection instance. Returns `0` on success or `-1` on error.

Parameter Description:

`emaneEventAntennaDirection *`	`pEvent`	Event instance pointer
`int`	`iNodeCount`	Number of node entries contained in event

12.5.2 `int emaneEventAntennaDirectionInitializeFromImport(emaneEventAntennaDirection * pEvent, const void * buf, size_t length)`

The `emaneEventAntennaDirectionInitializeFromImport` function initializes an `emaneEventAntennaDirection` instance from event data. Returns `0` on success or `-1` on error.

Parameter Description:

`emaneEventAntennaDirection *`	`pEvent`	Event instance pointer
`const void *`	`buf`	Buffer containing event data
`size_t`	`length`	Length of event data in bytes

12.5.3 `int emaneEventAntennaDirectionDestroy(emaneEventAntennaDirection * pEvent)`

The `emaneEventAntennaDirectionDestroy` function destroys an `emaneEventAntennaDirection` instance. Returns `0` on success or `-1` on error.

Parameter Description:

`emaneEventAntennaDirection *`	`pEvent`	Event instance pointer

12.5.4 `int emaneEventAntennaDirectionSetAntennaDirection(emaneEventAntennaDirection * pEvent, int iIndex, u_int16_t u16NEMId, float fElevationDegrees, float fAzimuthDegrees, float fBeamwidthElevationDegrees, float fBeamwidthAzimuthDegrees)`

The `emaneEventAntennaDirectionSetAntennaDirection` function sets the antenna direction information for the NEM located at the `iIndex` position of the antenna direction entry data. Returns `0` on success or `-1` on error.

Parameter Description:

`emaneEventAntennaDirection *`	`pEvent`	Event instance pointer
`int`	`iIndex`	Entry index
`u_int16_t`	`u16NEMId`	NEM id
`float`	`fElevationDegrees`	Elevation in degrees
`float`	`fAzimuthDegrees`	Azimuth in degrees
`float`	`fBeamwidthElevationDegrees`	Beamwidth elevation in degrees
`float`	`fBeamwidthAzimuthDegrees`	Beamwidth azimuth in degrees

12.5.5 `int emaneEventAntennaDirectionGetAntennaDirection(emaneEventAntennaDirection * pEvent, int iIndex, u_int16_t *pu16NEMId, float * pfElevationDegrees, float * pfAzimuthDegrees, float * pfBeamwidthElevationDegrees, float * pfBeamwidthAzimuthDegrees)`

The `emaneEventAntennaDirectionGetAntennaDirection` function gets the antenna direction information for the NEM located at the `iIndex` position of the antenna direction entry data. Returns `0` on success or `-1` on error.

Parameter Description:

`emaneEventAntennaDirection *`	`pEvent`	Event instance pointer
`int`	`iIndex`	Entry index
`u_int16_t *`	`u16NEMId`	NEM id
`float *`	`fElevationDegrees`	Pointer to store elevation in degrees
`float *`	`fAzimuthDegrees`	Pointer to store azimuth in degrees
`float *`	`fBeamwidthElevationDegrees`	Pointer to store beamwidth elevation in deg
`float *`	`fBeamwidthAzimuthDegrees`	Pointer to store beamwidth azimuth in deg

12.5.6 `int emaneEventAntennaDirectionNodeCount(emaneEventAntennaDirection * pEvent, int * piNodeCount)`

The `emaneEventAntennaDirectionNodeCount` function gets the total number of NEM entries contained in the pathloss message. Returns `0` on success or `-1` on error.

Parameter Description:

`emaneEventAntennaDirection *`	`pEvent`	Event instance pointer
`int *`	`piNodeCount`	Pointer to store node count

12.5.7 `int emaneEventAntennaDirectionExportSizeBytes(emaneEventAntennaDirection * pEvent)`

The `emaneEventAntennaDirectionExportSizeBytes` returns the length in bytes of the data returned by `emaneEventAntennaDirectionExport` or `-1` on error.

Parameter Description:

`emaneEventAntennaDirection * pEvent`	Event instance pointer

12.5.8 `const void * emaneEventAntennaDirectionExport(emaneEventAntennaDirection * pEvent)`

The `emaneEventAntennaDirectionExport` function returns the antenna direction event data in a format suitable for publication. Returns `0` on success or `NULL` on error.

Parameter Description:

`emaneEventAntennaDirection * pEvent`	Event instance pointer

12.5.9 `const char * emaneEventAntennaDirectionError(emaneEventAntennaDirection * pEvent)`

The `emaneEventAntennaDirectionError` function returns a string description of the current `emaneEventAntennaDirection` error.

Parameter Description:

`emaneEventAntennaDirection * pEvent`	Event instance pointer

Chapter 13

libemane

`libemane` exposes the builder classes used by the **emane**, **emanetransportd**, **emaneeventservice** and **emaneeventd** applications to construct and manage their internal components. With this library, developers can configure and run EMANE application containers programmatically from their own applications. Additionally, the `EMANE::Logger` class permits logging configuration.

This chapter describes the five classes that comprise the `libemane` interface and provides example usage: `EMANE::Logger`, `EMANE::NEMBuilder`, `EMANE::TransportBuilder`, `EMANE::EventGeneratorBuilder`, and `EMANE::EventAgentBuilder`.

The four `libemane` builder classes follow a similar design pattern as the EMANE builder classes discussed briefly in Section 2.2. Each provides methods for building a fundamental EMANE component: NEM layer, transport, event generator and event agent, and other methods for inserting these components into compatible managers. The managers provide methods to transition their contained components through the EMANE component lifecycle described in Chapter 3.

13.1 `EMANE::Logger`

The `EMANE::Logger` class provides a method to set the application log level and other methods to direct log output to one of the following destinations: a log file, the system logger or an ACE Log Server. The logger supports runtime control with commands received through a control port.

13.1.1 `void setLogLevel(const LogLevel level)`

Control logging verbosity.

Parameter Description:

`EMANE::LogLevel`	`level`	Log level. See Table 5.1 on page 26.

13.1.2 `void redirectLogsToSysLog(const char *program)`

Direct log statements to the system logger.

Parameter Description:

`const char *`	`program`	User specified string to demarcate system log statements

13.1.3 `void redirectLogsToFile(const char* filename)`

Direct log statements to a file.

Parameter Description:

`const char *`	`filename`	The log destination file

13.1.4 `void redirectLogsToRemoteLogger(const char *program, const char* address)`

Direct log statements to the remote logger.

Parameter Description:

`const char *`	`program`	User specified string to demarcate log statements
`const char *`	`address`	address:port of the remote logger

13.1.5 `void log(LogLevel level, const char *fmt, ...)`

Log a formatted string.

Parameter Description:

`EMANE::LogLevel`	`level`	Log level. See Table 5.1 on page 26.
`const char *`	`fmt`	A `printf` style format string
`vargs`	`...`	Format string arguments

13.1.6 Sample Usage

The `EMANE::Logger` is used in subsequent chapter examples: Listing 13.1, Listing 13.2 and Listing 13.3.

13.2 `EMANE::NEMBuilder`

An `EMANE::NEMBuilder` instance constructs, configures and executes a Platform Server and its enclosed NEMs and NEM layers. The builder enforces build rules that promote correct use. Each constructed `NEMLayer` instance may only be used in one `NEM`. Each constructed `NEM` instance may only be used in one `Platform`. Only one `Platform` can be built.

13.2.1 `EMANE::NEMLayer * buildPHYLayer(NEMId id, const std::string & sLibraryFile, const ConfigurationItems * pItems) throw(EMANEUtils::FactoryException, ConfigureException, InitializeException)`

Build a configured `PHYLayer` from a PHY library implementation.

Parameter Description:

`EMANE::NEMId`	`id`	NEM Id of this PHY layer
`const std::string &`	`sLibraryFile`	The name of the PHY library definition to load
`EMANE::ConfigurationItems *`	`pItems`	List of EMANE::ConfigurationItem

13.2.2 `EMANE::NEMLayer * buildMACLayer(NEMId id, const std::string & sLibraryFile, const ConfigurationItems * pItems) throw(EMANEUtils::FactoryException, ConfigureException, InitializeException)`

Build a configured `MACLayer` from a MAC library implementation.

Parameter Description:

`EMANE::NEMId`	`id`	NEM Id of this MAC layer.
`const std::string &`	`sLibraryFile`	The name of the MAC library definition to load
`EMANE::ConfigurationItems *`	`pItems`	List of EMANE::ConfigurationItem

13.2.3 `EMANE::NEMLayer * buildShimLayer(NEMId id, const std::string & sLibraryFile, const ConfigurationItems * pItems) throw(EMANEUtils::FactoryException, ConfigureException, InitializeException)`

Build a configured `ShimLayer` from a Shim library implementation.

Parameter Description:

`EMANE::NEMId`	`id`	NEM Id of this Shim layer.
`const std::string &`	`sLibraryFile`	The name of the Shim library definition to load
`EMANE::ConfigurationItems *`	`pItems`	List of EMANE::ConfigurationItem

13.2.4 `EMANE::NEM * buildNEM(NEMId id, const NEMLayers &layers, const ConfigurationItems * pItems) throw(ConfigureException, InitializeException, BuildException)`

Build a configured `NEM` containing the specified layers. The first layer in the list is placed at the top of the stack, closest to the transport. Each subsequent layer is placed in order, downstream from the preceding layer, with the last layer at the OTA boundary. `NEMLayer` instances cannot be placed in more than one `NEM`.

Parameter Description:

`EMANE::NEMId`	`id`	NEM Id of this NEM.
`EMANE::NEMLayers &`	`layers`	List of EMANE::NEMLayer
`EMANE::ConfigurationItems *`	`pItems`	List of EMANE::ConfigurationItem

13.2.5 `EMANE::Platform * buildPlatform(PlatformId id, const NEMs &nems, const ConfigurationItems * pItems) throw(ConfigureException, InitializeException, BuildException)`

Build a configured `Platform` instance containing the specified NEMs. A `NEM` instance cannot be placed in more than one `Platform` instance. The `NEMBuilder` will not contruct more than one `Platform` instance.

Parameter Description:

`EMANE::PlatformId`	`id`	platformid of this Platform.
`EMANE::NEMs &`	`nems`	List of EMANE::NEM
`EMANE::ConfigurationItems *`	`pItems`	List of EMANE::ConfigurationItem

13.2.6 Sample Usage

The `NEMBuilder` example is combined with the `TransportBuilder` example in Section 13.3.4.

13.3 `EMANE::TransportBuilder`

The `EMANE::TransportBuilder` constructs, configures and executes one or more `EMANE::Transport` instances. The collection of transports is managed by an `EMANE::TransportManager` container. Each `EMANE::Transport` is associated with one NEM and serves as its emulation/application boundary. An intermediate `EMANE::-TransportAdapter` instance wraps each `Transport`. It handles the socket communication with the associated NEM in a uniform way, irrespective of the transport implementation. It is configured with the same *transportendpoint* and *platformendpoint* configuration values as the corresponding NEM, as will be seen in the example.

13.3.1 `EMANE::Transport * buildTransport(const NEMId id, const std::string & sLibraryFile, const ConfigurationItems * pItems) throw(EMANEUtils::FactoryException, ConfigureException, InitializeException)`

Build a configured `Transport` from the specified library definition. Each `Transport` is associated with a single NEM.

Parameter Description:

`EMANE::NEMId`	`id`	NEM Id of the corresponding NEM.
`std::string &`	`sLibraryFile`	The name of the transport library definition to load
`EMANE::ConfigurationItems *`	`pItems`	List of EMANE::ConfigurationItem

13.3.2 `EMANE::TransportAdapter * buildTransportAdapter(EMANE::Transport * pTransport, const ConfigurationItems * pItems) throw(BuildException,ConfigureException,InitializeException)`

Build a configured `TransportAdapter` to wrap the provided `Transport` and enable connection to the NEM.

Parameter Description:

`EMANE::Transport *`	`pTransport`	Pointer to EMANE::Transport
`EMANE::ConfigurationItems *`	`pItems`	List of EMANE::ConfigurationItem

13.3.3 `EMANE::TransportManager * buildTransportManager(const TransportAdapters &adapters, const ConfigurationItems * pItems) throw(BuildException,ConfigureException,InitializeException)`

Build a configured `TransportManager` to manage the collection of transport adapters and transports.

Parameter Description:

`EMANE::TransportAdapters &`	`adapters`	List of EMANE::TransportAdapter
`EMANE::ConfigurationItems *`	`pItems`	List of EMANE::ConfigurationItem

13.3.4 Sample Usage

The example application in Listing 13.1 uses an `NEMBuilder` to build a Platform Server containing one NEM and an `EMANE::TransportBuilder` to build the NEM's associated transport. The NEM is constructed from the MAC layer developed in Chapter 6 and the PHY layer developed in Chapter 7. Both layers are configured identically to the earlier examples, but programmatically rather than via XML files.

The example may be usefully extended to instantiate multiple NEMs as in a centralized deployment.

The example also uses an `EMANE::Logger` instance to set the log level to `DEBUG_LEVEL`. This is equivalent to the setting specified by the command line option to the EMANE applications. Output 9 shows the output generated by the example.

```
#include <libemane/logger.h>
#include <libemane/nembuilder.h>
#include <libemane/transportbuilder.h>
#include <emaneutils/terminator.h>

#include <cstdlib>


/**
 *
 * Build A Platform Server with one NEM, an associated Transport and
 * launch them both.
 *
 */
int main(int argc, char** argv)
{
  // Read nemid from the command line
  if(argc != 2)
    {
      std::printf("usage: libemanenem nemid\n");
      return 1;
    }

  EMANE::NEMId nemid;
  std::sscanf(argv[1], "%hu", &nemid);


  // Set log level to debug
  EMANE::Logger logger;
  logger.setLogLevel(EMANE::DEBUG_LEVEL);


  std::printf("***** Build a Platform Server ****************************\n");

  // Build the NEM layers first. Start from top of the nem stack and work
  // your way down (mac before phy).
  EMANE::NEMBuilder nemBuilder;

  EMANE::ConfigurationItems macCfg;
  macCfg.push_back(EMANE::ConfigurationItem("boolvalue", "on"));
  macCfg.push_back(EMANE::ConfigurationItem("u16value", "65521"));
  macCfg.push_back(EMANE::ConfigurationItem("u32value", "6371000"));
  macCfg.push_back(EMANE::ConfigurationItem("stringvalue", "emane"));

  EMANE::NEMLayers nemlayers;
  nemlayers.push_back(nemBuilder.buildMACLayer(nemid,
                                               "devtrainingmac03",
                                               &macCfg));

  EMANE::ConfigurationItems phyCfg;
  phyCfg.push_back(EMANE::ConfigurationItem("boolvalue", "on"));
  phyCfg.push_back(EMANE::ConfigurationItem("u16value", "65521"));
  phyCfg.push_back(EMANE::ConfigurationItem("u32value", "6371000"));
  phyCfg.push_back(EMANE::ConfigurationItem("stringvalue", "emane"));

  nemlayers.push_back(nemBuilder.buildPHYLayer(nemid,
                                               "devtrainingphy04",
                                               &phyCfg));
```

```


    // Next build the NEM from the layers
    EMANE::ConfigurationItems nemCfg;
    char platformendpoint[64];
    std::sprintf(platformendpoint, "localhost:%u", 8800 + nemid);
    char transportendpoint[64];
    std::sprintf(transportendpoint, "localhost:%u", 8900 + nemid);

    nemCfg.push_back(EMANE::ConfigurationItem("platformendpoint",
                                              platformendpoint));
    nemCfg.push_back(EMANE::ConfigurationItem("transportendpoint",
                                              transportendpoint));

    EMANE::NEMs nems;
    nems.push_back(nemBuilder.buildNEM(nemid, nemlayers, &nemCfg));


    // Then the Platform from nems
    EMANE::PlatformId platformid = 1;
    EMANE::ConfigurationItems platformCfg;
    platformCfg.push_back(EMANE::ConfigurationItem("otamanagerchannelenable",
                                                   "off"));
    platformCfg.push_back(EMANE::ConfigurationItem("eventservicegroup",
                                                   "224.1.2.8:45703"));
    platformCfg.push_back(EMANE::ConfigurationItem("eventservicedevice",
                                                   "lo"));

    EMANE::Platform * platform = nemBuilder.buildPlatform(platformid,
                                                          nems,
                                                          &platformCfg);


    std::printf("***** Build a Transport Manager ***************************\n");

    EMANE::TransportBuilder transBuilder;

    // First a transport
    EMANE::ConfigurationItems transCfg;
    transCfg.push_back(EMANE::ConfigurationItem("boolvalue", "on"));
    transCfg.push_back(EMANE::ConfigurationItem("u16value", "65521"));
    transCfg.push_back(EMANE::ConfigurationItem("u32value", "6371000"));
    transCfg.push_back(EMANE::ConfigurationItem("stringvalue", "emane"));

    EMANE::Transport * t = transBuilder.buildTransport(nemid,
                                                       "devtrainingtrans07",
                                                       &transCfg);


    // Then a TransportAdapter. The adapter handles communication with the NEM
    // so share the platformendpoint/transportendpoint configuration
    EMANE::TransportAdapters adapters;
    adapters.push_back(transBuilder.buildTransportAdapter(t, &nemCfg));


    // And finally the manager
    EMANE::TransportManager * tmgr =
      transBuilder.buildTransportManager(adapters, NULL);


    std::printf("***** Start Platform Server ******************************\n");
    platform->start();
    platform->postStart();

    std::printf("***** Start Transport Manager*****************************\n");
    tmgr->start();
    tmgr->postStart();

    // wait for Control^C
    EMANEUtils::Terminator terminator;
    terminator.reactor(ACE_Reactor::instance());
    ACE_Reactor::instance()->run_reactor_event_loop();

    std::printf("***** Stop Transport Manager *****************************\n");
    tmgr->stop();
    tmgr->destroy();

```

```
  std::printf("***** Stop Platform Server ******************************\n");
  platform->stop();
  platform->destroy();
}
```

Listing 13.1: `libemane` Sample Usage, Platform Server and Transport Manager

```
# libemanenem 37
***** Build a Platform Server ****************************
15:48:58.117137 DEBUG MACI 037 TRAINING::DevTrainingMAC03::initialize line 52
15:48:58.117212 DEBUG MACI 037 TRAINING::DevTrainingMAC03::configure line 59
15:48:58.117402 DEBUG PHYI 037 TRAINING::DevTrainingPHY04::initialize line 55
15:48:58.117407 DEBUG PHYI 037 TRAINING::DevTrainingPHY04::configure line 62
15:48:58.117444 DEBUG NEMLayerStack::initialize
15:48:58.117462 DEBUG ConcretePlatform::initialize
15:48:58.117463 DEBUG ConcretePlatform::configure
***** Build a Transport Manager ***************************
15:48:58.117612 DEBUG PHYI 037 TRAINING::DevTrainingTrans07::DevTrainingTrans07 line 36
15:48:58.117621 DEBUG PHYI 037 TRAINING::DevTrainingTrans07::initialize line 50
15:48:58.117622 DEBUG PHYI 037 TRAINING::DevTrainingTrans07::configure line 57
***** Start Platform Server ******************************
15:48:58.117651 DEBUG ConcretePlatform::start debugport = 47000
15:48:58.117659 DEBUG ConcretePlatform::start debugportenable = 0
15:48:58.117661 DEBUG ConcretePlatform::start eventservicedevice = lo
15:48:58.117747 DEBUG ConcretePlatform::start eventservicegroupaddr = 224.1.2.8
15:48:58.117752 DEBUG ConcretePlatform::start otamanagerchannelenable = 0
15:48:58.117753 DEBUG Not Starting Channel Device
15:48:58.118657 DEBUG EventService::processEventMessage
15:48:58.118699 DEBUG EventService::receiveEventMessage
15:48:58.120680 DEBUG MACI 037 TRAINING::DevTrainingMAC03::start line 67
15:48:58.120696 DEBUG MACI 037 TRAINING::DevTrainingMAC03::start boolvalue = on
15:48:58.120703 DEBUG MACI 037 TRAINING::DevTrainingMAC03::start stringvalue = emane
15:48:58.120712 DEBUG MACI 037 TRAINING::DevTrainingMAC03::start u16value = 65521
15:48:58.120717 DEBUG MACI 037 TRAINING::DevTrainingMAC03::start u32value = 6371000
15:48:58.120833 DEBUG PHYI 037 TRAINING::DevTrainingPHY04::start line 70
15:48:58.120870 DEBUG PHYI 037 TRAINING::DevTrainingPHY04::start boolvalue = on
15:48:58.120879 DEBUG PHYI 037 TRAINING::DevTrainingPHY04::start stringvalue = emane
15:48:58.120884 DEBUG PHYI 037 TRAINING::DevTrainingPHY04::start u16value = 65521
15:48:58.120890 DEBUG PHYI 037 TRAINING::DevTrainingPHY04::start u32value = 6371000
15:48:58.120914 DEBUG MACI 037 TRAINING::DevTrainingMAC03::postStart line 163
15:48:58.120919 DEBUG PHYI 037 TRAINING::DevTrainingPHY04::postStart line 166
***** Start Transport Manager****************************
15:48:58.121579 DEBUG NEM 037 ConcreteTransportAdapter::start platformendpoint: 127.0.0.1
15:48:58.122132 DEBUG NEM 037: ConcreteTransportAdapter::start transportendpoint: 127.0.0.1
15:48:58.122183 DEBUG PHYI 037 TRAINING::DevTrainingTrans07::start line 65
15:48:58.122201 DEBUG PHYI 037 TRAINING::DevTrainingTrans07::start boolvalue = on
15:48:58.122208 DEBUG PHYI 037 TRAINING::DevTrainingTrans07::start stringvalue = emane
15:48:58.122215 DEBUG PHYI 037 TRAINING::DevTrainingTrans07::start u16value = 65521
15:48:58.122221 DEBUG PHYI 037 TRAINING::DevTrainingTrans07::start u32value = 6371000
15:48:58.122300 DEBUG PHYI 037 TRAINING::DevTrainingTrans07::postStart line 161
15:48:58.122398 DEBUG NEM 037 ConcreteTransportAdapter::processNetworkMessage
```

Output 9: EMANE Platform Server (Output from the example program **libemanenem** shown in Listing 13.1. The program takes the NEM Id as its only argument.

13.4 EMANE::EventGeneratorBuilder

The `EMANE::EventGeneratorBuilder` follows the pattern of the previous builders. A method is provided to construct and configure `EMANE::EventGenerator` instances from library implementations. The generators can then be added to an `EMANE::EventGeneratorManager` for execution.

13.4.1 `EMANE::EventGenerator * buildEventGenerator(const std::string & sLibraryFile, const ConfigurationItems * pItems) throw(EMANEUtils::FactoryException, ConfigureException, InitializeException)`

Build a configured `EventGenerator` from the named library implementation.

Parameter Description:

`std::string &`	`sLibraryFile`	The name of the event generator library implementation
`EMANE::ConfigurationItems *`	`pItems`	List of EMANE::ConfigurationItem

13.4.2 `EMANE::EventGeneratorManager * buildEventGeneratorManager(const EventGenerators &generators, const ConfigurationItems * pItems) throw(BuildException,ConfigureException,InitializeException)`

Build a configured `EventGeneratorManager` containing the provided generators.

Parameter Description:

`EMANE::EventGenerators &`	`generators`	List of EMANE::EventGenerator
`EMANE::ConfigurationItems *`	`pItems`	List of EMANE::ConfigurationItem

13.4.3 Sample Usage

Listing 13.2 shows an example of launching an `EventGenerator` with an `EventGeneratorBuilder`. The code closely resembles the previous `NEMBuilder` example. The client builds an instance of the `EventGenerator` implemented in Section 9.2, adds it to a `EventGeneratorManager` and starts the manager.

When run, the example generates the output shown in Output 10.

```
#include <libemane/logger.h>
#include <libemane/eventgeneratorbuilder.h>
#include <emaneutils/terminator.h>


/**
 *
 * Build an EventGenerator container with one generator and launch it.
 *
 */
int main(int argc, char** argv)
{
  // Set log level to debug
  EMANE::Logger logger;
  logger.setLogLevel(EMANE::DEBUG_LEVEL);


  std::printf("***** Build a EventGenerator Manager **********************\n");

  EMANE::EventGeneratorBuilder generatorBuilder;

  // First a transport
  EMANE::ConfigurationItems generatorCfg;
  generatorCfg.push_back(EMANE::ConfigurationItem("boolvalue", "on"));
  generatorCfg.push_back(EMANE::ConfigurationItem("u16value", "65521"));
  generatorCfg.push_back(EMANE::ConfigurationItem("u32value", "6371000"));
  generatorCfg.push_back(EMANE::ConfigurationItem("stringvalue", "emane"));
  generatorCfg.push_back(EMANE::ConfigurationItem("modelfilename", "model.txt"));

  EMANE::EventGenerators generators;
  generators.push_back(generatorBuilder.buildEventGenerator("devtraininggen06",
```

```
                                                        &generatorCfg));

  EMANE::ConfigurationItems managerCfg;
  managerCfg.push_back(EMANE::ConfigurationItem("eventservicegroup",
                                                "224.1.2.8:45703"));
  managerCfg.push_back(EMANE::ConfigurationItem("eventservicedevice", "lo"));

  // And finally the manager
  EMANE::EventGeneratorManager * generatorMgr =
    generatorBuilder.buildEventGeneratorManager(generators, &managerCfg);


  std::printf("***** Start EventGenerator Manager ************************\n");
  generatorMgr->start();
  generatorMgr->postStart();

  // wait for Control^C
  EMANEUtils::Terminator terminator;
  terminator.reactor(ACE_Reactor::instance());
  ACE_Reactor::instance()->run_reactor_event_loop();

  std::printf("***** Stop EventGenerator Manager *************************\n");
  generatorMgr->stop();
  generatorMgr->destroy();
}
```

Listing 13.2: `libemane` Sample Usage, Event Generator

```
# libemanegenerator
***** Build a EventGenerator Manager ***********************
18:54:13.603074 DEBUG TRAINING::DevTrainingGen06::DevTrainingGen06 line 43
18:54:13.603221 DEBUG TRAINING::DevTrainingGen06::initialize line 57
18:54:13.603226 DEBUG TRAINING::DevTrainingGen06::configure line 64
18:54:13.603263 DEBUG ConcreteEventGeneratorManager::configure
***** Start EventGenerator Manager ************************
18:54:13.603276 DEBUG ConcreteEventGeneratorManager::start Event Service Device: lo
18:54:13.603437 DEBUG ConcreteEventGeneratorManager::start Event Service Group: 224.1.2.8
18:54:13.603528 DEBUG TRAINING::DevTrainingGen06::start line 72
18:54:13.603546 DEBUG TRAINING::DevTrainingGen06::start boolvalue = on
18:54:13.603551 DEBUG TRAINING::DevTrainingGen06::start modelfilename = model.txt
18:54:13.603726 DEBUG TRAINING::DevTrainingGen06::start stringvalue = emane
18:54:13.603733 DEBUG TRAINING::DevTrainingGen06::start u16value = 65521
18:54:13.603737 DEBUG TRAINING::DevTrainingGen06::start u32value = 6371000
18:54:13.604043 DEBUG Event 32770 EMANE::EventService::sendEvent
18:54:13.604138 DEBUG TRAINING::DevTrainingGen06::postStart line 176
18:54:13.604111 DEBUG EventService::receiveEventMessage
18:54:13.604205 DEBUG EventService::processEventMessage
18:54:13.604282 DEBUG EventService Packet Received len: 36
18:54:13.604380 DEBUG ConcreteEventGeneratorManager::svc
18:54:13.604395 DEBUG ConcreteEventGeneratorManager Packet Received len: 36
18:54:14.604453 DEBUG Event 32770 EMANE::EventService::sendEvent
18:54:14.604555 DEBUG ConcreteEventGeneratorManager Packet Received len: 36
18:54:14.604587 DEBUG EventService Packet Received len: 36
18:54:15.604622 DEBUG Event 32770 EMANE::EventService::sendEvent
18:54:15.604725 DEBUG ConcreteEventGeneratorManager Packet Received len: 36
18:54:15.604766 DEBUG EventService Packet Received len: 36
18:54:16.604855 DEBUG Event 32770 EMANE::EventService::sendEvent
18:54:16.604950 DEBUG ConcreteEventGeneratorManager Packet Received len: 36
18:54:16.604991 DEBUG EventService Packet Received len: 36
18:54:17.605078 DEBUG Event 32770 EMANE::EventService::sendEvent
18:54:17.605178 DEBUG ConcreteEventGeneratorManager Packet Received len: 36
18:54:17.605210 DEBUG EventService Packet Received len: 36
```

Output 10: EMANE EventGenerator (Output from the example program **libemanegenerator** shown in Listing 13.2).

13.5 EMANE::EventAgentBuilder

The EMANE::EventAgentBuilder provides an interface for building EMANE::EventAgent instances and an EMANE::EventAgentManager container to execute them. The interface is nearly identical to the EMANE::EventGeneratorBuilder class interface, but for event agents instead of event generators.

13.5.1 EMANE::EventAgent * buildEventAgent(EMANE::NEMId nemId, const std::string & sLibraryFile, const ConfigurationItems * pItems) throw(EMANEUtils::FactoryException, ConfigureException, InitializeException)

Build a configured EventAgent from the named library implementation. The method includes an EMANE::NEMId parameter. This parameter can be used by agent implementations to filter received events. As an example, the GPS Location Agent acts on behalf of a single NEM to publish GPS location events; only events tagged with the specified nemid are published.

Parameter Description:

EMANE::NEMId	nemid	NEM Id of corresponding NEM
std::string &	sLibraryFile	The name of the event agent library implementation
EMANE::ConfigurationItems *	pItems	List of EMANE::ConfigurationItem

13.5.2 EMANE::EventAgentManager * buildEventAgentManager(const EventAgents &agents, const ConfigurationItems * pItems) throw(BuildException,ConfigureException,InitializeException)

Build a configured Event Agent manager containing the provided agents.

Parameter Description:

EMANE::EventAgents &	agents	List of EMANE::EventAgent
EMANE::ConfigurationItems *	pItems	List of EMANE::ConfigurationItem

13.5.3 Sample Usage

Listing 13.3 shows code for launching an instance of the EMANE::EventAgent implemented in Section 9.3.1. The code is nearly identical to Listing 13.2. Output 11 shows the text generated by the **libemaneagent** example when run with the **libemanegenerator** program from the previous section. The agent receives the generator's events.

```
#include <libemane/logger.h>
#include <libemane/eventagentbuilder.h>
#include <emaneutils/terminator.h>


/**
 *
 * Build an EventAgent container with one agent and launch it.
 *
 */
int main(int argc, char** argv)
{
  // Set log level to debug
  EMANE::Logger logger;
```

```
  logger.setLogLevel(EMANE::DEBUG_LEVEL);


  std::printf("***** Build a EventAgent Manager **************************\n");

  EMANE::EventAgentBuilder agentBuilder;

  // First a transport
  EMANE::ConfigurationItems agentCfg;
  agentCfg.push_back(EMANE::ConfigurationItem("boolvalue", "on"));
  agentCfg.push_back(EMANE::ConfigurationItem("u16value", "65521"));
  agentCfg.push_back(EMANE::ConfigurationItem("u32value", "6371000"));
  agentCfg.push_back(EMANE::ConfigurationItem("stringvalue", "emane"));

  EMANE::EventAgents agents;
  agents.push_back(agentBuilder.buildEventAgent(0, "devtrainingagent08", &agentCfg));

  EMANE::ConfigurationItems managerCfg;
  managerCfg.push_back(EMANE::ConfigurationItem("eventservicegroup",
                                                "224.1.2.8:45703"));
  managerCfg.push_back(EMANE::ConfigurationItem("eventservicedevice", "lo"));

  // And finally the manager
  EMANE::EventAgentManager * agentMgr =
    agentBuilder.buildEventAgentManager(agents, &managerCfg);


  std::printf("***** Start EventAgent Manager ***************************\n");
  agentMgr->start();
  agentMgr->postStart();

  // wait for Control^C
  EMANEUtils::Terminator terminator;
  terminator.reactor(ACE_Reactor::instance());
  ACE_Reactor::instance()->run_reactor_event_loop();

  std::printf("***** Stop EventAgent Manager ****************************\n");
  agentMgr->stop();
  agentMgr->destroy();
}
```

Listing 13.3: `libemane` Sample Usage, Event Agent

```
# libemaneagent
***** Build a EventAgent Manager ***************************
18:54:11.541636 DEBUG TRAINING::DevTrainingAgent08::DevTrainingAgent08 line 38
18:54:11.541786 DEBUG TRAINING::DevTrainingAgent08::initialize line 52
18:54:11.541791 DEBUG TRAINING::DevTrainingAgent08::configure line 59
***** Start EventAgent Manager ***************************
18:54:11.544194 DEBUG EventService::receiveEventMessage
18:54:11.544197 DEBUG TRAINING::DevTrainingAgent08::start line 67
18:54:11.544293 DEBUG TRAINING::DevTrainingAgent08::start boolvalue = on
18:54:11.544303 DEBUG TRAINING::DevTrainingAgent08::start stringvalue = emane
18:54:11.544309 DEBUG TRAINING::DevTrainingAgent08::start u16value = 65521
18:54:11.544316 DEBUG TRAINING::DevTrainingAgent08::start u32value = 6371000
18:54:11.544321 DEBUG EventService::processEventMessage
18:54:11.544388 DEBUG TRAINING::DevTrainingAgent08::postStart line 156
18:54:13.604166 DEBUG EventService Packet Received len: 36
18:54:13.604173 DEBUG ConcreteEventAgentManager Packet Received len: 36
18:54:13.604216 DEBUG TRAINING::DevTrainingAgent08::processEvent line 175
18:54:13.604235 DEBUG TRAINING::DevTrainingAgent08::processEvent 1 0% loss
18:54:13.604242 DEBUG TRAINING::DevTrainingAgent08::processEvent 2 0% loss
18:54:13.604245 DEBUG TRAINING::DevTrainingAgent08::processEvent 3 0% loss
18:54:13.604248 DEBUG TRAINING::DevTrainingAgent08::processEvent 4 0% loss
18:54:13.604252 DEBUG TRAINING::DevTrainingAgent08::processEvent 5 0% loss
18:54:14.604585 DEBUG EventService Packet Received len: 36
18:54:14.604621 DEBUG ConcreteEventAgentManager Packet Received len: 36
18:54:14.604634 DEBUG TRAINING::DevTrainingAgent08::processEvent line 175
18:54:14.604644 DEBUG TRAINING::DevTrainingAgent08::processEvent 1 10% loss
18:54:14.604647 DEBUG TRAINING::DevTrainingAgent08::processEvent 2 20% loss
18:54:14.604650 DEBUG TRAINING::DevTrainingAgent08::processEvent 3 30% loss
18:54:14.604652 DEBUG TRAINING::DevTrainingAgent08::processEvent 4 40% loss
18:54:14.604655 DEBUG TRAINING::DevTrainingAgent08::processEvent 5 50% loss
18:54:15.604753 DEBUG EventService Packet Received len: 36
18:54:15.604753 DEBUG ConcreteEventAgentManager Packet Received len: 36
18:54:15.604817 DEBUG TRAINING::DevTrainingAgent08::processEvent line 175
18:54:15.604830 DEBUG TRAINING::DevTrainingAgent08::processEvent 1 50% loss
18:54:15.604835 DEBUG TRAINING::DevTrainingAgent08::processEvent 2 20% loss
18:54:15.604839 DEBUG TRAINING::DevTrainingAgent08::processEvent 3 30% loss
18:54:15.604844 DEBUG TRAINING::DevTrainingAgent08::processEvent 4 20% loss
18:54:15.604848 DEBUG TRAINING::DevTrainingAgent08::processEvent 5 10% loss
18:54:16.604977 DEBUG EventService Packet Received len: 36
18:54:16.604977 DEBUG ConcreteEventAgentManager Packet Received len: 36
18:54:16.605027 DEBUG TRAINING::DevTrainingAgent08::processEvent line 175
18:54:16.605040 DEBUG TRAINING::DevTrainingAgent08::processEvent 1 0% loss
18:54:16.605048 DEBUG TRAINING::DevTrainingAgent08::processEvent 2 0% loss
18:54:16.605050 DEBUG TRAINING::DevTrainingAgent08::processEvent 3 0% loss
18:54:16.605053 DEBUG TRAINING::DevTrainingAgent08::processEvent 4 0% loss
18:54:16.605055 DEBUG TRAINING::DevTrainingAgent08::processEvent 5 0% loss
```

Output 11: EMANE EventAgent (Output from the example program **libemaneagent** shown in Listing 13.3 when run with the **libemanegenerator** example from Listing 13.2).

Bibliography

Erich Gamma, Richard Helm, Ralph Johnson, and John M. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional, illustrated edition, November 1994. ISBN 0201633612.