NoaNex

A Node Architecture for Netlet Execution

Denis Martin et al.

25 Aug 2009

Contents

Chapter 1

Overview

This is the reference documentation of the node architecture prototype. It will not describe the concepts themselves, only their implementations. For background information about the concepts, please refer to [1].

The code is pure C++/STL with some use of the Boost libraries (mainly header files). The daemon as well as the example implementations of Netlets etc. are designed to run in multiple target environments, such as Linux, Windows, or OMNeT++.

1.1 NoaNex Daemon

The node architecture daemon is the core system that provides the basic components as e. g. the repository, the Netlet selector, and the network access broker (see [1]). It interfaces with the system wrapper and it is able to load and instantiate architecture specific multiplexers (INetletMultiplexer) and Netlets (INetlet). Both of the latter are compiled as shared libraries, which in turn are loaded by the daemon.

Figure 1.1 gives an overview on the relavant classes implementing the Node Architecture. Classes are prefixed with "C", interfaces with "I". All classes depicted here are compiled into an object archive (static library). This archive is then linked to wrapper classes provided by the respective target systems to form the final executable.

With target system, we refer to the system environment where the daemon will be run. There are existing targets for the OMNeT++ simulator and UNIX systems supported by boost (the status of the latter is *pending*).

In this chapter, the main implementation decisions are described. For now, it comprises the internal message passing interfaces, the system wrapper classes, and selected utility classes.

1.2 Message processors

Entities within the node architecture communicate with each other via special message passing interfaces in order to allow for transparent multi-threading (this is work-in-progress). The internal message passing interfaces consist of two interface definitions: IMessageProcessor and IMessageScheduler. A message scheduler manages the input/output queues of one or more message processors. A message processor belongs to exactly one message scheduler at a given time.

A message processor sends its messages always via its associated scheduler. If the destination processor belongs to the same scheduler, the scheduler will enqueue the message for the processor and call it later on. However, if the destination processor belongs to another scheduler, inter-scheduler (and hence, inter-

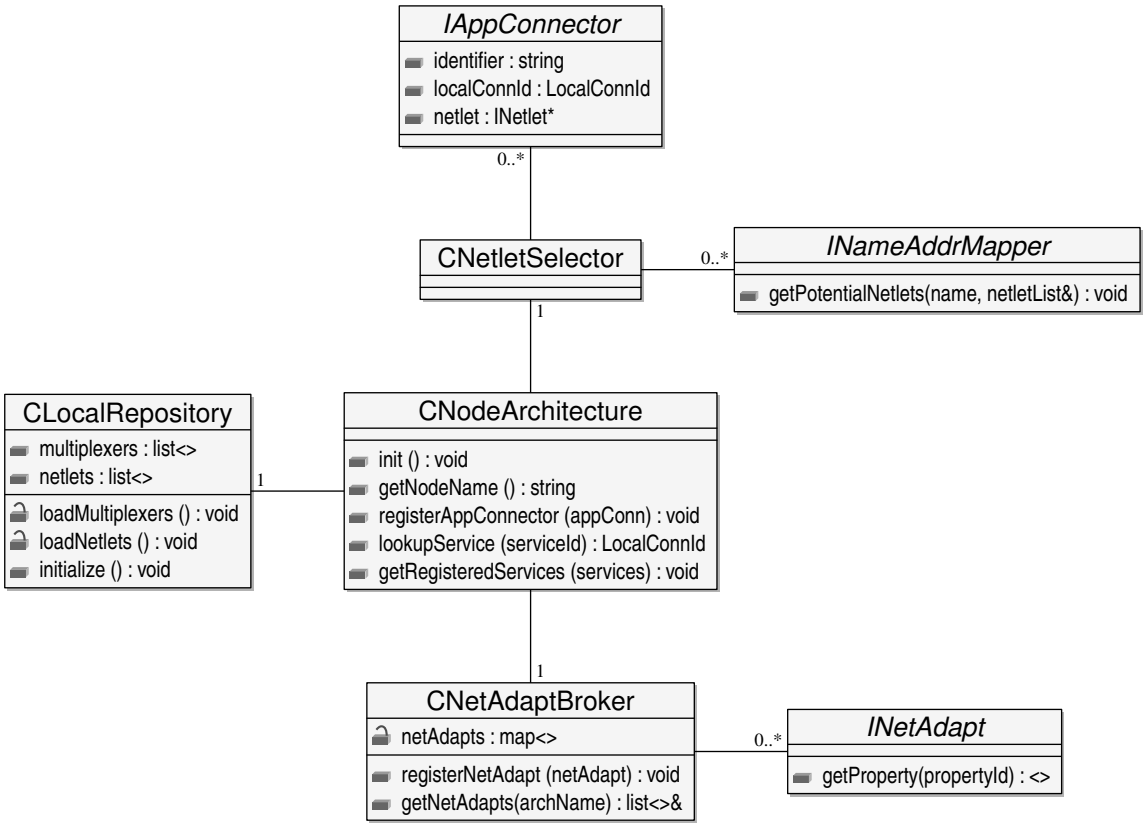


Figure 1.1: NoaNex Class Diagram

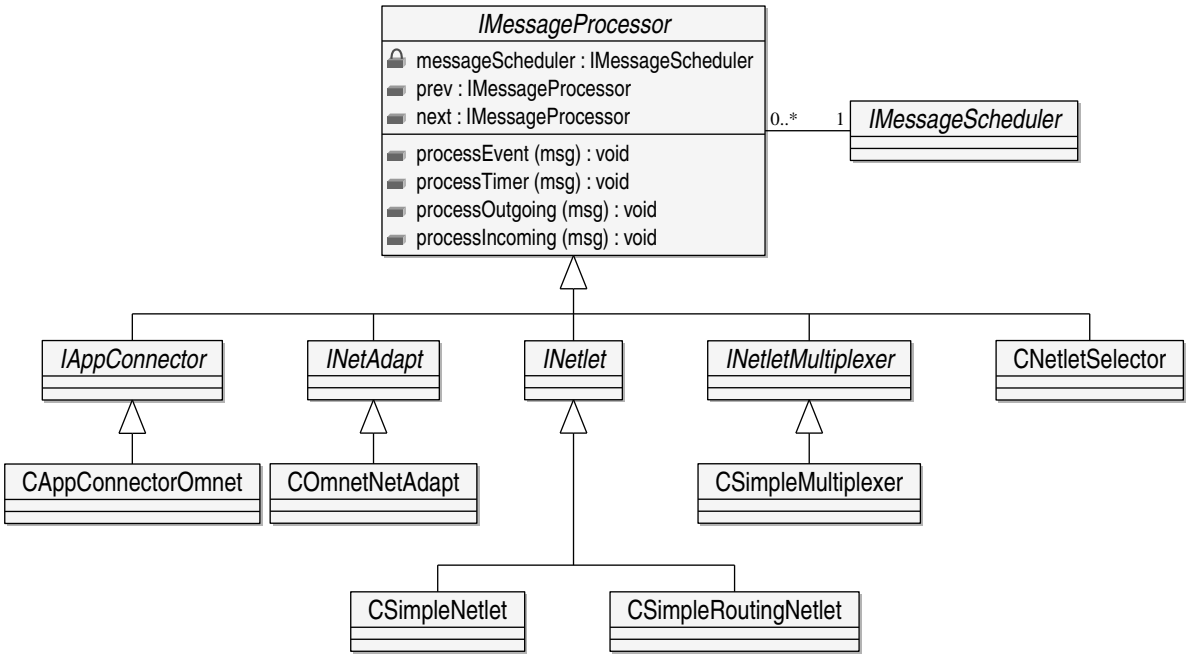


Figure 1.2: Message processors (inheritance tree)

thread) communication is needed. The communication between schedulers is handled by the schedulers. *Note:* These interfaces are *not* intended to provide message passing across different hosts as some existing MPIs do. Figure 1.2 shows relevant classes inheriting the IMessageProcessor interface. Each separate entity doing some message processing is a message processor.

This concept was mainly introduced to support multi-threading transparently to the message processors: Per thread, a single message scheduler takes care of cross-thread message passing to schedulers running in other threads. Thus, all message processors associated to the same scheduler will run in the same thread. The number of threads can be dynamically chosen depending on the needs or the host system. Currently, this feature is not yet implemented fully, so the scheduler essentially calls directly the processing routine of the destination processor. In OMNeT++, schedulers might be represented by different modules to emulate a sort of parallelism.

At least one scheduler implementing IMessageScheduler must be created for the target system. This is an abstraction for a thread and at least one must be returned by ISystemWrapper::getMainScheduler(). If no real multithreading is implemented (e.g. in a simulator, or a single threaded implementation for real systems like sensor networks), CSimpleMsgScheduler can be used as a basis.

An idea, how the messaging interfaces work, can be obtained from the unit test implementation in test/ut-messages/ut-messages.cpp or in the node architecture implementation.

1.3 System Wrapper

A system wrapper provides the necessary abstractions to access services of the underlying operating system (such as timers, threads, network send/receive, ...). The current implementation focuses on wrappers for OMNeT++ (CSystemOmnet) and real systems like Linux, BSD, or others where the Boost ASIO library is available (CSystemBoost). In total, a wrapper consists of implementations of several interfaces:

ISystemWrapper – The ISystemWrapper provides general system information and should be instantiated by the main initialization routine of the target system. It will also instantiate the CNodeArchitecture class

and will take care of the initialization of the system specific network adaptors and application APIs.

INetAdapt – This interface provides access to a medium where the outgoing messages are sent to (and incoming ones are expected from). At least one implementation of such a Network Access / Network Adaptor must exist. A note on naming: Network Access and Network Adaptor can be used interchangeably

IAppConnector – A system specific application interface can be provided by the system wrapper. The IAppConnector serves for interfacing with the node architecture (respectively, the Netlet selector). The interface towards the application can be completely dependent of the host system, but we will develop an enhanced application API that will support all features envisioned by this framework.

IMessageScheduler – A scheduler taking care about the message passing between message processors within the local node architecture. This should relate to a thread. At least one implementation must be provided by the target system (representing the main thread).

1.3.1 OMNeT++ Wrapper

The OMNeT++ implementation of the system wrapper interfaces consists of the following classes:

CSystemOmnet – TBD

COmnetNetAdapt – TBD

CAppConnectorOmnet – TBD

COmnetScheduler – TBD

1.3.2 Boost Wrapper

TBD

1.4 Simple Architecture

The Simple Architecture is an example architecture and mainly used for testing the concepts. It should be also used to get started with own implementations of Netlets.

CSimpleMultiplexer implements a multiplexer for the "Simple" Netlets. Forwarding of messages is done within the multiplexer. Note, that this is a "design decision" of the simple architecture – in general, forwarding may also be done within a Netlet.

CSimpleNameAddrMapper implements a name/address mapper for the Simple Architecture. This is not yet complete and should not be regarded too closely.

CSimpleComposedNetlet and CSimpleMultimediaNetlet are examples on how to use building blocks and IComposableNetlet. This is by no means complete and should not be looked at right now. More will follow soon.

1.5 Utilities

This section describes a selection of utility classes used within the Node Architecture.

1.5.1 Morphable value

CMorphableValue is a base class for various types. It may contain bools, integers, doubles, strings, and ranges (e. g. 2 - 42) and lists (e. g. 2, 4, 10) of those. In INetAdapt, it is used in conjunction with a std::map<> in order to provide a list of extensible network (access) properties. In IMessage, it is used to provide meta data for a message going through the daemon and Netlets.

1.6 Tutorials

This section provides brief tutorials or code-walkthroughs.

1.6.1 Getting started

In this section, we will give a little overview on how things are working together in the Node Architecture. Here, we will only consider the OMNeT implementation, so everything above the Netlet Selector and below the Multiplexer / Network Access Broker will differ a lot for real implementations.

The code within the latest-alpha tag should always compile and produce an OMNeT executable. Please have a look at the README file for instructions on how to compile and run the code.

When running the omnet_na binary, a small demo network with three nodes should be shown (rev78, 2009-07-28). Node n[0] sends periodic ping-requests to node n[2], which is supposed to answer it. Some additional messages are sent between all nodes to exchange routing information. After the routing information is up-to-date at all nodes, the ping will finally succeed.

When looking at the code, ping is realized as an application in the class CPingPong (src/targets/omnetpp/apps). Although this class has no OMNeT specific includes, it resides in the OMNeT target directory. The reason for this is, that applications running in a simulator generally don't have any user interaction and are specifically written for traffic generation.

The example application CPingPong implements the IAppConnector interface. This application connector interface class has a proxy role when communicating with the application. Since the application and the Node Architecture are in the same executable for simulators, CPingPong may just implement it directly.

IAppConnector itself inherits from IMessageProcessor which is a base class for all message processing entities within the Node Architecture. INetlet, INetletMultiplexer etc. will all inherit from this.

When CPingPong is created, it sets a timeout upon which it will send its first ping-request. This message will be sent to the CNetletSelector which, for the time being, is hard-wired to CSimpleNetlet. CSimpleNetlet "provides" some transport functionalities, which essentially means that it adds a header to identify the application that sent the message. After that, it just hands it over to the CSimpleMultiplexer.

CSimpleMultiplexer adds addressing information to the packet and realizes basic forwarding mechanisms. Note, that this is a decision of the "Simple Architecture". Forwarding may also be realized within Netlets, if this is desired. For forwarding, the CSimpleMultiplexer has a very simple forwarding information base (FIB), which is maintained by the CSimpleRoutingNetlet.

The CSimpleRoutingNetlet is a so-called control Netlet, which means that there is no application associated with it and that it runs on its own in the Node Architecture. It sends some route information exchange (RIX) messages via the known network adaptors / accesses and collects any incoming RIX messages. Based on these messages, the FIB of the CSimpleMultiplexer is updated. Note, that this is a very simple mechanism. It will suffer from many well-known routing problems and it won't select the best route, just any route.

The CSimpleMultiplexer selects the INetAdapt via which the outgoing messages are sent. Either it uses its FIB, or the emitting entity (application, Netlet) has already chosen the INetAdapt to use (as, e.g., the CSimpleRoutingNetlet does).

The INetAdapt interface is implemented in COmnetNetAdapt, which basically maps the network adaptor to an OMNeT port.

1.6.2 Creating an own Control Netlet

To create an own Control Netlet (i. e., a Netlet not associated to any user application), the best thing is to start with CSimpleRoutingNetlet and to construct it on top of the CSimpleMultiplexer. So, just copy the two files of CSimpleRoutingNetlet and name them appropriately (e.g. simpleControlNetlet.(h|cpp)). Edit the SConscript file and add an entry for the new Netlet. Go into the two class files and rename everything that is related to the 'Simple Routing' to your new name. Be sure, to also rename its ID (i.e., the string in SIMPLE_ROUTING_NETLET_NAME).

After you did all those changes and renaming, it should compile and create a new shared library in build/targets/netlets. When running the OMNeT simulation, it should also be instantiated automatically.

Chapter 2

Namespace Index

2.1 Namespace List

Here is a list of all documented namespaces with brief descriptions:

Chapter 3

Class Index

3.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 4

Class Index

4.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 5

File Index

5.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 6

Namespace Documentation

6.1 netletEdit Namespace Reference

Name space used by the NetletEdit tool.

Classes

- class SimpleMultimediaNetletConfig
  Auto-generated Netlet configuration.
- class SimpleTransportNetletConfig
  Auto-generated Netlet configuration.

6.1.1 Detailed Description

Name space used by the NetletEdit tool.

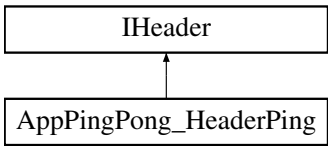
Chapter 7

Class Documentation

7.1 AppPingPong_HeaderPing Class Reference

Header class for testing.

Inheritance diagram for AppPingPong_HeaderPing::



Public Member Functions

- virtual void serialize (CSerialBuffer *buffer)

  Serialize all relevant data into a byte buffer. Remember to do Little/Big Endian conversion.

- virtual void deserialize (CSerialBuffer *buffer)

  De-serialize all relevant data from a byte buffer. Remember to do Little/Big Endian conversion.

Public Attributes

- std::string **testStr**
- int **testInt**
- bool **isPong**

7.1.1 Detailed Description

Header class for testing.

Header format is [A][B...][CCCC][D] where A is the size of testStr (one byte), B is testStr (variable length), C is testInt (four bytes), D is isPong (one byte).

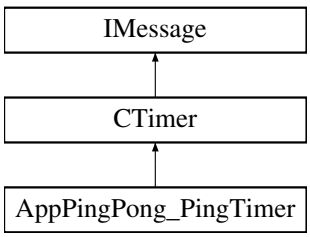
The documentation for this class was generated from the following file:

- src/targets/omnetpp/apps/pingPong.cpp

7.2 AppPingPong_PingTimer Class Reference

Ping timer.

Inheritance diagram for AppPingPong_PingTimer::



Public Member Functions

- AppPingPong_PingTimer (double timeout, IMessageProcessor *proc)
  Constructor.
- virtual ~AppPingPong_PingTimer ()
  Destructor.

7.2.1 Detailed Description

Ping timer.

Single shot timer.

7.2.2 Constructor & Destructor Documentation

7.2.2.1 AppPingPong_PingTimer::AppPingPong_PingTimer (double *timeout*, IMessageProcessor * *proc*) [inline]

Constructor.

Parameters:

timeout Timeout in seconds

proc Node arch entity the event is linked to (default = NULL)

The documentation for this class was generated from the following file:

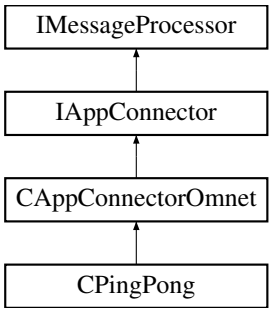
- src/targets/omnetpp/apps/pingPong.cpp

7.3 CAppConnectorOmnet Class Reference

Omnet implementation of the app connector interface. This actually does nothing since the main logic will be in the traffic generator applications themselves.

`#include <appConnectorOmnet.h>`

Inheritance diagram for CAppConnectorOmnet::



Public Member Functions

- **CAppConnectorOmnet** (IMessageScheduler *sched)

7.3.1 Detailed Description

Omnet implementation of the app connector interface. This actually does nothing since the main logic will be in the traffic generator applications themselves.

The documentation for this class was generated from the following files:

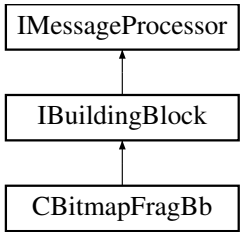
- src/targets/omnetpp/appConnectorOmnet.h
- src/targets/omnetpp/appConnectorOmnet.cpp

7.4 CBitmapFragBb Class Reference

Building block to fragment a bitmap into tiles.

`#include <bitmapFragBb.h>`

Inheritance diagram for CBitmapFragBb::



Public Member Functions

- **CBitmapFragBb** (IMessageScheduler ∗sched)
- virtual void processEvent (IMessage ∗msg) throw (EUnhandledMessage)
  Process an event message directed to this message processing unit.
- virtual void processTimer (IMessage ∗msg) throw (EUnhandledMessage)
  Process a timer message directed to this message processing unit.
- virtual void processOutgoing (IMessage ∗msg) throw (EUnhandledMessage)
  Process an outgoing message directed towards the network.
- virtual void processIncoming (IMessage ∗msg) throw (EUnhandledMessage)
  Process an incoming message directed towards the application.
- virtual Id getId ()
  Return ID of the building block.

7.4.1 Detailed Description

Building block to fragment a bitmap into tiles.

7.4.2 Member Function Documentation

7.4.2.1 void CBitmapFragBb::processEvent (IMessage ∗ *msg*) throw (EUnhandledMessage) [virtual]

Process an event message directed to this message processing unit.

Parameters:

msg Pointer to message

Implements IMessageProcessor.

7.4.2.2 void CBitmapFragBb::processTimer (IMessage * *msg*) throw (EUnhandledMessage) [virtual]

Process a timer message directed to this message processing unit.

Parameters:

msg Pointer to message

Implements IMessageProcessor.

7.4.2.3 void CBitmapFragBb::processOutgoing (IMessage * *msg*) throw (EUnhandledMessage) [virtual]

Process an outgoing message directed towards the network.

Parameters:

msg Pointer to message

Implements IMessageProcessor.

7.4.2.4 void CBitmapFragBb::processIncoming (IMessage * *msg*) throw (EUnhandledMessage) [virtual]

Process an incoming message directed towards the application.

Parameters:

msg Pointer to message

Implements IMessageProcessor.

The documentation for this class was generated from the following files:

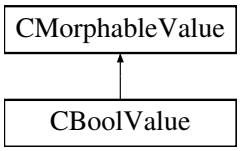
- src/buildingBlocks/bitmapFragBb.h
- src/buildingBlocks/bitmapFragBb.cpp

7.5 CBoolValue Class Reference

Facade for bool values.

`#include <morphableValue.h>`

Inheritance diagram for CBoolValue::



Public Member Functions

- **CBoolValue** (bool v=false)
- bool & **value** ()

7.5.1 Detailed Description

Facade for bool values.

The documentation for this class was generated from the following file:

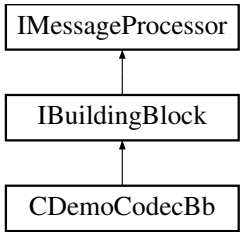
- include/morphableValue.h

7.6 CDemoCodecBb Class Reference

Demo codec building block.

`#include <demoCodecBb.h>`

Inheritance diagram for CDemoCodecBb::



Public Member Functions

- **CDemoCodecBb** (IMessageScheduler *sched)
- virtual void processEvent (IMessage *msg) throw (EUnhandledMessage)
  Process an event message directed to this message processing unit.
- virtual void processTimer (IMessage *msg) throw (EUnhandledMessage)
  Process a timer message directed to this message processing unit.
- virtual void processOutgoing (IMessage *msg) throw (EUnhandledMessage)
  Process an outgoing message directed towards the network.
- virtual void processIncoming (IMessage *msg) throw (EUnhandledMessage)
  Process an incoming message directed towards the application.
- virtual Id getId ()
  Return ID of the building block.

7.6.1 Detailed Description

Demo codec building block.

7.6.2 Member Function Documentation

7.6.2.1 void CDemoCodecBb::processEvent (IMessage * *msg*) throw (EUnhandledMessage) [virtual]

Process an event message directed to this message processing unit.

Parameters:

msg Pointer to message

Implements IMessageProcessor.

7.6.2.2 void CDemoCodecBb::processTimer (IMessage * *msg*) throw (EUnhandledMessage) [virtual]

Process a timer message directed to this message processing unit.

Parameters:

msg Pointer to message

Implements IMessageProcessor.

7.6.2.3 void CDemoCodecBb::processOutgoing (IMessage * *msg*) throw (EUnhandledMessage) [virtual]

Process an outgoing message directed towards the network.

Parameters:

msg Pointer to message

Implements IMessageProcessor.

7.6.2.4 void CDemoCodecBb::processIncoming (IMessage * *msg*) throw (EUnhandledMessage) [virtual]

Process an incoming message directed towards the application.

Parameters:

msg Pointer to message

Implements IMessageProcessor.

The documentation for this class was generated from the following files:

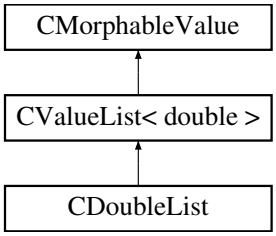
- src/buildingBlocks/demoCodecBb.h
- src/buildingBlocks/demoCodecBb.cpp

7.7 CDoubleList Class Reference

Facade for an int list.

`#include <morphableValue.h>`

Inheritance diagram for CDoubleList::



7.7.1 Detailed Description

Facade for an int list.

The documentation for this class was generated from the following file:

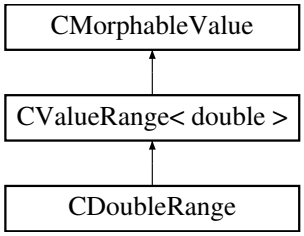
- include/morphableValue.h

7.8 CDoubleRange Class Reference

Facade for double ranges.

`#include <morphableValue.h>`

Inheritance diagram for CDoubleRange::



Public Member Functions

- **CDoubleRange** (double lower=0, double upper=0)

7.8.1 Detailed Description

Facade for double ranges.

The documentation for this class was generated from the following file:

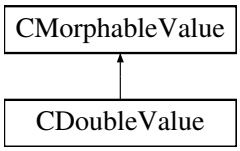
- include/morphableValue.h

7.9 CDoubleValue Class Reference

Facade for int values.

`#include <morphableValue.h>`

Inheritance diagram for CDoubleValue::



Public Member Functions

- **CDoubleValue** (double v=0)
- double & **value** ()

7.9.1 Detailed Description

Facade for int values.

The documentation for this class was generated from the following file:

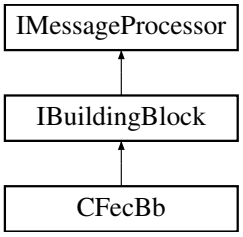
- include/morphableValue.h

7.10 CFecBb Class Reference

FEC building block.

`#include <fecBb.h>`

Inheritance diagram for CFecBb::



Public Member Functions

- **CFecBb** (IMessageScheduler *sched)
- virtual void processEvent (IMessage *msg) throw (EUnhandledMessage)

  Process an event message directed to this message processing unit.
- virtual void processTimer (IMessage *msg) throw (EUnhandledMessage)

  Process a timer message directed to this message processing unit.
- virtual void processOutgoing (IMessage *msg) throw (EUnhandledMessage)

  Process an outgoing message directed towards the network.
- virtual void processIncoming (IMessage *msg) throw (EUnhandledMessage)

  Process an incoming message directed towards the application.
- virtual Id getId ()

  Return ID of the building block.

7.10.1 Detailed Description

FEC building block.

7.10.2 Member Function Documentation

7.10.2.1 void CFecBb::processEvent (IMessage * *msg*) throw (EUnhandledMessage) `[virtual]`

Process an event message directed to this message processing unit.

Parameters:

msg Pointer to message

Implements IMessageProcessor.

7.10.2.2 void CFecBb::processTimer (IMessage * *msg*) throw (EUnhandledMessage) [virtual]

Process a timer message directed to this message processing unit.

Parameters:

msg Pointer to message

Implements IMessageProcessor.

7.10.2.3 void CFecBb::processOutgoing (IMessage * *msg*) throw (EUnhandledMessage) [virtual]

Process an outgoing message directed towards the network.

Parameters:

msg Pointer to message

Implements IMessageProcessor.

7.10.2.4 void CFecBb::processIncoming (IMessage * *msg*) throw (EUnhandledMessage) [virtual]

Process an incoming message directed towards the application.

Parameters:

msg Pointer to message

Implements IMessageProcessor.

The documentation for this class was generated from the following files:

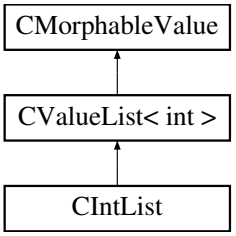
- src/buildingBlocks/fecBb.h
- src/buildingBlocks/fecBb.cpp

7.11 CIntList Class Reference

Facade for an int list.

`#include <morphableValue.h>`

Inheritance diagram for CIntList::



7.11.1 Detailed Description

Facade for an int list.

The documentation for this class was generated from the following file:

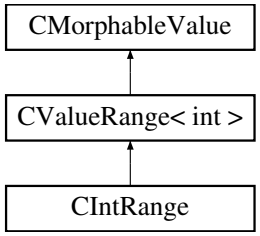
- include/morphableValue.h

7.12 CIntRange Class Reference

Facade for int ranges.

`#include <morphableValue.h>`

Inheritance diagram for CIntRange::



Public Member Functions

- **CIntRange** (int lower=0, int upper=0)

7.12.1 Detailed Description

Facade for int ranges.

The documentation for this class was generated from the following file:

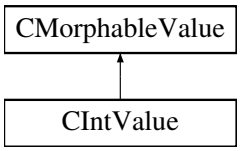
- include/morphableValue.h

7.13 CIntValue Class Reference

Facade for int values.

`#include <morphableValue.h>`

Inheritance diagram for CIntValue::



Public Member Functions

- **CIntValue** (int v=0)
- int & **value** ()

7.13.1 Detailed Description

Facade for int values.

The documentation for this class was generated from the following file:

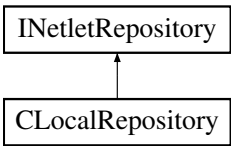
- include/morphableValue.h

7.14 CLocalRepository Class Reference

Local Netlet repository.

`#include <localRepository.h>`

Inheritance diagram for CLocalRepository::



Public Member Functions

- **CLocalRepository** (CNodeArchitecture *nodeArch)
- virtual void initialize ()

  Initialize the repository, e.g. load Netlets etc.

Protected Member Functions

- std::string getSoFileName (const std::string soName)

  Return shared object file name.

- virtual void loadMultiplexers ()

  Load available multiplexers.

- virtual void loadNetlets ()

  Load available Netlets.

7.14.1 Detailed Description

Local Netlet repository.

Loads Netlets available as shared object files.

7.14.2 Member Function Documentation

7.14.2.1 std::string CLocalRepository::getSoFileName (const std::string *soName*) [protected]

Return shared object file name.

Get OS dependent file name of shared object (library).

TODO: Only UNIX type naming is supported. Windows (*.dll) and others should be added.

Parameters:

soName Name of shared object (e.g. name of netlet)

7.14.2.2 void CLocalRepository::loadMultiplexers () [protected, virtual]

Load available multiplexers.

Scan for and load multiplexers.

7.14.2.3 void CLocalRepository::loadNetlets () [protected, virtual]

Load available Netlets.

Scan for and load Netlets.

The documentation for this class was generated from the following files:

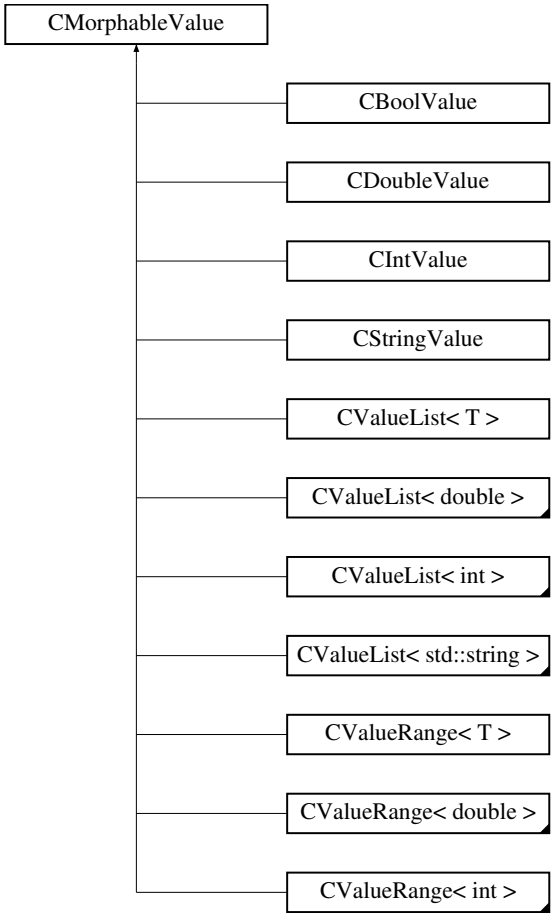
- src/daemon/localRepository.h
- src/daemon/localRepository.cpp

7.15 CMorphableValue Class Reference

Base class for morphable values, e.g. properties, config parameters etc.

`#include <morphableValue.h>`

Inheritance diagram for CMorphableValue::



Public Types

- enum **ValueType** {
  vt_none, **vt_bool**, **vt_int**, **vt_double**,
  vt_string }
- enum **ContainerType** { **ct_simple**, **ct_range**, **ct_list** }

Public Member Functions

- **CMorphableValue** (ValueType vt=vt_none, ContainerType ct=ct_simple)
- ValueType **getValueType** ()
- ContainerType **getContainerType** ()
- template<class T>
  T ∗ **cast** () throw (EValueTypeMismatch)

Protected Attributes

- ValueType **valueType**
- ContainerType **containerType**

Classes

- class **EContainerTypeMismatch**
- class **EValueTypeMismatch**
- class **Range**

7.15.1 Detailed Description

Base class for morphable values, e.g. properties, config parameters etc.

This is not a variant ;)

The documentation for this class was generated from the following file:

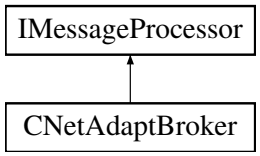
- include/morphableValue.h

7.16 CNetAdaptBroker Class Reference

Network Adaptor Broker.

`#include <netAdaptBroker.h>`

Inheritance diagram for CNetAdaptBroker::



Public Member Functions

- CNetAdaptBroker (IMessageScheduler *sched)
- virtual ~CNetAdaptBroker ()
- virtual void processEvent (IMessage *msg) throw (EUnhandledMessage)

  Process an event message directed to this message processing unit.

- virtual void processTimer (IMessage *msg) throw (EUnhandledMessage)

  Process a timer message directed to this message processing unit.

- virtual void processOutgoing (IMessage *msg) throw (EUnhandledMessage)

  Process an outgoing message directed towards the network.

- virtual void processIncoming (IMessage *msg) throw (EUnhandledMessage)

  Process an incoming message directed towards the application.

- void registerNetAdapt (INetAdapt *netAdapt)

  Register a network adaptor.

- std::list< INetAdapt * > & getNetAdapts (std::string archName)

  Return a list network adaptors fitting to the given architecture.

Protected Attributes

- std::map< std::string, std::list< INetAdapt * > > netAdapts

  List of adaptors per architecture.

7.16.1 Detailed Description

Network Adaptor Broker.

TODO: Maybe this should not be an IMessageProcessor. Instead, the Multiplexers should connect directly to the network accesses.

7.16.2 Constructor & Destructor Documentation

7.16.2.1 CNetAdaptBroker::CNetAdaptBroker (IMessageScheduler * *sched*)

Constructor

7.16.2.2 CNetAdaptBroker::~CNetAdaptBroker () [virtual]

Destructor

7.16.3 Member Function Documentation

7.16.3.1 void CNetAdaptBroker::processEvent (IMessage * *msg*) throw (EUnhandledMessage) [virtual]

Process an event message directed to this message processing unit.

Parameters:

msg Pointer to message

Implements IMessageProcessor.

7.16.3.2 void CNetAdaptBroker::processTimer (IMessage * *msg*) throw (EUnhandledMessage) [virtual]

Process a timer message directed to this message processing unit.

Parameters:

msg Pointer to message

Implements IMessageProcessor.

7.16.3.3 void CNetAdaptBroker::processOutgoing (IMessage * *msg*) throw (EUnhandledMessage) [virtual]

Process an outgoing message directed towards the network.

Parameters:

msg Pointer to message

Implements IMessageProcessor.

7.16.3.4 void CNetAdaptBroker::processIncoming (IMessage * *msg*) throw (EUnhandledMessage) [virtual]

Process an incoming message directed towards the application.

Parameters:

msg Pointer to message

Implements IMessageProcessor.

7.16.3.5 void CNetAdaptBroker::registerNetAdapt (INetAdapt * *netAdapt*)

Register a network adaptor.

Parameters:

archName Name of architecture spoken on the network the adaptor connects to

netAdapt Pointer to INetAdapt

7.16.3.6 std::list< INetAdapt * > & CNetAdaptBroker::getNetAdapts (std::string *archName*)

Return a list network adaptors fitting to the given architecture.

Parameters:

archName Name of architecture

Returns:

List of INetAdapt pointers

The documentation for this class was generated from the following files:

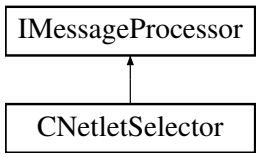
- src/daemon/netAdaptBroker.h
- src/daemon/netAdaptBroker.cpp

7.17 CNetletSelector Class Reference

Implements Netlet selection functionality.

`#include <netletSelector.h>`

Inheritance diagram for CNetletSelector::



Public Member Functions

- CNetletSelector (CNodeArchitecture *nodeA, IMessageScheduler *sched)

  Constructor.

- virtual ~CNetletSelector ()

  Destructor.

- virtual void processEvent (IMessage *msg) throw (EUnhandledMessage)

  Process an event message directed to this message processing unit.

- virtual void processTimer (IMessage *msg) throw (EUnhandledMessage)

  Process a timer message directed to this message processing unit.

- virtual void processOutgoing (IMessage *msg) throw (EUnhandledMessage)

  Process an outgoing message directed towards the network.

- virtual void processIncoming (IMessage *msg) throw (EUnhandledMessage)

  Process an incoming message directed towards the application.

- virtual void registerAppConnector (IAppConnector *appConn)

  Register an application connector ("socket").

- virtual LocalConnId lookupService (ServiceId serviceId)

  Lookup a service ID ("port") and return a local connection ID which identifies the application connector (thus, the application providing the service).

- virtual void getRegisteredServices (std::list< ServiceId > &services)

  Fills the provided list with known service IDs. The list will be cleared if it is not empty.

- virtual void registerNameAddrMapper (INameAddrMapper *mapper)

  Register a name/addr mapper that will be consulted in case an app connector has no Netlet associated with it yet.

- virtual void unregisterNameAddrMapper (INameAddrMapper *mapper)

  Unregister a name/addr mapper.

Protected Member Functions

- virtual void registerService (ServiceId serviceId, IAppConnector *appConn)

  Register an application-layer service. serviceId is an arbitrary string for now and can be seen as some sort of "port number".

- virtual void unregisterService (ServiceId serviceId, IAppConnector *appConn)

  Unregister an application-layer service. serviceId is an arbitrary string for now and can be seen as some sort of "port number".

Protected Attributes

- CNodeArchitecture * **nodeArch**
- unsigned int **lastAppId**
- std::map< LocalConnId, IAppConnector * > appConns

  id -> connector

- std::map< IAppConnector *, LocalConnId > appIds

  connector -> id (reverse lookup)

- std::map< ServiceId, IAppConnector * > services

  TODO map with multiple entries?

- std::list< INameAddrMapper * > nameAddrMappers

  list of known name/addr mappers

7.17.1 Detailed Description

Implements Netlet selection functionality.

7.17.2 Member Function Documentation

7.17.2.1 void CNetletSelector::registerService (ServiceId *serviceId*, IAppConnector * *appConn*) [protected, virtual]

Register an application-layer service. serviceId is an arbitrary string for now and can be seen as some sort of "port number".

Parameters:

serviceId String identifying the service

appConn Application connector ("socket")

7.17.2.2 void CNetletSelector::unregisterService (ServiceId *serviceId*, IAppConnector * *appConn*) [protected, virtual]

Unregister an application-layer service. serviceId is an arbitrary string for now and can be seen as some sort of "port number".

Parameters:

serviceId String identifying the service

appConn Application connector ("socket")

7.17.2.3 void CNetletSelector::processEvent (IMessage * *msg*) throw (EUnhandledMessage) [virtual]

Process an event message directed to this message processing unit.

Parameters:

msg Pointer to message

Implements IMessageProcessor.

7.17.2.4 void CNetletSelector::processTimer (IMessage * *msg*) throw (EUnhandledMessage) [virtual]

Process a timer message directed to this message processing unit.

Parameters:

msg Pointer to message

Implements IMessageProcessor.

7.17.2.5 void CNetletSelector::processOutgoing (IMessage * *msg*) throw (EUnhandledMessage) [virtual]

Process an outgoing message directed towards the network.

Parameters:

msg Pointer to message

Implements IMessageProcessor.

7.17.2.6 void CNetletSelector::processIncoming (IMessage * *msg*) throw (EUnhandledMessage) [virtual]

Process an incoming message directed towards the application.

Parameters:

msg Pointer to message

Implements IMessageProcessor.

7.17.2.7 LocalConnId CNetletSelector::lookupService (ServiceId *serviceId*) [virtual]

Lookup a service ID ("port") and return a local connection ID which identifies the application connector (thus, the application providing the service).

Parameters:

serviceId String identifying the service

Returns:

Local connection ID of an application providing the service

7.17.2.8 void CNetletSelector::getRegisteredServices (std::list< ServiceId > & *services*) [virtual]

Fills the provided list with known service IDs. The list will be cleared if it is not empty.

Parameters:

services Reference to the list that is to be filled

The documentation for this class was generated from the following files:

- src/daemon/netletSelector.h
- src/daemon/netletSelector.cpp

7.18 CNodeArchitecture Class Reference

Node architecture daemon class.

`#include <nodeArchitecture.h>`

Public Member Functions

- CNodeArchitecture (ISystemWrapper *sys)
  Constructor.
- virtual ~CNodeArchitecture ()
  Destructor.
- void init ()
  initialize daemon
- std::string getNodeName ()
  Return an arbitrary name for the node.
- CNetAdaptBroker * getNetAdaptBroker ()
  Return the NA Broker.
- CNetletSelector * getNetletSelector ()
  Return the Netlet selector.
- virtual void registerAppConnector (IAppConnector *appConn)
  Register an application connector ("socket").
- virtual LocalConnId lookupService (ServiceId serviceId)
  Lookup a service ID ("port") and return a local connection ID which identifies the application connector (thus, the application providing the service).
- virtual void getRegisteredServices (std::list< ServiceId > &services)
  Fills the provided list with known service IDs. The list will be cleared if it is not empty.
- virtual void getNetlets (std::string archName, std::list< INetlet * > &netletList)
  Get all instantiated Netlets belonging to architecture archName.
- virtual INetletMultiplexer * getMultiplexer (std::string archName)
  Return the multiplexer for the given architecture.
- virtual INetlet * getInstanceOf (const std::string &netletName)
  Returns an instance of the given Netlet.

Public Attributes

- IMessageScheduler * defaultScheduler
  Pointer to default scheduler.

7.18.1 Detailed Description

Node architecture daemon class.

Conceptually, this is a singleton class. But since it may run in a simulator, more than one instance may exist in the same executable.

TODO: All public methods should be thread-safe.

7.18.2 Constructor & Destructor Documentation

7.18.2.1 CNodeArchitecture::CNodeArchitecture (ISystemWrapper * *sys*)

Constructor.

Set up initial daemon configuration

7.18.2.2 CNodeArchitecture::~CNodeArchitecture () [virtual]

Destructor.

Constructor.

7.18.3 Member Function Documentation

7.18.3.1 void CNodeArchitecture::init ()

initialize daemon

Initialize daemon.

Loading multiplexers, Netlets, initialize network accesses, application interface, etc.

7.18.3.2 LocalConnId CNodeArchitecture::lookupService (ServiceId *serviceId*) [virtual]

Lookup a service ID ("port") and return a local connection ID which identifies the application connector (thus, the application providing the service).

Parameters:

serviceId String identifying the service

Returns:

Local connection ID of an application providing the service

7.18.3.3 void CNodeArchitecture::getRegisteredServices (std::list< ServiceId > & *services*) [virtual]

Fills the provided list with known service IDs. The list will be cleared if it is not empty.

Parameters:

services Reference to the list that is to be filled

7.18.3.4 void CNodeArchitecture::getNetlets (std::string *archName*, std::list< INetlet * > & *netletList*) [virtual]

Get all instantiated Netlets belonging to architecture archName.

Returns an existing instance of netletName.

Parameters:

archName Name of architecture

list List to be filled (will be cleared if non-empty)

ID of the Netlet instance requested.

Returns:

An existing instance of netletName. NULL if there is no such instance. Get all instantiated Netlets belonging to architecture archName.

Parameters:

archName Name of architecture

list List to be filled (will be cleared if non-empty)

7.18.3.5 INetletMultiplexer * CNodeArchitecture::getMultiplexer (std::string *archName*) [virtual]

Return the multiplexer for the given architecture.

Parameters:

archName Name or architecture

The documentation for this class was generated from the following files:

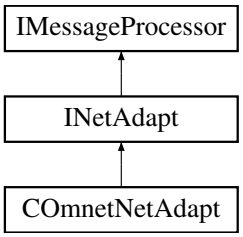
- src/daemon/nodeArchitecture.h
- src/daemon/nodeArchitecture.cpp

7.19 COmnetNetAdapt Class Reference

Simple OMNeT++ network access.

`#include <netAdaptOmnet.h>`

Inheritance diagram for COmnetNetAdapt::



Public Member Functions

- **COmnetNetAdapt** (CNodeArchitecture *nodeA, CSystemOmnet *sys, std::string gate, int gateIndex=-1)
- virtual void processEvent (IMessage *msg) throw (EUnhandledMessage)

  Process an event message directed to this message processing unit.

- virtual void processTimer (IMessage *msg) throw (EUnhandledMessage)

  Process a timer message directed to this message processing unit.

- virtual void processOutgoing (IMessage *msg) throw (EUnhandledMessage)

  Process an outgoing message directed towards the network.

- virtual void processIncoming (IMessage *msg) throw (EUnhandledMessage)

  Process an incoming message directed towards the application.

- virtual std::string getName ()

  Return a name for this Network Access.

- void handleOmnetPacket (COmnetPacket *pkt)

Protected Attributes

- CSystemOmnet * **sys**
- std::string gate

  cSimpleModule gate to use

- int gateIndex

  Index of gate vector.

- std::string **outputGate**
- std::string **inputGate**

7.19.1 Detailed Description

Simple OMNeT++ network access.

7.19.2 Member Function Documentation

7.19.2.1 void COmnetNetAdapt::processEvent (IMessage * *msg*) throw (EUnhandledMessage) [virtual]

Process an event message directed to this message processing unit.

Parameters:

msg Pointer to message

Implements IMessageProcessor.

7.19.2.2 void COmnetNetAdapt::processTimer (IMessage * *msg*) throw (EUnhandledMessage) [virtual]

Process a timer message directed to this message processing unit.

Parameters:

msg Pointer to message

Implements IMessageProcessor.

7.19.2.3 void COmnetNetAdapt::processOutgoing (IMessage * *msg*) throw (EUnhandledMessage) [virtual]

Process an outgoing message directed towards the network.

Parameters:

msg Pointer to message

Implements IMessageProcessor.

7.19.2.4 void COmnetNetAdapt::processIncoming (IMessage * *msg*) throw (EUnhandledMessage) [virtual]

Process an incoming message directed towards the application.

Parameters:

msg Pointer to message

Implements IMessageProcessor.

7.19.2.5 void COmnetNetAdapt::handleOmnetPacket (COmnetPacket * *opkt*)

Invoked when a packet is received via omnet

The documentation for this class was generated from the following files:

- src/targets/omnetpp/netAdaptOmnet.h
- src/targets/omnetpp/netAdaptOmnet.cpp

7.20 COmnetPacket Class Reference

OMNeT++ packet.

`#include <netAdaptOmnet.h>`

Public Member Functions

- **COmnetPacket** (CPacket *dataUnit=NULL)

Public Attributes

- CPacket * **dataUnit**
- CSerialBuffer * **buffer**
- unsigned char * **frame**
- unsigned int **frameSize**

7.20.1 Detailed Description

OMNeT++ packet.

The documentation for this class was generated from the following files:

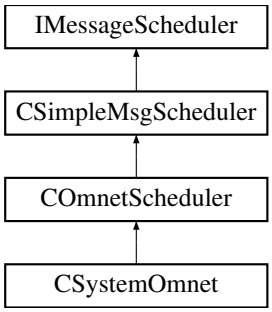
- src/targets/omnetpp/netAdaptOmnet.h
- src/targets/omnetpp/netAdaptOmnet.cpp

7.21 COmnetScheduler Class Reference

OMNeT implementation of a message scheduler.

`#include <messagesOmnet.h>`

Inheritance diagram for COmnetScheduler::



Public Member Functions

- virtual void setTimer (CTimer *timer)
  Set a single shot time.

Protected Member Functions

- virtual void initialize ()
  From cSimpleModule.
- virtual void handleMessage (cMessage *msg)
  Handle an OMNeT message.

Protected Attributes

- EventMap events
  List of pending events.

7.21.1 Detailed Description

OMNeT implementation of a message scheduler.

7.21.2 Member Function Documentation

7.21.2.1 void COmnetScheduler::initialize () [protected, virtual]

From cSimpleModule.

cSimpleModule::initialize()

Reimplemented in CSystemOmnet.

7.21.2.2 void COmnetScheduler::handleMessage (cMessage * *msg*) [protected, virtual]

Handle an OMNeT message.

cSimpleModule::handleMessage()

Reimplemented in CSystemOmnet.

7.21.2.3 void COmnetScheduler::setTimer (CTimer * *timer*) [virtual]

Set a single shot time.

Parameters:

timer Timer to be set

Reimplemented from CSimpleMsgScheduler.

The documentation for this class was generated from the following files:

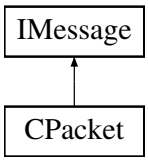
- src/targets/omnetpp/messagesOmnet.h
- src/targets/omnetpp/messagesOmnet.cpp

7.22 CPacket Class Reference

Container for a data frame.

`#include <packets.h>`

Inheritance diagram for CPacket::



Public Member Functions

- CPacket (IMessageProcessor *from=NULL, IMessageProcessor *to=NULL, Type type=t_outgoingPacket, CSerialBuffer *serialBuffer=NULL)

  Constructor.

- virtual ~CPacket ()

  Destructor.

- virtual INetAdapt * **getNetAdapt** ()
- virtual void **setNetAdapt** (INetAdapt *netAdapt)
- void pushHeader (IHeader *header)

  Prepends a header.

- template<class T>
  T * popHeader ()

  Removes the front most header and returns it.

- void serialize (CSerialBuffer *buffer)

  Serialize the packet.

- CSerialBuffer * **getSerialBuffer** ()

Protected Attributes

- INetAdapt * **netAdapt**

7.22.1 Detailed Description

Container for a data frame.

7.22.2 Constructor & Destructor Documentation

7.22.2.1 CPacket::CPacket (IMessageProcessor * *from* = NULL, IMessageProcessor * *to* = NULL, Type *type* = t_outgoingPacket, CSerialBuffer * *serialBuffer* = NULL) [inline]

Constructor.

Parameters:

from Which entity sent sent this packet

to Which entity will receive this packet

type Message type

serialBuffer Buffer to use for (de-)serialization

7.22.3 Member Function Documentation

7.22.3.1 template<class T> T* CPacket::popHeader () [inline]

Removes the front most header and returns it.

The template parameter identifies the header class to use.

The documentation for this class was generated from the following file:

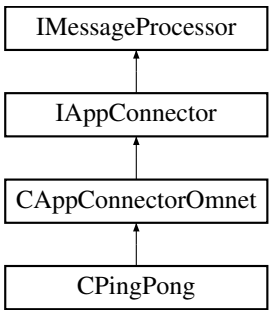
- include/packets.h

7.23 CPingPong Class Reference

Ping pong example application for OMNeT.

`#include <pingPong.h>`

Inheritance diagram for CPingPong::



Public Member Functions

- **CPingPong** (CNodeArchitecture *nodeArch, IMessageScheduler *sched)
- virtual void processEvent (IMessage *msg) throw (EUnhandledMessage)

  Process an event message directed to this message processing unit.

- virtual void processTimer (IMessage *msg) throw (EUnhandledMessage)

  Process a timer message directed to this message processing unit.

- virtual void processOutgoing (IMessage *msg) throw (EUnhandledMessage)

  Process an outgoing message directed towards the network.

- virtual void processIncoming (IMessage *msg) throw (EUnhandledMessage)

  Process an incoming message directed towards the application.

- void handlePingTimer ()

  Handle ping timeout.

7.23.1 Detailed Description

Ping pong example application for OMNeT.

Note: In Omnet, every app can inherited directly from the app connector. On real system, one may want to consider the connector only as a proxy class.

7.23.2 Member Function Documentation

7.23.2.1 void CPingPong::processEvent (IMessage * *msg*) throw (EUnhandledMessage) [virtual]

Process an event message directed to this message processing unit.

Parameters:

msg Pointer to message

Implements IMessageProcessor.

7.23.2.2 void CPingPong::processTimer (IMessage * *msg*) throw (EUnhandledMessage) [virtual]

Process a timer message directed to this message processing unit.

Parameters:

msg Pointer to message

Implements IMessageProcessor.

7.23.2.3 void CPingPong::processOutgoing (IMessage * *msg*) throw (EUnhandledMessage) [virtual]

Process an outgoing message directed towards the network.

Parameters:

msg Pointer to message

Implements IMessageProcessor.

7.23.2.4 void CPingPong::processIncoming (IMessage * *msg*) throw (EUnhandledMessage) [virtual]

Process an incoming message directed towards the application.

Parameters:

msg Pointer to message

Implements IMessageProcessor.

The documentation for this class was generated from the following files:

- src/targets/omnetpp/apps/pingPong.h
- src/targets/omnetpp/apps/pingPong.cpp

7.24 CSerialBuffer Class Reference

Buffer for packet serialization.

`#include <packets.h>`

Public Member Functions

- CSerialBuffer (unsigned int reservedSize=2048)

  Buffer for outgoing packets.

- CSerialBuffer (unsigned char *buffer, unsigned int size)

  Buffer for incoming packets.

- void **dbgPrintBuffer** ()
- unsigned char * **getBuffer** ()
- unsigned int **getSize** ()
- void **copyRemainingBufferFrom** (CSerialBuffer *sourceBuffer)
- unsigned char * **getRemainingBuffer** ()
- unsigned int **getRemainingSize** ()
- void **push_ulong** (unsigned long h)
- void **push_ushort** (unsigned short h)
- void **push_uchar** (unsigned char h)
- void **push_string** (std::string s)
- unsigned long **pop_ulong** ()
- unsigned short **pop_ushort** ()
- unsigned char **pop_uchar** ()
- std::string **pop_string** (unsigned int size)

7.24.1 Detailed Description

Buffer for packet serialization.

The documentation for this class was generated from the following file:

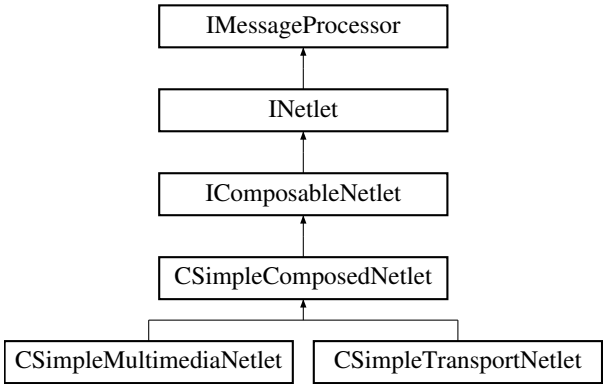
- include/packets.h

7.25 CSimpleComposedNetlet Class Reference

A Netlet taking its actual building block configuration from a generated source file.

`#include <simpleComposedNetlet.h>`

Inheritance diagram for CSimpleComposedNetlet::



Public Member Functions

- CSimpleComposedNetlet (CNodeArchitecture *nodeA, IMessageScheduler *sched)
- virtual ~CSimpleComposedNetlet ()
- virtual void processEvent (IMessage *msg) throw (EUnhandledMessage)

  Process an event message directed to this message processing unit.

- virtual void processTimer (IMessage *msg) throw (EUnhandledMessage)

  Process a timer message directed to this message processing unit.

- virtual void processOutgoing (IMessage *msg) throw (EUnhandledMessage)

  Process an outgoing message directed towards the network.

- virtual void processIncoming (IMessage *msg) throw (EUnhandledMessage)

  Process an incoming message directed towards the application.

- SimpleHash hash (const std::string &str)

7.25.1 Detailed Description

A Netlet taking its actual building block configuration from a generated source file.

This is a base class for composed Netlets within the Simple Architecture.

Per architecture, a single template for a composable Netlet would be sufficient.

7.25.2 Constructor & Destructor Documentation

7.25.2.1 CSimpleComposedNetlet::CSimpleComposedNetlet (CNodeArchitecture * *nodeA*, IMessageScheduler * *sched*)

Constructor

7.25.2.2 CSimpleComposedNetlet::~CSimpleComposedNetlet () [virtual]

Destructor

7.25.3 Member Function Documentation

7.25.3.1 void CSimpleComposedNetlet::processEvent (IMessage * *msg*) throw (EUnhandledMessage) [virtual]

Process an event message directed to this message processing unit.

Parameters:

msg Pointer to message

Implements IMessageProcessor.

7.25.3.2 void CSimpleComposedNetlet::processTimer (IMessage * *msg*) throw (EUnhandledMessage) [virtual]

Process a timer message directed to this message processing unit.

Parameters:

msg Pointer to message

Implements IMessageProcessor.

7.25.3.3 void CSimpleComposedNetlet::processOutgoing (IMessage * *msg*) throw (EUnhandledMessage) [virtual]

Process an outgoing message directed towards the network.

Parameters:

msg Pointer to message

Implements IMessageProcessor.

7.25.3.4 void CSimpleComposedNetlet::processIncoming (IMessage * *msg*) throw (EUnhandledMessage) [virtual]

Process an incoming message directed towards the application.

Parameters:

msg Pointer to message

Implements IMessageProcessor.

7.25.3.5 SimpleHash CSimpleComposedNetlet::hash (const std::string & *str*)

http://codesnippets.joyent.com/user/wastepixel/tag/hashing

TODO: to be replaced by something better

The documentation for this class was generated from the following files:

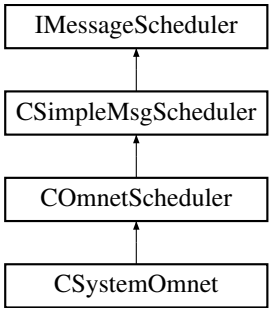
- src/netlets/simpleArch/simpleComposedNetlet.h
- src/netlets/simpleArch/simpleComposedNetlet.cpp

7.26 CSimpleMsgScheduler Class Reference

Simple scheduler. May only be used if ∗every∗ message processor runs in the same thread.

`#include <messages.h>`

Inheritance diagram for CSimpleMsgScheduler::



Public Member Functions

- virtual void sendMessage (IMessage ∗msg) throw (EUnknowMessageProcessor, EMessageLoop)

  Send a message to another processing unit.

- virtual void setTimer (CTimer ∗timer)

  Set a single shot time.

- virtual void processMessages () throw (EUnknowMessageProcessor)

  Process all waiting messages and call respective receivers. Returns if all queues are empty.

7.26.1 Detailed Description

Simple scheduler. May only be used if ∗every∗ message processor runs in the same thread.

7.26.2 Member Function Documentation

7.26.2.1 virtual void CSimpleMsgScheduler::sendMessage (IMessage ∗ *msg*) throw (EUnknowMessageProcessor, EMessageLoop) `[inline, virtual]`

Send a message to another processing unit.

Parameters:

msg Message to send

Implements IMessageScheduler.

7.26.2.2 virtual void CSimpleMsgScheduler::setTimer (CTimer * *timer*) [inline, virtual]

Set a single shot time.

Parameters:

timer Timer to be set

Implements IMessageScheduler.

Reimplemented in COmnetScheduler.

The documentation for this class was generated from the following file:

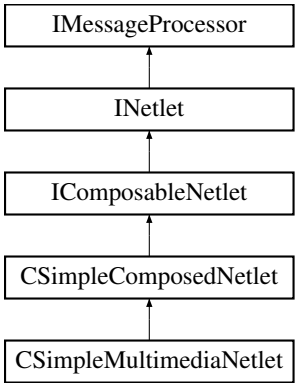
- include/messages.h

7.27 CSimpleMultimediaNetlet Class Reference

A Netlet taking its actual building block configuration from a generated source file.

`#include <simpleMultimediaNetlet.h>`

Inheritance diagram for CSimpleMultimediaNetlet::



Public Member Functions

- CSimpleMultimediaNetlet (CNodeArchitecture *nodeA, IMessageScheduler *sched)
- virtual ~CSimpleMultimediaNetlet ()
- virtual INetletMetaData * getMetaData ()

  Returns the Netlet's meta data.

7.27.1 Detailed Description

A Netlet taking its actual building block configuration from a generated source file.

Per architecture, a single template for a composable Netlet would be sufficient.

7.27.2 Constructor & Destructor Documentation

7.27.2.1 CSimpleMultimediaNetlet::CSimpleMultimediaNetlet (CNodeArchitecture * *nodeA*, IMessageScheduler * *sched*)

Constructor

7.27.2.2 CSimpleMultimediaNetlet::~CSimpleMultimediaNetlet () `[virtual]`

Destructor

The documentation for this class was generated from the following files:

- src/netlets/simpleArch/simpleMultimediaNetlet.h
- src/netlets/simpleArch/simpleMultimediaNetlet.cpp

7.28 CSimpleMultimediaNetletMetaData Class Reference

Simple composed Netlet meta data.

`#include <simpleMultimediaNetlet.h>`

Public Member Functions

- CSimpleMultimediaNetletMetaData ()
- virtual ~CSimpleMultimediaNetletMetaData ()
- virtual const std::string & getArchName ()
- virtual const std::string & getName ()
- virtual bool isControlNetlet ()

  Returns true if the Netlet does not offer any transport service for applications, false otherwise.

- virtual INetlet * createNetlet (CNodeArchitecture *nodeA, IMessageScheduler *sched)

7.28.1 Detailed Description

Simple composed Netlet meta data.

The Netlet meta data class has two purposes: On one hand, it describes the Netlet's properties, capabilities, etc., on the other hand, it provides a factory functions that will be used by the node architecture daemon to instantiate new Netlets of this type.

7.28.2 Constructor & Destructor Documentation

7.28.2.1 CSimpleMultimediaNetletMetaData::CSimpleMultimediaNetletMetaData ()

Constructor

7.28.2.2 CSimpleMultimediaNetletMetaData::~CSimpleMultimediaNetletMetaData () [virtual]

Destructor

7.28.3 Member Function Documentation

7.28.3.1 const std::string & CSimpleMultimediaNetletMetaData::getArchName () [virtual]

Return Netlet name

7.28.3.2 const std::string & CSimpleMultimediaNetletMetaData::getName () [virtual]

Return Netlet name

7.28.3.3 INetlet * CSimpleMultimediaNetletMetaData::createNetlet (CNodeArchitecture * *nodeA*, IMessageScheduler * *sched*) [virtual]

Create an instance of the Netlet

The documentation for this class was generated from the following files:

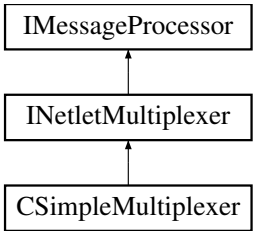
- src/netlets/simpleArch/simpleMultimediaNetlet.h
- src/netlets/simpleArch/simpleMultimediaNetlet.cpp

7.29 CSimpleMultiplexer Class Reference

Simple Netlet multiplexer.

`#include <simpleMultiplexer.h>`

Inheritance diagram for CSimpleMultiplexer::



Public Member Functions

- CSimpleMultiplexer (IMultiplexerMetaData *metaData, CNodeArchitecture *nodeA, IMessageScheduler *sched)
- virtual ~CSimpleMultiplexer ()
- virtual void processEvent (IMessage *msg) throw (EUnhandledMessage)

  Process an event message directed to this message processing unit.

- virtual void processTimer (IMessage *msg) throw (EUnhandledMessage)

  Process a timer message directed to this message processing unit.

- virtual void processOutgoing (IMessage *msg) throw (EUnhandledMessage)

  Process an outgoing message directed towards the network.

- virtual void processIncoming (IMessage *msg) throw (EUnhandledMessage)

  Process an incoming message directed towards the application.

- virtual void refreshNetlets ()

  Called if new Netlets of this architecture were added to the system.

- virtual SimpleFib & **getFib** ()
- virtual void dbgPrintFib ()

  Print the FIB for debugging purposes.

- SimpleHash hash (const std::string &str)

Protected Attributes

- std::map< SimpleHash, INetlet * > **hashedNetlets**
- CSimpleNameAddrMapper * **nameAddrMapper**
- SimpleFib fib

  Forwarding information base; TODO: access to this should be thread-safe...

7.29.1 Detailed Description

Simple Netlet multiplexer.

7.29.2 Constructor & Destructor Documentation

7.29.2.1 CSimpleMultiplexer::CSimpleMultiplexer (IMultiplexerMetaData * *metaData*, CNodeArchitecture * *nodeA*, IMessageScheduler * *sched*)

Constructor

7.29.2.2 CSimpleMultiplexer::~CSimpleMultiplexer () [virtual]

Constructor

7.29.3 Member Function Documentation

7.29.3.1 void CSimpleMultiplexer::processEvent (IMessage * *msg*) throw (EUnhandledMessage) [virtual]

Process an event message directed to this message processing unit.

Parameters:

msg Pointer to message

Implements IMessageProcessor.

7.29.3.2 void CSimpleMultiplexer::processTimer (IMessage * *msg*) throw (EUnhandledMessage) [virtual]

Process a timer message directed to this message processing unit.

Parameters:

msg Pointer to message

Implements IMessageProcessor.

7.29.3.3 void CSimpleMultiplexer::processOutgoing (IMessage * *msg*) throw (EUnhandledMessage) [virtual]

Process an outgoing message directed towards the network.

Parameters:

msg Pointer to message

Implements IMessageProcessor.

7.29.3.4 void CSimpleMultiplexer::processIncoming (IMessage * *msg*) throw (EUnhandledMessage) [virtual]

Process an incoming message directed towards the application.

Parameters:

msg Pointer to message

Implements IMessageProcessor.

7.29.3.5 SimpleHash CSimpleMultiplexer::hash (const std::string & *str*)

http://codesnippets.joyent.com/user/wastepixel/tag/hashing

TODO: to be replaced by something better

The documentation for this class was generated from the following files:

- src/netlets/simpleArch/simpleMultiplexer.h
- src/netlets/simpleArch/simpleMultiplexer.cpp

7.30 CSimpleMultiplexerMetaData Class Reference

Simple multiplexer meta data class.

`#include <simpleMultiplexer.h>`

Public Member Functions

- CSimpleMultiplexerMetaData ()
- virtual ~CSimpleMultiplexerMetaData ()
- virtual std::string getArchName ()
- virtual INetletMultiplexer * createMultiplexer (CNodeArchitecture *nodeA, IMessageScheduler *sched)

  factory function

7.30.1 Detailed Description

Simple multiplexer meta data class.

The multiplexer meta data class has two purposes: On one hand, it describes the architecure's properties, on the other hand, it provides a factory functions that will be used by the node architecture daemon to instantiate new "architecture".

7.30.2 Constructor & Destructor Documentation

7.30.2.1 CSimpleMultiplexerMetaData::CSimpleMultiplexerMetaData ()

Constructor

7.30.2.2 CSimpleMultiplexerMetaData::~CSimpleMultiplexerMetaData () `[virtual]`

Destructor

7.30.3 Member Function Documentation

7.30.3.1 std::string CSimpleMultiplexerMetaData::getArchName () `[virtual]`

Return Netlet name

7.30.3.2 INetletMultiplexer * CSimpleMultiplexerMetaData::createMultiplexer (CNodeArchitecture * *nodeA*, IMessageScheduler * *sched*) `[virtual]`

factory function

Create an instance of the Netlet

The documentation for this class was generated from the following files:

- src/netlets/simpleArch/simpleMultiplexer.h

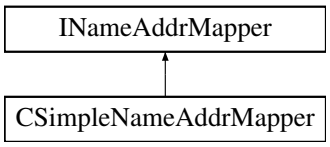
- src/netlets/simpleArch/simpleMultiplexer.cpp

7.31 CSimpleNameAddrMapper Class Reference

Name/address mapper implementation for SimpleArchitecture.

`#include <simpleMultiplexer.h>`

Inheritance diagram for CSimpleNameAddrMapper::



Public Member Functions

- CSimpleNameAddrMapper (CNodeArchitecture *nodeA)

  Constructor.

- virtual ~CSimpleNameAddrMapper ()

  Destructor.

- virtual void getPotentialNetlets (const std::string &name, std::list< INetletMetaData * > &netletList)

  Adds meta-data classes to the supplied list, providing information about Netlets that are able to communicate with the given name.

- virtual std::string resolve (const std::string &name)

  Resolve a name into an address.

7.31.1 Detailed Description

Name/address mapper implementation for SimpleArchitecture.

7.31.2 Member Function Documentation

7.31.2.1 void CSimpleNameAddrMapper::getPotentialNetlets (const std::string & *name*, std::list< INetletMetaData * > & *netletList*) [virtual]

Adds meta-data classes to the supplied list, providing information about Netlets that are able to communicate with the given name.

Currently, we assume that every non-control Netlet can handle all given names.

Parameters:

name Name of the peer or object that should be resolved.

netletList List to which the Netlets' meta data is to be added.

The documentation for this class was generated from the following files:

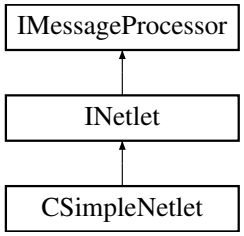
- src/netlets/simpleArch/simpleMultiplexer.h
- src/netlets/simpleArch/simpleMultiplexer.cpp

7.32 CSimpleNetlet Class Reference

Simple Netlet example.

`#include <simpleNetlet.h>`

Inheritance diagram for CSimpleNetlet::



Public Member Functions

- CSimpleNetlet (CNodeArchitecture ∗nodeA, IMessageScheduler ∗sched)
- virtual ∼CSimpleNetlet ()
- virtual void processEvent (IMessage ∗msg) throw (EUnhandledMessage)

  Process an event message directed to this message processing unit.

- virtual void processTimer (IMessage ∗msg) throw (EUnhandledMessage)

  Process a timer message directed to this message processing unit.

- virtual void processOutgoing (IMessage ∗msg) throw (EUnhandledMessage)

  Process an outgoing message directed towards the network.

- virtual void processIncoming (IMessage ∗msg) throw (EUnhandledMessage)

  Process an incoming message directed towards the application.

- virtual INetletMetaData ∗ getMetaData ()

  Returns the Netlet's meta data.

- SimpleHash hash (const std::string &str)

Protected Attributes

- std::map< unsigned int, std::string > **services**

7.32.1 Detailed Description

Simple Netlet example.

7.32.2 Constructor & Destructor Documentation

7.32.2.1 CSimpleNetlet::CSimpleNetlet (CNodeArchitecture * *nodeA*, IMessageScheduler * *sched*)

Constructor

7.32.2.2 CSimpleNetlet::~CSimpleNetlet () [virtual]

Destructor

7.32.3 Member Function Documentation

7.32.3.1 void CSimpleNetlet::processEvent (IMessage * *msg*) throw (EUnhandledMessage) [virtual]

Process an event message directed to this message processing unit.

Parameters:

msg Pointer to message

Implements IMessageProcessor.

7.32.3.2 void CSimpleNetlet::processTimer (IMessage * *msg*) throw (EUnhandledMessage) [virtual]

Process a timer message directed to this message processing unit.

Parameters:

msg Pointer to message

Implements IMessageProcessor.

7.32.3.3 void CSimpleNetlet::processOutgoing (IMessage * *msg*) throw (EUnhandledMessage) [virtual]

Process an outgoing message directed towards the network.

Parameters:

msg Pointer to message

Implements IMessageProcessor.

7.32.3.4 void CSimpleNetlet::processIncoming (IMessage * *msg*) throw (EUnhandledMessage) [virtual]

Process an incoming message directed towards the application.

Parameters:

msg Pointer to message

Implements IMessageProcessor.

7.32.3.5 SimpleHash CSimpleNetlet::hash (const std::string & *str*)

http://codesnippets.joyent.com/user/wastepixel/tag/hashing

TODO: to be replaced by something better

The documentation for this class was generated from the following files:

- src/netlets/simpleArch/simpleNetlet.h
- src/netlets/simpleArch/simpleNetlet.cpp

7.33 CSimpleNetletMetaData Class Reference

Simple Netlet meta data.

`#include <simpleNetlet.h>`

Public Member Functions

- CSimpleNetletMetaData ()
- virtual ~CSimpleNetletMetaData ()
- virtual const std::string & getArchName ()
- virtual const std::string & getName ()
- virtual bool isControlNetlet ()

  Returns true if the Netlet does not offer any transport service for applications, false otherwise.

- virtual INetlet * createNetlet (CNodeArchitecture *nodeA, IMessageScheduler *sched)

7.33.1 Detailed Description

Simple Netlet meta data.

The Netlet meta data class has two purposes: On one hand, it describes the Netlet's properties, capabilities, etc., on the other hand, it provides a factory functions that will be used by the node architecture daemon to instantiate new Netlets of this type.

7.33.2 Constructor & Destructor Documentation

7.33.2.1 CSimpleNetletMetaData::CSimpleNetletMetaData ()

Constructor

7.33.2.2 CSimpleNetletMetaData::~CSimpleNetletMetaData () `[virtual]`

Destructor

7.33.3 Member Function Documentation

7.33.3.1 const std::string & CSimpleNetletMetaData::getArchName () `[virtual]`

Return Netlet name

7.33.3.2 const std::string & CSimpleNetletMetaData::getName () `[virtual]`

Return Netlet name

7.33.3.3 INetlet * CSimpleNetletMetaData::createNetlet (CNodeArchitecture * *nodeA*, IMessageScheduler * *sched*) [virtual]

Create an instance of the Netlet

The documentation for this class was generated from the following files:

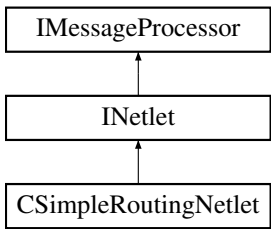
- src/netlets/simpleArch/simpleNetlet.h
- src/netlets/simpleArch/simpleNetlet.cpp

7.34 CSimpleRoutingNetlet Class Reference

Simple neighbor discovery protocol.

`#include <simpleRoutingNetlet.h>`

Inheritance diagram for CSimpleRoutingNetlet::



Public Member Functions

- **CSimpleRoutingNetlet** (CNodeArchitecture ∗nodeA, IMessageScheduler ∗sched)
- virtual void processEvent (IMessage ∗msg) throw (EUnhandledMessage)

  Process an event message directed to this message processing unit.
- virtual void processTimer (IMessage ∗msg) throw (EUnhandledMessage)

  Process a timer message directed to this message processing unit.
- virtual void processOutgoing (IMessage ∗msg) throw (EUnhandledMessage)

  Process an outgoing message directed towards the network.
- virtual void processIncoming (IMessage ∗msg) throw (EUnhandledMessage)

  Process an incoming message directed towards the application.
- virtual INetletMetaData ∗ getMetaData ()

  Returns the Netlet's meta data.
- void handleTimeout ()

  Send out RIX messages.

7.34.1 Detailed Description

Simple neighbor discovery protocol.

7.34.2 Member Function Documentation

7.34.2.1 void CSimpleRoutingNetlet::processEvent (IMessage ∗ *msg*) throw (EUnhandledMessage) [virtual]

Process an event message directed to this message processing unit.

Parameters:

msg Pointer to message

Implements IMessageProcessor.

7.34.2.2 void CSimpleRoutingNetlet::processTimer (IMessage * *msg*) throw (EUnhandledMessage) [virtual]

Process a timer message directed to this message processing unit.

Parameters:

msg Pointer to message

Implements IMessageProcessor.

7.34.2.3 void CSimpleRoutingNetlet::processOutgoing (IMessage * *msg*) throw (EUnhandledMessage) [virtual]

Process an outgoing message directed towards the network.

Parameters:

msg Pointer to message

Implements IMessageProcessor.

7.34.2.4 void CSimpleRoutingNetlet::processIncoming (IMessage * *msg*) throw (EUnhandledMessage) [virtual]

Process an incoming message directed towards the application.

Parameters:

msg Pointer to message

Implements IMessageProcessor.

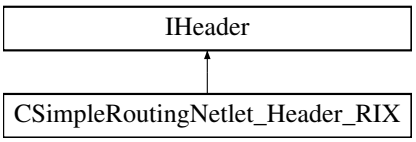
The documentation for this class was generated from the following files:

- src/netlets/simpleArch/simpleRoutingNetlet.h
- src/netlets/simpleArch/simpleRoutingNetlet.cpp

7.35 CSimpleRoutingNetlet_Header_RIX Class Reference

Header class for routing information exchange (RIX) message. This packet is sent blindly of an interface and contains node names that we (the sender) can reach via ∗another∗ interface.

Inheritance diagram for CSimpleRoutingNetlet_Header_RIX::



Public Member Functions

- virtual void serialize (CSerialBuffer ∗buffer)

  Serialize all relevant data into a byte buffer. Remember to do Little/Big Endian conversion.

- virtual void deserialize (CSerialBuffer ∗buffer)

  De-serialize all relevant data from a byte buffer. Remember to do Little/Big Endian conversion.

Public Attributes

- list< string > **nodes**

7.35.1 Detailed Description

Header class for routing information exchange (RIX) message. This packet is sent blindly of an interface and contains node names that we (the sender) can reach via ∗another∗ interface.

7.35.2 Member Function Documentation

7.35.2.1 virtual void CSimpleRoutingNetlet_Header_RIX::serialize (CSerialBuffer ∗ *buffer*)

`[inline, virtual]`

Serialize all relevant data into a byte buffer. Remember to do Little/Big Endian conversion.

Header format is [A][B][C...] where A is the number of list entries (one byte) B is the size of node name (one byte) C is the node name (variable length)

Implements IHeader.

7.35.2.2 virtual void CSimpleRoutingNetlet_Header_RIX::deserialize (CSerialBuffer ∗ *buffer*)

`[inline, virtual]`

De-serialize all relevant data from a byte buffer. Remember to do Little/Big Endian conversion.

Header format is [A][B][C...] where A is the number of list entries (one byte) B is the size of node name (one byte) C is the node name (variable length)

Implements IHeader.

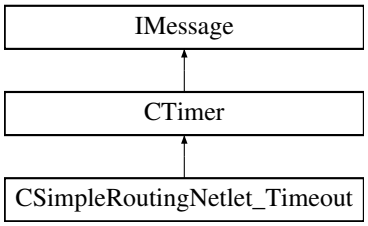
The documentation for this class was generated from the following file:

- src/netlets/simpleArch/simpleRoutingNetlet.cpp

7.36 CSimpleRoutingNetlet_Timeout Class Reference

Timeout to send route information.

Inheritance diagram for CSimpleRoutingNetlet_Timeout::



Public Member Functions

- CSimpleRoutingNetlet_Timeout (double timeout, IMessageProcessor *proc)
  Constructor.
- virtual ~CSimpleRoutingNetlet_Timeout ()
  Destructor.

7.36.1 Detailed Description

Timeout to send route information.

Single shot timer.

7.36.2 Constructor & Destructor Documentation

7.36.2.1 CSimpleRoutingNetlet_Timeout::CSimpleRoutingNetlet_Timeout (double *timeout*, IMessageProcessor * *proc*) [inline]

Constructor.

Parameters:

timeout Timeout in seconds

proc Node arch entity the event is linked to (default = NULL)

The documentation for this class was generated from the following file:

- src/netlets/simpleArch/simpleRoutingNetlet.cpp

7.37 CSimpleRoutingNetletMetaData Class Reference

Simple Routing Netlet meta data.

`#include <simpleRoutingNetlet.h>`

Public Member Functions

- CSimpleRoutingNetletMetaData ()
- virtual ~CSimpleRoutingNetletMetaData ()
- virtual const std::string & getArchName ()
- virtual const std::string & getName ()
- virtual bool isControlNetlet ()

  Returns true if the Netlet does not offer any transport service for applications, false otherwise.

- virtual INetlet * createNetlet (CNodeArchitecture *nodeA, IMessageScheduler *sched)

7.37.1 Detailed Description

Simple Routing Netlet meta data.

The Netlet meta data class has two purposes: On one hand, it describes the Netlet's properties, capabilities, etc., on the other hand, it provides a factory functions that will be used by the node architecture daemon to instantiate new Netlets of this type.

7.37.2 Constructor & Destructor Documentation

7.37.2.1 CSimpleRoutingNetletMetaData::CSimpleRoutingNetletMetaData ()

Constructor

7.37.2.2 CSimpleRoutingNetletMetaData::~CSimpleRoutingNetletMetaData () [virtual]

Destructor

7.37.3 Member Function Documentation

7.37.3.1 const std::string & CSimpleRoutingNetletMetaData::getArchName () [virtual]

Return Netlet name

7.37.3.2 const std::string & CSimpleRoutingNetletMetaData::getName () [virtual]

Return Netlet name

7.37.3.3 INetlet * CSimpleRoutingNetletMetaData::createNetlet (CNodeArchitecture * *nodeA*, IMessageScheduler * *sched*) [virtual]

Create an instance of the Netlet

The documentation for this class was generated from the following files:

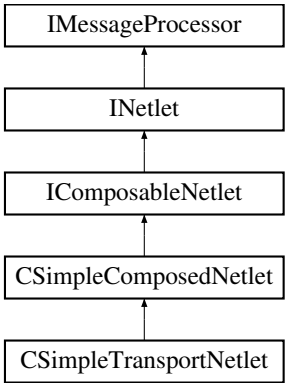
- src/netlets/simpleArch/simpleRoutingNetlet.h
- src/netlets/simpleArch/simpleRoutingNetlet.cpp

7.38 CSimpleTransportNetlet Class Reference

A Netlet taking its actual building block configuration from a generated source file.

`#include <simpleTransportNetlet.h>`

Inheritance diagram for CSimpleTransportNetlet::



Public Member Functions

- CSimpleTransportNetlet (CNodeArchitecture *nodeA, IMessageScheduler *sched)
- virtual ~CSimpleTransportNetlet ()
- virtual INetletMetaData * getMetaData ()

  Returns the Netlet's meta data.

7.38.1 Detailed Description

A Netlet taking its actual building block configuration from a generated source file.

Per architecture, a single template for a composable Netlet would be sufficient.

7.38.2 Constructor & Destructor Documentation

7.38.2.1 CSimpleTransportNetlet::CSimpleTransportNetlet (CNodeArchitecture * *nodeA*, IMessageScheduler * *sched*)

Constructor

7.38.2.2 CSimpleTransportNetlet::~CSimpleTransportNetlet () `[virtual]`

Destructor

The documentation for this class was generated from the following files:

- src/netlets/simpleArch/simpleTransportNetlet.h
- src/netlets/simpleArch/simpleTransportNetlet.cpp

7.39 CSimpleTransportNetletMetaData Class Reference

Simple composed Netlet meta data.

`#include <simpleTransportNetlet.h>`

Public Member Functions

- CSimpleTransportNetletMetaData ()
- virtual ~CSimpleTransportNetletMetaData ()
- virtual const std::string & getArchName ()
- virtual const std::string & getName ()
- virtual bool isControlNetlet ()

  Returns true if the Netlet does not offer any transport service for applications, false otherwise.

- virtual INetlet * createNetlet (CNodeArchitecture *nodeA, IMessageScheduler *sched)

7.39.1 Detailed Description

Simple composed Netlet meta data.

The Netlet meta data class has two purposes: On one hand, it describes the Netlet's properties, capabilities, etc., on the other hand, it provides a factory functions that will be used by the node architecture daemon to instantiate new Netlets of this type.

7.39.2 Constructor & Destructor Documentation

7.39.2.1 CSimpleTransportNetletMetaData::CSimpleTransportNetletMetaData ()

Constructor

7.39.2.2 CSimpleTransportNetletMetaData::~CSimpleTransportNetletMetaData () [virtual]

Destructor

7.39.3 Member Function Documentation

7.39.3.1 const std::string & CSimpleTransportNetletMetaData::getArchName () [virtual]

Return Netlet name

7.39.3.2 const std::string & CSimpleTransportNetletMetaData::getName () [virtual]

Return Netlet name

7.39.3.3 INetlet * CSimpleTransportNetletMetaData::createNetlet (CNodeArchitecture * *nodeA*, IMessageScheduler * *sched*) [virtual]

Create an instance of the Netlet

The documentation for this class was generated from the following files:

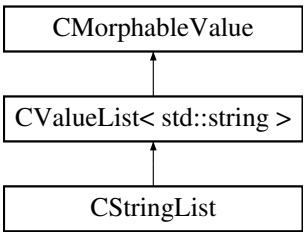
- src/netlets/simpleArch/simpleTransportNetlet.h
- src/netlets/simpleArch/simpleTransportNetlet.cpp

7.40 CStringList Class Reference

Facade for a string list.

`#include <morphableValue.h>`

Inheritance diagram for CStringList::



7.40.1 Detailed Description

Facade for a string list.

The documentation for this class was generated from the following file:

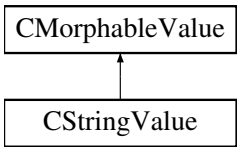
- include/morphableValue.h

7.41 CStringValue Class Reference

Facade for string values.

`#include <morphableValue.h>`

Inheritance diagram for CStringValue::



Public Member Functions

- **CStringValue** (std::string v=0)
- std::string & **value** ()

7.41.1 Detailed Description

Facade for string values.

The documentation for this class was generated from the following file:

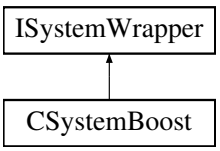
- include/morphableValue.h

7.42 CSystemBoost Class Reference

Boost ASIO wrapper.

`#include <systemBoost.h>`

Inheritance diagram for CSystemBoost::



Public Member Functions

- CSystemBoost ()

  Constructor.

- virtual ~CSystemBoost ()

  Destructor.

- virtual void run ()

  Main system loop.

- virtual void initNetAdapts ()

  Initialize network accesses.

- virtual void releaseNetAdapts ()

  Release network accesses.

- virtual void initAppInterface ()

  Initiliaze application interface.

- virtual void closeAppInterface ()

  Tear down application interface.

- virtual void setTimer (CTimer *timer)

  Start a timer.

- virtual std::string getNodeName ()

  Return the current node's name.

7.42.1 Detailed Description

Boost ASIO wrapper.

7.42.2 Constructor & Destructor Documentation

7.42.2.1 CSystemBoost::CSystemBoost ()

Constructor.

System wrapper for Boost supported systems (e.g. Linux, Windows).

systemBoost.cpp

(c) 2008-2009 Institut fuer Telematik, Universitaet Karlsruhe (TH), Germany

Created on: Dec 10, 2008 Author: denis Constructor

7.42.2.2 CSystemBoost::~CSystemBoost () [virtual]

Destructor.

Destructor

7.42.3 Member Function Documentation

7.42.3.1 void CSystemBoost::run () [virtual]

Main system loop.

Main event loop

7.42.3.2 void CSystemBoost::initNetAdapts () [virtual]

Initialize network accesses.

Scan for or create Network Accesses

7.42.3.3 void CSystemBoost::releaseNetAdapts () [virtual]

Release network accesses.

Release previously initialized network accesses

Implements ISystemWrapper.

7.42.3.4 void CSystemBoost::initAppInterface () [virtual]

Initiliaze application interface.

Initialize application interface

also: initialize traffic generators if applicable

7.42.3.5 void CSystemBoost::closeAppInterface () [virtual]

Tear down application interface.

Release / tear down application interface

7.42.3.6 void CSystemBoost::setTimer (CTimer * *timer*) [virtual]

Start a timer.

Set a timer.

Parameters:

timer Timer event containing the timeout properties.

7.42.3.7 std::string CSystemBoost::getNodeName () [virtual]

Return the current node's name.

Return an arbitrary name for the node

Implements ISystemWrapper.

The documentation for this class was generated from the following files:

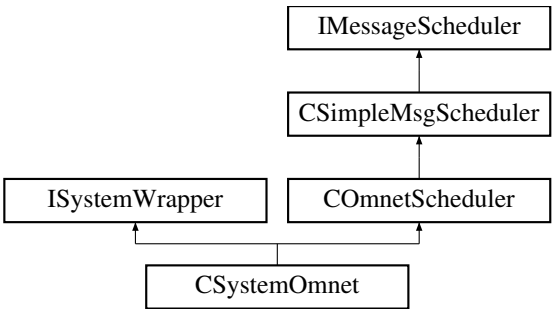
- src/targets/boost/systemBoost.h
- src/targets/boost/systemBoost.cpp

7.43 CSystemOmnet Class Reference

OMNeT++ Wrapper.

`#include <systemOmnet.h>`

Inheritance diagram for CSystemOmnet::



Public Member Functions

- CSystemOmnet ()
  Constructor.
- virtual ~CSystemOmnet ()
  Destructor.
- virtual IMessageScheduler * getMainScheduler ()
  Returns the system-specific main scheduler (e.g. main thread).
- virtual IMessageScheduler * schedulerFactory ()
  Returns a system-specific scheduler.
- virtual void getNetAdapts (std::list< INetAdapt * > &netAdapts)
  Initialize network accesses.
- virtual void releaseNetAdapts ()
  Release network accesses.
- virtual std::string getNodeName ()
  Return the current node's name.

Protected Member Functions

- virtual void initialize ()
  From cSimpleModule.
- virtual void handleMessage (cMessage *msg)
  From cSimpleModule.

7.43.1 Detailed Description

OMNeT++ Wrapper.

CSystemOmnet is a Omnet++ module that registers itself at Omnet. It gets instantiated by Omnet for every node as defined in the .ned-file. During initialization, it instantiates the Node Architecture - hence, we have ∗multiple∗ instances within the Omnet address space. This is different to standalone targets, where the Node Architecture is really a singleton.

7.43.2 Constructor & Destructor Documentation

7.43.2.1 CSystemOmnet::CSystemOmnet ()

Constructor.

Constructor

7.43.2.2 CSystemOmnet::~CSystemOmnet () [virtual]

Destructor.

Destructor

7.43.3 Member Function Documentation

7.43.3.1 void CSystemOmnet::initialize () [protected, virtual]

From cSimpleModule.

cSimpleModule::initialize()

Reimplemented from COmnetScheduler.

7.43.3.2 void CSystemOmnet::handleMessage (cMessage ∗ *msg*) [protected, virtual]

From cSimpleModule.

cSimpleModule::handleMessage()

Reimplemented from COmnetScheduler.

7.43.3.3 void CSystemOmnet::getNetAdapts (std::list< INetAdapt ∗ > & *netAdapts*) [virtual]

Initialize network accesses.

Scan for or create Network Accesses

Implements ISystemWrapper.

7.43.3.4 void CSystemOmnet::releaseNetAdapts () [virtual]

Release network accesses.

Release previously initialized network accesses

Implements ISystemWrapper.

7.43.3.5 std::string CSystemOmnet::getNodeName () [virtual]

Return the current node's name.

Return an arbitrary name for the node

Implements ISystemWrapper.

The documentation for this class was generated from the following files:

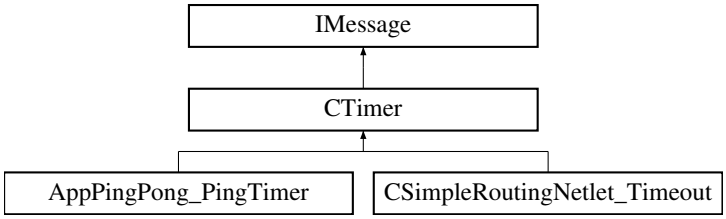
- src/targets/omnetpp/systemOmnet.h
- src/targets/omnetpp/systemOmnet.cpp

7.44 CTimer Class Reference

Timer events.

`#include <messages.h>`

Inheritance diagram for CTimer::



Public Member Functions

- CTimer (double timeout, IMessageProcessor *proc)
  Constructor.
- virtual ~CTimer ()
  Destructor.

Public Attributes

- double timeout
  Timeout in seconds.

7.44.1 Detailed Description

Timer events.

Single shot timer.

7.44.2 Constructor & Destructor Documentation

7.44.2.1 CTimer::CTimer (double *timeout*, IMessageProcessor * *proc*) [inline]

Constructor.

Parameters:

timeout Timeout in seconds
proc Node arch entity the event is linked to

The documentation for this class was generated from the following file:

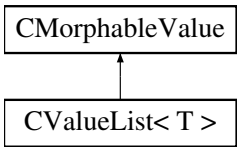
- include/messages.h

7.45 CValueList< T > Class Template Reference

Internal use only; container class for a value range.

`#include <morphableValue.h>`

Inheritance diagram for CValueList< T >::



Public Member Functions

- **CValueList** (ValueType vt=vt_none)
- std::list< T > & **list** () throw (EValueTypeMismatch, EContainerTypeMismatch)

Protected Attributes

- std::list< T > **list_v**

7.45.1 Detailed Description

template<class T> class CValueList< T >

Internal use only; container class for a value range.

The documentation for this class was generated from the following file:

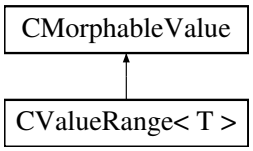
- include/morphableValue.h

7.46 CValueRange< T > Class Template Reference

Internal use only; container class for a value range.

`#include <morphableValue.h>`

Inheritance diagram for CValueRange< T >::



Public Member Functions

- **CValueRange** (ValueType vt=vt_none)
- Range< T > & **range** () throw (EValueTypeMismatch, EContainerTypeMismatch)

Protected Attributes

- Range< T > **range_v**

7.46.1 Detailed Description

template<class T> class CValueRange< T >

Internal use only; container class for a value range.

The documentation for this class was generated from the following file:

- include/morphableValue.h

7.47 Debug Class Reference

Debug class.

`#include <debug.h>`

Static Public Member Functions

- static void print (std::string msg)
  Print a message.
- static void print (boost::format fmt)
  Print a formatted message.
- static void error (boost::format fmt)
  Print an error message.
- static void fail (boost::format fmt)
  Print an error message and terminate execution.

Static Public Attributes

- static DebugTimeFunction **time** = NULL

7.47.1 Detailed Description

Debug class.

Please don't use this directly. Use the macros defined above instead.

The documentation for this class was generated from the following files:

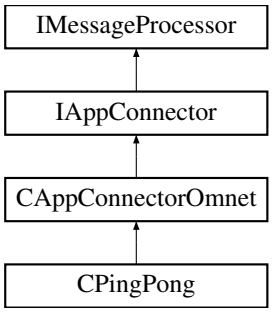
- include/debug.h
- src/daemon/nodeArchitecture.cpp

7.48 IAppConnector Class Reference

Stub for Application Interfaces. There is one connector per connection.

`#include <appConnector.h>`

Inheritance diagram for IAppConnector::



Public Member Functions

- **IAppConnector** (IMessageScheduler *sched)
- virtual std::string getIdentifier ()
  Return ID of the application/connection/service.
- virtual void setIdentifier (const std::string &id)
  Set ID of the application/connection/service.
- virtual LocalConnId getLocalConnId ()
  Return connection ID (only locally valid).
- virtual void setLocalConnId (const LocalConnId id)
  Set connection ID (only locally valid).
- virtual INetlet * getNetlet ()
  Return Netlet associated with the app connector. NULL if none is associated.
- virtual void setNetlet (INetlet *netlet)
  Set Netlet association.

Protected Attributes

- std::string identifier
  ID of the application/connection/service.
- LocalConnId connId
  connection ID, is only locally valid
- INetlet * netlet
  associated Netlet

7.48.1 Detailed Description

Stub for Application Interfaces. There is one connector per connection.

The documentation for this class was generated from the following file:

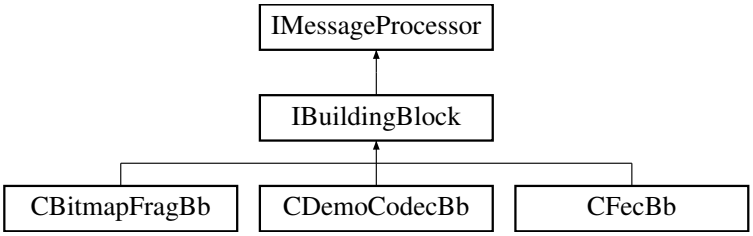
- include/appConnector.h

7.49 IBuildingBlock Class Reference

Interface for a building block.

`#include <composableNetlet.h>`

Inheritance diagram for IBuildingBlock::



Public Types

- typedef std::string Id
  type for building block ID

Public Member Functions

- IBuildingBlock (IMessageScheduler *sched)
  Constructor.
- virtual ~IBuildingBlock ()
  Destructor.
- virtual Id getId ()=0
  Return ID of the building block.

7.49.1 Detailed Description

Interface for a building block.

The documentation for this class was generated from the following file:

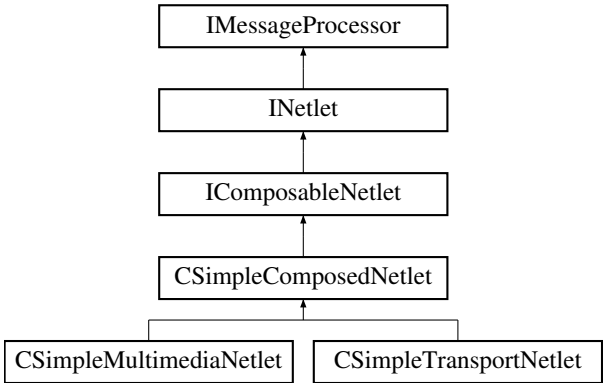
- include/composableNetlet.h

7.50 IComposableNetlet Class Reference

A base class for a Netlet composed of building blocks.

`#include <composableNetlet.h>`

Inheritance diagram for IComposableNetlet::



Public Member Functions

- IComposableNetlet (CNodeArchitecture *nodeA, IMessageScheduler *sched)

  Constructor.

- virtual ~IComposableNetlet ()

  Destructor.

- virtual void rewire ()

  Rewire the building blocks according to outgoingChain and incomingChain.

Protected Attributes

- std::map< IBuildingBlock::Id, IBuildingBlock * > buildingBlocks

  Map of all available blocks.

- Config * **config**

Classes

- class Config

  Simple Building Block configuration. This class will be generated by the Netlet editor.

- class EUnknownBuildingBlock

  Thrown if a building block cannot be found.

7.50.1 Detailed Description

A base class for a Netlet composed of building blocks.

Currently, only very simple compositions are supported. There are only firmly wired chains of building blocks allowed, one for outgoing data, one for incoming data. Furthermore, no signaling of the used BB chain is supported, neither in-band, nor out-of-band.

More sophisticated mechanisms supporting building block graphs including branches and dynamic signaling may follow in the future.

The documentation for this class was generated from the following file:

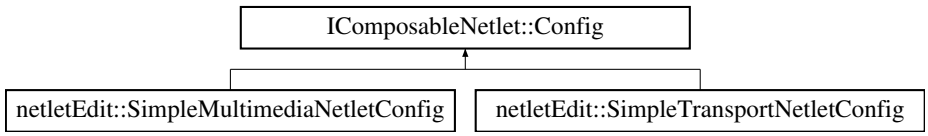
- include/composableNetlet.h

7.51 IComposableNetlet::Config Class Reference

Simple Building Block configuration. This class will be generated by the Netlet editor.

`#include <composableNetlet.h>`

Inheritance diagram for IComposableNetlet::Config::



Public Attributes

- std::list< IBuildingBlock::Id > outgoingChain

  Outgoing chain of building blocks from top to bottom.

- std::list< IBuildingBlock::Id > incomingChain

  Incoming chain of building blocks from bottom to top.

7.51.1 Detailed Description

Simple Building Block configuration. This class will be generated by the Netlet editor.

More sophisticated mechanisms supporting building block graphs including branches and dynamic signaling may follow in the future.

7.51.2 Member Data Documentation

7.51.2.1 std::list<IBuildingBlock::Id> IComposableNetlet::Config::outgoingChain

Outgoing chain of building blocks from top to bottom.

This is used by rewire() to rewire the building blocks.

7.51.2.2 std::list<IBuildingBlock::Id> IComposableNetlet::Config::incomingChain

Incoming chain of building blocks from bottom to top.

This is used by rewire() to rewire the building blocks.

The documentation for this class was generated from the following file:

- include/composableNetlet.h

7.52 IComposableNetlet::EUnknownBuildingBlock Class Reference

Thrown if a building block cannot be found.

`#include <composableNetlet.h>`

Public Member Functions

- EUnknownBuildingBlock () throw ()
  Default constructor.
- EUnknownBuildingBlock (const std::string &msg) throw ()
  Constructor with exception message.
- virtual ~EUnknownBuildingBlock () throw ()
  Destructor.
- virtual const char * what () const throw ()
  Return the message associated with this exception.

Protected Attributes

- std::string msg
  Message associated with this exception.

7.52.1 Detailed Description

Thrown if a building block cannot be found.

The documentation for this class was generated from the following file:

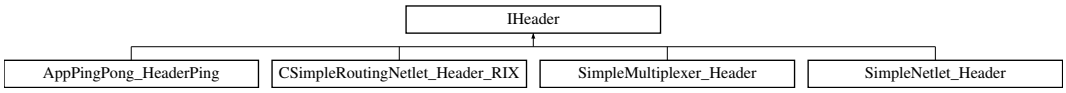
- include/composableNetlet.h

7.53 IHeader Class Reference

Single protocol (Functional Block) header.

`#include <packets.h>`

Inheritance diagram for IHeader::



Public Member Functions

- virtual void serialize (CSerialBuffer *buffer)=0

  Serialize all relevant data into a byte buffer. Remember to do Little/Big Endian conversion.

- virtual void deserialize (CSerialBuffer *buffer)=0

  De-serialize all relevant data from a byte buffer. Remember to do Little/Big Endian conversion.

7.53.1 Detailed Description

Single protocol (Functional Block) header.

The documentation for this class was generated from the following file:

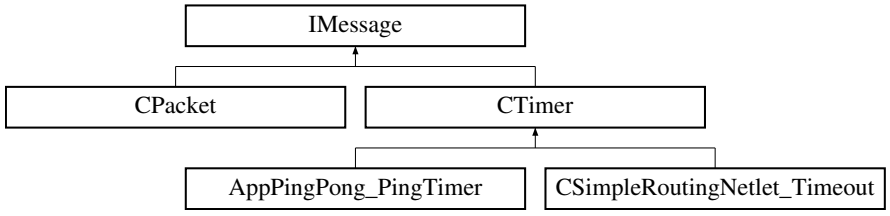
- include/packets.h

7.54 IMessage Class Reference

Generic message interface.

`#include <messages.h>`

Inheritance diagram for IMessage::



Public Types

- enum **Type** {
  t_none = 0, **t_timer**, **t_event**, **t_incomingPacket**,
  t_outgoingPacket, **t_max** }
- enum **PropertyId** {
  p_none = 0, **p_destId**, **p_srcId**, **p_destLoc**,
  p_srcLoc, **p_netletId**, **p_userBase** = 1000 }

Public Member Functions

- IMessage (IMessageProcessor *from=NULL, IMessageProcessor *to=NULL, Type type=t_none)
  Constructor.
- virtual ~IMessage ()
  Destructor.
- std::string getName ()
  Return name identifying this class.
- virtual IMessageProcessor * **getFrom** ()
- virtual void **setFrom** (IMessageProcessor *from)
- virtual IMessageProcessor * **getTo** ()
- virtual void **setTo** (IMessageProcessor *to)
- virtual Type **getType** ()
- virtual void **setType** (Type type)
- virtual LocalConnId **getLocalConnId** ()
- virtual void **setLocalConnId** (LocalConnId connId)
- virtual ServiceId **getServiceId** ()
- virtual void **setServiceId** (ServiceId servId)
- template<class T>
  T * cast () throw (ETypeMismatch)
  Safe cast.

- void **setProperty** (PropertyId pid, CMorphableValue *val)
- template<typename T>
  T **getProperty** (PropertyId pid) throw (EPropertyNotDefined)
- void flushVisitedProcessors ()

Public Attributes

- std::list< IMessageProcessor * > **visitedProcessors**

Protected Attributes

- IMessageProcessor * **from**
- IMessageProcessor * **to**
- Type **type**
- LocalConnId **connId**
- ServiceId **serviceId**
- std::string **name**
- std::map< PropertyId, CMorphableValue * > properties

  Cross-"layer" properties.

Classes

- class EPropertyNotDefined

  Thrown if the requested property is not set.

- class ETypeMismatch

  Thrown if the expected IMessage::Type differs from the actual one.

7.54.1 Detailed Description

Generic message interface.

A message can be a packet, a local event, or a timer.

7.54.2 Constructor & Destructor Documentation

7.54.2.1 IMessage::IMessage (IMessageProcessor * *from* = NULL, IMessageProcessor * *to* = NULL, Type *type* = t_none) [inline]

Constructor.

Parameters:

from Sender

to Receiver

type Message type

7.54.3 Member Function Documentation

7.54.3.1 void IMessage::flushVisitedProcessors () [inline]

If messages are sent in circles by purpose, this method must be called in order to flush the loop detection stack.

The documentation for this class was generated from the following file:

- include/messages.h

7.55 IMessage::EPropertyNotDefined Class Reference

Thrown if the requested property is not set.

`#include <messages.h>`

7.55.1 Detailed Description

Thrown if the requested property is not set.

The documentation for this class was generated from the following file:

- include/messages.h

7.56 IMessage::ETypeMismatch Class Reference

Thrown if the expected IMessage::Type differs from the actual one.

`#include <messages.h>`

7.56.1 Detailed Description

Thrown if the expected IMessage::Type differs from the actual one.

The documentation for this class was generated from the following file:

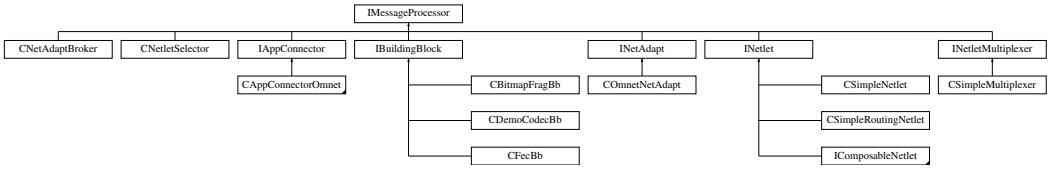
- include/messages.h

7.57 IMessageProcessor Class Reference

Any class processing a message.

`#include <messages.h>`

Inheritance diagram for IMessageProcessor::



Public Member Functions

- IMessageProcessor (IMessageScheduler ∗sched)

  Constructor.

- virtual ∼IMessageProcessor ()

  Destructor.

- std::string getName ()

  Return name identifying this class.

- virtual void processMessage (IMessage ∗msg) throw (EUnhandledMessage)

  Process a message directed to this message processing unit. This method calls the respective process∗ functions depending on the message's type.

- virtual void processEvent (IMessage ∗msg)=0 throw (EUnhandledMessage)

  Process an event message directed to this message processing unit.

- virtual void processTimer (IMessage ∗msg)=0 throw (EUnhandledMessage)

  Process a timer message directed to this message processing unit.

- virtual void processOutgoing (IMessage ∗msg)=0 throw (EUnhandledMessage)

  Process an outgoing message directed towards the network.

- virtual void processIncoming (IMessage ∗msg)=0 throw (EUnhandledMessage)

  Process an incoming message directed towards the application.

- virtual IMessageScheduler ∗ getMessageScheduler ()

  Return message scheduler of this message processor.

- virtual void setMessageScheduler (IMessageScheduler ∗sched)

  Set message scheduler of this message processor.

- virtual IMessageProcessor ∗ getPrev ()

  Return message processor that is called prior to this one (from application's point of view).

- virtual void setPrev (IMessageProcessor *processor)

  Set message processor that should be called prior to this one (from application's point of view).

- virtual IMessageProcessor * getNext ()

  Return message processor that is called next to this one (from application's point of view).

- virtual void setNext (IMessageProcessor *processor)

  Set message processor that should be called next to this one (from application's point of view).

Protected Member Functions

- void sendMessage (IMessage *msg) throw (IMessageScheduler::EUnknowMessageProcessor, IMessageScheduler::EMessageLoop)

  Send a message via the scheduler (convenience function).

Protected Attributes

- IMessageScheduler * scheduler

  associated scheduler

- IMessageProcessor * prev

  "upper", towards application

- IMessageProcessor * next

  "lower", towards network

- std::string name

  name identifying this class

Classes

- class EUnhandledMessage

  Thrown if the message cannot be handled by the addressed message processor.

7.57.1 Detailed Description

Any class processing a message.

Each message processor is connected to a scheduler which manages its incoming and outgoing queues. The scheduler will also handle interprocessor communications.

7.57.2 Constructor & Destructor Documentation

7.57.2.1 IMessageProcessor::IMessageProcessor (IMessageScheduler * *sched*) [inline]

Constructor.

Parameters:

sched Scheduler that manages the queues of this message processor

7.57.3 Member Function Documentation

7.57.3.1 virtual void IMessageProcessor::processMessage (IMessage * *msg*) throw (EUnhandledMessage) [inline, virtual]

Process a message directed to this message processing unit. This method calls the respective process* functions depending on the message's type.

Parameters:

msg Pointer to message

7.57.3.2 virtual void IMessageProcessor::processEvent (IMessage * *msg*) throw (EUnhandledMessage) [pure virtual]

Process an event message directed to this message processing unit.

Parameters:

msg Pointer to message

Implemented in CBitmapFragBb, CDemoCodecBb, CFecBb, CNetAdaptBroker, CNetletSelector, CSimpleComposedNetlet, CSimpleMultiplexer, CSimpleNetlet, CSimpleRoutingNetlet, CPingPong, and COmnetNetAdapt.

7.57.3.3 virtual void IMessageProcessor::processTimer (IMessage * *msg*) throw (EUnhandledMessage) [pure virtual]

Process a timer message directed to this message processing unit.

Parameters:

msg Pointer to message

Implemented in CBitmapFragBb, CDemoCodecBb, CFecBb, CNetAdaptBroker, CNetletSelector, CSimpleComposedNetlet, CSimpleMultiplexer, CSimpleNetlet, CSimpleRoutingNetlet, CPingPong, and COmnetNetAdapt.

7.57.3.4 virtual void IMessageProcessor::processOutgoing (IMessage * *msg*) throw (EUnhandledMessage) [pure virtual]

Process an outgoing message directed towards the network.

Parameters:

msg Pointer to message

Implemented in CBitmapFragBb, CDemoCodecBb, CFecBb, CNetAdaptBroker, CNetletSelector, CSimpleComposedNetlet, CSimpleMultiplexer, CSimpleNetlet, CSimpleRoutingNetlet, CPingPong, and COmnetNetAdapt.

7.57.3.5 virtual void IMessageProcessor::processIncoming (IMessage * *msg*) throw (EUnhandledMessage) [pure virtual]

Process an incoming message directed towards the application.

Parameters:

msg Pointer to message

Implemented in CBitmapFragBb, CDemoCodecBb, CFecBb, CNetAdaptBroker, CNetletSelector, CSimpleComposedNetlet, CSimpleMultiplexer, CSimpleNetlet, CSimpleRoutingNetlet, CPingPong, and COmnetNetAdapt.

The documentation for this class was generated from the following file:

- include/messages.h

7.58 IMessageProcessor::EUnhandledMessage Class Reference

Thrown if the message cannot be handled by the addressed message processor.

`#include <messages.h>`

Public Member Functions

- EUnhandledMessage () throw ()
  Default constructor.
- EUnhandledMessage (const std::string &msg) throw ()
  Constructor with exception message.
- virtual ~EUnhandledMessage () throw ()
  Destructor.
- virtual const char * what () const throw ()
  Return the message associated with this exception.

Protected Attributes

- std::string msg
  Message associated with this exception.

7.58.1 Detailed Description

Thrown if the message cannot be handled by the addressed message processor.

The documentation for this class was generated from the following file:

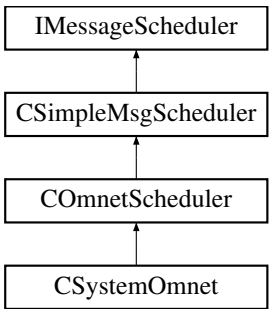
- include/messages.h

7.59 IMessageScheduler Class Reference

Interface to message scheduler.

`#include <messages.h>`

Inheritance diagram for IMessageScheduler::



Public Member Functions

- IMessageScheduler ()

  Constructor.

- virtual ~IMessageScheduler ()

  Destructor.

- std::string getName ()

  Return name identifying this class.

- virtual void sendMessage (IMessage *msg)=0 throw (EUnknowMessageProcessor, EMessageLoop)

  Send a message to another processing unit.

- virtual void setTimer (CTimer *timer)=0

  Set a single shot time.

- virtual void processMessages ()=0 throw (EUnknowMessageProcessor)

  Process all waiting messages and call respective receivers. Returns if all queues are empty.

- virtual void registerMessageProcessor (IMessageProcessor *proc) throw (EUnknowMessageProcessor, EAlreadyRegistered)

  Register a new message processor belonging to this scheduler.

- virtual void unregisterMessageProcessor (IMessageProcessor *proc) throw (EUnknowMessageProcessor)

  Unregister a new message processor belonging to this scheduler.

Protected Attributes

- std::map< IMessageProcessor *, MessageQueue > **queues**
- std::string **name**

Classes

- class EAlreadyRegistered

  Message processor already exists.

- class EMessageLoop

  Message loop detected.

- class EUnknowMessageProcessor

  Message processor is unknown to scheduler.

- class **MessageQueue**

7.59.1 Detailed Description

Interface to message scheduler.

This may represent a thread on a real system.

7.59.2 Member Function Documentation

7.59.2.1 virtual void IMessageScheduler::sendMessage (IMessage * *msg*) throw (EUnknowMessageProcessor, EMessageLoop) [pure virtual]

Send a message to another processing unit.

Parameters:

msg Message to send

Implemented in CSimpleMsgScheduler.

7.59.2.2 virtual void IMessageScheduler::setTimer (CTimer * *timer*) [pure virtual]

Set a single shot time.

Parameters:

timer Timer to be set

Implemented in COmnetScheduler, and CSimpleMsgScheduler.

The documentation for this class was generated from the following file:

- include/messages.h

7.60 IMessageScheduler::EAlreadyRegistered Class Reference

Message processor already exists.

`#include <messages.h>`

Public Member Functions

- **EAlreadyRegistered** (const std::string &msg) throw ()
- virtual const char * **what** () const throw ()

Protected Attributes

- std::string **msg**

7.60.1 Detailed Description

Message processor already exists.

The documentation for this class was generated from the following file:

- include/messages.h

7.61 IMessageScheduler::EMessageLoop Class Reference

Message loop detected.

`#include <messages.h>`

Public Member Functions

- **EMessageLoop** (const std::string &msg) throw ()
- virtual const char * **what** () const throw ()

Protected Attributes

- std::string **msg**

7.61.1 Detailed Description

Message loop detected.

The documentation for this class was generated from the following file:

- include/messages.h

7.62 IMessageScheduler::EUnknowMessageProcessor Class Reference

Message processor is unknown to scheduler.

`#include <messages.h>`

Public Member Functions

- **EUnknowMessageProcessor** (const std::string &msg) throw ()
- virtual const char * **what** () const throw ()

Protected Attributes

- std::string **msg**

7.62.1 Detailed Description

Message processor is unknown to scheduler.

The documentation for this class was generated from the following file:

- include/messages.h

7.63 IMultiplexerMetaData Class Reference

Multiplexer meta data.

`#include <netletMultiplexer.h>`

Public Member Functions

- virtual std::string getArchName ()=0

  Return the name of the architecture.

- virtual INetletMultiplexer * createMultiplexer (CNodeArchitecture *nodeA, IMessageScheduler *sched)=0

  Factory function.

7.63.1 Detailed Description

Multiplexer meta data.

The multiplexer meta data class has two purposes: On one hand, it describes the architecure's properties, on the other hand, it provides a factory functions that will be used by the node architecture daemon to instantiate new "architecture".

7.63.2 Member Function Documentation

7.63.2.1 virtual INetletMultiplexer* IMultiplexerMetaData::createMultiplexer (CNodeArchitecture * *nodeA*, IMessageScheduler * *sched*) [pure virtual]

Factory function.

Returns an instance of the architecture specific multiplexer. On a single node, there should be only ONE multiplexer per architecture.

The documentation for this class was generated from the following file:

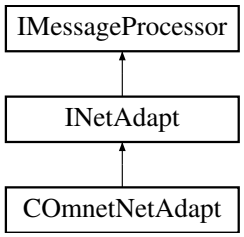
- include/netletMultiplexer.h

7.64 INetAdapt Class Reference

Generic Network Adaptor Interface.

`#include <netAdapt.h>`

Inheritance diagram for INetAdapt::



Public Types

- enum **PropertyId** {
  p_min = 0, **p_generic_min**, **p_name**, **p_archid**,
  p_up, **p_linkencap**, **p_bandwidth**, **p_broadcast**,
  p_rx_packets, **p_rx_errors**, **p_rx_dropped**, **p_tx_packets**,
  p_tx_errors, **p_tx_dropped**, **p_mtu**, **p_duplex**,
  p_virtualized, **p_generic_max**, **p_phy_min** = 1000, **p_phy_max**,
  p_virt_min = 2000, **p_virt_max**, **p_mac_min** = 3000, **p_mac_max**,
  p_pow_min = 4000, **p_pow_sleepmodes**, **p_pow_powerctrl**, **p_pow_max**,
  p_qos_min = 5000, **p_qos_bandwidth**, **p_qos_avgdelay**, **p_qos_jitter**,
  p_qos_pktloss_rate, **p_qos_biterror_rate**, **p_qos_max**, **p_max** }

Public Member Functions

- **INetAdapt** (CNodeArchitecture *nodeA, IMessageScheduler *sched)
- virtual std::string getName ()=0

  Return name identifying this class.

- template<typename T>
  T **getProperty** (PropertyId pid) throw (EPropertyNotDefined)

Protected Attributes

- CNodeArchitecture * **nodeArch**
- std::map< PropertyId, CMorphableValue * > **properties**

Classes

- class **EPropertyNotDefined**

7.64.1 Detailed Description

Generic Network Adaptor Interface.

The documentation for this class was generated from the following file:

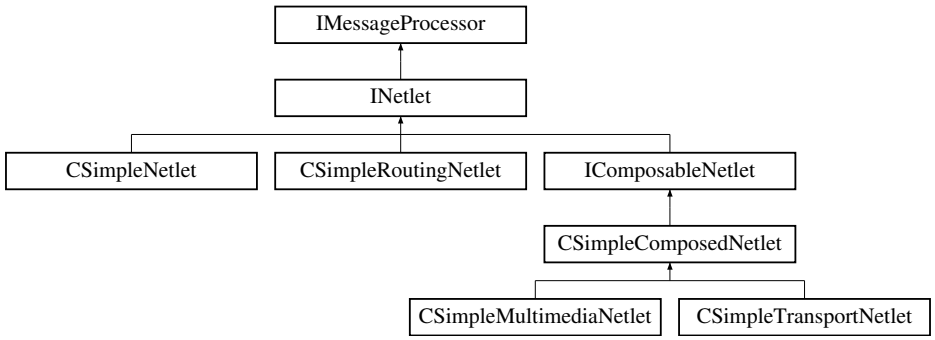
- include/netAdapt.h

7.65 INetlet Class Reference

Generic Netlet Interface.

`#include <netlet.h>`

Inheritance diagram for INetlet::



Public Member Functions

- INetlet (CNodeArchitecture *nodeA, IMessageScheduler *sched)

  Constructor.

- virtual ~INetlet ()

  Destructor.

- virtual INetletMetaData * getMetaData ()=0

  Returns the Netlet's meta data.

Protected Attributes

- CNodeArchitecture * **nodeArch**

7.65.1 Detailed Description

Generic Netlet Interface.

Must be implemented by all Netlets.

The documentation for this class was generated from the following file:

- include/netlet.h

7.66 INetletMetaData Class Reference

Netlet meta data.

`#include <netlet.h>`

Public Member Functions

- virtual const std::string & getArchName ()=0

  Must return name of architecture the Netlet belongs to.

- virtual const std::string & getName ()=0

  Must return a unique name of the Netlet type.

- virtual bool isControlNetlet ()=0

  Returns true if the Netlet does not offer any transport service for applications, false otherwise.

- virtual INetlet * createNetlet (CNodeArchitecture *nodeA, IMessageScheduler *sched)=0

  Netlet factory function.

7.66.1 Detailed Description

Netlet meta data.

The Netlet meta data class has two purposes: On one hand, it describes the Netlet's properties, capabilities, etc., on the other hand, it provides a factory functions that will be used by the node architecture daemon to instantiate new Netlets of this type.

7.66.2 Member Function Documentation

7.66.2.1 virtual INetlet* INetletMetaData::createNetlet (CNodeArchitecture * *nodeA*, IMessageScheduler * *sched*) `[pure virtual]`

Netlet factory function.

Creates an instance of the Netlet type and returns a pointer to it.

The documentation for this class was generated from the following file:

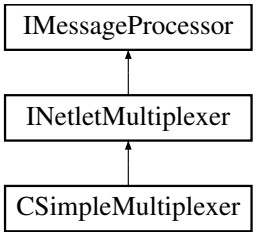
- include/netlet.h

7.67 INetletMultiplexer Class Reference

Generic Netlet multiplexer interface.

`#include <netletMultiplexer.h>`

Inheritance diagram for INetletMultiplexer::



Public Member Functions

- **INetletMultiplexer** (IMultiplexerMetaData *metaData, CNodeArchitecture *nodeA, IMessageScheduler *sched)
- IMultiplexerMetaData * getMetaData ()

  Return pointer to meta data.

- virtual void refreshNetlets ()=0

  Called if new Netlets of this architecture were added to the system.

Protected Attributes

- IMultiplexerMetaData * meta

  Pointer to meta data.

- CNodeArchitecture * **nodeArch**

7.67.1 Detailed Description

Generic Netlet multiplexer interface.

The documentation for this class was generated from the following file:

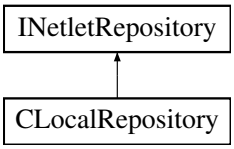
- include/netletMultiplexer.h

7.68 INetletRepository Class Reference

Interface for Netlet repository.

`#include <netletRepository.h>`

Inheritance diagram for INetletRepository::



Public Member Functions

- **INetletRepository** (CNodeArchitecture *nodeArch)
- virtual void initialize ()=0

  Initialize the repository, e.g. load Netlets etc.

Public Attributes

- std::list< INetletMultiplexer * > multiplexers

  Instantiated, arch specific multiplexers.

- std::list< INetlet * > netlets

  Instantiated, arch specific Netlets.

Protected Attributes

- CNodeArchitecture * **na**

7.68.1 Detailed Description

Interface for Netlet repository.

The documentation for this class was generated from the following file:

- include/netletRepository.h

7.69 IPacketMetaData Class Reference

Meta data for packets sent/received.

`#include <packets.h>`

7.69.1 Detailed Description

Meta data for packets sent/received.

This is architecture specific and must be implemented per architecture.

The documentation for this class was generated from the following file:

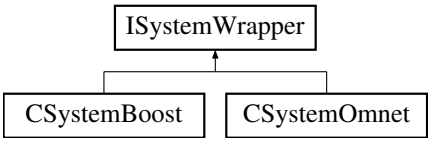
- include/packets.h

7.70 ISystemWrapper Class Reference

Host system wrapper.

`#include <systemWrapper.h>`

Inheritance diagram for ISystemWrapper::



Public Member Functions

- ISystemWrapper ()
  Constructor.
- virtual ~ISystemWrapper ()
  Destructor.
- virtual IMessageScheduler * getMainScheduler ()=0
  Returns the system-specific main scheduler (e.g. main thread).
- virtual IMessageScheduler * schedulerFactory ()=0
  Returns a system-specific scheduler.
- virtual void getNetAdapts (std::list< INetAdapt * > &netAdapts)=0
  Scan for or create Network Accesses.
- virtual void releaseNetAdapts ()=0
  Release previously initialized network accesses.
- virtual std::string getNodeName ()=0
  Return an arbitrary name for the node.

7.70.1 Detailed Description

Host system wrapper.

May not be static since there may be several instances when used in a simulator.

The documentation for this class was generated from the following file:

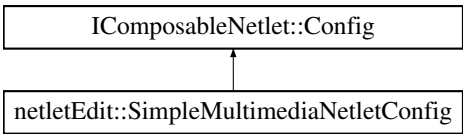
- include/systemWrapper.h

7.71 netletEdit::SimpleMultimediaNetletConfig Class Reference

Auto-generated Netlet configuration.

`#include <bb_simpleMultimediaNetlet.h>`

Inheritance diagram for netletEdit::SimpleMultimediaNetletConfig::



7.71.1 Detailed Description

Auto-generated Netlet configuration.

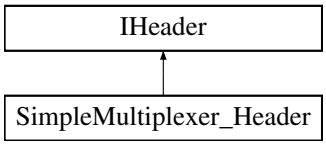
The documentation for this class was generated from the following file:

- src/netlets/simpleArch/configs/bb_simpleMultimediaNetlet.h

7.72 SimpleMultiplexer_Header Class Reference

Minimum header containing the source and destination IDs.

Inheritance diagram for SimpleMultiplexer_Header::



Public Member Functions

- virtual void serialize (CSerialBuffer *buffer)

  Serialize all relevant data into a byte buffer. Remember to do Little/Big Endian conversion.

- virtual void deserialize (CSerialBuffer *buffer)

  De-serialize all relevant data from a byte buffer. Remember to do Little/Big Endian conversion.

Public Attributes

- std::string **destNodeName**
- std::string **srcNodeName**
- SimpleHash **netletHash**

7.72.1 Detailed Description

Minimum header containing the source and destination IDs.

Header format is [A][B...][C][D...][EEEE] A is the string length of destNodeName (one byte) B is the string of destNodeName (variable length) C is the string length of srcNodeName (one byte) D is the string of srcNodeName (variable length) E is the hash of the Netlet identifier (four bytes)

The documentation for this class was generated from the following file:

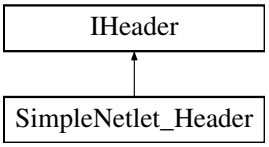
- src/netlets/simpleArch/simpleMultiplexer.cpp

7.73 SimpleNetlet_Header Class Reference

Minimum header containing a hash of the application's service ID.

`#include <simpleMultiplexer.h>`

Inheritance diagram for SimpleNetlet_Header::



Public Member Functions

- virtual void serialize (CSerialBuffer *buffer)

  Serialize all relevant data into a byte buffer. Remember to do Little/Big Endian conversion.

- virtual void deserialize (CSerialBuffer *buffer)

  De-serialize all relevant data from a byte buffer. Remember to do Little/Big Endian conversion.

Public Attributes

- SimpleHash **serviceHash**

7.73.1 Detailed Description

Minimum header containing a hash of the application's service ID.

7.73.2 Member Function Documentation

7.73.2.1 virtual void SimpleNetlet_Header::serialize (CSerialBuffer * *buffer*) [inline, virtual]

Serialize all relevant data into a byte buffer. Remember to do Little/Big Endian conversion.

Header format is [AAAA] A is the local connection id

Implements IHeader.

7.73.2.2 virtual void SimpleNetlet_Header::deserialize (CSerialBuffer * *buffer*) [inline, virtual]

De-serialize all relevant data from a byte buffer. Remember to do Little/Big Endian conversion.

Header format is [AAAA] A is the local connection id

Implements IHeader.

The documentation for this class was generated from the following file:

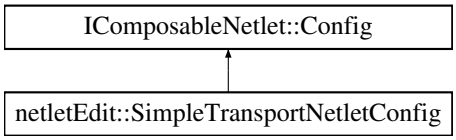
- src/netlets/simpleArch/simpleMultiplexer.h

7.74 netletEdit::SimpleTransportNetletConfig Class Reference

Auto-generated Netlet configuration.

`#include <bb_simpleTransportNetlet.h>`

Inheritance diagram for netletEdit::SimpleTransportNetletConfig::



7.74.1 Detailed Description

Auto-generated Netlet configuration.

The documentation for this class was generated from the following file:

- src/netlets/simpleArch/configs/bb_simpleTransportNetlet.h

Chapter 8

File Documentation

8.1 doc/reference/mainpage.h File Reference

Doxygen main page content.

8.1.1 Detailed Description

Doxygen main page content.

8.2 include/appConnector.h File Reference

```
#include "messages.h"
#include "netlet.h"
```

Classes

- class IAppConnector

  Stub for Application Interfaces. There is one connector per connection.

8.2.1 Detailed Description

appInterface.h

Created on: Jul 21, 2009 Author: denis

8.3 include/composableNetlet.h File Reference

Generic classes for message passing within the same address space.

```
#include "netlet.h"
#include "messages.h"
#include <string>
#include <map>
```

Classes

- class IBuildingBlock

  Interface for a building block.

- class IComposableNetlet

  A base class for a Netlet composed of building blocks.

- class IComposableNetlet::EUnknownBuildingBlock

  Thrown if a building block cannot be found.

- class IComposableNetlet::Config

  Simple Building Block configuration. This class will be generated by the Netlet editor.

8.3.1 Detailed Description

Generic classes for message passing within the same address space.

composableNetlet.h

(c) 2008-2009 Institut fuer Telematik, Universitaet Karlsruhe (TH), Germany

Created on: Aug 14, 2009 Author: denis

8.4 include/debug.h File Reference

Some general functions for debugging.

```
#include <string>
#include <iostream>
#include <stdlib.h>
#include <boost/format.hpp>
```

Classes

- class Debug

  Debug class.

Defines

- #define **FMT** boost::format
- #define **DBG_PRINT** Debug::print
- #define **DBG_ERROR**(msg...) Debug::error(boost::format("%1%:%2%: %3%") % __FILE__ % __LINE__ % (msg))
- #define **DBG_FAIL**(msg...) Debug::fail(boost::format("%1%:%2%: %3%") % __FILE__ % __LINE__ % (msg))

Typedefs

- typedef double(∗ **DebugTimeFunction**)()

8.4.1 Detailed Description

Some general functions for debugging.

debug.h

Please use only the following macros:

DBG_PRINT("Message"); // to print an arbitrary message

DBG_PRINT(FMT("Format %1, %2") % "Param1" % 42); // to print a formatted message

DBG_ERROR("Message"); // to print an error message

DBG_FAIL("Message"); // to print an error message and stop the execution immediately

(c) 2008-2009 Institut fuer Telematik, Universitaet Karlsruhe (TH), Germany

Created on: Dec 9, 2008 Author: denis

8.5 include/messages.h File Reference

Generic classes for message passing within the same address space.

```
#include "debug.h"
#include "morphableValue.h"
#include <stddef.h>
#include <list>
#include <map>
#include <exception>
#include <string>
```

Classes

- class IMessage

  Generic message interface.

- class IMessage::ETypeMismatch

  Thrown if the expected IMessage::Type differs from the actual one.

- class IMessage::EPropertyNotDefined

  Thrown if the requested property is not set.

- class CTimer

  Timer events.

- class IMessageScheduler

  Interface to message scheduler.

- class IMessageScheduler::EUnknowMessageProcessor

  Message processor is unknown to scheduler.

- class IMessageScheduler::EAlreadyRegistered

  Message processor already exists.

- class IMessageScheduler::EMessageLoop

  Message loop detected.

- class **IMessageScheduler::MessageQueue**
- class IMessageProcessor

  Any class processing a message.

- class IMessageProcessor::EUnhandledMessage

  Thrown if the message cannot be handled by the addressed message processor.

- class CSimpleMsgScheduler

  Simple scheduler. May only be used if ∗every∗ message processor runs in the same thread.

Typedefs

- typedef unsigned int LocalConnId
- typedef std::string ServiceId

8.5.1 Detailed Description

Generic classes for message passing within the same address space.

message.h

(c) 2008-2009 Institut fuer Telematik, Universitaet Karlsruhe (TH), Germany

Created on: Jul 13, 2009 Author: denis

8.5.2 Typedef Documentation

8.5.2.1 typedef unsigned int LocalConnId

Local ID to identify the application connection. This must not be used to identify the application/connection across the network (the mapping to the local connection ID has to be done by the respective architecture). 0 means "not an application connection"

8.5.2.2 typedef std::string ServiceId

Service identifier this is a (for now) arbitrary string to identify the remote service. An empty string means "no service".

8.6 include/morphableValue.h File Reference

Classes for a morphable value, e.g. a property, config parameter etc.

```
#include <string>
#include <list>
#include <exception>
```

Classes

- class CMorphableValue

  Base class for morphable values, e.g. properties, config parameters etc.

- class **CMorphableValue::Range< T >**
- class **CMorphableValue::EValueTypeMismatch**
- class **CMorphableValue::EContainerTypeMismatch**
- class CValueRange< T >

  Internal use only; container class for a value range.

- class CValueList< T >

  Internal use only; container class for a value range.

- class CBoolValue

  Facade for bool values.

- class CIntValue

  Facade for int values.

- class CDoubleValue

  Facade for int values.

- class CStringValue

  Facade for string values.

- class CIntRange

  Facade for int ranges.

- class CDoubleRange

  Facade for double ranges.

- class CIntList

  Facade for an int list.

- class CDoubleList

  Facade for an int list.

- class CStringList

  Facade for a string list.

8.6.1 Detailed Description

Classes for a morphable value, e.g. a property, config parameter etc.

morphableValue.h

Created on: Jun 30, 2009 Author: denis

8.7 include/nameAddrMapper.h File Reference

Interface for a name/address mapper.

```
#include "netlet.h"
#include <string>
#include <list>
```

Classes

- class **INameAddrMapper**

8.7.1 Detailed Description

Interface for a name/address mapper.

nameAddrMapper.h

There should be at least one implementation per architecture. Basic assumption: The name is a string and globally unique (e.g., an URL). The address format is completely undefined and architecture independent.

Created on: Aug 3, 2009 Author: denis

8.8 include/netAdapt.h File Reference

Generic interface for network accesses.

```
#include "messages.h"
#include "morphableValue.h"
#include <string>
#include <map>
```

Classes

- class INetAdapt

  Generic Network Adaptor Interface.

- class **INetAdapt::EPropertyNotDefined**

8.8.1 Detailed Description

Generic interface for network accesses.

netAdapt.h

(c) 2008-2009 Institut fuer Telematik, Universitaet Karlsruhe (TH), Germany

Created on: Dec 9, 2008 Author: denis

8.9 include/netlet.h File Reference

Generic interface for all Netlets and Netlet meta data.

```
#include "messages.h"
#include "debug.h"
#include <map>
#include <string>
```

Classes

- class INetlet

  Generic Netlet Interface.

- class INetletMetaData

  Netlet meta data.

Typedefs

- typedef std::string **NetletName**
- typedef std::map< std::string, std::map< std::string, INetletMetaData * > > NetletFactories

  Type for global collection of Netlet factories.

Variables

- NetletFactories netletFactories

  Global collection of available Netlet factories (allocated in Node Arch Daemon).

8.9.1 Detailed Description

Generic interface for all Netlets and Netlet meta data.

netlet.h

(c) 2008-2009 Institut fuer Telematik, Universitaet Karlsruhe (TH), Germany

Created on: Dec 9, 2008 Author: denis

8.9.2 Variable Documentation

8.9.2.1 NetletFactories netletFactories

Global collection of available Netlet factories (allocated in Node Arch Daemon).

Netlet factory for self-registering of Netlets

Note: Within omnet++, this factory class is only instantiated once for all node architecture instances.

Index of first map is the architecture ID, index of second map is the Netlet ID

8.10 include/netletMultiplexer.h File Reference

Generic interface for architecture specific multiplexers and their meta data.

```
#include "messages.h"
#include "debug.h"
#include <map>
#include <string>
```

Classes

- class INetletMultiplexer
  Generic Netlet multiplexer interface.
- class IMultiplexerMetaData
  Multiplexer meta data.

Typedefs

- typedef std::map< std::string, IMultiplexerMetaData * > MultiplexerFactories
  Type for global collection of multiplexer factories.

Variables

- MultiplexerFactories multiplexerFactories
  Global collection of multiplexer factories (allocated in Node Arch Daemon).

8.10.1 Detailed Description

Generic interface for architecture specific multiplexers and their meta data.

netletMultiplexer.h

(c) 2008-2009 Institut fuer Telematik, Universitaet Karlsruhe (TH), Germany

Created on: Dec 9, 2008 Author: denis

8.10.2 Variable Documentation

8.10.2.1 MultiplexerFactories multiplexerFactories

Global collection of multiplexer factories (allocated in Node Arch Daemon).

Multiplexer factory for self-registering of multiplexers.

Note: Within omnet++, this factory class is only instantiated once for all node architecture instances.

Map index is the ID / name of the architecture

8.11 include/netletRepository.h File Reference

Netlet repository interface.

```
#include "netlet.h"
#include "netletMultiplexer.h"
#include <list>
```

Classes

- class INetletRepository

  Interface for Netlet repository.

8.11.1 Detailed Description

Netlet repository interface.

netletSelector.h

(c) 2008-2009 Institut fuer Telematik, Universitaet Karlsruhe (TH), Germany

Created on: Apr 30, 2009 Author: denis

8.12 include/packets.h File Reference

Generic packet types.

```
#include "messages.h"
#include <list>
#include <vector>
#include <boost/format.hpp>
#include <netinet/in.h>
#include <string.h>
#include <assert.h>
```

Classes

- class CSerialBuffer

  Buffer for packet serialization.

- class IHeader

  Single protocol (Functional Block) header.

- class IPacketMetaData

  Meta data for packets sent/received.

- class CPacket

  Container for a data frame.

8.12.1 Detailed Description

Generic packet types.

packets.h

(c) 2008-2009 Institut fuer Telematik, Universitaet Karlsruhe (TH), Germany

Created on: Apr 9, 2009 Author: denis

8.13 include/systemWrapper.h File Reference

Generic interface for system wrappers.

```
#include <string>
#include <list>
#include "messages.h"
#include "netAdapt.h"
```

Classes

- class ISystemWrapper

  Host system wrapper.

8.13.1 Detailed Description

Generic interface for system wrappers.

systemWrapper.h

Created on: Dec 10, 2008 Author: denis

8.14 src/buildingBlocks/bitmapFragBb.cpp File Reference

Fragment a bitmap into tiles.

```
#include "bitmapFragBb.h"
#include "packets.h"
```

8.14.1 Detailed Description

Fragment a bitmap into tiles.

bitmapFragBb.cpp

Created on: Aug 17, 2009 Author: denis

8.15 src/buildingBlocks/bitmapFragBb.h File Reference

Fragment a bitmap into tiles.

```
#include "composableNetlet.h"
#include "messages.h"
```

Classes

- class CBitmapFragBb

  Building block to fragment a bitmap into tiles.

8.15.1 Detailed Description

Fragment a bitmap into tiles.

bitmapFragBb.h

Created on: Aug 17, 2009 Author: denis

8.16 src/buildingBlocks/demoCodecBb.cpp File Reference

Demo codec building block.

```
#include "demoCodecBb.h"
#include "packets.h"
```

8.16.1 Detailed Description

Demo codec building block.

demoCodecBb.cpp

Created on: Aug 17, 2009 Author: denis

8.17 src/buildingBlocks/demoCodecBb.h File Reference

Demo codec building block.

```
#include "composableNetlet.h"
#include "messages.h"
```

Classes

- class CDemoCodecBb

  Demo codec building block.

8.17.1 Detailed Description

Demo codec building block.

demoCodecBb.h

Created on: Aug 17, 2009 Author: denis

8.18 src/buildingBlocks/fecBb.cpp File Reference

FEC building block.

```
#include "fecBb.h"
#include "packets.h"
```

8.18.1 Detailed Description

FEC building block.

fecBb.cpp

Created on: Aug 14, 2009 Author: denis

8.19 src/buildingBlocks/fecBb.h File Reference

FEC building block.

```
#include "composableNetlet.h"
#include "messages.h"
```

Classes

- class CFecBb

  FEC building block.

8.19.1 Detailed Description

FEC building block.

fecBb.h

Created on: Aug 14, 2009 Author: denis

8.20 src/daemon/localRepository.cpp File Reference

Local Netlet repository.

```
#include "localRepository.h"
#include "nodeArchitecture.h"
#include <boost/filesystem.hpp>
#include <dlfcn.h>
```

8.20.1 Detailed Description

Local Netlet repository.

localRepository.cpp

Created on: May 8, 2009 Author: denis

8.21 src/daemon/localRepository.h File Reference

Local Netlet repository.

```
#include "netletRepository.h"
#include <list>
#include <string>
#include <boost/extension/shared_library.hpp>
```

Classes

- class CLocalRepository
  Local Netlet repository.

8.21.1 Detailed Description

Local Netlet repository.

localRepository.h

(c) 2008-2009 Institut fuer Telematik, Universitaet Karlsruhe (TH), Germany

Created on: May 8, 2009 Author: denis

8.22 src/daemon/netAdaptBroker.h File Reference

Network Access Broker (Manager).

```
#include "messages.h"
#include "netAdapt.h"
#include <map>
#include <list>
#include <string>
```

Classes

- class CNetAdaptBroker
  Network Adaptor Broker.

8.22.1 Detailed Description

Network Access Broker (Manager).

netAdaptBroker.h

(c) 2008-2009 Institut fuer Telematik, Universitaet Karlsruhe (TH), Germany

Created on: Dec 9, 2008 Author: denis

8.23 src/daemon/netletSelector.h File Reference

Netlet selection implementation.

```
#include "messages.h"
#include "appConnector.h"
#include "nameAddrMapper.h"
#include <map>
#include <list>
```

Classes

- class CNetletSelector

  Implements Netlet selection functionality.

8.23.1 Detailed Description

Netlet selection implementation.

netletSelector.h

(c) 2008-2009 Institut fuer Telematik, Universitaet Karlsruhe (TH), Germany

Created on: Feb 12, 2009 Author: denis

8.24 src/daemon/nodeArchitecture.h File Reference

Node architecture daemon.

```
#include "netlet.h"
#include "netletMultiplexer.h"
#include "netletRepository.h"
#include "appConnector.h"
#include "systemWrapper.h"
```

Classes

- class CNodeArchitecture

  Node architecture daemon class.

Variables

- MultiplexerFactories multiplexerFactories
- NetletFactories netletFactories

8.24.1 Detailed Description

Node architecture daemon.

nodeArchitecture.h

(c) 2008-2009 Institut fuer Telematik, Universitaet Karlsruhe (TH), Germany

Created on: Dec 8, 2008 Author: denis

8.24.2 Variable Documentation

8.24.2.1 MultiplexerFactories multiplexerFactories

Multiplexer factory for self-registering of multiplexers.

Note: Within omnet++, this factory class is only instantiated once for all node architecture instances.

Map index is the ID / name of the architecture

8.24.2.2 NetletFactories netletFactories

Netlet factory for self-registering of Netlets

Note: Within omnet++, this factory class is only instantiated once for all node architecture instances.

Index of first map is the architecture ID, index of second map is the Netlet ID

8.25 src/netlets/simpleArch/configs/bb_simpleMultimediaNetlet.h File Reference

```
#include "composableNetlet.h"
```

Namespaces

- namespace netletEdit

Classes

- class netletEdit::SimpleMultimediaNetletConfig

  Auto-generated Netlet configuration.

8.25.1 Detailed Description

bb_simpleComposedNetlet.h

TODO: This file should be AUTOGENERATED

Created on: Aug 14, 2009 Author: denis

8.26 src/netlets/simpleArch/configs/bb_simpleTransportNetlet.h File Reference

```
#include "composableNetlet.h"
```

Namespaces

- namespace netletEdit

Classes

- class netletEdit::SimpleTransportNetletConfig

  Auto-generated Netlet configuration.

8.26.1 Detailed Description

bb_simpleTransportNetlet.h

TODO: This file should be AUTOGENERATED

Created on: Aug 14, 2009 Author: denis

8.27 src/netlets/simpleArch/simpleComposedNetlet.cpp File Reference

Simple composed Netlet for Simple Architecture.

```
#include "simpleComposedNetlet.h"
#include "simpleMultiplexer.h"
#include "nodeArchitecture.h"
#include "netAdaptBroker.h"
#include "packets.h"
```

8.27.1 Detailed Description

Simple composed Netlet for Simple Architecture.

simpleComposedNetlet.cpp

Created on: Aug 14, 2009 Author: denis

8.28 src/netlets/simpleArch/simpleComposedNetlet.h File Reference

Simple composed Netlet for Simple Architecture.

```
#include "composableNetlet.h"
#include "simpleMultiplexer.h"
```

Classes

- class CSimpleComposedNetlet

  A Netlet taking its actual building block configuration from a generated source file.

8.28.1 Detailed Description

Simple composed Netlet for Simple Architecture.

simpleComposedNetlet.h

Created on: Aug 14, 2009 Author: denis

8.29 src/netlets/simpleArch/simpleMultimediaNetlet.cpp File Reference

Simple composed Netlet.

```
#include "simpleMultimediaNetlet.h"
#include "simpleMultiplexer.h"
#include "nodeArchitecture.h"
#include "netAdaptBroker.h"
#include "packets.h"
#include "bitmapFragBb.h"
#include "demoCodecBb.h"
#include "fecBb.h"
#include "configs/bb_simpleMultimediaNetlet.h"
```

8.29.1 Detailed Description

Simple composed Netlet.

simpleComposedNetlet.cpp

Created on: Aug 14, 2009 Author: denis

8.30 src/netlets/simpleArch/simpleMultimediaNetlet.h File Reference

Simple example of a composed multimedia Netlet.

```
#include "simpleComposedNetlet.h"
#include "simpleMultiplexer.h"
```

Classes

- class CSimpleMultimediaNetlet

  A Netlet taking its actual building block configuration from a generated source file.

- class CSimpleMultimediaNetletMetaData

  Simple composed Netlet meta data.

Variables

- const std::string & **SIMPLE_MULTIMEDIA_NETLET_NAME** = "netlet://tm.uka.de/itm/SimpleArchitecture/SimpleMultimediaNetlet"

8.30.1 Detailed Description

Simple example of a composed multimedia Netlet.

simpleMultimediaNetlet.h

(c) 2008-2009 Institut fuer Telematik, Universitaet Karlsruhe (TH), Germany

Created on: Aug 14, 2009 Author: denis

8.31 src/netlets/simpleArch/simpleMultiplexer.h File Reference

Multiplexer for "Simple Architecture".

```
#include "netletMultiplexer.h"
#include "nameAddrMapper.h"
#include "packets.h"
```

Classes

- class SimpleNetlet_Header
  Minimum header containing a hash of the application's service ID.
- class CSimpleNameAddrMapper
  Name/address mapper implementation for SimpleArchitecture.
- class CSimpleMultiplexer
  Simple Netlet multiplexer.
- class CSimpleMultiplexerMetaData
  Simple multiplexer meta data class.

Typedefs

- typedef unsigned long **SimpleHash**
- typedef std::map< std::string, INetAdapt * > **SimpleFib**

Variables

- const std::string & **SIMPLE_ARCH_NAME**

8.31.1 Detailed Description

Multiplexer for "Simple Architecture".

simpleMultiplexer.h

The Simple Architecture has the following features:

- A simple transport Netlet. At the moment, it just hands the application's data to the multiplexer.
- A simple routing Netlet. A very dumb one, just creating a forwarding information base (FIB).
- A forwarding mechanism based on the FIB. This mechanism is realized within the multiplexer. Note, that other architectures may do the forwarding in a Netlet.

Created on: Dec 9, 2008 Author: denis

8.32 src/netlets/simpleArch/simpleNetlet.h File Reference

Simple example Netlet.

```
#include "netlet.h"
#include "simpleMultiplexer.h"
```

Classes

- class CSimpleNetlet
  Simple Netlet example.
- class CSimpleNetletMetaData
  Simple Netlet meta data.

Variables

- const std::string & **SIMPLE_NETLET_NAME** = "netlet://tm.uka.de/itm/SimpleArchitecture/SimpleNetlet"

8.32.1 Detailed Description

Simple example Netlet.

simpleNetlet.h

Created on: Dec 9, 2008 Author: denis

8.33 src/netlets/simpleArch/simpleRoutingNetlet.h File Reference

Simple routing Netlet.

```
#include "netlet.h"
#include "messages.h"
#include "netAdapt.h"
```

Classes

- class CSimpleRoutingNetlet

  Simple neighbor discovery protocol.

- class CSimpleRoutingNetletMetaData

  Simple Routing Netlet meta data.

Variables

- const std::string & **SIMPLE_ROUTING_NETLET_NAME** = "net-let://tm.uka.de/itm/SimpleArchitecture/SimpleRoutingNetlet"

8.33.1 Detailed Description

Simple routing Netlet.

simpleRoutingNetlet.h

The Simple Architecture has the following features:

- A simple transport Netlet. At the moment, it just hands the application's data to the multiplexer.
- A simple routing Netlet. A very dumb one, just creating a forwarding information base (FIB).
- A forwarding mechanism based on the FIB. This mechanism is realized within the multiplexer. Note, that other architectures may do the forwarding in a Netlet.

The routing uses the node names as identifier and locator. Thus, we're basically doing the same mistake all over again in this dumb demo architecture.

Created on: May 26, 2009 Author: denis

8.34 src/netlets/simpleArch/simpleTransportNetlet.cpp File Reference

Simple example of a composed transport Netlet.

```
#include "simpleTransportNetlet.h"
#include "simpleMultiplexer.h"
#include "nodeArchitecture.h"
#include "netAdaptBroker.h"
#include "packets.h"
#include "bitmapFragBb.h"
#include "demoCodecBb.h"
#include "fecBb.h"
#include "configs/bb_simpleTransportNetlet.h"
```

8.34.1 Detailed Description

Simple example of a composed transport Netlet.

simpleTransportNetlet.cpp

(c) 2008-2009 Institut fuer Telematik, Universitaet Karlsruhe (TH), Germany

Created on: Aug 14, 2009 Author: denis

8.35 src/netlets/simpleArch/simpleTransportNetlet.h File Reference

Simple example of a composed transport Netlet.

```
#include "simpleComposedNetlet.h"
#include "simpleMultiplexer.h"
```

Classes

- class CSimpleTransportNetlet

  A Netlet taking its actual building block configuration from a generated source file.

- class CSimpleTransportNetletMetaData

  Simple composed Netlet meta data.

Variables

- const std::string & **SIMPLE_TRANSPORT_NETLET_NAME** = "netlet://tm.uka.de/itm/SimpleArchitecture/SimpleTransportNetlet"

8.35.1 Detailed Description

Simple example of a composed transport Netlet.

simpleTransportNetlet.h

Created on: Aug 14, 2009 Author: denis

8.36 src/targets/boost/systemBoost.h File Reference

System wrapper for Boost supported systems (e.g. Linux, Windows).

```
#include "systemWrapper.h"
```

Classes

- class CSystemBoost

  Boost ASIO wrapper.

8.36.1 Detailed Description

System wrapper for Boost supported systems (e.g. Linux, Windows).

systemBoost.h

(c) 2008-2009 Institut fuer Telematik, Universitaet Karlsruhe (TH), Germany

Created on: Dec 10, 2008 Author: denis

8.37 src/targets/omnetpp/systemOmnet.h File Reference

OMNeT++ system wrapper.

```
#include "systemWrapper.h"
#include "messagesOmnet.h"
#include "netAdaptOmnet.h"
#include <string>
#include <map>
```

Classes

- class CSystemOmnet
  OMNeT++ Wrapper.

Defines

- #define **SYS_OMNET_MTU** 2048

8.37.1 Detailed Description

OMNeT++ system wrapper.

systemOmnet.h

CSystemOmnet is a Omnet++ module that registers itself at Omnet. It gets instantiated by Omnet for every node as defined in the .ned-file. During initialization, it instantiates the Node Architecture - hence, we have ∗multiple∗ instances within the Omnet address space. This is different to standalone targets, where the Node Architecture is really a singleton.

(c) 2008-2009 Institut fuer Telematik, Universitaet Karlsruhe (TH), Germany

Created on: Jan 14, 2009 Author: denis

Index

Bibliography

[1] L. Völker, D. Martin, I. El Khayat, C. Werle, and M. Zitterbart, “A Node Architecture for 1000 Future Networks”, in *Proceedings of the International Workshop on the Network of the Future 2009*. Dresden, Germany: IEEE, Jun. 2009.