Tuscarora Framework API documentation

Version 2.1

Generated by Doxygen 1.8.11

Contents

Chapter 1

Introduction and Naming Conventions

1.1 Abbreviation

- PWI : Pattern Writer's Interface
- PAL : Platform Abstraction Layer
- WF : Waveform

1.2 Layering

The Tuscarora framework has 3 layers, from bottom to top these are:

1. **The Waveform layer**: This layer interacts with the radio. The framework defines the APIs of this layer, but the implementation of this layer can be provided by the platform provider. For ns-3/DCE simulations the layer is provided by Samraksh. The APIs and implementations for this layer are under the Waveform namespace.
2. **Framework Layer**: This layer is the core of Tuscarora, it defines and implements the Pattern Writer's Interface (PWI) and the Core framework modules. Internally it also contains the 'Core' namespace, which defines and implements internal functions not exposed to a pattern writer. A pattern writter typical needs to be familiar only with the APIs definted in the PWI namespace.
3. **Pattern Layer**: This is implemented by a pattern writer, using the PWI interfaces.

Apart from the above layers, a number of Platform Abstraction Layer modules have defined to make the porting of Tuscarora easy. These are under the PAL namespace.

1.3 Naming Convention

- **Class Names**: Class and Method names are PolyCapitalized.
- **Template Class Names**: Templated class names end with a capital "T" and are PolyCapitalized. These must not be confused with names ending with a "_t" which imply typedef types.
- **Variable Names**: Variables and method parameters begin with a lower case letter, but are polyCapitalized.

- **Interfaces**: Classes that end with an "I" are interface definitions. C++ interfaces are "Pure Virtual Classes", i.e., they have at least one "Pure Virtual Method". These classes cannot be directly instantiated and need to be inherited and extended to be used. Interfaces may be templated in which case they still end with an I.
- **Enums**: Enum names end with an "E" and are PolyCapitalized.
- **Structs vs Classes**: There are only minor differences between a Struct and Class in C++. One of these is that in a Struct all members are by default "Public". However, we use a Struct when no Methods are part of the object, and use a Class when Methods are part of the object. So a Struct is simply a way of bundling a bunch of variables.

Chapter 2

Namespace Index

2.1 Namespace List

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

Chapter 3

Hierarchical 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

Namespace Documentation

5.1 Core Namespace Reference

Frameworks core services.

Namespaces

- Dataflow

  Types and classes related to the dataflow service.
- Discovery

  Types and classes related to the network discovery service.
- Estimation

  Estimation related classes and types.
- Naming

  Types and classes related to Naming service.
- Policy

  Types and classes related to Policy Manager service.

Classes

- class ConfigureFramework_I
- struct Link

  Structure to store information about a link in the framework and pattern layers including its metrics, 33b.
- struct LinkMetrics

  Structure used by the framwork to expose link metrics to the pattern, 28 bytes with the use of float for metrics.
- class MessageT

  A class that facilitates data exchange between Patterns and the framework.
- class NeighborTableI
- struct NeighborUpdateParam

  Structure to be used as the parameter for the function handling the neighbor change events.
- struct NodeInfo
- class PiggyBackeeBase
- class PiggyBackeeI
- class PiggyBackerI

This class will be implemented by the framework to provide piggybacking service.

- struct WF_Link

  Structure to store information about a link received from the waveform layer.
- struct WF_LinkEstimates
- struct WF_LinkEstimationParam

  Structure to be used as the parameter for the function handling the neighbor change events.
- class WF_LinkId
- struct WF_LinkMetrics

Typedefs

- typedef WF_LinkId< NodeId_t > **LinkId**
- typedef WF_LinkEstimationParam< uint64_t > WF_LinkEstimationParam_n64_t

  A delegate definition which will be called to notify events about a node or link.
- typedef Delegate< void, WF_LinkEstimationParam< uint64_t > > **WF_LinkChangeDelegate_n64_t**
- typedef Delegate< void, NeighborUpdateParam > **NeighborDelegate**
- typedef MessageT< NodeId_t > **FMessage_t**

Enumerations

- enum LinkTypeE { TwoWay, SinkOnly, SourceOnly }

  Enum for different types of links.
- enum NeighborChangeTypeE { **NBR_NEW**, NBR_DEAD, NBR_UPDATE }

  Enum for the type of neighbor change event.
- enum NeighborTypeE { **REGULAR_NODE**, RADIOSILENT_NODE, C3_NODE }

  A neighbor can be one of these types.
- enum **PB_TypeE** { **PB_TEST**, **PB_ASNP_CONTROL**, **PB_TIMESYNC**, **PB_LINK_ESTIMATION** }

Functions

- template<class WF_AddressType >
  struct Core::WF_LinkEstimationParam **__attribute__** ((packed, aligned(1)))

Variables

- struct Core::NeighborUpdateParam **__attribute__**
- const uint8_t **RELAY_FLAG** = 0x01
- const uint8_t **JAMMER_FLAG** = 0x02
- const uint8_t **BLACK_LISTED_FLAG** = 0x04

5.1.1 Detailed Description

Frameworks core services.

5.1.2 Enumeration Type Documentation

5.1.2.1 enum Core::LinkTypeE

Enum for different types of links.

Enumerator

TwoWay Two way communicaiton possible using the link.

SinkOnly Link can only be used to send messages, the node at the other end will not transmit.

SourceOnly Link can only be use to receive message, the node at the other end will not receive messages.

5.1.2.2 enum Core::NeighborChangeTypeE

Enum for the type of neighbor change event.

Enumerator

NBR_DEAD New neighbor detected.

NBR_UPDATE A previously existing neighbor is now dead. An update about an existing neighbor

5.1.2.3 enum Core::NeighborTypeE

A neighbor can be one of these types.

Enumerator

RADIOSILENT_NODE Comms are possible with this node.

C3_NODE Special node, can only receive messages.

5.2 Core::Dataflow Namespace Reference

Types and classes related to the dataflow service.

Classes

- struct DataStatusParam

  A structure that is passed as the parameter to the DataNotifier Delegate.
- class FragmentAndReassembleI

  Specifies a definition for the fragementation and reassembly service.

Typedefs

- typedef Delegate< void, DataStatusParam > **DataflowDelegate_t**

Enumerations

- enum DataStatusTypeE {
  PDN_ERROR, **PDN_ERROR_PKT_TOOBIG**, **PDN_FW_RECV**, PDN_WF_RECV,
  PDN_WF_SENT, PDN_DST_RECV, PDN_BROADCAST_NOT_SUPPORTED }
- enum **FragmentStatusTypeE** {
  FRAG_DOES_NOT_EXIST = 0, **FRAG_FRAG_CREATED** = 1, **FRAG_WF_RECV** = 2, **FRAG_WF_SENT** = 3,
  FRAG_DST_RECV = 4 }
- enum WF_DataStatusTypeE { **WDN_WF_RECV** =1, WDN_WF_SENT, WDN_DST_RECV }

Variables

- const uint8_t **PDN_WF_RECV_MASK** =0x02
- const uint8_t **PDN_WF_SENT_MASK** =0x04
- const uint8_t **PDN_RECV_DEST_MASK** =0x08

5.2.1 Detailed Description

Types and classes related to the dataflow service.

5.2.2 Enumeration Type Documentation

5.2.2.1 enum Core::Dataflow::DataStatusTypeE

Enumerator

PDN_WF_RECV Acknowledgement when framework reaceived message.

PDN_WF_SENT Acknowlegment that Waveform received the messsage.

PDN_DST_RECV Acknowlegment that Waveform sent the messsage out on channel.

PDN_BROADCAST_NOT_SUPPORTED Acknowlegment that (some module on) destination received the message.

5.2.2.2 enum Core::Dataflow::WF_DataStatusTypeE

Enumerator

WDN_WF_SENT Acknowlegment that Waveform received the messsage.

WDN_DST_RECV Acknowlegment that Waveform sent the messsage out on channel.

5.3 Core::Discovery Namespace Reference

Types and classes related to the network discovery service.

Classes

- class DiscoveryLogHeader
- class NetworkDiscoveryI

  Inteface definition for a network discovery service.
- struct PotentialNeighbor
- struct PotentialNeighborDelegateParam
- class PotentialNeighborRegistryI

Typedefs

- typedef Delegate< void, PotentialNeighborDelegateParam > PotentialNeighborDelegate

  A delegate definition which will be called to notify events about a node or link.

Enumerations

- enum **NetworkDiscoveryTypeE** {
  GLOBAL_DISCOVERY, **ORACLE_DISCOVERY**, **LONG_LINK_DISCOVERY**, **ZSC_DISCOVERY**,
  FLOOD_DISCOVERY, **NULL_DISCOVERY** }
- enum PNbrUpdateTypeE { **PNBR_INVALID**, PNBR_UPDATE }

  Enum for the type of neighbor change event.
- enum PotentialNeighborTypeE { **BEYOND_COMM_PNODE**, ALTERNATEMODE_PNODE, SCHEDULED↩
  _PNODE, HOSTILE_PNODE }

  Enum for the type of potential neighbor.

5.3.1 Detailed Description

Types and classes related to the network discovery service.

5.3.2 Enumeration Type Documentation

5.3.2.1 enum Core::Discovery::PNbrUpdateTypeE

Enum for the type of neighbor change event.

Enumerator

PNBR_UPDATE Previous info about tihs pnbr is not valid anymore. An update a potential nbr

5.3.2.2 enum Core::Discovery::PotentialNeighborTypeE

Enum for the type of potential neighbor.

Enumerator

ALTERNATEMODE_PNODE Node is beyond comm range currently.

SCHEDULED_PNODE Comms currently not possible, but may be possible with a mode change.

HOSTILE_PNODE This node has a predictable location/mobility. Comms currently not possible, but might be available for comms in future.

5.4 Core::Estimation Namespace Reference

Estimation related classes and types.

Classes

- class EstimationLogI
- class EstimatorCallback_I
- class LinkEstimatorI

  Interface definition for a link estimator which is implemented by the framework.
- class NeighborhoodControlI

  Defines the interface that must be implemented by the Pattern to customize its neighborhood.
- struct SendEstMsgParam

Typedefs

- typedef struct Core::Estimation::SendEstMsgParam **SendEstMsgParam**
- typedef Delegate< void, SendEstMsgParam > **SendEstimationMessageDelegate_t**

Enumerations

- enum **EstimationLogE** {
  LINK_ADDED = 1, **LINK_REMOVED** = 2, **LINK_UPDATED** = 3, **LINK_SKIPPED** = 4,
  LINK_SENT = 5, **LINK_POTENTIAL_ADD** = 6, **LINK_POTENTIAL_DEL** = 7 }
- enum EstimatorTypeE {
  WF_PROXY, FW_RANDOM_EST, FW_SCHEDULED_EST, FW_CONFLICTAWARE_EST,
  FW_TEST_EST }

  Enum for type of framework estimator.

5.4.1 Detailed Description

Estimation related classes and types.

5.4.2 Enumeration Type Documentation

5.4.2.1 enum Core::Estimation::EstimatorTypeE

Enum for type of framework estimator.

Enumerator

FW_RANDOM_EST provided by waveform

FW_SCHEDULED_EST Provided by framework, a randomized periodic estimator.

FW_CONFLICTAWARE_EST Provided by framework, based on a schedule that is known.

FW_TEST_EST Provided by framework, based on a schedule that is known, also identifies message loss due to schedule collision.

5.5 Core::Naming Namespace Reference

Types and classes related to Naming service.

Classes

- class PatternNameServiceI

  Generic interface specification for a name service.

Enumerations

- enum PatternTypeE {
  UNKNOWN_PTN, **COP_PTN**, **OLSR_PTN**, **CENSUS_PTN**,
  FWP_PTN, **RBCAST_PTN**, **IWAVE_PTN**, **NTKR_PTN**,
  SENTRY_PTN, **ELG_PTN**, **GOSSIP_PTN**, **SHARE_PTN**,
  NON_STD_PTN }

  Enum for the support Pattern Types.

Functions

- PatternId_t **GetNewPatternInstanceId** (PatternTypeE ptype, const char uniqueName[128])

5.5.1 Detailed Description

Types and classes related to Naming service.

5.6 Core::Policy Namespace Reference

Types and classes related to Policy Manager service.

Classes

- class PolicyManagerI

  Defines the interface for the policy manager.

Enumerations

- enum PacketSchedulingPolicyE { **PRIORITY_FCFS**, **FCFS** }

  types of packet scheduling currently
- enum **LinkSelectionPolicyE** {
  PATTERN_DEFINED, **BEST_QUALITY**, **LEAST_DELAY**, **LEAST_ENERGY**,
  PREFFERED_WAVEFORM }

5.6.1 Detailed Description

Types and classes related to Policy Manager service.

5.7 ExternalServices Namespace Reference

Definitions and APIs for External Services. Tuscarora does not provide these services.

Classes

- class GlobalNameI
- struct GPSLocation
- struct Location2D
- struct Location3D
- class LocationServiceI
- class NeighborTimeI
- class TimeStampingI
- class TopologyInformationI

  Defines an interface for the topology information service. In our current thinking, this is just another Network Discovery service, hence has the same APIs.

5.7.1 Detailed Description

Definitions and APIs for External Services. Tuscarora does not provide these services.

5.8 Patterns Namespace Reference

Patern Writers Interface definition and types.

Classes

- class PatternBase

  Defines the abstract base class for patterns.
- class PatternNeighborTableI

  Interface definition for the Neighbor service, that maintains a sorted list of neighbors based on their metric.
- class PatternShim_I

Typedefs

- typedef AVLBSTElement< NodeId_t > **NeigborContainerElementType**
- typedef AVLBST_T< NodeId_t > **NeigborContainerType**

Enumerations

- enum **PatternStateE** { **UNREGISTERED**, **REGISTERED**, **EXECUTING**, **ERROR** }
- enum **PatternRequestStateE** { **NONE_PENDING**, **WAITING_FOR_CONTROL_RESPONSE**, **WAITING_FOR_DATA_RESPONSE** }

Functions

- Framework_I & **GetPatternShim** ()

5.8.1 Detailed Description

Patern Writers Interface definition and types.

5.9 PWI Namespace Reference

Patern Writers Interface definition and types.

Namespaces

- Neighborhood

  Types and APIs for the Neighborhood service.

Classes

- struct AddDestinationResponse_Data
- struct CancelDataResponse_Data
- struct ControlResponseParam

  Structure used as parameter by the framework when sending the Control Response Event.
- struct DataStatusResponse_Data
- class Framework_I

  Defines the generic interations between the Pattern and the Framework.
- struct FrameworkAttributes

  Describtes the attributes of the framework.
- struct NewPatternInstanceResponse_Data
- class PatternEvent
- class PatternEventDispatch_I
- struct RegistrationResponse_Data
- struct ReplacePayloadResponse_Data
- struct SelectDataNotificationResponse_Data
- struct SetLinkThresholdResponse_Data

Typedefs

- typedef PatternEvent< PTN_CONT_RES_EVT, ControlResponseParam > PTN_ControlResponseEvent_t

  Event sent to Pattern as a response to control requests.
- typedef PatternEvent< PTN_NBR_CHG_EVT, NeighborUpdateParam > PTN_NeighborUpdateEvent_t

  Event sent to Pattern to notify it of updates in the neighborhood.
- typedef PatternEvent< PTN_RECV_MSG_EVT, FMessage_t & > PTN_ReceivedMessageEvent_t

  Event sent to Pattern when a new message is received on one of the waveforms for the pattern.
- typedef PatternEvent< PTN_DATA_NTY_EVT, DataStatusParam > PTN_DatNotificationEvent_t

  Event sent to the pattern to notify about the status of message the pattern sent to the framework.
- typedef Delegate< void, FMessage_t & > **RecvMessageDelegate_t**
- typedef Delegate< void, ControlResponseParam > **ControlResponseDelegate_t**

Enumerations

- enum **PTN_EventNameE** { **PTN_NBR_CHG_EVT**, **PTN_RECV_MSG_EVT**, **PTN_DATA_NTY_EVT**, **PTN_CONT_RES_EVT** }
- enum ControlResponseTypeE {
  PTN_RegisterResponse =2, **PTN_AttributeResponse** =3, **PTN_SetLinkThresholdResponse** =4, **PTN_AddDestinationResponse** =5,
  PTN_ReplacePayloadResponse =6, **PTN_CancelDataResponse** =7, **PTN_SelectDataNotificationResponse** =8 }

  Enum that describes the types of constrol responses sent by the framework to the pattern.

Functions

- Framework_I & GetFrameworkInterface ()

  Returns a reference to an implentation of the pattern interface.
- Framework_I & **GetFrameworkShim** ()
- void **SetFrameworkInterface** (Framework_I &fi)
- void **GetWaveformIDs** (WaveformId_t *idArray, uint16_t noOfWaveforms)
- uint16_t **GetNumberOfWaveforms** ()

5.9.1 Detailed Description

Patern Writers Interface definition and types.

5.10 PWI::Neighborhood Namespace Reference

Types and APIs for the Neighborhood service.

Classes

- class LinkComparatorI

Enumerations

- enum LinkComparatorTypeE { **QUALITY_LC**, **COST_LC**, **QUAL_XMIT_DELAY_LC**, **ENERGY_LC** }

  Approved list of link comparators.

5.10.1 Detailed Description

Types and APIs for the Neighborhood service.

5.11 Waveform Namespace Reference

Waveform Interface namespace.

Classes

- class AddressMapper
- struct FragmentationHeader
- class PacketAdaptor
- struct PacketMetadata

  Structure for packet metadata that should be supported by all waveforms.
- struct SoftwareAcknowledgement
- class Waveform_64b_I
- class Waveform_I

  Defines the generic iteractions for waveforms.
- class WaveformAddressMapper
- class WaveformAddressMapper< uint64_t >
- class WaveformBase
- struct WF_AddDestinationResponse_Param
- struct WF_Attributes
- struct WF_CancelDataResponse_Param
- struct WF_ControlResponseParam
- struct WF_DataStatusParam
- class WF_Event
- class WF_MessageBase

  Base class for all waveform messages.
- class WF_MessageT

  A class that facilitates data exchange between the Waveform and the framework.
- struct WF_RecvMsgParam
- struct WF_ReplacePayloadResponse_Param
- struct WF_ScheduleUpdateParam

Typedefs

- typedef WF_Event< WF_CONT_RES_EVT, WF_ControlResponseParam > **WF_ControlResponseEvent_t**
- typedef WF_Event< WF_SCHD_UPD_EVT, WF_ScheduleUpdateParam > **WF_ScheduleUpdateEvent_t**
- typedef WF_MessageT< uint32_t > **WF_Message_n32_t**
- typedef WF_MessageT< uint64_t > **WF_Message_n64_t**
- typedef WF_DataStatusParam< uint32_t > **WF_DataStatusParam_n32_t**
- typedef WF_DataStatusParam< uint64_t > **WF_DataStatusParam_n64_t**

Enumerations

- enum **WF_EventNameE** {
  WF_GREET_EVT, **WF_LINK_EST_EVT**, **WF_RECV_MSG_EVT**, **WF_DATA_NTY_EVT**,
  WF_CONT_RES_EVT, **WF_SCHD_UPD_EVT** }
- enum **WF_MessageFlagsE** { **WF_FLG_PIGGYBACKING** = 0x01, **WF_FLG_TIMESTAMP** = 0x02, **WF_F↩
  LG_FRAGMENTED** = 0x04 }
- enum WF_MessageStatusE {
  WF_ST_BUSY, **WF_ST_PKT_TOO_BIG**, **WF_ST_SUCCESS**, **WF_ST_BCAST_NOT_SUPPTD**,
  WF_ST_BUFFER_FULL, **WF_ST_CHANNEL_BUSY**, **WF_ST_ACK_NOT_RECV**, WF_ST_MAX_RETRI↩
  ES_ERR,
  WF_ST_DEADNBR }
- enum WF_AddressTypeE { **WFA_UINT64**, **WFA_ETH_MAC** }

  Enum defining the supported waveform address types.
- enum WF_AntennaTypeE { **OMNI_ANT** = 1, **DIRECTIONAL_ANT** = 2 }

  Enum for supported Antenna types.
- enum WF_TypeE { **ALWAYS_ON_LOCAL_WF** = 2, **TDMA_LOCAL_WF** = 3, **LONG_LINK_WF** = 1 }

  Enum for supported Waveform types.
- enum WF_ModeE {
  WF_MODE_NORMAL =1, **WF_MODE_AJ** =2, **WF_MODE_LPD** =4, **WF_MODE_LOWPOWER** =8,
  WF_MODE_RADIO_SILENCE =16 }

  Enum for the waveform modes supported by framework.
- enum WF_EstimatorTypeE { **WF_FULL_EST**, WF_NO_EST }

  Enum for type of waveform estimator.
- enum **WF_ControlP_StatusE** {
  WF_NORMAL, **WF_BUSY**, **WF_ERROR**, **WF_BUFFER_LOW**,
  WF_BUFFER_FULL }
- enum **WF_ScheduleTypeE** { **WF_RecvSchd**, **WF_SendSchd**, **WF_LinkEstSchd** }
- enum **WF_ControlResponseE** {
  AttributeResponse, **AddDestinationResponse**, **CancelPacketResponse**, **ReplacePayloadResponse**,
  ControlStatusResponse, **DataStatusResponse**, **SetScheduleResponse** }

Functions

- struct Waveform::PacketMetadata **__attribute__** ((packed, aligned(1)))

Variables

- struct Waveform::FragmentationHeader **__attribute__**

5.11.1 Detailed Description

Waveform Interface namespace.

5.11.2 Enumeration Type Documentation

5.11.2.1 enum Waveform::WF_EstimatorTypeE

Enum for type of waveform estimator.

Enumerator

WF_NO_EST Waveform provides the complete estimation. Waveform does not provide estimates

5.11.2.2 enum Waveform::WF_MessageStatusE

Enumerator

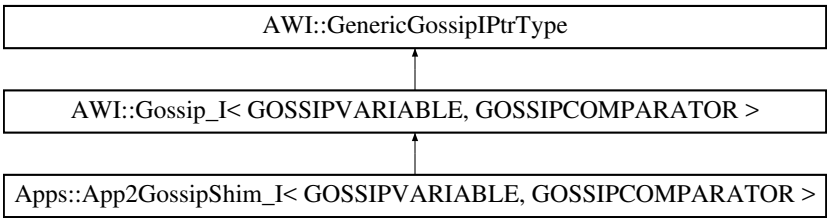
WF_ST_MAX_RETRIES_ERR WF_MessageTimeout,.

Chapter 6

Class Documentation

6.1 Apps::App2GossipShim_I< GOSSIPVARIABLE, GOSSIPCOMPARATOR > Class Template Reference

Inheritance diagram for Apps::App2GossipShim_I< GOSSIPVARIABLE, GOSSIPCOMPARATOR >:



Public Member Functions

- virtual void RegisterGossipVariableUpdateDelegate (GossipVariableUpdateDelegate_t *_gvu_del)=0

  Used by the application to register its GossipVariableUpdateDelegate_t, which in turn is used by the pattern to notify the application about the changes in the GOSSIPVARIABLE.

Additional Inherited Members

6.1.1 Member Function Documentation

6.1.1.1 template<typename GOSSIPVARIABLE , typename GOSSIPCOMPARATOR = SimpleGenericComparator<GOSSIPVARIABLE>> virtual void Apps::App2GossipShim_I< GOSSIPVARIABLE, GOSSIPCOMPARATOR >::RegisterGossipVariableUpdateDelegate (GossipVariableUpdateDelegate_t * *_gvu_del*) [pure virtual]

Used by the application to register its GossipVariableUpdateDelegate_t, which in turn is used by the pattern to notify the application about the changes in the GOSSIPVARIABLE.

The Gossip pattern stores delegates registered to itself and invokes these delegates whenever there is an update on the GOSSIPVARIABLE. The update can be as a result of a reception of a GossipMessage from another node as well as the UpdateGossipVariable request from another application connected to the Pattern locally.

Parameters

GossipVariableUpdate↩ Delegate_t	The delegate that specifies the function/method used for receiving updates from pattern.

Returns

void.

Implements AWI::Gossip_I< GOSSIPVARIABLE, GOSSIPCOMPARATOR >.

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

- TuscaroraFW/Include/Interfaces/App/App2GossipShim_I.h

6.2 AWI::FWP2AppShim_I Class Reference

Public Member Functions

- virtual void **ReceiveMessage** (void *data, uint16_t size)=0
- virtual void **RegisterAppReceiveDelegate** (AppId_t _app_id, AppRecvMessageDelegate_t *_gvu_del)

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

- TuscaroraFW/Include/Interfaces/AWI/Fwp_I.h

6.3 AWI::Fwp_I Class Reference

Public Member Functions

- virtual void **Send** (void *data, uint16_t size)=0
- virtual void **RegisterAppReceiveDelegate** (AppId_t _app_id, AppRecvMessageDelegate_t *_gvu_del)=0
- virtual ~Fwp_I ()

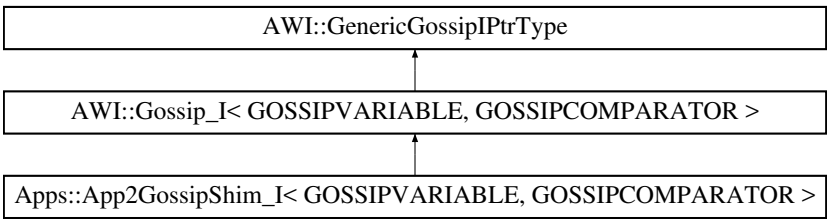
  Virtual destructor.

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

- TuscaroraFW/Include/Interfaces/AWI/Fwp_I.h

6.4 AWI::GenericGossipIPtrType Class Reference

Inheritance diagram for AWI::GenericGossipIPtrType:



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

- TuscaroraFW/Include/Interfaces/AWI/Gossip_I.h

6.5 AWI::Gossip2AppShim_I< GOSSIPVARIABLE > Class Template Reference

Public Member Functions

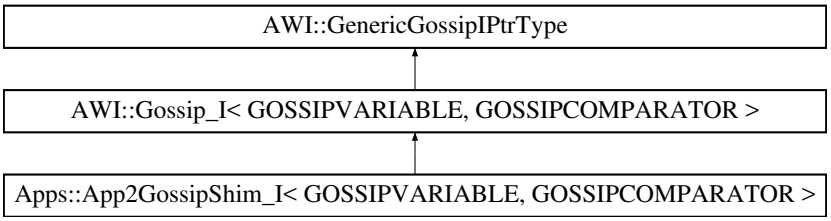
- virtual void **ReceiveUpdatedGossipVariable** (GOSSIPVARIABLE &msg)=0
- virtual void **RegisterGossipVariableUpdateDelegate** (GossipVariableUpdateDelegate_t_I *_gvu_del)

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

- TuscaroraFW/Include/Interfaces/AWI/Gossip_I.h

6.6 AWI::Gossip_I< GOSSIPVARIABLE, GOSSIPCOMPARATOR > Class Template Reference

Inheritance diagram for AWI::Gossip_I< GOSSIPVARIABLE, GOSSIPCOMPARATOR >:



Public Types

- typedef Delegate< void, GOSSIPVARIABLE & > **GossipVariableUpdateDelegate_t**

Public Member Functions

- virtual void RegisterGossipVariableUpdateDelegate (GossipVariableUpdateDelegate_t *_gvu_del)=0

  Used by the application to register its GossipVariableUpdateDelegate_t, which in turn is used by the pattern to notify the application about the changes in the GOSSIPVARIABLE.
- virtual void UpdateGossipVariable (GOSSIPVARIABLE &newgossipVariable)=0

  Used by the applications to update the underlying GOSSIPVARIABLE of the pattern.
- virtual ~Gossip_I ()

  Virtual destructor.

6.6.1 Member Function Documentation

6.6.1.1 template<typename GOSSIPVARIABLE, typename GOSSIPCOMPARATOR = SimpleGenericComparator<GOSSIPVARIABLE>> virtual void AWI::Gossip_I< GOSSIPVARIABLE, GOSSIPCOMPARATOR >::RegisterGossipVariableUpdateDelegate (GossipVariableUpdateDelegate_t * *_gvu_del*) [pure virtual]

Used by the application to register its GossipVariableUpdateDelegate_t, which in turn is used by the pattern to notify the application about the changes in the GOSSIPVARIABLE.

The Gossip pattern stores delegates registered to itself and invokes these delegates whenever there is an update on the GOSSIPVARIABLE. The update can be as a result of a reception of a GossipMessage from another node as well as the UpdateGossipVariable request from another application connected to the Pattern locally.

Parameters

GossipVariableUpdate_t	The delegate that specifies the function/method used for receiving updates from pattern.

Returns

void.

Implemented in Apps::App2GossipShim_I< GOSSIPVARIABLE, GOSSIPCOMPARATOR >.

6.6.1.2 template<typename GOSSIPVARIABLE, typename GOSSIPCOMPARATOR = SimpleGenericComparator<GOSSIPVARIABLE>> virtual void AWI::Gossip_I< GOSSIPVARIABLE, GOSSIPCOMPARATOR >::UpdateGossipVariable (GOSSIPVARIABLE & *newgossipVariable*) [pure virtual]

Used by the applications to update the underlying GOSSIPVARIABLE of the pattern.

This method lets a application to request changing the variable being gossiped with a newVariable. The newgossipVariable is compared with the existing one. If the existing variable is larger(based on the comparator), no further action is taken. Else if, the newvariable is larger(based on the comparator), the gossip variable is updated, and all registered GossipVariableUpdateDelegate_t are invoked with the new GOSSIPVARIABLE.

Parameters

newgossipVariable	The new gossip

Returns

void.

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

- TuscaroraFW/Include/Interfaces/AWI/Gossip_I.h

6.7 Core::ConfigureFramework_I Class Reference

Public Member Functions

- void **RegisterLinkEstimator** (LinkEstimatorI< uint64_t > &est, EstimatorTypeE type, WaveformId_t wid)

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

- TuscaroraFW/Include/Interfaces/Core/ConfigureFrameworkI.h

6.8 Core::Dataflow::DataStatusParam Struct Reference

A structure that is passed as the parameter to the DataNotifier Delegate.

`#include <DataFlowI.h>`

Public Member Functions

- **DataStatusParam** (uint32_t _messageId)

Public Attributes

- DataStatusTypeE **statusType** [MAX_DEST]
- bool **statusValue** [MAX_DEST]
- NodeId_t **destArray** [MAX_DEST]
- uint8_t **noOfDest**
- FMessageId_t **messageId**
- bool **readyToReceive**
- Nonce_t **nonce**

6.8.1 Detailed Description

A structure that is passed as the parameter to the DataNotifier Delegate.

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

- TuscaroraFW/Include/Interfaces/Core/DataFlowI.h

6.9 Core::Dataflow::FragmentAndReassembleI Class Reference

Specifies a definition for the fragementation and reassembly service.

`#include <DataFlowI.h>`

Public Member Functions

- virtual bool Fragment (FMessage_t *org, uint16_t maxSize, FMessage_t *resultFragments, uint16_t *numberOfFragments)=0

  Fragments a framework packet into a number of packets and returns them in an array pointed to by the 3rd parameter.
- virtual bool Reassemble (FMessage_t *fragments, uint16_t *numberOfFragments, FMessage_t *result)=0

  Reassembles an array of individual fragment packet.
- virtual ~FragmentAndReassembleI ()

  Destructor for abstract virtual class.

6.9.1 Detailed Description

Specifies a definition for the fragementation and reassembly service.

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

- TuscaroraFW/Include/Interfaces/Core/DataFlowI.h

6.10 Core::Discovery::DiscoveryLogHeader Class Reference

Public Attributes

- uint32_t **nodeid**
- uint32_t **beaconPeriod**
- uint8_t **inactivityPeriod**
- uint16_t **range**
- uint8_t **linkEstimationType**
- uint8_t **discoveryType**
- uint16_t **numNodes**
- uint16_t **runTime**
- uint8_t **density**
- uint8_t **mobilityModel**
- uint16_t **speed**

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

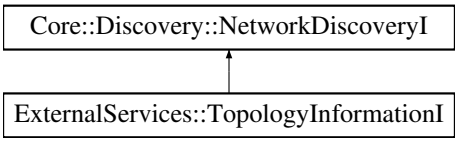
- TuscaroraFW/Include/Interfaces/Core/NetworkDiscoveryI.h

6.11 Core::Discovery::NetworkDiscoveryI Class Reference

Inteface definition for a network discovery service.

`#include <NetworkDiscoveryI.h>`

Inheritance diagram for Core::Discovery::NetworkDiscoveryI:



Public Member Functions

- virtual bool RegisterDelegate (PotentialNeighborDelegate &del)=0

  Registers the change delegates with the discovery service. The delegate callback occurs when a network discovery event happens.
- virtual uint16_t NetworkSize (uint16_t waveformId)=0

  Returns the number of neighbors associated with a particular waveform.
- virtual uint16_t NumberOfPotentialNeighbors (NodeId_t node)=0

  Returns the number of potential (local) neighbors for a given node.
- virtual uint16_t PotentialNeighborsList (NodeId_t node, NodeInfo *nodeInfoArray, uint16_t arraySize)=0
- virtual void **Start** ()=0
- virtual void **ProcessMessage** (FMessage_t &msg)=0

6.11.1 Detailed Description

Inteface definition for a network discovery service.

6.11.2 Member Function Documentation

6.11.2.1 virtual uint16_t Core::Discovery::NetworkDiscoveryI::PotentialNeighborsList (NodeId_t *node*, NodeInfo * *nodeInfoArray*, uint16_t *arraySize*) [pure virtual]

Copies potential neighbors information for a given node into the array pointed to by parameter 'nodeInfoArray' and returns the number of nodes which where copied. The 'nodeInfoArray' parameter should be atleast of size 'arraySize' (parameter 3)

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

- TuscaroraFW/Include/Interfaces/Core/NetworkDiscoveryI.h

6.12 Core::Discovery::PotentialNeighbor Struct Reference

Public Attributes

- LinkId **linkId**
- PotentialNeighborTypeE **nodeType**
- uint8_t **normalizedRssi**
- UFixed_7_8_t **potential**
- void * otherInfo

  Measure of potential to become a neighbor.

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

- TuscaroraFW/Include/Interfaces/Core/NetworkDiscoveryI.h

6.13 Core::Discovery::PotentialNeighborDelegateParam Struct Reference

Public Attributes

- PNbrUpdateTypeE **changeType**
- PotentialNeighbor * **pnbr**

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

- TuscaroraFW/Include/Interfaces/Core/NetworkDiscoveryI.h

6.14 Core::Discovery::PotentialNeighborRegistryI Class Reference

Public Member Functions

- virtual uint16_t GetNumberOfNodes ()=0

  Returns the number of nodes currently reachable that satisfies the Patterns neighbor criteria.
- virtual uint16_t GetPotentialNeighbors (UFixed_7_8_t threshPotential, PotentialNeighbor *array, uint16_t sizeOfArray)=0

  ,
- virtual PotentialNeighbor * AddNode (LinkId id, PotentialNeighborTypeE type, uint8_t normalizedRssi, UFixed_7_8_t potential)=0

  Adds a new node to the registry . Discovery modules should use this to update the registry.
- virtual PotentialNeighbor * **GetNode** (LinkId id)=0
- virtual bool UpdateNode (LinkId id, PotentialNeighbor _info)=0

  Updates info a existing node.
- virtual bool RemoveNode (LinkId id)=0

  Updates info a existing node.

6.14.1 Member Function Documentation

6.14.1.1 virtual uint16_t Core::Discovery::PotentialNeighborRegistryI::GetPotentialNeighbors (UFixed_7_8_t *threshPotential,* PotentialNeighbor * *array,* uint16_t *sizeOfArray*) [pure virtual]

,

Returns an array of pottential neighbors, which are above the potential specified by the _potential specified

Parameters

threshPotential	Threshold to be satisfied
array	Array into which the potential neighbor info is copied
sizeOfArray	Size of the array provided. If the number of potential neighbors satisfying the threshold is higher the size of the array, then the top sizOfArray number of nodes info is copied to the array

Returns

uint16_t

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

- TuscaroraFW/Include/Interfaces/Core/PotentialNeighborRegistryI.h

6.15 Core::Estimation::EstimationLogI Class Reference

Public Member Functions

- virtual void **LogEvent** (EstimationLogE event, NodeId_t nbr)=0

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

- TuscaroraFW/Include/Interfaces/Core/EstimationLogI.h

6.16 Core::Estimation::EstimatorCallback_I< AddressType > Class Template Reference

Public Member Functions

- virtual bool **SendEstimationMessage** (WaveformId_t wfId, FMessage_t *msg)=0
- virtual WF_Attributes **GetWaveformAttributes** (WaveformId_t wfId)=0
- virtual void **LinkEstimatorChangeHandler** (WF_LinkEstimationParam< AddressType > _param)=0

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

- TuscaroraFW/Include/Interfaces/Core/EstimationCallbackI.h

6.17 Core::Estimation::LinkEstimatorI< AddressType > Class Template Reference

Interface definition for a link estimator which is implemented by the framework.

`#include <LinkEstimatorI.h>`

Public Member Functions

- **LinkEstimatorI** (EstimatorCallback_I< AddressType > &callback)
- virtual bool **RegisterDelegate** (Core::WF_LinkChangeDelegate_n64_t &del)=0
- virtual void **SetParam** (uint32_t beaconPeriod, uint32_t inactivePeriod)=0
- virtual LinkMetrics * OnPacketReceive (Waveform::WF_MessageBase *rcvPkt)=0

  This function will be called each time a new Waveform packet arrives with that packet. This should return the new link metric of the src of the packet.
- virtual void **PotentialNeighborUpdate** (Discovery::PotentialNeighbor &pnbr, Discovery::PNbrUpdateTypeE type)=0

6.17.1 Detailed Description

template<class AddressType>
class Core::Estimation::LinkEstimatorI< AddressType >

Interface definition for a link estimator which is implemented by the framework.

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

- TuscaroraFW/Include/Interfaces/Core/LinkEstimatorI.h

6.18 Core::Estimation::NeighborhoodControlI Class Reference

Defines the interface that must be implemented by the Pattern to customize its neighborhood.

`#include <LinkEstimatorI.h>`

Public Member Functions

- virtual void RegisterBlackListCallback (Delegate< void, NodeList > *blacklistDelegate)=0

  This API registers a callback, that will be called by the policy manager module to provide a blacklist of nodes.

6.18.1 Detailed Description

Defines the interface that must be implemented by the Pattern to customize its neighborhood.

Defines an interface that can be used to take information from a policy module about nodes

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

- TuscaroraFW/Include/Interfaces/Core/LinkEstimatorI.h

6.19 Core::Estimation::SendEstMsgParam Struct Reference

Public Attributes

- WaveformId_t **wfId**
- FMessage_t * **msg**

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

- TuscaroraFW/Include/Interfaces/Core/LinkEstimatorI.h

6.20 Core::Link Struct Reference

Structure to store information about a link in the framework and pattern layers including its metrics, 33b.

`#include <Link.h>`

Public Attributes

- LinkId **linkId**
- LinkTypeE **type**
- LinkMetrics **metrics**

6.20.1 Detailed Description

Structure to store information about a link in the framework and pattern layers including its metrics, 33b.

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

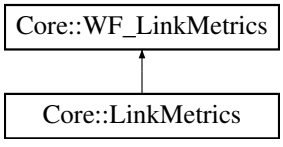
- TuscaroraFW/Include/Interfaces/Core/Link.h

6.21 Core::LinkMetrics Struct Reference

Structure used by the framwork to expose link metrics to the pattern, 28 bytes with the use of float for metrics.

`#include <Link.h>`

Inheritance diagram for Core::LinkMetrics:



Public Member Functions

- LinkMetrics ()

  Tuscarora's estimate of the latency for the next transmit.
- **LinkMetrics** (WF_LinkMetrics _param)

Additional Inherited Members

6.21.1 Detailed Description

Structure used by the framwork to expose link metrics to the pattern, 28 bytes with the use of float for metrics.

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

- TuscaroraFW/Include/Interfaces/Core/Link.h

6.22 Core::MessageT< NodeIDType > Class Template Reference

A class that facilitates data exchange between Patterns and the framework.

`#include <MessageT.h>`

Public Member Functions

- **MessageT** (uint16_t payload_size)
- NodeIDType **GetSource** ()
- WaveformId_t **GetWaveform** ()
- void **SetSource** (NodeIDType _src)
- void **SetWaveform** (WaveformId_t _wfid)
- MessageTypeE **GetType** ()
- void **SetType** (MessageTypeE _type)
- uint16_t **GetPayloadSize** ()
- void **SetPayloadSize** (uint16_t _size)
- uint8_t * **GetPayload** ()
- void **SetPayload** (uint8_t *_payload)

6.22.1 Detailed Description

template<class NodeIDType>
class Core::MessageT< NodeIDType >

A class that facilitates data exchange between Patterns and the framework.

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

- TuscaroraFW/Include/Interfaces/Core/MessageT.h

6.23 Core::Naming::PatternNameServiceI Class Reference

Generic interface specification for a name service.

`#include <PatternNamingI.h>`

Public Member Functions

- PatternId_t GetNewPatternInstanceId (PatternTypeE ptype, const char uniqueName[128])

  Returns a unique Id for use as the pattern Id, takes the pattern type as parameter.
- PatternTypeE **GetPatternType** (PatternId_t id)

6.23.1 Detailed Description

Generic interface specification for a name service.

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

- TuscaroraFW/Include/Interfaces/Core/PatternNamingI.h

6.24 Core::NeighborTableI Class Reference

Public Member Functions

- virtual uint16_t GetNumberOfNeighbors ()=0

  Returns the number of nodes currently reachable that satisfies the Patterns neighbor criteria.
- virtual uint16_t GetBestQualityNeighbors (uint16_t numNeighbors, NodeId_t *nodeIds)=0

  Returns a number (specified by the first parameter) of best neighbor sorted by quality, in the array specified by the second parameter.
- virtual uint16_t GetLeastLatencyNeighbors (uint16_t numNeighbors, NodeId_t *nodeIds)=0

  Returns a number(specified by the first parameter) of neighbors of least expected transmit latency, in the array specified by the second parameter.
- virtual NodeInfo * **GetNeighborInfo** (NodeId_t nodeId)
- virtual NodeId_t GetNeighborID (uint16_t table_index)

  Given an index in the table returns the NodeId of the corresponding node.
- virtual int32_t AddNeighbor (NodeId_t id, NodeTypeE type)
- virtual int32_t UpdateNeighbor (NodeId_t id, NodeInfo _info)

  Updates info about nbr, if node does not exist, it is created.
- virtual void **RemoveNeighbor** (NodeId_t id)
- virtual uint16_t GetPendingPackets (NodeId_t neighbor)

  Returns the number of packets pending for the neighbor.

6.24.1 Member Function Documentation

6.24.1.1 virtual int32_t Core::NeighborTableI::AddNeighbor (NodeId_t *id*, NodeTypeE *type*) [virtual]

Adds a new neighbor to the neighbor table, if the node already exist, the information is updated Used to change the type of neighbor type or add potential neighbor for who link metrics are not available.

6.24.1.2 virtual uint16_t Core::NeighborTableI::GetLeastLatencyNeighbors (uint16_t *numNeighbors,* NodeId_t * *nodeIds*) [pure virtual]

Returns a number(specified by the first parameter) of neighbors of least expected transmit latency, in the array specified by the second parameter.

Returns the NeighborInfo object for the specified node id.

6.24.1.3 virtual int32_t Core::NeighborTableI::UpdateNeighbor (NodeId_t *id,* **NodeInfo** *_info*) [virtual]

Updates info about nbr, if node does not exist, it is created.

Removes a neighbor from the neighbor table. Used by the Network Discovery module.

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

- TuscaroraFW/Include/Interfaces/Core/NeighborTableI.h

6.25 Core::NeighborUpdateParam Struct Reference

Structure to be used as the parameter for the function handling the neighbor change events.

`#include <Link.h>`

Public Member Functions

- **NeighborUpdateParam** (WF_LinkEstimationParam_n64_t _param, NodeId_t nodeid)

Public Attributes

- Link **link**
- NodeId_t **nodeId**
- NeighborChangeTypeE **changeType**

6.25.1 Detailed Description

Structure to be used as the parameter for the function handling the neighbor change events.

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

- TuscaroraFW/Include/Interfaces/Core/Link.h

6.26 Core::NodeInfo Struct Reference

Public Attributes

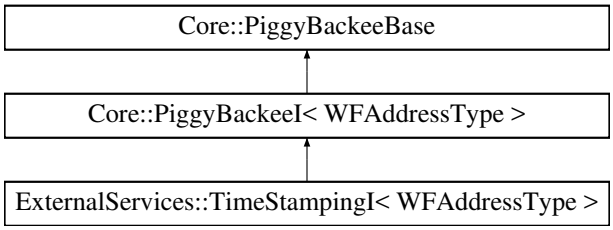
- NodeId_t **nodeId**
- uint8_t **flags**
- NeighborTypeE **type**
- Link * **linkArray**
- uint8_t **numberOfLinks**

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

- TuscaroraFW/Include/Interfaces/Core/Node.h

6.27 Core::PiggyBackeeBase Class Reference

Inheritance diagram for Core::PiggyBackeeBase:



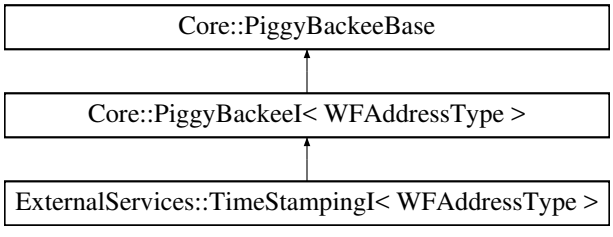
The documentation for this class was generated from the following file:

- TuscaroraFW/Include/Interfaces/Core/PiggyBackingI.h

6.28 Core::PiggyBackeeI< WFAddressType > Class Template Reference

`#include <PiggyBackingI.h>`

Inheritance diagram for Core::PiggyBackeeI< WFAddressType >:



Public Member Functions

- **PiggyBackeel** (PiggyBackerI< WFAddressType > &sendDelegate)
- virtual uint16_t CreateData (WFAddressType dest, bool unaddressed, uint8_t *&data, uint8_t dataSize)=0

  Creates the piggybacked data to be sent out. This method gets called by the framework on the sending side, whenever their is enough space for piggybacking in an outgoing packet.
- virtual void ReceiveData (WFAddressType src, uint8_t *data, uint8_t dataSize, U64NanoTime receiveTimestamp)=0

  Receives the piggybacked data and handles it, whenever such data is received by framework. This method gets called by the framework on the receiving side.

6.28.1 Detailed Description

template<class WFAddressType>
class Core::PiggyBackeel< WFAddressType >

Interface that must be implemented by modules that want to send piggybacked data on outgoing packets. Please note that the module needs to be registered successfully with the framework before piggybacking can happen. The interface is meant to be used only by the TimeStamping and TimeSync services. No other service is supported at this point. Any other module trying to use this service will result in correct system.

6.28.2 Member Function Documentation

6.28.2.1 template<class WFAddressType> virtual uint16_t Core::PiggyBackeel< WFAddressType >::CreateData (WFAddressType *dest*, bool *unaddressed*, uint8_t *& *data*, uint8_t *dataSize*) [pure virtual]

Creates the piggybacked data to be sent out. This method gets called by the framework on the sending side, whenever their is enough space for piggybacking in an outgoing packet.

Parameters

dest	The destination address the packet is meant for
unaddressed	Boolen. If set to true, the packet is not addressed to anyone in particular and will be sent out on the channel for anyone who can receive it.
data	Pointer to the piggybacked data. This method should create the data array and set the address to this pointer
dataSize	Max size of the piggybacked data that can be accomdated

Returns

uint16_t Actucal size of the piggybacked data created.

Implemented in ExternalServices::TimeStampingI< WFAddressType >.

6.28.2.2 template<class WFAddressType> virtual void Core::PiggyBackeel< WFAddressType >::ReceiveData (WFAddressType *src*, uint8_t * *data*, uint8_t *dataSize*, U64NanoTime *receiveTimestamp*) [pure virtual]

Receives the piggybacked data and handles it, whenever such data is received by framework. This method gets called by the framework on the receiving side.

Parameters

src	Node Id of the sending node.
data	Pointer to the piggybacked data
dataSize	Size of the piggybacked data. If IMPLICIT_SYNC_TIMESTAMPING is used, the datasize will be set to 0 and only the receiveTimestamp field will have valid value.
receiveTimestamp	Time at which the packet was received at the waveform layer

Returns

void

Implemented in ExternalServices::TimeStampingI< WFAddressType >.

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

- TuscaroraFW/Include/Interfaces/Core/PiggyBackingI.h

6.29 Core::PiggyBackerI< WFAddressType > Class Template Reference

This class will be implemented by the framework to provide piggybacking service.

`#include <PiggyBackingI.h>`

Public Member Functions

- **PiggyBackerI** (Dataflow::U64AddressSendDataDelegate_t &sendDelegate)
- virtual bool **RegisterPiggyBackee** (PB_TypeE type, Waveform::WF_AddressTypeE addressType, uint8_t dataSize, const std::string &authString, PiggyBackeeI< WFAddressType > &receivingModule, WaveformId_t wId)=0
- virtual uint8_t **GetNumberOfRegisteredPiggyBackees** (WaveformId_t wid)=0
- virtual bool SendDataNow (WaveformId_t wid, const std::string &authString, MessageTypeE msgType, uint8_t dataSize, uint8_t *data, WFAddressType dest, bool unaddressed)=0

  Allows a Piggebackee to send a message on demand. This is a best effort service. The framework can choose to ignore this message.

6.29.1 Detailed Description

template<class WFAddressType>
class Core::PiggyBackerI< WFAddressType >

This class will be implemented by the framework to provide piggybacking service.

6.29.2 Member Function Documentation

6.29.2.1 template<class WFAddressType> virtual bool Core::PiggyBackerI< WFAddressType >::SendDataNow (WaveformId_t *wid*, const std::string & *authString*, MessageTypeE *msgType*, uint8_t *dataSize*, uint8_t * *data*, WFAddressType *dest*, bool *unaddressed*) [pure virtual]

Allows a Piggebackee to send a message on demand. This is a best effort service. The framework can choose to ignore this message.

Parameters

wid	Waveform Id on which the message should be sent out.
authString	Authentication string of the piggebackee to verify its allowed to send
dataSize	size of the data
data	The pointer to data

Returns

Return type is bool. Returns true if data was sent to waveform successfully. Note this does not mean

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

- TuscaroraFW/Include/Interfaces/Core/PiggyBackingI.h

6.30 Core::Policy::PolicyManagerI Class Reference

Defines the interface for the policy manager.

`#include <PolicyManagerI.h>`

Public Member Functions

- bool SetPacketPolicy (PacketSchedulingPolicyE _policy)

  Set the policy for the framework packet scheduler. The default is FCFS.
- bool SetAsnpPriority (uint16_t patternInstanceId, uint8_t priority)

  Can set indivitual pattern priorities, if the policy is PriorityFCFS. Has no effect if PacketPolicy is FCFS.
- uint16_t GetAsnpPriority (uint16_t asnpInstanceId)

  Get the priority of a pattern instance.
- bool AddNode (NodeId_t id, NodeInfo info)

  Add a node not discovered by framework, like special nodes.
- bool RemoveNode (NodeId_t id)

  Remove a special nodes from network.
- bool AddRadioSilenceNode (NodeId_t id, NodeInfo info, double lat, double long, uint64_t radius)
- bool SetBlackList (NodeId_t *nodeArray, uint16_t noOfNodes)

  Blacks lists the nodes passed as parameters. The black listed nodes will be removed from neighborhood even if they are detected.
- bool ClearBlackList ()

  Clears the current black-listed nodes.
- bool IsPiggyBackingEnabled ()

  Returns the status of piggy backing in the framework.
- uint16_t GetMaxFWPacketPayloadSize ()

  Returns the maximum payload size that the framework accepts.
- bool **IsValidPiggyBackee** (PB_TypeE type, Waveform::WF_AddressTypeE addressType, uint8_t dataSize, const std::string &authString)

6.30.1 Detailed Description

Defines the interface for the policy manager.

6.30.2 Member Function Documentation

6.30.2.1 bool Core::Policy::PolicyManagerI::AddRadioSilenceNode (NodeId_t *id*, NodeInfo *info*, double *lat*, double *long*, uint64_t *radius*)

Add radio silence noes to network, the 3 and 4 parameter specify the lattitude and longitude of a GPS location and 5 paramerter specify the radius around the GPS location, where the node should be added to the network. Any node which is inside this circle will see this radio silence node in its neighbor table

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

- TuscaroraFW/Include/Interfaces/Core/PolicyManagerI.h

6.31 Core::WF_Link< WF_AddressType > Struct Template Reference

Structure to store information about a link received from the waveform layer.

`#include <Link.h>`

Public Attributes

- WF_LinkId< WF_AddressType > **linkId**
- LinkTypeE **type**
- WF_LinkMetrics **metrics**

6.31.1 Detailed Description

template<class WF_AddressType>
struct Core::WF_Link< WF_AddressType >

Structure to store information about a link received from the waveform layer.

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

- TuscaroraFW/Include/Interfaces/Core/Link.h

6.32 Core::WF_LinkEstimates< WF_AddressType > Struct Template Reference

Public Attributes

- WF_Link< WF_AddressType > * **linkArray**
- uint16_t **size**

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

- TuscaroraFW/Include/Interfaces/Core/Link.h

6.33 Core::WF_LinkEstimationParam< WF_AddressType > Struct Template Reference

Structure to be used as the parameter for the function handling the neighbor change events.

`#include <Link.h>`

Public Attributes

- LinkTypeE **type**
- NodeId_t **nodeId**
- WF_LinkMetrics **metrics**
- WF_AddressType **linkAddress**
- NeighborChangeTypeE **changeType**
- WaveformId_t **wfid**

6.33.1 Detailed Description

template<class WF_AddressType>
struct Core::WF_LinkEstimationParam< WF_AddressType >

Structure to be used as the parameter for the function handling the neighbor change events.

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

- TuscaroraFW/Include/Interfaces/Core/Link.h

6.34 Core::WF_LinkId< WF_AddressType > Class Template Reference

Public Member Functions

- **WF_LinkId** (WF_AddressType _id)
- **WF_LinkId** (WF_AddressType _id, WaveformId_t _wid)
- bool **operator==** (const WF_LinkId< WF_AddressType > rhs) const
- bool **operator<** (const WF_LinkId< WF_AddressType > rhs) const

Public Attributes

- WF_AddressType **nodeId**
- WaveformId_t **waveformId**

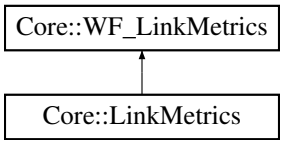
The documentation for this class was generated from the following file:

- TuscaroraFW/Include/Interfaces/Core/Link.h

6.35 Core::WF_LinkMetrics Struct Reference

`#include <Link.h>`

Inheritance diagram for Core::WF_LinkMetrics:



Public Attributes

- UFixed_7_8_t quality

  A quality metric expressed as a real number between [0,1].
- UFixed_2_8_t dataRate

  Datarate expressed as log_2(bps)
- Fixed_7_16_t avgLatency

  Average latency expressed as log_2(milliseconds)
- UFixed_2_8_t energy

  Average energy to transmit a packet expressed as log_2(pica-joules)
- Fixed_14_16_t cost

  Average cost of sending a packet.

6.35.1 Detailed Description

Quality. The probability of delivery of transmissions with successful receipt of acknowledgement on the link, expressed in [0, 1]. Quality has type UFixed_8_7_t, where the last 7 of the 8 bit unsigned fixed point type are the radix point bits.

DataRate. Let si be defined as the estimation of the average sustained rate in bits per second. Then DataRate will be defined as log_2(si) DataRate has type UFixed_8_2_t, where the 16 bit fixed point type consists, 6 bits before the radix point and 2 radix point bits. Its maximum value is 63.75 and minimum value is 0. This corresponds to data rates from 1 bps to 15.5 Pb, with a representational accuracy of plus or minus 25%.

Latency. Let be defined as the latency in seconds. Then the latency is defined to be log_2().Latency has type UFixed_8_2_t, which contains a sign bit, 5 bits before the decimal point, and 2 after. This can represent a number from 230 ps to 136 years to accuracies of plus or minus 25%.

Energy. Let be the average network-wide number of pico-joules (e.g., combined energy of the transmitter and the receiver) per byte. Then the energy is defined as log_2().Energy has type UFixed_8_2_t, which defines an unsigned number with 6 bits before the decimal and 2 after. This would correspond to levels from 1 pJ/Byte to 15 MJ/Byte.

cost. The cost sending one packet using the waveform.

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

- TuscaroraFW/Include/Interfaces/Core/Link.h

6.36 EventTimeStamp Struct Reference

Structure for implementing event timestamping between two neighbors.

`#include <EventTimeStamp.h>`

Public Attributes

- U64NanoTime **senderEventTime**
- uint64_t senderDelay

  Receive time stamp of the packet in local clock time.

6.36.1 Detailed Description

Structure for implementing event timestamping between two neighbors.

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

- TuscaroraFW/Include/Interfaces/Waveform/EventTimeStamp.h

6.37 ExternalServices::GlobalNameI Class Reference

Public Member Functions

- uint128_t **GetMyNodeName** ()
- uint128_t **GetWaveformName** (WaveformId_t id)

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

- TuscaroraFW/Include/Interfaces/ExternalServices/GlobalNodeNameI.h

6.38 ExternalServices::GPSLocation Struct Reference

Public Attributes

- double **latitude**
- double **longitude**

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

- TuscaroraFW/Include/Interfaces/ExternalServices/LocationServiceI.h

6.39 ExternalServices::Location2D Struct Reference

Public Attributes

- double **x**
- double **y**

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

- TuscaroraFW/Include/Interfaces/ExternalServices/LocationServiceI.h

6.40 ExternalServices::Location3D Struct Reference

Public Attributes

- double **x**
- double **y**
- double **z**

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

- TuscaroraFW/Include/Interfaces/ExternalServices/LocationServiceI.h

6.41 ExternalServices::LocationServiceI< LocationType > Class Template Reference

Public Member Functions

- LocationType **GetLocation** ()

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

- TuscaroraFW/Include/Interfaces/ExternalServices/LocationServiceI.h

6.42 ExternalServices::Mobility::MobilityControlI Class Reference

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

- TuscaroraFW/Include/Interfaces/ExternalServices/MobilityControlI.h

6.43 ExternalServices::NeighborTimeI< TimeA, TimeB, NodeIDType > Class Template Reference

Public Member Functions

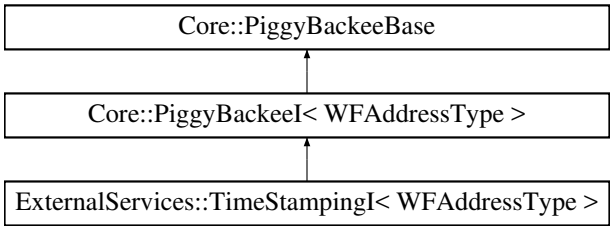
- virtual TimeB **GetNeighborTime** (NodeIDType nbrId, TimeA localtime)=0
- virtual TimeA **GetLocalTime** (NodeIDType nbrId, TimeB nbrTime=0)

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

- TuscaroraFW/Include/Interfaces/ExternalServices/NeighborTime.h

6.44 ExternalServices::TimeStampingI< WFAddressType > Class Template Reference

Inheritance diagram for ExternalServices::TimeStampingI< WFAddressType >:



Public Member Functions

- **TimeStampingI** (PiggyBackerI< WFAddressType > &piggyBacker)
- virtual uint16_t CreateData (WFAddressType dest, bool unaddressed, uint8_t *&data, uint8_t dataSize)=0

  Creates the piggybacked data to be sent out. This method gets called by the framework on the sending side, whenever their is enough space for piggybacking in an outgoing packet.
- virtual void ReceiveData (WFAddressType src, uint8_t *data, uint8_t dataSize, U64NanoTime _receiveTimestamp)=0

  Receives the piggybacked data and handles it, whenever such data is received by framework. This method gets called by the framework on the receiving side.

6.44.1 Member Function Documentation

6.44.1.1 template<class WFAddressType > virtual uint16_t ExternalServices::TimeStampingI< WFAddressType >::CreateData (WFAddressType *dest*, bool *unaddressed*, uint8_t *&* *data*, uint8_t *dataSize*) [pure virtual]

Creates the piggybacked data to be sent out. This method gets called by the framework on the sending side, whenever their is enough space for piggybacking in an outgoing packet.

Parameters

dest	The destination address the packet is meant for
unaddressed	Boolen. If set to true, the packet is not addressed to anyone in particular and will be sent out on the channel for anyone who can receive it.
data	Pointer to the piggybacked data. This method should create the data array and set the address to this pointer
dataSize	Max size of the piggybacked data that can be accomdated

Returns

uint16_t Actucal size of the piggybacked data created.

Implements Core::PiggyBackeeI< WFAddressType >.

6.44.1.2 template<class WFAddressType > virtual void ExternalServices::TimeStampingI< WFAddressType >::ReceiveData (WFAddressType *src*, uint8_t * *data*, uint8_t *dataSize*, U64NanoTime *receiveTimestamp*) [pure virtual]

Receives the piggybacked data and handles it, whenever such data is received by framework. This method gets called by the framework on the receiving side.

Parameters

src	Node Id of the sending node.
data	Pointer to the piggybacked data
dataSize	Size of the piggybacked data. If IMPLICIT_SYNC_TIMESTAMPING is used, the datasize will be set to 0 and only the receiveTimestamp field will have valid value.
receiveTimestamp	Time at which the packet was received at the waveform layer

Returns

void

Implements Core::PiggyBackeeI< WFAddressType >.

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

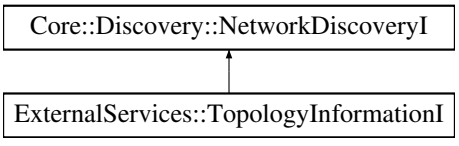
- TuscaroraFW/Include/Interfaces/ExternalServices/TimeStampingI.h

6.45 ExternalServices::TopologyInformationI Class Reference

Defines an interface for the topology information service. In our current thinking, this is just another Network Discovery service, hence has the same APIs.

```
#include <TopologyInformationI.h>
```

Inheritance diagram for ExternalServices::TopologyInformationI:



Additional Inherited Members

6.45.1 Detailed Description

Defines an interface for the topology information service. In our current thinking, this is just another Network Discovery service, hence has the same APIs.

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

- TuscaroraFW/Include/Interfaces/ExternalServices/TopologyInformationI.h

6.46 Patterns::PatternBase Class Reference

Defines the abstract base class for patterns.

`#include <PatternBase.h>`

Public Member Functions

- **PatternBase** (PatternTypeE type, char _uniqueName[128])
- virtual bool Start ()=0

  Starts the pattern, must be implemented by each pattern.
- virtual bool **Stop** ()=0
- virtual ~PatternBase ()

  Virtual destructor.

Public Attributes

- RecvMessageDelegate_t * **recvDelegate**
- NeighborDelegate * **nbrDelegate**
- DataflowDelegate_t * **dataflowDelegate**
- ControlResponseDelegate_t * **controlDelegate**

Protected Member Functions

- virtual void **ReceiveMessageEvent** (FMessage_t &msg)=0
- virtual void **NeighborUpdateEvent** (NeighborUpdateParam nbrUpdate)=0
- virtual void **DataStatusEvent** (DataStatusParam notification)=0
- virtual void **ControlResponseEvent** (ControlResponseParam response)=0
- void **RegisterPatternDelegates** (PatternId_t pid, PatternTypeE _type)
- void **Handle_RegisterResponse** (ControlResponseParam response)
- void **RandomLocalSpray** (PatternId_t pid, FMessage_t &msg, SprayTypeE spraytype, bool israndomselection, Patterns::PatternNeighborTableI &ptnNbrTable, uint16_t nonce)
- bool **RandomlySelectNeighbor** (NeigborContainerType &selectedNeighborList, Patterns::PatternNeighborTableI &ptnNbrTable, UniformRandomInt *rand=NULL)
- void **Send2SelectedNeighbors** (PatternId_t pid, NeigborContainerType &selectedNeighborList, FMessage_t &msg, uint16_t nonce)

Protected Attributes

- char **uniqueName** [128]
- PatternId_t **PID**
- Framework_I * **FRAMEWORK**
- bool **registered**
- PatternRequestStateE **requestState**
- PatternStateE **patternState**
- uint32_t **n_ExpectedFrameworkResponses**
- UniformRandomInt * **randInt**

6.46.1 Detailed Description

Defines the abstract base class for patterns.

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

- TuscaroraFW/Include/Interfaces/Pattern/PatternBase.h

6.47 Patterns::PatternNeighborTableI Class Reference

Interface definition for the Neighbor service, that maintains a sorted list of neighbors based on their metric.

`#include <PatternNeighborTableI.h>`

Public Member Functions

- **PatternNeighborTableI** (LinkComparatorTypeE lcType)
- virtual uint16_t GetNumberOfNeighbors ()=0

  Sets the comparator used to sort the neighbors.
- virtual PatternNeighborIterator Begin ()=0

  Returns the top of list of neighbor sorted by quality in an iterator.
- virtual PatternNeighborIterator End ()=0

  Returns the end of list of neighbor sorted by quality in an iterator.
- virtual Core::Link * GetNeighborLink (NodeId_t nodeId)=0

  Returns the information about the neighbor specified by the nodeId.
- virtual Link * GetBestLink (WaveformId_t wid)=0

  Returns the best link on a given waveform specified by wid.
- virtual void UpdateTable (NeighborUpdateParam _param)=0

  Updates the neighor table with received neighbor update parameter.
- virtual ~PatternNeighborTableI ()

  Virtual empty destructor. Needed for abstract virtual class in C++.
- virtual void **PrintTable** ()=0

6.47.1 Detailed Description

Interface definition for the Neighbor service, that maintains a sorted list of neighbors based on their metric.

6.47.2 Member Function Documentation

6.47.2.1 virtual uint16_t Patterns::PatternNeighborTableI::GetNumberOfNeighbors ( ) [pure virtual]

Sets the comparator used to sort the neighbors.

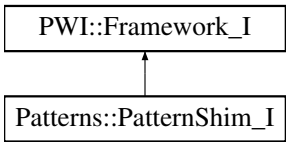
Returns the number of waveforms as the return valude and their Ids in the parameter Returns the number of nodes currently reachable that satisfies the Patterns neighbor criteria.

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

- TuscaroraFW/Include/Interfaces/PWI/PatternNeighborTableI.h

6.48 Patterns::PatternShim_I Class Reference

Inheritance diagram for Patterns::PatternShim_I:



Public Member Functions

- virtual void **RegisterDelegates** (PatternId_t _pid, RecvMessageDelegate_t *_recvDelegate, NeighborDelegate *_nbrDelegate, DataflowDelegate_t *_dataNotifierDelegate, ControlResponseDelegate_t *_controlDelegate)=0

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

- TuscaroraFW/Include/Interfaces/Pattern/PatternShim_I.h

6.49 PWI::AddDestinationResponse_Data Struct Reference

Public Attributes

- FMessageId_t **msgId**
- uint16_t **noOfDest**
- NodeId_t **destArray** [Core::MAX_DEST]
- bool **add_status** [Core::MAX_DEST]
- bool **status**

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

- TuscaroraFW/Include/Interfaces/PWI/FrameworkResponces.h

6.50 PWI::CancelDataResponse_Data Struct Reference

Public Member Functions

- void **SetAllStatusNegative** ()

Public Attributes

- FMessageId_t **msgId**
- bool **status**
- NodeId_t **destArray** [Core::MAX_DEST]
- bool **cancel_status** [Core::MAX_DEST]
- uint16_t **noOfDest**

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

- TuscaroraFW/Include/Interfaces/PWI/FrameworkResponces.h

6.51 PWI::ControlResponseParam Struct Reference

Structure used as parameter by the framework when sending the Control Response Event.

```
#include <Framework_I.h>
```

Public Attributes

- uint32_t **sequenceNo**
- ControlResponseTypeE **type**
- uint16_t **dataSize**
- void * **data**

6.51.1 Detailed Description

Structure used as parameter by the framework when sending the Control Response Event.

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

- TuscaroraFW/Include/Interfaces/PWI/Framework_I.h

6.52 PWI::DataStatusResponse_Data Struct Reference

Public Attributes

- FMessageId_t **msgId**
- NodeId_t **destArray** [MAX_DEST]
- DataStatusTypeE **ackStatus** [MAX_DEST]
- bool **status**

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

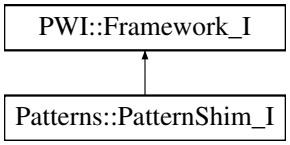
- TuscaroraFW/Include/Interfaces/PWI/FrameworkResponces.h

6.53 PWI::Framework_I Class Reference

Defines the generic interations between the Pattern and the Framework.

`#include <Framework_I.h>`

Inheritance diagram for PWI::Framework_I:



Public Member Functions

- virtual void SendData (PatternId_t pid, NodeId_t destArray[MAX_DEST], uint16_t noOfDest, FMessage_t &msg, uint16_t nonce, bool noDestAck=false)=0

  Sends message provided as parameter to the set of neighbors indicated by destArray.
- virtual void **SendData** (PatternId_t pid, NodeId_t destArray[MAX_DEST], uint16_t noOfDest, LinkComparatorTypeE lcType, FMessage_t &msg, uint16_t nonce, bool noDestAck=false)=0
- virtual void ReplacePayloadRequest (PatternId_t patternId, FMessageId_t msgId, void *payload, uint16_t sizeOfPayload)=0

  Replaces the payload of a packet already sent to the framework. The previous payload will be discarded and destroyed.
- virtual void RegisterPatternRequest (PatternId_t patternId, const char uniqueName[128], PatternTypeE type)=0

  Generates a pattern ID, based on the pattern type and an optional unique string.
- virtual void AddDestinationRequest (PatternId_t patternId, FMessageId_t msgId, NodeId_t destArray[MAX_DEST], uint16_t noOfNbrs)=0

  Adds more destinations to a message already handed over to the framework. If the original message was sent using BroadcastData method, this new request is ignored.
- virtual void **AddDestinationRequest** (PatternId_t patternId, FMessageId_t msgId, NodeId_t destArray[MAX_DEST], uint16_t noOfNbrs, LinkComparatorTypeE lcType)=0
- virtual void CancelDataRequest (PatternId_t patternId, FMessageId_t msgId, NodeId_t destArray[MAX_DEST], uint16_t noOfDest)=0

  Cancels the sending of a message to one or more destinations of a message previously handed over to the framework.

- virtual void DataStatusRequest (PatternId_t patternId, FMessageId_t msgId)=0

  Request the status of a data message sent to the framework.
- virtual void SelectDataNotificationRequest (PatternId_t patternId, uint8_t notifierMask)=0

  Lets the Pattern Select which type of Data Notification Updates the Pattern wants to recieve.
- virtual void SetLinkThresholdRequest (PatternId_t patternId, LinkMetrics threshold)=0

  Sets the link metric threshold to qualify as pattern neighbors.
- virtual void FrameworkAttributesRequest (PatternId_t patternId)=0

  Requests the attributes of the framework.
- virtual void SelectLinkComparatorRequest (PatternId_t patternId, LinkComparatorTypeE lcType)=0

  Selects the link comparator to use while comparing link qualities. Methods lets patterns to specify a link comparator from a list of existing library of comparators.
- virtual ~Framework_I ()

  Virtual destructor.

6.53.1 Detailed Description

Defines the generic interations between the Pattern and the Framework.

Interface class that defines a asynchornous message passing interface for interactions between the Pattern and the framework. A pattern may either be built as part of same process or as seperate process on the same node. When the pattern and the framework are running as two seperate processes, a shim-layer should be implementation that will translate the calls between the two processes.

The interactions in general consist of a command/request orginating from the Pattern to the framework and the framework responding to the commands using an Event response. There are 12 Commands that the pattern can issue to framework and 5 types of Events that the framework sends to the pattern. A command is a message in a specific format (based on a the command being invoked) sent from the pattern to the framework. Event is a message in a specific format sent from the framework to the pattern. While Commands and Events are messages, they are invoked using function syntax, just like invoking and providing functions. To send a command to the framework, a pattern uses a FRAMEWORK reference provided by the shim layer and to handle a event from the framework a pattern implements 4 methods one for each type of event. The shim layer takes care of invoking the correct method when the corresponding event message arrives.

The 4 event types are:

1. Control Response Event (PTN_ControlResponseEvent_t): This is dispatched in response to one of the *↩ Request message, with an appropriate response type, response status (sucuss/failure), any additional data and its size.
2. Received Message Event (PTN_RcvMessageEvent_t) : This event is dispatched whenever a message is received for the pattern, with the message as the parameter
3. Data Notification Event (PTN_DatNotificationEvent_t): This is dispatched to convey the status of the Send↩ Data and BroadcastData calls. The event provides the cummulative status of a message. Each notification contains an array called ‘statusType' that specifies the status type corresponding to each destination and another boolen array called the ‘statusValue' that specifies if that operation was a success or failure. For each destination, the status progresses in the following sequence: PDN_FW_RECV (framework received the packet), PDN_WF_RECV (waveform received the packet), PDN_WF_SENT (waveform sent the messsage out) and finally PDN_DST_RECV (destination received the message). The sequence stops at PDN_WF_S↩ ENT for messages sent using BroadcastData api, for which destination received notification is not supported. Also if the status type has progressed to the next in sequence, the previous type is implicitly successfull. For example, for a given destination the status type is 'PDN_WF_SENT', then it also means that PDN_FW_RECV and PDN_WF_RECV are true.
4. Neighbor Update Event (PTN_NeighborUpdateEvent_t): This event is dispatched to provide updates about the status of the nodes neighborhood. A neighbor update can either be about a new neighbor, a dead neighor or change in the quality of an existing neighbor.

6.53.2 Member Function Documentation

6.53.2.1 virtual void PWI::Framework_I::AddDestinationRequest (PatternId_t *patternId,* FMessageId_t *msgId,* NodeId_t *destArray[MAX_DEST],* uint16_t *noOfNbrs*) [pure virtual]

Adds more destinations to a message already handed over to the framework. If the original message was sent using BroadcastData method, this new request is ignored.

Adds destinations to a message already sent to the framework. If the new destinations specifed using is destArray parameter is split across two or more waveforms, it is possible that the request can succeed for some of the new destinations and can fail for others. A Control Response Event will be generated by the framework with the result of the operation.

Parameters

patternId	Pattern's instance ID.
msgId	Unique message ID.
destArray	Array of destinations to be added.
noOfNbrs	Number of new destinations.

Returns

void.

6.53.2.2 virtual void PWI::Framework_I::CancelDataRequest (PatternId_t *patternId,* FMessageId_t *msgId,* NodeId_t *destArray[MAX_DEST],* uint16_t *noOfDest*) [pure virtual]

Cancels the sending of a message to one or more destinations of a message previously handed over to the framework.

Cancels the sending of a message previously handed over to the framework. If no destination is specified or if the original message was sent using BroadcastData method, entire message is cancelled and destroyed. Otherwise sending is cancelled only to the desitnations specified in the destArray parameter. A Control Response Event is generated with the result of the operation.

Parameters

patternId	Pattern's instance ID.
msgId	Unique message ID.
destArray	Array of destinations that need to be cancelled
noOfDest	Number of destinations to be cancelled.

Returns

void.

6.53.2.3 virtual void PWI::Framework_I::DataStatusRequest (PatternId_t *patternId,* FMessageId_t *msgId*) [pure virtual]

Request the status of a data message sent to the framework.

If for any reason a pattern has ambiguity about the status of a message that was sent to the framework, the pattern can use this method to request the status of the message. The framework will generate a Data Notification Event in response to this command, with the cummulative status of the message till that point.

Parameters

patternId	Pattern's instance ID.
msgId	ID of the message for which status is being requested.

Returns

void

6.53.2.4 virtual void PWI::Framework_I::FrameworkAttributesRequest (PatternId_t *patternId*) [pure virtual]

Requests the attributes of the framework.

The method is used to know the current attributes of the framework. The addritbutes are defined in the FrameworkAttributes struct. A pattern should first querry this method to find out the attributes of the framework before starting messaging. Critical information such as Max packet size, number of waveforms in the system, their IDs, etc. are returned through the FrameworkAttributes struct. A Control Response Event is generated in response to this command, with the FrameworkAttributes copied into the data section of the event. It is possible for the framework to generate a Control Response Event with the type set to 'PTN_AttributeResponse', even without a pattern requesting it, if the framework attributes changes while a pattern is active.

Parameters

patternId	Pattern's instance ID.

Returns

void.

6.53.2.5 virtual void PWI::Framework_I::RegisterPatternRequest (PatternId_t *patternId,* const char *uniqueName[128],* PatternTypeE *type*) [pure virtual]

Generates a pattern ID, based on the pattern type and an optional unique string.

The method is used to get a unique pattern ID for each pattern using the framework. The ID itself is generated by a Pattern Name Service. The framework acts as an pass through for this service. The ID is return through a Control Response Event. The uniqueName parameter can be used for fail-safe reasons. For example, if a pattern which was running crashes and reboot, it could get the ID for the previous instance by using the same uniqueName of the previous instance.

Parameters

ptype	Pattern type.
uniqueName	An optional parameter using which a pattern can identify itself to the framework. The Pattern Name Service will rememeber the string the pattern provides and will generate the same ID, if both the Pattern type and the uniqueName string match.

Returns

void.

virtual void NewPatternInstanceRequest(PatternTypeE ptype, char uniqueName[128] = NULL)= 0; Registers a pattern with the framework.

Patterns can use this method to register with the framework and optionally request for a pattern ID. A pattern will need to register first to be able to access the services of the framework. A Control Response Event will be generate to notify the pattern of the result of the registration. If the pattern does not have an ID, it sets the input variable "patternId" to zero and uses Framework generates a new ID for the pattern in this case. The Control Response Event will include the new pattern ID in this case.

If the pattern already has an ID either generated through an external service or obtained previously, it can use the existing patternID. In this case, the input uniqueName is ignored by the Framework.

Parameters

patternId	Unique ID of the Pattern
uniqueName	Unique string indicating the pattern instance used in pattern id generation
type	Pattern type

Returns

void.

6.53.2.6 virtual void PWI::Framework_I::ReplacePayloadRequest (PatternId_t *patternId,* FMessageId_t *msgId,* void * *payload,* uint16_t *sizeOfPayload*) [pure virtual]

Replaces the payload of a packet already sent to the framework. The previous payload will be discarded and destroyed.

Replaces the payload of a packet already sent to the framework. The previous payload will be discarded and destroyed. In response to this command one or more Control Response Event(s) may be generated by the framework. If the message has already been sent to the waveform, but not sent out by the waveform, the framework will issue commands to the waveform to replace the payload, however the waveform may return a failure. Hence, if the original message was sent to more than one destination, it is possible that this operation can fail for some destination and can succeed for others.

Parameters

patternId	Id of the pattern sending the message.
msgId	ID of the message in which the payload should be replaced
payload	Pointer to new payload.
sizeOfPayload	Size of the new payload.

Returns

void.

6.53.2.7 virtual void PWI::Framework_I::SelectDataNotificationRequest (PatternId_t *patternId,* uint8_t *notifierMask*) [pure virtual]

Lets the Pattern Select which type of Data Notification Updates the Pattern wants to recieve.

This methods lets the pattern to customize data notifications that it wishes to receive. The customization is achived by specifying the mask of the data status types that pattern is interested in receiving. While this could decrease the number and frequency of notifications the pattern receives, the framework cannot completely eliminate the status types that the pattern is not interested in since the data notification is an cummulative.

Parameters

patternId	Pattern's instance ID.
notifierMask	Mask of the notifications that are requested.

Returns

void

6.53.2.8 virtual void PWI::Framework_I::SelectLinkComparatorRequest (PatternId_t *patternId,* **LinkComparatorTypeE** *lcType*) [pure virtual]

Selects the link comparator to use while comparing link qualities. Methods lets patterns to specify a link comparator from a list of existing library of comparators.

This method lets a pattern to customize the comparison of the link metrics between two links. While there are 5 standard link metrics, how to select one link as better than another can be customized using this method, by selecting one of the many pre-approved comparators. Please see documentation for 'LinkComparatorTypeE' for available comparators.

Parameters

patternId	Pattern's instance ID.
lcType	Type of the link comparator. Enum.

Returns

void.

6.53.2.9 virtual void PWI::Framework_I::SendData (PatternId_t *pid,* NodeId_t *destArray[MAX_DEST],* uint16_t *noOfDest,* **FMessage_t** & *msg,* uint16_t *nonce,* bool *noDestAck* = false) [pure virtual]

Sends message provided as parameter to the set of neighbors indicated by destArray.

This is the primary method a pattern uses to request that a message be transmitted to its neighbors. The first two parameters are self-explanatory. The third parameter is an array of one or more neighbors to which the message is to be sent, each of which are identified by the neighbor's waveform address. The Framework may use the given MessageId to cancel or replace the payload in future (via calls to the next two methods). The parameter noDestAck specifies whether a notification is requested when this message has reached its intended destination(s).

A true value for this field means the Framework does not need notifications for this message. If data notifications/acknowledgements are not supported by the waveform, this parameter may be ignored.

Data Notifications: Framework returns one or more data notification events corresponding to each message sent. The Data Notifications are cummulative in nature. Each notification contains an array called 'statusType' that specifies the status type corresponding to each destination and another boolen array called the 'statusValue' that specifies if that operation was a success or failure.

Nonce: Nonce is an unsigned integer that the pattern uses to identify the message, before the framework can return a message Id. Framework uses the nonce in the data notification and also returns a Message ID generated by the framework for the message. The nonce need not be unique for every message, but must be different for consequent messages. If two messages have the same once, then the framework assumes that the data notificaiton it sent to the pattern for the previous message is lost and hence the pattern is resending the same message with the same nonce.

Parameters

pid	Id of the pattern sending the message.
destArray	List of destinations.
noOfDest	Number of destinations specified in the destination array.
lcType	The link comparator type to be used to differentiate between multiple links to the same neighbor
msg	Reference of the message to be send out. Tuscarora uses 'handover' semantics for parameters passed as referece. That is the Pattern SHOULD NOT deallocate the message that is sent out. The shim layer will deallocate the message once it has been sent out.
nonce	An unsigned integer specified by the pattern to differentiate this message from previous message. It is used for error recovery between the pattern and the framework.
noDestAck	Indicates if the pattern is not requesting destination acknowledgement for this message. A true value means no destination ack is requested. Default value is false.

Returns

void

6.53.2.10 virtual void PWI::Framework_I::SetLinkThresholdRequest (PatternId_t *patternId,* LinkMetrics *threshold*) [pure virtual]

Sets the link metric threshold to qualify as pattern neighbors.

This method lets a Pattern customize its neighbor and the neighborhood updates. A pattern can specify the minimum metric that a link should meet to qualify as its neighbor. This simplifies neighbor maintanence and look up.Also, neighborhood updates will be sent to patterns only about neighbors who meet this threshold, hence the number of Neighbor Update Events sent to the pattern will decrease. If a pattern does not set this threshold, all neighbors know to the framework will be published to the pattern. A Control Response Event is generated in response to this command to indicate the status of the operation.

Parameters

patternId	Pattern's instance ID.
threshold	Link metric threshold to be set.

Returns

void.

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

- TuscaroraFW/Include/Interfaces/PWI/Framework_I.h

6.54 PWI::FrameworkAttributes Struct Reference

Describtes the attributes of the framework.

`#include <Framework_I.h>`

Public Member Functions

- FrameworkAttributes ()

  Maximum size of a packet that can be sent unfragmented on all of the existing waveforms.

Public Attributes

- uint8_t **numberOfWaveforms**
- WaveformId_t waveformIds [MAX_WAVEFORMS]

  Number of waveforms supported/accessible to the pattern on the node.
- uint16_t maxFrameworkPacketSize

  Waveform IDs of the waveforms.

6.54.1 Detailed Description

Describtes the attributes of the framework.

Any pattern should first query the FrameworkAttributes before it starts messaging.

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

- TuscaroraFW/Include/Interfaces/PWI/Framework_I.h

6.55 PWI::Neighborhood::LinkComparatorI Class Reference

Public Member Functions

- virtual bool BetterThan (Core::LinkMetrics &A, Core::LinkMetrics &B)=0

  Returns true if A is better than B, false otherwise.

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

- TuscaroraFW/Include/Interfaces/PWI/LinkCompartorI.h

6.56 PWI::NewPatternInstanceResponse_Data Struct Reference

Public Attributes

- bool **status**
- char **uniqueName** [128]
- PatternId_t **pid**
- PatternTypeE **type**

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

- TuscaroraFW/Include/Interfaces/PWI/FrameworkResponces.h

6.57 PWI::PatternEvent< eventName, Param0 > Class Template Reference

Public Member Functions

- bool **Invoke** (PatternId_t pid, Param0 param0)

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

- TuscaroraFW/Include/Interfaces/Pattern/PatternEvent.h

6.58 PWI::PatternEventDispatch_I Class Reference

Public Member Functions

- virtual bool **InvokeControlEvent** (PatternId_t pid, ControlResponseParam _param0)=0
- virtual bool **InvokeRecvMsgEvent** (PatternId_t pid, FMessage_t &_param0)=0
- virtual bool **InvokeDataStatusEvent** (PatternId_t pid, DataStatusParam _param0)=0
- virtual bool **InvokeNeighborUpdateEvent** (PatternId_t pid, NeighborUpdateParam _param0)=0
- virtual void **RegisterDelegates** (PatternId_t pid, RecvMessageDelegate_t *_recvDelegate, NeighborDelegate *_nbrDelegate, DataflowDelegate_t *_dataNotifierDelegate, ControlResponseDelegate_t *_controlDelegate)

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

- TuscaroraFW/Include/Interfaces/PWI/PatternEventDispatch_I.h

6.59 PWI::RegistrationResponse_Data Struct Reference

Public Attributes

- bool **status**
- uint8_t **priority**
- uint16_t **noOfOutStandingPkts**
- PatternId_t **patternId**

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

- TuscaroraFW/Include/Interfaces/PWI/FrameworkResponces.h

6.60 PWI::ReplacePayloadResponse_Data Struct Reference

Public Member Functions

- void **SetAllStatusNegative** ()

Public Attributes

- FMessageId_t **msgId**
- bool **status**
- NodeId_t **noOfDest**
- NodeId_t **destArray** [Core::MAX_DEST]
- bool **replace_status** [Core::MAX_DEST]

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

- TuscaroraFW/Include/Interfaces/PWI/FrameworkResponces.h

6.61 PWI::SelectDataNotificationResponse_Data Struct Reference

Public Attributes

- bool **status**

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

- TuscaroraFW/Include/Interfaces/PWI/FrameworkResponces.h

6.62 PWI::SetLinkThresholdResponse_Data Struct Reference

Public Attributes

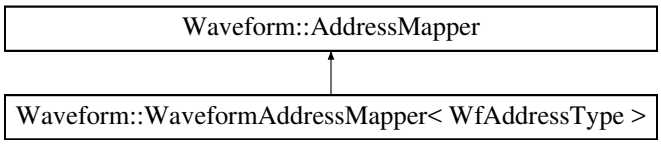
- bool **status**

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

- TuscaroraFW/Include/Interfaces/PWI/FrameworkResponces.h

6.63 Waveform::AddressMapper Class Reference

Inheritance diagram for Waveform::AddressMapper:



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

- TuscaroraFW/Include/Interfaces/Waveform/AddressMapper.h

6.64 Waveform::FragmentationHeader Struct Reference

Public Attributes

- uint8_t **frag_number**
- uint16_t **total_payload_size**

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

- TuscaroraFW/Include/Interfaces/Waveform/WF_MessageT.h

6.65 Waveform::PacketAdaptor< addressType > Class Template Reference

Public Member Functions

- WF_MessageT< addressType > & **Convert_FM_to_WM** (FMessage_t &msg, bool copyDest)
- FMessage_t & **Convert_WM_to_FM** (WF_MessageT< addressType > &wfMsg, bool copyDest)

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

- TuscaroraFW/Include/Interfaces/Waveform/WF_MessageT.h

6.66 Waveform::PacketMetadata Struct Reference

Structure for packet metadata that should be supported by all waveforms.

`#include <WF_MessageT.h>`

Public Attributes

- uint8_t **rss**
- uint8_t sinr

  Receiver Signal Strength.
- U64NanoTime recvTimeStamp

  Signal to noise and interference ratio.
- bool IsOverHeard

  Receive time stamp of the packet in local clock time.

6.66.1 Detailed Description

Structure for packet metadata that should be supported by all waveforms.

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

- TuscaroraFW/Include/Interfaces/Waveform/WF_MessageT.h

6.67 Waveform::SoftwareAcknowledgement Struct Reference

Public Attributes

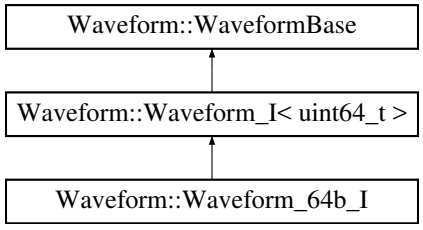
- WaveformId_t **wfId**
- WF_MessageId_t **wfMsgId**
- NodeId_t **src**

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

- TuscaroraFW/Include/Interfaces/Waveform/WF_MessageT.h

6.68 Waveform::Waveform_64b_I Class Reference

Inheritance diagram for Waveform::Waveform_64b_I:



Public Member Functions

- **Waveform_64b_I** (WaveformId_t _wId, WF_TypeE _type, WF_EstimatorTypeE _estType, char *_deviceName)
- virtual void SendData (WF_Message_n64_t &_msg, uint16_t _payloadSize, uint64_t *_destArray, uint16_t _noOfDest, WF_MessageId_t _msgId, bool _noAck=false)=0

  Send message indicated by the MessageId to a set of neighbors indicated by destArray, with a given payload size.
- virtual void BroadcastData (WF_Message_n64_t &_msg, uint16_t _payloadSize, WF_MessageId_t _msgId)=0

  Sends a broadcast on a waveform. Destination received data notification will not be generated If the waveform does not support physical layer broadcasting, the implementation is upto the waveform or the waveform might simply not implement this interface.
- virtual void AddDestinationRequest (RequestId_t _rId, WF_MessageId_t _msgId, uint64_t *_destArray, uint16_t _noOfDestinations)=0

  Adds one or more destinations to a packet, that has already been sent to the waveform. Waveform will send a Control Response Event as reply to this command.
- virtual void CancelDataRequest (RequestId_t _rId, WF_MessageId_t _msgId, uint64_t *_destArray, uint16_t _noOfDestinations)=0

  Requests the Cancellation of a data message already sent to the waveform for one or more destinations.
- virtual void ReplacePayloadRequest (RequestId_t rId, WF_MessageId_t msgId, uint8_t *payload, uint16_t payloadSize)=0

  Requests the replacement the payload of a message, passed in an earlier call to 'SendData' or 'BroadcastData' method with the given MessageId. The previous payload will be discarded.
- virtual void AttributesRequest (RequestId_t rId)=0

  Request the attributes of the waveform.
- virtual void ControlStatusRequest (RequestId_t rId)=0

  Requests the control status of waveform. A WF_StatusE enum type is sent as the data of the response message, to indicate the status of the variable.
- virtual void DataStatusRequest (RequestId_t rId, WF_MessageId_t mId)=0

  Requests the status of a data message already sent to the waveform.
- virtual void **DataNotification** (WF_DataStatusParam_n64_t)=0
- virtual void **ProcessIncomingMessage** (WaveformId_t wfid, uint16_t msglen, unsigned char *buf)=0

Additional Inherited Members

6.68.1 Member Function Documentation

6.68.1.1 virtual void Waveform::Waveform_64b_I::AddDestinationRequest (RequestId_t *_rId,* WF_MessageId_t *_msgId,* uint64_t * *_destArray,* uint16_t *_noOfDestinations*) [pure virtual]

Adds one or more destinations to a packet, that has already been sent to the waveform. Waveform will send a Control Response Event as reply to this command.

Semantic Behavior: Waveform should add a destination to the outgoing message if it has not already sent out the message identified by the message ID and send a positive response. If the waveform has already sent out the message to all the destinations and has cleaned its local buffers of the message, it should send a negative response to this request. Once destination has been added to a message, the waveform should communicate the result of the operation using Data Status Response Events as usual.

A response to this request should be generated through the Control Response Event, with the same RequestID, that should include a status which will be set to false if the message has been sent out or if the waveform is otherwise unable to add destinations to the message. Otherwise the status should be set to true in the response message.

Parameters

_rId	Specifies the ID of the request, which the waveform will use in its response message
_msgId	ID of the message that to which the destinations needs to be added
_destArray	New destinations to be added to the message
_noOfDestinations	Number of destinations in the destArray. If this is 0, the request will be ignored by waveform.

Returns

void

Implements Waveform::Waveform_I< uint64_t >.

6.68.1.2 virtual void Waveform::Waveform_64b_I::AttributesRequest (RequestId_t *rId*) [pure virtual]

Request the attributes of the waveform.

Semantic Behavior: Waveform should return its attributes to the Framework by generating a response through the Control Response Event, with the same request ID. The status of the response message should be true, unless otherwise the attributes are not available to be sent to the Framework. The response contains attributes in the WF_Attributes structure, which is copied into the data portion of the WF_ControlResponseParam.

Parameters

rId	Specifies the ID of the request, which the waveform will use to in its response message.

Returns

void.

Implements Waveform::WaveformBase.

6.68.1.3 virtual void Waveform::Waveform_64b_I::BroadcastData (WF_Message_n64_t & *_msg,* uint16_t *_payloadSize,* WF_MessageId_t *_msgId*) [pure virtual]

Sends a broadcast on a waveform. Destination received data notification will not be generated If the waveform does not support physical layer broadcasting, the implementation is upto the waveform or the waveform might simply not implement this interface.

Semantic Behavior: Broadcast is an optional feature for waveforms. They can choose not to implement it. If implemented, the waveform should set the 'broadcastSupport' field in its Attribute as true whenever it responds to the AttributeRequest. When the Framework makes a broadcast request, it is expected that the waveform will transmit the message without any specific addressing and without any expectation of destination acknowledgements. The waveform should perform sufficient effort such that neighboring nodes as well as nodes that could be a neighbor might sometimes receive the packet under representative operating conditions.

Efficiency is the key intent of the broadcast method. For example, if a waveform has neighbors across 7 different spectral-temporal channels and to reach all of the known neighbors it has to send the same message 7 times, once

on each channel, then the nominal implementation would randomly select a channel and send one message on that channel. The randomization ensures that any given neighbor will be reached at least sometimes.

The MessageId may be used by the pattern to cancel or replace the payload in future. The waveform is expected to generate 2 notifications through the Data Notification Event, for every message sent to it, the first indicating that the waveform has received the message and the second indicating that it has sent out the message. The waveform can generate a third notification indicating that the destination(s) has received the message if it can support destination acknowledgements.

Parameters

msg	Reference to the packet to be broadcasted
payloadSize	Size of the payload in the packet.
msgId	Specifies the id of the framework will use to identify this packet in future calls.

Returns

void.

Reimplemented from Waveform::Waveform_I< uint64_t >.

6.68.1.4 virtual void Waveform::Waveform_64b_I::CancelDataRequest (RequestId_t *_rId,* WF_MessageId_t *_msgId,* uint64_t * *_destArray,* uint16_t *_noOfDestinations*) [pure virtual]

Requests the Cancellation of a data message already sent to the waveform for one or more destinations.

Semantic Behavior: Waveform should cancel the transmission of the message, passed in an earlier call to SendData or BroadcastData method with the given message ID, if the message has not been sent out. The message transmission should be cancelled only for the destinations passed as parameters. If the noOfDestinations parameter is 0, then the entire message should be cancelled. If the message ID is for a Broadcast message the whole message should be cancelled.

A response message should be generated through the Control Response Event, with the same RequestID, that should include a status which will be set to false if the message has been sent out or if the waveform is otherwise unable to cancel the transmissions. Otherwise the status should be set to true in the response message.

Parameters

rId	Specifies the ID of the request, which the waveform will use to in its response message.
msgId	ID of the message that needs to be cancelled.
destArray	Array of destinations to which the packet should be cancelled.
noOfDestinations	Number of destinations in the destArray. If the number of destinations is 0, the entire message is cancelled.

Returns

void

Implements Waveform::Waveform_I< uint64_t >.

6.68.1.5 virtual void Waveform::Waveform_64b_I::ControlStatusRequest (RequestId_t *rId*) [pure virtual]

Requests the control status of waveform. A WF_StatusE enum type is sent as the data of the response message, to indicate the status of the variable.

Semantic Behavior: The Framework expects the waveform to be in one of the data plane states defined by the WF_ControlP_StatusE enum. The waveform should return the current status of its data plane by generating a response through the Control Response Event, with the same RequestId. The status that response message should be true. The actual data plane status should be copied into the data part of the response

Parameters

rId	Specifies the ID of the request, which the waveform will use to in its response message.

Returns

void.

Implements Waveform::WaveformBase.

6.68.1.6 virtual void Waveform::Waveform_64b_I::DataStatusRequest (RequestId_t *rId,* WF_MessageId_t *mId*) [pure virtual]

Requests the status of a data message already sent to the waveform.

Semantic Behavior: If the Framework is uncertain about the status of a message that it sent to the waveform, it can send the DataStatusRequest command. A response message should be generated through the Control Response Event, with the same RequestId, that should have the status set as 'true' if the waveform received the particular message and a WF_DataNotifierParam object must be created that reflects the current status of the message and should be copied into the data part of the WF_ControlResponseParam.

Parameters

rId	Specifies the ID of the request, which the waveform will use to in its response message.
mId	Specified the ID of the message, for which status is requested.

Returns

void.

Implements Waveform::WaveformBase.

6.68.1.7 virtual void Waveform::Waveform_64b_I::ReplacePayloadRequest (RequestId_t *rId,* WF_MessageId_t *msgId,* uint8_t * *payload,* uint16_t *payloadSize*) [pure virtual]

Requests the replacement the payload of a message, passed in an earlier call to 'SendData' or 'BroadcastData' method with the given MessageId. The previous payload will be discarded.

Semantic Behavior: Waveform should replace the payload of a message passed in an earlier call to 'SendData' or 'BroadcastData' method with the given message ID, if the message has not been sent out. A response message should be generated through the Control Response Event, with the same request ID, that should include a status which will be set to false if the message has been sent out or if the waveform is unable to replace the payload of a message. Otherwise, the status should be set to true in the response message.

Parameters

rId	Specifies the ID of the request, which the waveform will use to in its response message.
msgId	specifies the message ID of the packet.
payload	pointer to the new payload.
payloadSize	size of the new payload.

Returns

void.

Implements Waveform::WaveformBase.

6.68.1.8 virtual void Waveform::Waveform_64b_I::SendData (WF_Message_n64_t & *_msg*, uint16_t *_payloadSize*, uint64_t * *_destArray*, uint16_t *_noOfDest*, WF_MessageId_t *_msgId*, bool *_noAck* = false) [pure virtual]

Send message indicated by the MessageId to a set of neighbors indicated by destArray, with a given payload size.

Semantic Behavior: This is the primary method the Framework uses to request that a message be transmitted by the waveform. The first two parameters are self-explanatory. The third parameter is an array of one or more neighbors to which the message is to be sent, each of which are identified by the neighbor's waveform address. The Framework may use the given MessageId to cancel or replace the payload in future . The parameter _noAck specifies whether a notification is requested when this message has reached its intended destination(s). A true value for this field means the Framework does not need notifications for this message. If data notifications/acknowledgements are not supported by the waveform, this parameter may be ignored.

Data Notifications: It is recommended that Waveforms support acknowledgements from destinations for unicast and multi-cast (in local neighborhood) messages. However this is optional. If the waveform choose to implement receiver acks, it should set the 'destReceiveAckSupport' field in the Attribute record as true whenever it responds to the AttributeRequest. If the waveform does not support this feature, then the framework will implement destination acknowledgements by replying to the source node on receiving messages from the waveform. The MessageId may be used by the Framework to cancel or replace the payload in future. The waveform is expected to generate at least 2 notifications through the Data Notification Event, for every message sent to it, the first indicating that the waveform has received the message and the second indicating that it has sent out the message. The waveform can generate a third notification indicating that the destination(s) has received the message if it chooses to support destination acknowledgement.

Performance: A request to send to multiple nodes should be fulfilled via a technique that is no less efficient than a sequence of transmissions to each node. The intent is that more advanced waveforms may internally optimize beyond this base performance level, by combining transmissions when possible and not unduly difficult. Any optimizations that: 1) are not naturally supported by the hardware and the waveform, 2) require non-local information, or 3) consume an order of magnitude more computational resources than required for the core send operation, should not be supported

Parameters

msg	Reference to the packet be sent.
payloadSize	Size of the payload in the packet.
destArray	Array of destinations to which the packet should be sent.
noOfDest	Number of destinations in the destArray.
msgId	Specifies the packet id the framework will use to identify this packet in future calls.
noAck	Specifies if acks should not be generated. Defaults option generates acks.

Returns

void.

Implements Waveform::Waveform_I< uint64_t >.

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

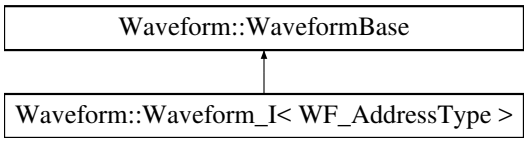
- TuscaroraFW/Include/Interfaces/Waveform/Waveform_64b_I.h

6.69 Waveform::Waveform_I< WF_AddressType > Class Template Reference

Defines the generic iteractions for waveforms.

`#include <WF_I.h>`

Inheritance diagram for Waveform::Waveform_I< WF_AddressType >:



Public Types

- typedef WF_Event< WF_RECV_MSG_EVT, WF_RecvMsgParam< WF_AddressType > > **WF_RcvMessageEvent_t**
- typedef WF_Event< WF_DATA_NTY_EVT, WF_DataStatusParam< WF_AddressType > > **WF_DataStatusEvent_t**
- typedef WF_Event< WF_LINK_EST_EVT, WF_LinkEstimationParam< WF_AddressType > > **WF_LinkEstimateEvent_t**

Public Member Functions

- Waveform_I (WaveformId_t _wId, WF_TypeE _type, WF_EstimatorTypeE _estType, char *_deviceName)

  Constructor. Creates a waveform object.
- virtual void SendData (WF_MessageT< WF_AddressType > &_msg, uint16_t _payloadSize, WF_AddressType *_destArray, uint16_t _noOfDest, WF_MessageId_t _msgId, bool _noAck=false)=0

  Send message indicated by the MessageId to a set of neighbors indicated by destArray, with a given payload size.
- virtual void BroadcastData (WF_MessageT< WF_AddressType > &_msg, uint16_t _payloadSize, WF_MessageId_t _msgId)

  Sends a broadcast on a waveform. Destination received data notification will not be generated If the waveform does not support physical layer broadcasting, the implementation is upto the waveform or the waveform might simply not implement this interface.
- virtual void CancelDataRequest (RequestId_t _rId, WF_MessageId_t _msgId, WF_AddressType *_destArray, uint16_t _noOfDestinations)=0

  Requests the Cancellation of a data message already sent to the waveform for one or more destinations.
- virtual void AddDestinationRequest (RequestId_t _rId, WF_MessageId_t _msgId, WF_AddressType *_destArray, uint16_t _noOfDestinations)=0

  Adds one or more destinations to a packet, that has already been sent to the waveform. Waveform will send a Control Response Event as reply to this command.
- virtual ~Waveform_I ()

  virtual destructor for interface

Protected Attributes

- WaveformId_t **WID**
- WF_TypeE **wfType**
- char **deviceName** [32]
- WF_EstimatorTypeE **estType**

6.69.1 Detailed Description

template<class WF_AddressType>
class Waveform::Waveform_I< WF_AddressType >

Defines the generic iteractions for waveforms.

The interface defines a asynchornous message passing interface for interacting with a waveform running either as part of same process or as seperate process on the same node. When running as two seperate processes, a shim-layer will be implementation that will translate the calls between the two processes.

The control flow in general consist of a command/request orginating from the framework to the waveform and the waveform responding to the command/request using a event response. There four types of events that the waveform can send to the framework :

1. Response Event (WF_ControlResponseEvent_t): This is called in response to one of the *Request message, with an appropriate response type, response status (sucuss/failure), any additional data and its size
2. Received Message Event (WF_RcvMessageEvent_t) : This delegate is called whenever a message is received on the waveform, with the message as the parameter
3. Data notification delegate (WF_DataNotifierEvent_t): This is called to convey the status of the SendData and BroadcastData calls. The delegates are might be called upto three types for each message, once to acknowledge the message has been received by the waveform, once when it is sent out and once when the message is received at the destination.
4. Link estimation delegate (WF_LinkEstimatesEvent_t): This is called by waveform to provide link estiamtions about neighbors to the framework, if the waveform can provide link estimates.

6.69.2 Constructor & Destructor Documentation

6.69.2.1 template<class WF_AddressType> Waveform::Waveform_I< WF_AddressType >::Waveform_I (WaveformId_t *_wid,* WF_TypeE *_type,* WF_EstimatorTypeE *_estType,* char * *_deviceName*)

Constructor. Creates a waveform object.

Semantic Behavior: The constructor is expected to create and initialize the waveform object. The constructor is also expected to “connect” to the Framework process. The mechanism of connecting may be platform specific. The data plane of the waveform is expected to be in WF_NORMAL state at the end of this call.

The WaveformId is allocated statically during compile time in the current version of Tuscarora. Other parameters of the constructor are decided by the waveform provider.

Parameters

_wId	Unique ID of the waveform. The WaveformId is allocated when the Tuscarora is initialized. A waveform writter usually doesnot have to worry about how they are generated. A waveform is expected to store this ID and use it to uniquely identify itself when communicating with the framework.
_type	Type of the waveform. This is one of the types of the waveform the framework supports.
_estType	Type of the estimator provided by the waveform. Indicated weather or not the waveform supports link estimation.
_deviceName	Name of the device which will be used by the waveform, if device names exist on the platform used to deploy the waveform.

6.69.3 Member Function Documentation

6.69.3.1 template<class WF_AddressType> virtual void Waveform::Waveform_I< WF_AddressType >::AddDestinationRequest (RequestId_t *_rId,* WF_MessageId_t *_msgId,* WF_AddressType * *_destArray,* uint16_t *_noOfDestinations*) [pure virtual]

Adds one or more destinations to a packet, that has already been sent to the waveform. Waveform will send a Control Response Event as reply to this command.

Semantic Behavior: Waveform should add a destination to the outgoing message if it has not already sent out the message identified by the message ID and send a positive response. If the waveform has already sent out the message to all the destinations and has cleaned its local buffers of the message, it should send a negative response to this request. Once destination has been added to a message, the waveform should communicate the result of the operation using Data Status Response Events as usual.

A response to this request should be generated through the Control Response Event, with the same RequestID, that should include a status which will be set to false if the message has been sent out or if the waveform is otherwise unable to add destinations to the message. Otherwise the status should be set to true in the response message.

Parameters

_rId	Specifies the ID of the request, which the waveform will use in its response message
_msgId	ID of the message that to which the destinations needs to be added
_destArray	New destinations to be added to the message
_noOfDestinations	Number of destinations in the destArray. If this is 0, the request will be ignored by waveform.

Returns

void

Implemented in Waveform::Waveform_64b_I.

6.69.3.2 template<class WF_AddressType> virtual void Waveform::Waveform_I< WF_AddressType >::BroadcastData (WF_MessageT< WF_AddressType > & *_msg,* uint16_t *_payloadSize,* WF_MessageId_t *_msgId*) [inline], [virtual]

Sends a broadcast on a waveform. Destination received data notification will not be generated If the waveform does not support physical layer broadcasting, the implementation is upto the waveform or the waveform might simply not implement this interface.

Semantic Behavior: Broadcast is an optional feature for waveforms. They can choose not to implement it. If implemented, the waveform should set the 'broadcastSupport' field in its Attribute as true whenever it responds to the AttributeRequest. When the Framework makes a broadcast request, it is expected that the waveform will transmit the message without any specific addressing and without any expectation of destination acknowledgements. The waveform should perform sufficient effort such that neighboring nodes as well as nodes that could be a neighbor might sometimes receive the packet under representative operating conditions.

Efficiency is the key intent of the broadcast method. For example, if a waveform has neighbors across 7 different spectral-temporal channels and to reach all of the known neighbors it has to send the same message 7 times, once on each channel, then the nominal implementation would randomly select a channel and send one message on that channel. The randomization ensures that any given neighbor will be reached at least sometimes.

The MessageId may be used by the pattern to cancel or replace the payload in future. The waveform is expected to generate 2 notifications through the Data Notification Event, for every message sent to it, the first indicating that the waveform has received the message and the second indicating that it has sent out the message. The waveform can generate a third notification indicating that the destination(s) has received the message if it can support destination acknowledgements.

Parameters

msg	Reference to the packet to be broadcasted
payloadSize	Size of the payload in the packet.
msgId	Specifies the id of the framework will use to identify this packet in future calls.

Returns

void.

Reimplemented in Waveform::Waveform_64b_I.

6.69.3.3 template<class WF_AddressType> virtual void Waveform::Waveform_I< WF_AddressType >::CancelDataRequest (RequestId_t *_rId,* WF_MessageId_t *_msgId,* WF_AddressType * *_destArray,* uint16_t *_noOfDestinations*) [pure virtual]

Requests the Cancellation of a data message already sent to the waveform for one or more destinations.

Semantic Behavior: Waveform should cancel the transmission of the message, passed in an earlier call to SendData or BroadcastData method with the given message ID, if the message has not been sent out. The message transmission should be cancelled only for the destinations passed as parameters. If the noOfDestinations parameter is 0, then the entire message should be cancelled. If the message ID is for a Broadcast message the whole message should be cancelled.

A response message should be generated through the Control Response Event, with the same RequestID, that should include a status which will be set to false if the message has been sent out or if the waveform is otherwise unable to cancel the transmissions. Otherwise the status should be set to true in the response message.

Parameters

rId	Specifies the ID of the request, which the waveform will use to in its response message.
msgId	ID of the message that needs to be cancelled.
destArray	Array of destinations to which the packet should be cancelled.
noOfDestinations	Number of destinations in the destArray. If the number of destinations is 0, the entire message is cancelled.

Returns

void

Implemented in Waveform::Waveform_64b_I.

6.69.3.4 template<class WF_AddressType> virtual void Waveform::Waveform_I< WF_AddressType >::SendData (WF_MessageT< WF_AddressType > & *_msg,* uint16_t *_payloadSize,* WF_AddressType * *_destArray,* uint16_t *_noOfDest,* WF_MessageId_t *_msgId,* bool *_noAck* = false) [pure virtual]

Send message indicated by the MessageId to a set of neighbors indicated by destArray, with a given payload size.

Semantic Behavior: This is the primary method the Framework uses to request that a message be transmitted by the waveform. The first two parameters are self-explanatory. The third parameter is an array of one or more neighbors to which the message is to be sent, each of which are identified by the neighbor's waveform address. The Framework may use the given MessageId to cancel or replace the payload in future . The parameter _noAck specifies whether a notification is requested when this message has reached its intended destination(s). A true value for this field means the Framework does not need notifications for this message. If data notifications/acknowledgements are not supported by the waveform, this parameter may be ignored.

Data Notifications: It is recommended that Waveforms support acknowledgements from destinations for unicast and multi-cast (in local neighborhood) messages. However this is optional. If the waveform choose to implement receiver acks, it should set the 'destReceiveAckSupport' field in the Attribute record as true whenever it responds to the AttributeRequest. If the waveform does not support this feature, then the framework will implement destination acknowledgements by replying to the source node on receiving messages from the waveform. The MessageId may be used by the Framework to cancel or replace the payload in future. The waveform is expected to generate at least 2 notifications through the Data Notification Event, for every message sent to it, the first indicating that the waveform has received the message and the second indicating that it has sent out the message. The waveform can generate a third notification indicating that the destination(s) has received the message if it chooses to support destination acknowledgement.

Performance: A request to send to multiple nodes should be fulfilled via a technique that is no less efficient than a sequence of transmissions to each node. The intent is that more advanced waveforms may internally optimize beyond this base performance level, by combining transmissions when possible and not unduly difficult. Any optimizations that: 1) are not naturally supported by the hardware and the waveform, 2) require non-local information, or 3) consume an order of magnitude more computational resources than required for the core send operation, should not be supported

Parameters

msg	Reference to the packet be sent.
payloadSize	Size of the payload in the packet.
destArray	Array of destinations to which the packet should be sent.
noOfDest	Number of destinations in the destArray.
msgId	Specifies the packet id the framework will use to identify this packet in future calls.
noAck	Specifies if acks should not be generated. Defaults option generates acks

Returns

void.

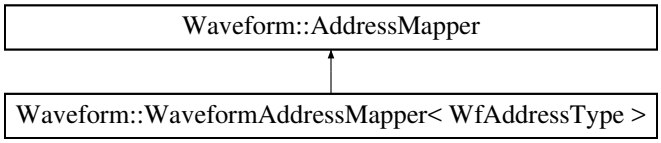
Implemented in Waveform::Waveform_64b_I.

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

- TuscaroraFW/Include/Interfaces/Waveform/WF_I.h

6.70 Waveform::WaveformAddressMapper< WfAddressType > Class Template Reference

Inheritance diagram for Waveform::WaveformAddressMapper< WfAddressType >:



Static Public Member Functions

- static WfAddressType **GetWaveformAdress** (NodeId_t id)
- static NodeId_t **GetFrameworkId** (WfAddressType addr)

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

- TuscaroraFW/Include/Interfaces/Waveform/AddressMapper.h

6.71 Waveform::WaveformAddressMapper< uint64_t > Class Template Reference

Static Public Member Functions

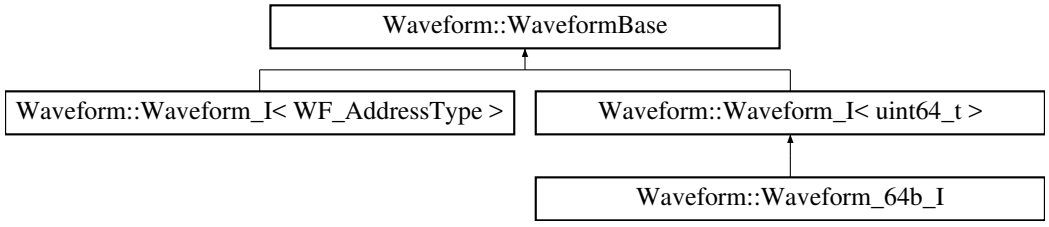
- static uint64_t **GetWaveformAdress** (NodeId_t id)
- static NodeId_t **GetFrameworkId** (uint64_t addr)

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

- TuscaroraFW/Include/Interfaces/Waveform/AddressMapper.h

6.72 Waveform::WaveformBase Class Reference

Inheritance diagram for Waveform::WaveformBase:



Public Member Functions

- virtual void ReplacePayloadRequest (RequestId_t rId, WF_MessageId_t msgId, uint8_t *payload, uint16_t payloadSize)=0

  Requests the replacement the payload of a message, passed in an earlier call to 'SendData' or 'BroadcastData' method with the given MessageId. The previous payload will be discarded.
- virtual void AttributesRequest (RequestId_t rId)=0

  Request the attributes of the waveform.
- virtual void ControlStatusRequest (RequestId_t rId)=0

  Requests the control status of waveform. A WF_StatusE enum type is sent as the data of the response message, to indicate the status of the variable.
- virtual void DataStatusRequest (RequestId_t rId, WF_MessageId_t mId)=0

  Requests the status of a data message already sent to the waveform.
- virtual void SetScheduleRequest (RequestId_t rId, NodeId_t nodeId, WF_ScheduleTypeE type, ScheduleBaseI &schedule, uint16_t scheduleSize)

  Sends information about a schedule for one of 3 kinds of functions: link estmation, sending or receiving to the waveform.
- virtual void **ScheduleRequest** (RequestId_t rId, NodeId_t nodeid, WF_ScheduleTypeE type)

6.72.1 Member Function Documentation

6.72.1.1 virtual void Waveform::WaveformBase::AttributesRequest (RequestId_t *rId*) [pure virtual]

Request the attributes of the waveform.

Semantic Behavior: Waveform should return its attributes to the Framework by generating a response through the Control Response Event, with the same request ID. The status of the response message should be true, unless otherwise the attributes are not available to be sent to the Framework. The response contains attributes in the WF_Attributes structure, which is copied into the data portion of the WF_ControlResponseParam.

Parameters

rId	Specifies the ID of the request, which the waveform will use to in its response message.

Returns

void.

Implemented in Waveform::Waveform_64b_I.

6.72.1.2 virtual void Waveform::WaveformBase::ControlStatusRequest (RequestId_t *rId*) [pure virtual]

Requests the control status of waveform. A WF_StatusE enum type is sent as the data of the response message, to indicate the status of the variable.

Semantic Behavior: The Framework expects the waveform to be in one of the data plane states defined by the WF_ControlP_StatusE enum. The waveform should return the current status of its data plane by generating a response through the Control Response Event, with the same RequestId. The status that response message should be true. The actual data plane status should be copied into the data part of the response

Parameters

rId	Specifies the ID of the request, which the waveform will use to in its response message.

Returns

void.

Implemented in Waveform::Waveform_64b_I.

6.72.1.3 virtual void Waveform::WaveformBase::DataStatusRequest (RequestId_t *rId,* WF_MessageId_t *mId*) [pure virtual]

Requests the status of a data message already sent to the waveform.

Semantic Behavior: If the Framework is uncertain about the status of a message that it sent to the waveform, it can send the DataStatusRequest command. A response message should be generated through the Control Response Event, with the same RequestId, that should have the status set as 'true' if the waveform received the particular message and a WF_DataNotifierParam object must be created that reflects the current status of the message and should be copied into the data part of the WF_ControlResponseParam.

Parameters

rId	Specifies the ID of the request, which the waveform will use to in its response message.
mId	Specified the ID of the message, for which status is requested.

Returns

void.

Implemented in Waveform::Waveform_64b_I.

6.72.1.4 virtual void Waveform::WaveformBase::ReplacePayloadRequest (RequestId_t *rId,* WF_MessageId_t *msgId,* uint8_t * *payload,* uint16_t *payloadSize*) [pure virtual]

Requests the replacement the payload of a message, passed in an earlier call to 'SendData' or 'BroadcastData' method with the given MessageId. The previous payload will be discarded.

Semantic Behavior: Waveform should replace the payload of a message passed in an earlier call to 'SendData' or 'BroadcastData' method with the given message ID, if the message has not been sent out. A response message should be generated through the Control Response Event, with the same request ID, that should include a status which will be set to false if the message has been sent out or if the waveform is unable to replace the payload of a message. Otherwise, the status should be set to true in the response message.

Parameters

rId	Specifies the ID of the request, which the waveform will use to in its response message.
msgId	specifies the message ID of the packet.
payload	pointer to the new payload.
payloadSize	size of the new payload.

Returns

void.

Implemented in Waveform::Waveform_64b_I.

6.72.1.5 virtual void Waveform::WaveformBase::SetScheduleRequest (RequestId_t *rId,* NodeId_t *nodeId,* WF_ScheduleTypeE *type,* ScheduleBaseI & *schedule,* uint16_t *scheduleSize*) [inline], [virtual]

Sends information about a schedule for one of 3 kinds of functions: link estmation, sending or receiving to the waveform.

Semantic Behavior: This method enables the Framework to pass on some scheduling information to the waveform. The expectation is that this method will be implemented only if the waveform needs scheduling information from the Framework or an external third party provider. The method enables receiving scheduling information specific to a neighbor specified by the _nodeId parameter. The schedule information is sent as abstract schedule object represented by the _schedule object and the size of this object is specified by the _scheduleSize parameter. The expectation is that, if a waveform needs scheduling information, it will derive from the ScheduleBaseI class and will implement its own scheduling class. The Framework defines three types of waveform schedules as the WF_↩ ScheduleTypeE enum (one each for link estimation, sending, and receiving messages).

The waveform should regenerate a response to this request through the Control Response Event, with the same RequestId. The status of that response message should be true if the waveform able to successfully use the scheduler information or false otherwise.

Since this is an optional feature for waveforms, if it choose to implement it, the waveform should set the scheduler↩ InfoSupport field in its Attribute as true whenever it responds to the AttributeRequest.

Parameters

rId	Specifies the ID of the request, which the waveform will use to in its response message.
nodeid	Specifies the node ID for which the schedule should be used.
type	Enum that specifies the type of the schedule. There are 3 types of schedules, sending, receiving and link estimation.
schedule	Reference to an abstract schedule object. It is expected that this class will be derived and implemented by different waveforms.
scheduleSize	Size of the schedule object.

Returns

void.

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

- TuscaroraFW/Include/Interfaces/Waveform/WF_I.h

6.73 Waveform::WF_AddDestinationResponse_Param Struct Reference

Public Attributes

- WF_MessageId_t **msgId**

- bool **status**
- uint64_t **destArray** [MAX_DEST]
- bool **add_status** [MAX_DEST]
- uint16_t **noOfDest**

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

- TuscaroraFW/Include/Interfaces/Waveform/WF_Event.h

6.74 Waveform::WF_Attributes Struct Reference

`#include <WF_Types.h>`

Public Attributes

- WaveformId_t **wfId**
- WF_TypeE type

  Waveform Ids given by the framework.
- WF_EstimatorTypeE estType

  Type of waveform.
- WF_AntennaTypeE antType

  Type of estimator provided by the waveform.
- WF_ModeE mode

  Type of antenna used by the waveform.
- bool broadcastSupport

  Current mode of the waveform.
- bool destReceiveAckSupport

  Does waveform support broadcastSupport.
- bool scheduleInfoSupport

  Does waveform support acknowlegements from receiver.
- bool senderTimeStampSupport

  Does waveform expose its scheduling information to framework.
- bool **receiverTimeStampSupport**
- uint16_t **ifindex**
- char name [32]

  Waveform OS/Platform interfce index number.
- uint16_t ackTimerPeriod

  Name of waveform.
- uint16_t **headerSize**
- uint16_t maxPayloadSize

  Size of the waveform header in bytes.
- uint16_t maxPacketSize

  Maximum payload that can be sent on the waveform in bytes.
- uint16_t minPacketSize

  Maximum packet size that can be sent on the waveform (in bytes), including the waveform header.
- uint32_t minInterPacketDelay

  Minimum packet size that can be sent on the waveform (in bytes), including the waveform header.
- uint32_t channelRate

Minimum interpacket rate in micro seconds.

- uint32_t maxBurstRate

  The channel reate in bytes per second.

- UFixed_2_8_t energy

  Maximum data burst rate in bytes per second.

- Fixed_14_16_t cost

  Energy in joules per bit.

6.74.1 Detailed Description

Structure for information about the waveform. The waveform will create an instance of it and populate it with information which can be querried by framework

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

- TuscaroraFW/Include/Interfaces/Waveform/WF_Types.h

6.75 Waveform::WF_CancelDataResponse_Param Struct Reference

Public Attributes

- WF_MessageId_t **msgId**
- bool **status**
- uint64_t **destArray** [16]
- bool **cancel_status** [16]
- uint16_t **noOfDest**

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

- TuscaroraFW/Include/Interfaces/Waveform/WF_Event.h

6.76 Waveform::WF_ControlResponseParam Struct Reference

Public Attributes

- WaveformId_t **wfid**
- RequestId_t **id**
- WF_ControlResponseE **type**
- uint32_t **dataSize**
- void * **data**

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

- TuscaroraFW/Include/Interfaces/Waveform/WF_Types.h

6.77 Waveform::WF_DataStatusParam< WF_AddressType > Struct Template Reference

`#include <WF_MessageT.h>`

Public Member Functions

- **WF_DataStatusParam** (FMessageId_t _msgId, WaveformId_t _wfId)

Public Attributes

- WaveformId_t **wfId**
- WF_MessageId_t **wfMsgId**
- uint8_t **noOfDest**
- bool **readyToReceive**
- Core::Dataflow::WF_DataStatusTypeE **statusType** [MAX_DEST]
- WF_MessageStatusE **statusValue** [MAX_DEST]
- WF_AddressType **destArray** [MAX_DEST]

6.77.1 Detailed Description

template<class WF_AddressType>
struct Waveform::WF_DataStatusParam< WF_AddressType >

Structure for the parameter passed to the acknowledgement of the packets. The framework will implement this callback which will be called by waveform the acknowledge packets sent to it by framework

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

- TuscaroraFW/Include/Interfaces/Waveform/WF_MessageT.h

6.78 Waveform::WF_Event< evtName, Param0 > Class Template Reference

`#include <WF_Event.h>`

Public Member Functions

- **WF_Event** (WaveformId_t id)
- bool **Invoke** (Param0 param0)

6.78.1 Detailed Description

template<WF_EventNameE evtName, typename Param0>
class Waveform::WF_Event< evtName, Param0 >

Received Message Event

Invoke Method Definition

bool WF_RcvMessageEvent_t::Invoke (WF_RecvMsgParam param);

The Received Message Event class has a single public method called Invoke, which can be use do send the event to the framework, with WF_RecvMsgParam as the parameter.

Expected Behavior:

When the waveform receives a valid data message through its radio interface, the waveform should event the Framework using the “Receive Message Event". Upon receipt of a message, waveform will evaluate metadata (RSSI, SINR, Receive Timestamp) associated with the message and the metadata will be added to the packet structure and passed in the parameter to the event. An operating mode wherein the waveform forwards up to the Framework each and every packets that it receives and decodes correctly irrespective of whether the packet was destined for the node is commonly referred to as “promiscuous mode". The API does not take a position on whether or not waveforms should support promiscuous mode. (Hence, no API is currently defined to configure such a mode). If a future programmatic decision is made to support promiscuous mode of waveforms, a policy manager (through the Framework) will explicitly configure the waveform to operate in such a mode. Unless explicitly configured a waveform should not send packets not intended for the node to the Framework. Note to Pattern designers: If a waveform does send packets to the Framework that are not meant for a particular node, the Framework has no way of identifying such packets. The Framework cannot guarantee to a Pattern that it will not receive such “unintentional” packets. If a Pattern does receive such packets they would have originated from the same Pattern instance on a neighboring node, but are not intended to be received on the current node (i.e., right pattern, wrong node). We recommend that Pattern designers consider the possibility of receiving some unintentional packets and make their patterns robust enough to handle them.

Event creation and invocation:

The details of the WF_RecvMsgParam parameter, and the code for creation and invocation of the Received Message are shown below. The waveform provider creates the event object and the parameter object. Further, it initializes the parameter object with the received data message, the rcvMsgId is assigned a MessageId generated by the waveform for the messages received by it. Once the parameter is initialized, the waveform can simply invoke the event with the parameter. The event class will convert this into a message of the appropriate format and send it using the shim layer to the Framework, where it will be routed to the particular module that will process the received waveform data message.

Data Notification Event

Invoke Method Definition

bool WF_DataNotifierEvent_t::Invoke (WF_DataNotifierParam param); The Data Notification class has a single public method called Invoke, which can be use do send the event to the framework, with WF_DataNotifierParam as the parameter.

Expected Behavior:

When a 'SendData' or 'BroadcastData' command is received by the Waveform, the Framework expects a Data Notification Event in response. The ackType in the parameter of the Data Notification Event indicates the type of notification. For each SendData command received with a particular MessageId, the waveform should send back at least two events, one of the ACK_WF_RECV type, indicating that the waveform received the message successfully and one of the ACK_WF_SENT type, indicating the waveform sent the message out through its radio. A third type named ACK_DST_RECV can also be sent if the waveform supports acks from the destination. For the Broadcast↩ Data command, the first two types are required and the third is not required. The 'dest' field in the param is an array that determines for which destinations that notification is being sent. For example, if the Waveform chooses to send multiple physical messages corresponding to the same "MessageId" once for each destination, then a notification event can be sent once per destination or it can all be grouped together and sent as a single notification.

Event Parameters:

The parameter creation and invocation is quite similar to that of the other events. The waveform needs to create an object of WF_DataNotifierEvent_t, create an object of the WF_DataNotifierParam and invoke the event with the param. The parameter for this event is show below.

Link Estimation Event

Invoke Method Definition

bool WF_LinkEstimateEvent_t::Invoke (WF_LinkEstimateParam param); The Link Estimation Event class has a single public method called Invoke, which can be use do send the event to the framework, with WF_LinkEstimate↩ Param as the parameter.

Expected Behavior:

Link estimation of known neighbors is an optional function that the waveforms can choose to provide. If supported, the waveform should set the 'estType' field in its Attribute to WF_EST_FULL whenever it responds to the Attribute↩ Request. Waveform that provide link estimation should provide link estimation events, and follow a standard format. The Framework defines a link metric structure that the waveform should compute for each neighbor. The WF_↩ LinkMetrics structure is show below, it consist of four metrics that are defined as follows:

1. Quality: Abstract quality metric expressed as a real number between [0,1].
2. Data rate: Link data rate expressed as log_2(bps).
3. Average Latency: Average latency in sending a packet to the destination expressed as log_2(seconds).
4. Energy: Average energy to transmit a packet expressed as log_2(pica-joules).

The waveform is expected to generate a event for each neighbor whenever one of the three events happens:

1. New Neighbor: Whenever the waveform detects a new neighbor and has an initial estimate of its link metrics. It uses the value NBR_NEW for the 'changeType' field of the event parameter.
2. Dead Neighbor: Whenever a previous neighbor no longer exist in the neighbor. Use the value NBR_DEAD for the 'changeType' field of the event parameter
3. Change in Metrics: Whenever the metrics has changed enough to notify the Framework. It uses the value NBR_UPDATE for the 'changeType' field of the event parameter.

Event Parameters:

The WF_LinkEstimateParam structure (shown above) is used as the parameter for the Link Estimation Event. The structure contains the waveform address, the link type, its link metrics and the type of change in neighborhood being conveyed by the message.

Control Response Event

Invoke Method Definition

bool WF_ControlResponseEvent_t::Invoke (WF_ControlResponseParam param); The Control Response Event class has a single public method called Invoke, which can be use do send the event to the framework, with WF_↩ ControlResponseParam as the parameter.

Expected Behavior:

Whenever a waveform receives any request command, i.e., a command whose name has the suffix Request, it should use the Control Response Event to send a response. The response should use the same “RequestId” as in the request command. The Framework tracks the RequestId that it uses for each waveform and generates a new RequestId for every request command. If the Framework fails to receive a response for a request, the Framework may resend the request using the same RequestId.

Event Parameters:

The WF_ControlResponseParam parameter associated with the Control Response is shown below. The key field is ‘type’, which is an enum of type WF_ControlResponseE. This enumerator defines a response for each of the Request commands. The waveform should set this type appropriately based on the command it is responding too. The parameter also contains a generic ‘data’ field, whose interpretation by the Framework depends upon on the type of the response. Based on the type of Request command that is being responded to, an appropriate structure is copied into the ‘data’ field, and the number of bytes sent through the data field should be set appropriately in the dataSize field.

Schedule Update Event

Invoke Method Definition

bool WF_ScheduleUpdateEvent_t::Invoke (WF_ScheduleUpdateParam param); The Schedule Update Event class has a single public method called Invoke, which can be use do send the event to the framework, with WF_↩ ScheduleUpdateParam as the parameter.

Expected Behavior:

Schedule updates are an optional feature by which waveforms that maintain schedules for functions such as sending, receiving or link estimation can share information about the schedules with the Framework. If the feature is supported, the waveform should set as true the ‘schedulerInfoSupport’ field in its Attribute whenever it responds to the AttributeRequest. This event is expected to be implemented by waveforms that have timing characteristics or TDMA schedules. The eventing should be implemented if the knowledge of a waveform's slot schedules will help the Framework in improving the overall data efficiency. Note: If the waveform is responding to the ‘ScheduleRequest’ command, then it should use the Control Response Event; and, if it is sending ongoing schedule updates, it should use the Schedule Updates Event.

Event Parameter

The WF_ScheduleUpdateParam structure that is used as the parameter of the Schedule Update Event is shown below. The parameter contains information about one of three types of schedules, send, receive or link estimation, which is indicated by setting the 'type' field appropriately. The 'data' field is a pointer of type ScheduleBaseI which points to the object that contains the actual schedule information for the neighbor indicated by 'nodeid'. The dataSize field contains the size of the actual schedule object that is pointed to by the data field.

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

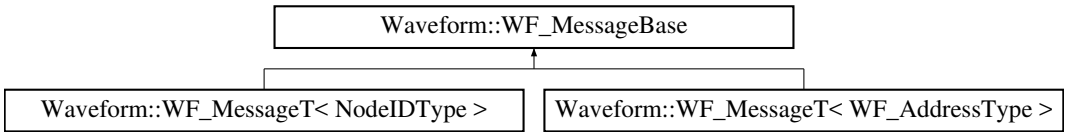
- TuscaroraFW/Include/Interfaces/Waveform/WF_Event.h

6.79 Waveform::WF_MessageBase Class Reference

Base class for all waveform messages.

`#include <WF_MessageT.h>`

Inheritance diagram for Waveform::WF_MessageBase:



Public Member Functions

- **WF_MessageBase** (uint16_t payload_size)
- WF_MessageId_t **GetWaveformMessageID** ()
- void **SetWaveformMessageID** (WF_MessageId_t _id)
- void **SetType** (MessageTypeE _type)
- MessageTypeE **GetType** ()
- void **SetSrcNodeID** (NodeId_t _node)
- NodeId_t **GetSrcNodeID** ()
- void **SetInstance** (uint16_t _inst)
- uint16_t **GetInstance** ()
- bool **IsFlagSet** (WF_MessageFlagsE _flag)
- void **SetFlag** (WF_MessageFlagsE _flag)
- void **ClearFlag** (WF_MessageFlagsE _flag)
- void **SetWaveform** (WaveformId_t wfId)
- WaveformId_t **GetWaveform** ()
- uint16_t **GetPayloadSize** ()
- void **SetPayloadSize** (uint16_t _size)
- uint8_t * **GetPayload** ()
- void **SetPayload** (uint8_t *_payload)
- PacketMetadata **GetMetaData** ()
- void **SetMetaData** (PacketMetadata _metadata)
- U64NanoTime **GetRecvTimestamp** ()
- void **SetRecvTimestamp** (U64NanoTime _time)

Public Attributes

- uint16_t **inst**
- uint8_t **flags**
- NodeId_t **nodeID**
- uint16_t **payloadSize**
- WaveformId_t **waveformId**
- MessageTypeE **type**
- PacketMetadata **metadata**
- uint8_t * **payload**

6.79.1 Detailed Description

Base class for all waveform messages.

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

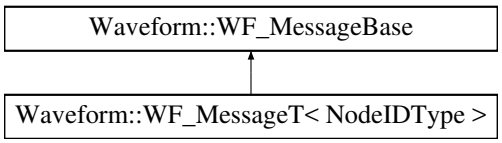
- TuscaroraFW/Include/Interfaces/Waveform/WF_MessageT.h

6.80 Waveform::WF_MessageT< NodeIDType > Class Template Reference

A class that facilitates data exchange between the Waveform and the framework.

`#include <WF_MessageT.h>`

Inheritance diagram for Waveform::WF_MessageT< NodeIDType >:



Public Member Functions

- bool **IsValid** ()
- bool **IsLocked** ()
- bool **Lock** ()
- bool **Unlock** ()
- void **SetValid** (bool value)
- **WF_MessageT** (uint16_t _payloadSize)
- uint8_t **GetNumberOfDest** ()
- void **SetNumberOfDest** (uint8_t _noOfDest)
- NodeIDType **GetSource** ()
- void **SetSource** (NodeIDType _src)
- void **CopyFrom** (FMessage_t &_msg)
- void **CopyIthSegmentFrom** (FMessage_t &_msg, uint8_t segment_number, uint16_t max_segment_size)
- void **CopyTo** (FMessage_t &_msg)

Static Public Member Functions

- static uint16_t **GetHeaderSize** ()

Additional Inherited Members

6.80.1 Detailed Description

template<class NodeIDType>
class Waveform::WF_MessageT< NodeIDType >

A class that facilitates data exchange between the Waveform and the framework.

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

- TuscaroraFW/Include/Interfaces/Waveform/WF_MessageT.h

6.81 Waveform::WF_RecvMsgParam< WF_AddressType > Struct Template Reference

Public Attributes

- FMessageId_t **wrn**
- WF_MessageT< WF_AddressType > * **msg**

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

- TuscaroraFW/Include/Interfaces/Waveform/WF_Types.h

6.82 Waveform::WF_ReplacePayloadResponse_Param Struct Reference

Public Attributes

- WF_MessageId_t **msgId**
- bool **status**
- uint64_t **destArray** [16]
- bool **replace_status** [16]
- uint16_t **noOfDest**

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

- TuscaroraFW/Include/Interfaces/Waveform/WF_Event.h

6.83 Waveform::WF_ScheduleUpdateParam Struct Reference

Public Attributes

- NodeId_t **nodeid**
- ScheduleTypeE **type**
- ScheduleBaseI * **data**
- uint32_t **size**

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

- TuscaroraFW/Include/Interfaces/Waveform/WF_Types.h

Index