JTRS LOS Physical API Service Definition

V1.0
December 15, 2000

Prepared for the
Joint Tactical Radio System (JTRS) Joint Program Office

Prepared by the
Modular Software-programmable Radio Consortium
Under Contract No. DAAB15-00-3-0001

Revision Summary

1.0	Initial release

Table of Contents

1 INTRODUCTION 1
1.1 OVERVIEW 1
1.2 SERVICE LAYER DESCRIPTION 1
1.2.1 Non-Real-Time Services 1
1.2.2 Real-Time Services 2
1.3 MODES OF SERVICE 2
1.4 SERVICE STATES 2
1.5 REFERENCED DOCUMENTS 2
2 UUID 2
3 SERVICES 2
3.1 NON-REAL-TIME SERVICES 4
3.1.1 CommonUtilities 4
3.1.2 setMode Service 4
3.1.3 setXmit Mode Service 4
3.1.4 Transceiver Setup 4
3.1.5 MediaSetup 5
3.1.6 RadioMode 5
3.2 REAL-TIME SERVICES 6
3.2.1 Physical Layer To Receive 6
3.2.2 Transmit/Receive Packet and Flow Control 7
3.2.3 PacketSignals 8
4 SERVICE PRIMITIVES 9
4.1 COMMON UTILITIES 9
4.1.1 setMode Service 9
4.1.2 setXmitMode Service 9
4.2 TRANSCEIVER_SETUP 10
4.2.1 setUpReceiverParams 10
4.2.2 setUpTransmitterParams 10
4.2.3 setDate 11
4.3 MEDIA_SETUP 11
4.3.1 Set Up Media Type 11
4.4 RADIO_MODE 12
4.4.1 setMode 12
4.5 RECEIVE COMMAND 12
4.5.1 receive 12
4.6 TRANSMIT / RECEIVE PACKET AND FLOW CONTROL 13
4.6.1 pushPacket 13
4.6.2 spaceAvailable 14
4.6.3 enableFlowControlSignals 14
4.6.4 enableEmptySignals 15
4.6.5 setNumofPriorityQueues 15

5 ALLOWABLE SEQUENCE OF SERVICE PRIMITIVES. 17

6 PRECEDENCE OF SERVICE PRIMITIVES 17

7 SERVICE USER GUIDELINES. 17

8 SERVICE PROVIDER-SPECIFIC INFORMATION. 17

9 IDL 17

10 UML. 24

10.1 CONTROLLER RELATIONSHIP. 24
10.2 USER RELATIONSHIP. 29

List of Figures

Figure 1-1. JTRS Service Layers 1
Figure 3-1. Common Utilities 4
Figure 3-2. Transceiver Setup 4
Figure 3-3. Media Setup 5
Figure 3-4. Radio Mode 5
Figure 3-5. Radio Mode State Chart 6
Figure 3-6. Receive Command 6
Figure 3-7. Receive Packet Control Structure 7
Figure 3-8. UlongPacket 7
Figure 3-9. Packet Signals 8
Figure 10-1. Controller Diagram 24
Figure 10-2. Common Utilities Diagram 25
Figure 10-3. Transceiver Setup Diagram 26
Figure 10-4. Media Setup Diagram 27
Figure 10-5. Radio Mode Diagram 28
Figure 10-6. User Relationship Diagram 29
Figure 10-7. Receive Command Diagram 30
Figure 10-8. UlongPacket Diagram 31

List of Tables

Table 1. Cross-Reference of Services and Primitives 3

1 INTRODUCTION.

1.1 OVERVIEW.

The LOS Physical Real-Time API provides standardized interfaces to the Physical Layer for the LOS waveform. Physical Real-Time Services are grouped into Building Block classes and structures to define abstract services to foster software reuse and commonality between different Service Definition implementations. Physical Real-Time Services provide Service Users with methods to signal the Service User that an event has occurred. Location of the Physical Layer with respect to other JTRS layers is shown in the following diagram. The Physical real-time is the "**A**" interface into the Physical Layer.

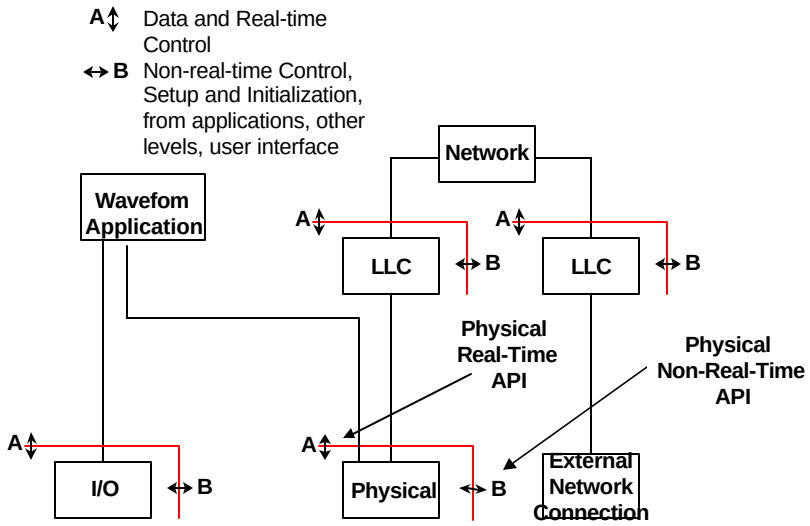

Figure 1-1. JTRS Service Layers

1.2 SERVICE LAYER DESCRIPTION.

The Service User for the Physical Real-Time API is the Logical Link layer. The Service Provider for the Physical Real-Time API is the Physical Layer. Table 1.0 defines this API's service groups and building blocks.

1.2.1 Non-Real-Time Services.

The non-real-time part of the Physical API is realized by instantiating the Physical non-real-time Building Block for LOS. The following Service Groups are included in the Physical Layer for non-real-time.

- Common Utilities
- Transceiver Setup

- Media Setup
- Radio Mode

Each Non-real-time Service Group is described in detail within Section 3.

1.2.2 Real-Time Services.

The real-time part of the Physical API is realized by instantiating the Physical real-time Building Block for LOS. The following Service Groups are included in the Physical Layer for real-time.

- Command Physical Layer To Receive
- Transmit / Receive Packet and Flow Control

Each Real-time Service Group is described in detail within Section 3.

1.3 MODES OF SERVICE.

There are no specific Modes of Service.

1.4 SERVICE STATES.

The API is used for all modes of operations except Initialization.

1.5 REFERENCED DOCUMENTS.

Document No.	Document Title
MSRC-5000SCA	Software Communications Architecture Specification
MSRC-5000API	Application Program Interface Supplement to the Software Communications Architecture Specification, Appendix C Generic Packet Building Block Service Definition

2 UUID.

The UUID for this API is `841df940-d1d3-11d4-8cc8-00104b23b8a2`.

3 SERVICES.

The LOS Physical interface is defined in terms of the services provided by the Service Provider, and the individual primitives that may flow between the Service User and Service Provider.

Table 1 defines the real-time and non-real-time service subsections.

Table 1. Cross-Reference of Services and Primitives

Service Group	Service	Primitives or Structure Attributes
CommonUtilities	Common Utilities	setMode(ChannelMode : in ChannelModeType) : boolean setXmitMode(PTTindicator : in boolean) : boolean
Transceiver_Setup	Set Up Receiver Parameters	SetUpReceiverParams(RecvParams : in RecvParams_Type) : boolean
	Set Up Transmitter Parameters	SetUpTransmitterParams(TransParams : in TransParams_Type) : boolean
	SetDate	SetDate(date : in DateType) : boolean
Media_Setup	Set Up Media Type	SetUpMediaType(MediaParams : in MediaParams_Type) : boolean
Radio_Mode	Set Radio Mode	SetMode(Mode : in RadioMode_Mode_Type) : boolean
Command Physical Layer to Receive	Receive Command	Receive(control : RcvPacketControl) : void
Transmit / Receive Packet and Flow Control	PacketSignals	SignalHighWatermark (priorityQueueID : in octet) : void signalLowWaterMark (priorityQueueID : in octet) : void signalEmpty() : void
	UlongPacket	pushPacket(priority : in octet, control : in LOS_API:: LOS::NullControl, payload : in LOS _API::PortTypes:: UlongSequence) : void spaceAvailable(priorityQueueID : in octet) : short enableFlowControlSignals(enable : in boolean) : void enableEmptySignal(enable : in boolean) : void setNumOfPriorityQueues(numOfPriorities : in octet) : void

3.1 NON-REAL-TIME SERVICES.

3.1.1 CommonUtilities.

This class sets up the transceiver channel mode. Refer to Figure 3-1.

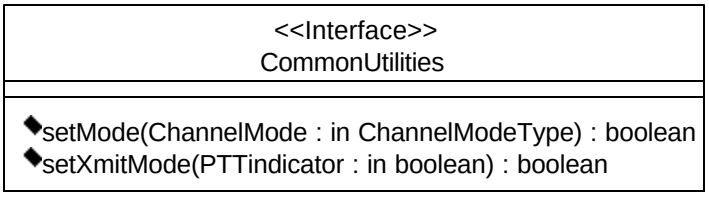

Figure 3-1. Common Utilities

3.1.2 setMode Service.

The setMode Service provides Service Users with a method to communicate to the Physical Layer the selected waveform.

3.1.3 setXmit Mode Service.

The setXmit Mode Service controls Push-To-Talk transmit activities.

3.1.4 Transceiver Setup.

This class sets up the transceiver parameters that are not modulation or media dependent. Refer to Figure 3-2.

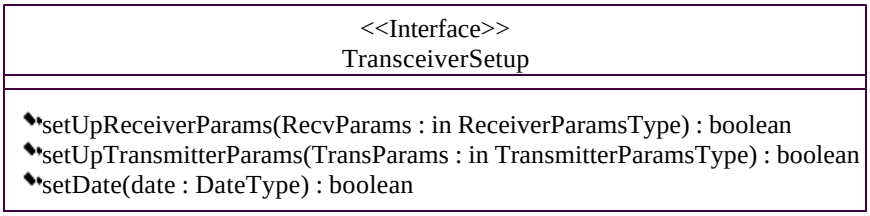

Figure 3-2. Transceiver Setup

3.1.4.1 setUpReceiveParams.

setUpReceiveParams sets the receive channel bandwidth to 25 kHz and sets the demodulator to AM.

3.1.4.2 setUpTransmitParams.

setUpTransmitParams sets the transmit channel bandwidth to 25 kHz and sets the modulator to AM.

3.1.4.3 setDate.

setDate sets the current day-of-year and year.

3.1.5 MediaSetup.

This service group provides the setup necessary for squelch control. Refer to Figure 3-3.

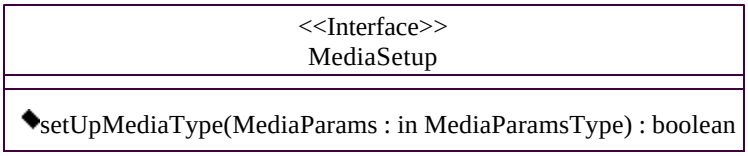

Figure 3-3. Media Setup

3.1.6 RadioMode.

This service group sets the transceiver mode to off, initialize, operational, or test. Refer to Figure 3-4.

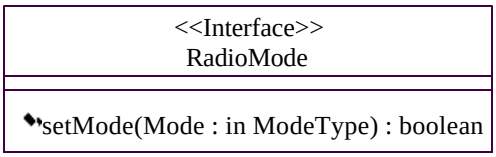

Figure 3-4. Radio Mode

These modes are defined as follows:

Off

means that the waveform is no longer active on the transceiver. Power may be turned off to the Physical Layer, if possible. This is typically a low power mode.

Operate

is normal operation, transmit and receive operations allowed.

Test

is a special mode of the radio to verify correct operation. Normal transmit and receive are disabled, but test transmit and receive can occur.

3.1.6.1 setMode.

setMode sets the Physical Layer to either standby, operate, test, or off according the state diagram shown below.

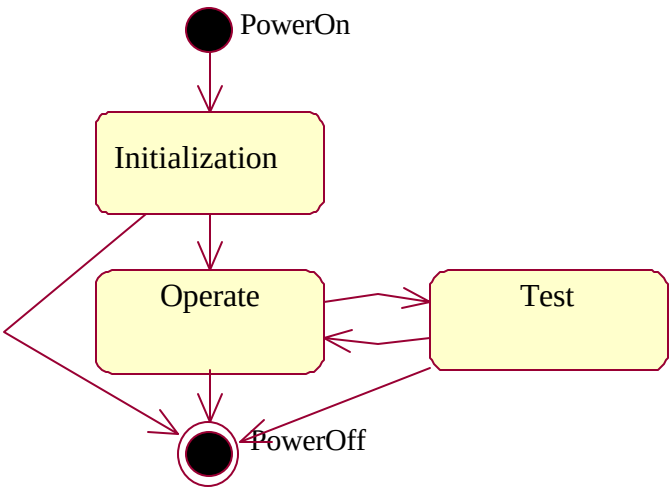

Figure 3-5. Radio Mode State Chart

3.2 REAL-TIME SERVICES.

Real-time control and data will be pushed upstream and downstream using control and data packets. The data packets include a control header for transferring real time control information with the data.

3.2.1 Physical Layer To Receive.

The Command Physical Layer to Receive service group consists of the Receive Command as shown in Figure 3-6.

3.2.1.1 ReceiveCommand.

The ReceiveCommand provides a method to command the Physical Layer to receive. The RcvPacket Control structure contains elements specific to the receive operation.

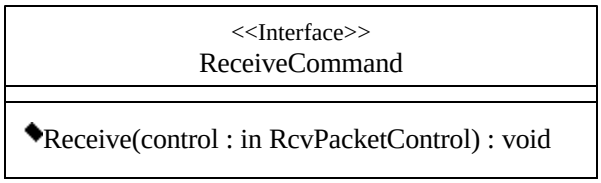

Figure 3-6. Receive Command

3.2.1.2 RcvPacketControl Structure.

This structure (Figure 3-7) can be used by pushPacket to communicate real time information to the Service Provider.

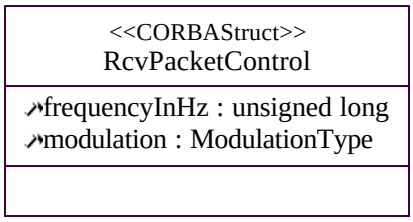

Figure 3-7. Receive Packet Control Structure

3.2.1.2.1 frequencyInHz.

The current operating frequency.

3.2.1.2.2 modulation.

Defined by the inheriting API.

3.2.2 Transmit/Receive Packet and Flow Control.

This service group consists of the Ulong Packet and PacketSignals.

3.2.2.1 UlongPacket.

The UlongPacket (Figure 3-8) provides a method for the Service User to push data to the Service Provider. The priority field in the pushPacket indicates the priority of the packet. The control field contains all the information necessary to transmit the packet.

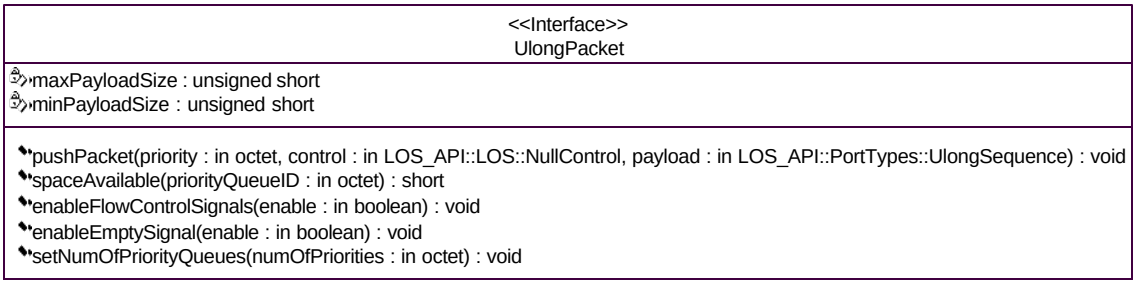

Figure 3-8. UlongPacket

3.2.2.2 pushPacket.

This operation is used to push Client data to the Server with a Control element and a Payload element.

3.2.2.3 spaceAvailable.

This operation shall be overridden and shall specify no functionality for HQ operations. There is no queuing mechanism implemented for this waveform.

3.2.2.4 enableFlowControlSignals.

This operation shall be overridden and shall specify no functionality for HQ operations. There is no queuing mechanism implemented for this waveform.

3.2.2.5 enableEmptySignal.

This operation shall be overridden and shall specify no functionality for HQ operations. There is no queuing mechanism implemented for this waveform.

3.2.2.6 setNumofPriorityQueues.

This operation shall be overridden and shall specify no functionality for HQ operations. There is no queuing mechanism implemented for this waveform.

3.2.3 PacketSignals.

The PacketSignals (Figure 3-9) provides a method for the Service Provider to give status to the Service User. After the Service Provider receives a pushPacket, it will provide the Service User with response.

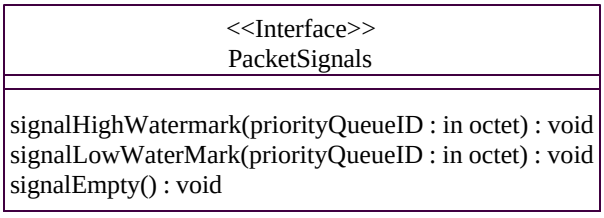

Figure 3-9. Packet Signals

4 SERVICE PRIMITIVES.

4.1 COMMON UTILITIES.

This primitive sets up the parameters in the transceiver related to the configuration of the radio channel.

4.1.1 setMode Service.

4.1.1.1 Synopsis.

setMode(ChannelMode : ChannelModeType) : boolean

4.1.1.2 Parameters.

ChannelModeType is a structure containing the parameters specific to configuration.

4.1.1.3 State.

This command is valid in all states.

4.1.1.4 New State.

This command does cause a state change.

4.1.1.5 Response.

True if mode set successfully, false otherwise.

4.1.1.6 Originator.

The service user.

4.1.1.7 Error/Exceptions.

None.

4.1.2 setXmitMode Service.

4.1.2.1 Synopsis.

setXmitMode(PTTindicator : in boolean) : boolean

4.1.2.2 Parameters.

PTTindicator is a boolean controlling the state of Push-to-Talk.

4.1.2.3 State.

This command is valid in all states.

4.1.2.4 New State.

This command does cause a state change.

4.1.2.5 Response.

True if mode set successfully, false otherwise.

4.1.2.6 Originator.

The service user.

4.1.2.7 Error/Exceptions.

None.

4.2 TRANSCEIVER_SETUP.

4.2.1 setUpReceiverParams.

This primitive sets up the parameters for the receiver function.

4.2.1.1 Synopsis.

boolean setUpReceiverParams(RecvParams :in ReceiverParamsType) : boolean

4.2.1.2 Parameters.

ReceiverParamsType is a structure containing parameters specific to Receive operations.

4.2.1.3 State.

This command is valid in all states.

4.2.1.4 New State.

This command does cause a state change.

4.2.1.5 Response.

TRUE if Receiver Parameters are set as specified in the call; FALSE otherwise.

4.2.1.6 Originator.

The service user.

4.2.1.7 Error/Exceptions.

None.

4.2.2 setUpTransmitterParams.

This primitive sets up the parameters for the transmitter function.

4.2.2.1 Synopsis.

boolean setUpTransmitterParams(TransParams :in TransmitterType) : boolean

4.2.2.2 Parameters.

TransmitterType is a structure containing parameters specific to transmit operations.

4.2.2.3 State.

This command is valid in all states.

4.2.2.4 New State.

This command does cause a state change.

4.2.2.5 Response.

TRUE if the Rx Antenna is set as specified in the call; FALSE otherwise.

4.2.2.6 Originator.

The service user.

4.2.2.7 Error/Exceptions.

None.

4.2.3 setDate.

This primitive sets up the parameters for the date in the DSP.

4.2.3.1 Synopsis.

boolean setDate(date :in DateType)

4.2.3.2 Parameters.

DateType is a structure containing parameters long year and long day of year.

4.2.3.3 State.

This command is valid in all states.

4.2.3.4 New State.

This command does cause a state change.

4.2.3.5 Response.

TRUE if the Rx Antenna is set as specified in the call; FALSE otherwise.

4.2.3.6 Originator.

The service user.

4.2.3.7 Error/Exceptions.

None.

4.3 MEDIA_SETUP.

4.3.1 Set Up Media Type.

This primitive sets up the parameters in the receiver related to squelch control.

4.3.1.1 Synopsis.

boolean setUpMediaType(MediaParams : in MediaParamsType)

4.3.1.2 Parameters.

MediaParamsType is a structure containing all the parameters specific to squelch control.

4.3.1.3 State.

This command is valid in all states.

4.3.1.4 New State.

This command does cause a state change.

4.3.1.5 Response.

TRUE if the Rx Antenna is set as specified in the call; FALSE otherwise.

4.3.1.6 Originator.

None.

4.3.1.7 Error/Exceptions.

None.

4.4 RADIO_MODE.

4.4.1 setMode.

This primitive sets the mode of the radio, e.g., Off, Standby, Operate, or Test.

4.4.1.1 Synopsis.

boolean setMode(Mode : in ModeType).

4.4.1.2 Parameters.

ModeType is an enumeration type listing all of the settable modes of the transceiver.

4.4.1.3 State.

This command operates in all states.

4.4.1.4 New State.

The state specified in the command.

4.4.1.5 Response.

TRUE if the CarrierThreshold is set; FALSE otherwise.

4.4.1.6 Originator.

This primitive is initiated by the Service User.

4.4.1.7 Errors/Exceptions.

None.

4.5 RECEIVE COMMAND.

4.5.1 receive.

receive provides the ability to command the Physical Real-Time Layer to receive data based on the control parameter.

4.5.1.1 Synopsis.

void receive(control : RcvPacketControl)

4.5.1.2 Parameters.

RcvPacketControl is a structure for Receiver Packet Control.

ModulationType is an Enum type listing all of the modulation types.

4.5.1.3 State.

Defined by the inheriting API.

4.5.1.4 New State.

Defined by the inheriting API.

4.5.1.5 Response.

No return.

4.5.1.6 Originator.

Service User.

4.5.1.7 Error/Exceptions.

Defined by the inheriting API.

4.6 TRANSMIT / RECEIVE PACKET AND FLOW CONTROL.

4.6.1 pushPacket.

This operation is used to push Client data to the Server with a Control element and a Payload element.

4.6.1.1 Synopsis.

void pushPacket(priority : in octet, control : in LOS_API::LOS::NullControl, payload : in LOS_API::PortTypes:: UlongSequence) : void

4.6.1.2 Parameters.

priority, control, payload.

4.6.1.2.1 priority.

priority is a single octet. This parameter is not used for a LOS implementation. It shall be set to 0 to indicate a Null priority.

4.6.1.2.2 control.

control is of type NullControl.

4.6.1.2.3 payload.

payload is of type payloadType. This parameter represents the sequence of the data stream to be pushed as a UlongSequence.

4.6.1.3 State.

TBD.

4.6.1.4 New State.

TBD.

4.6.1.5 Response.

TBD.

4.6.1.6 Originator.

Service User

4.6.1.7 Errors/Exceptions.

None.

4.6.2 spaceAvailable.

This operation shall be overridden and shall specify no functionality for LOS operations. There is no queuing mechanism implemented for this waveform.

4.6.2.1 Synopsis.

short spaceAvailable (priorityQueueId : in octet)

4.6.2.2 Parameters.

priorityQueueId is of type short.

4.6.2.3 State.

TBD.

4.6.2.4 New State.

TBD.

4.6.2.5 Response.

TBD.

4.6.2.6 Originator.

Service User

4.6.2.7 Errors/Exceptions.

None.

4.6.3 enableFlowControlSignals.

This operation shall be overridden and shall specify no functionality for LOS operations. There is no queuing mechanism implemented for this waveform.

4.6.3.1 Synopsis.

void enableFlowControlSignals (enable : in boolean)

4.6.3.2 Parameters.

enable is of type boolean. This parameter is not used for a LOS implementation.

4.6.3.3 State.

TBD.

4.6.3.4 New State.

TBD.

4.6.3.5 Response.

TBD.

4.6.3.6 Originator.

Service User

4.6.3.7 Errors/Exceptions.

None.

4.6.4 enableEmptySignals.

This operation shall be overridden and shall specify no functionality for LOS operations. There is no queuing mechanism implemented for this waveform.

4.6.4.1 Synopsis.

void enableEmptyControlSignals (enable : in boolean)

4.6.4.2 Parameters.

enable

enable is of type boolean. This parameter is not used for a LOS implementation.

4.6.4.3 State.

TBD.

4.6.4.4 New State.

TBD.

4.6.4.5 Response.

TBD.

4.6.4.6 Originator.

Service User

4.6.4.7 Errors/Exceptions.

None.

4.6.5 setNumofPriorityQueues.

This operation shall be overridden and shall specify no functionality for LOS operations. There is no queuing mechanism implemented for this waveform.

4.6.5.1 Synopsis.

void setNumofPriorityQueues (enable : in boolean)

4.6.5.2 Parameters.

enable

enable is of type boolean. This parameter is not used for a LOS implementation.

4.6.5.3 State.

TBD.

4.6.5.4 New State.

TBD.

4.6.5.5 Response.

TBD.

4.6.5.6 Originator.

Service User.

4.6.5.7 Errors/Exceptions.

None.

5 ALLOWABLE SEQUENCE OF SERVICE PRIMITIVES.

Intentionally left blank.

6 PRECEDENCE OF SERVICE PRIMITIVES.

There is no precedence of primitives.

7 SERVICE USER GUIDELINES.

There are no Service User Guidelines for the LOS Physical API provided.

8 SERVICE PROVIDER-SPECIFIC INFORMATION.

Not applicable.

9 IDL.

The LOS interface design depicted in IDL source code is shown below.

```
//Source file: c:/program files/devstudio/vc/atl/include/LOS.idl

#ifndef __LOS_DEFINED
#define __LOS_DEFINED

/* CmIdentification
  %X% %Q% %Z% %W% */

module LOS {

      /* Control of packet for los with nothing in it. */

      struct NullControl {
      };

      module Physical  {

            enum ModeType {
                  Off,              /* AM_Stop_Msg #4
                                       FM_Stop_Msg #5 */
                  Operate,          /* AM_Start_Msg #1
                                       FM_Start_Msg #2 */
                  Test              /* DSP_Run_BIT_Msg #34 */
            };

            interface RadioMode {
                  /*
                  @roseuid 39F463E40245 */
                  boolean setMode (
                        in ModeType Mode
                        );

            };
```

```
enum squelchType {
      AGC,
      NOISE
};

struct MediaParamsType {
      /* Gen_Set_Squelch_Msg #20
         Gen_AGC_Squelch_Msg #87 */
      unsigned long squelchLevelOffToOn;
      squelchType squelchType;
};

interface MediaSetup {
      /*
      @roseuid 39F464AE033B */
      boolean setUpMediaType (
            in MediaParamsType MediaParams
            );

};

/* Provide a pushPacket with a octet payload */

interface OctetPacket {
      /* The maxPacketSize is a read only attribute set by the
      Packet Server and the get operation reports back the
      maximum number of traffic units allowed in one pushPacket
      call. */

      attribute unsigned short maxPayloadSize;
      attribute unsigned short minPayloadSize;

      /* This operation is used to push Client data to the Server
      with a Control element and a Payload element.
      @roseuid 39EF48840287 */
      void pushPacket (
            in octet priority,
            in LOS::NullControl control,
            in CF::OctetSequence payload
            );

      /* The operation returns the space available in the Servers
      queue(s) in terms of the implementers defined Traffic
      Units.
      @roseuid 39EF4884029A */
      short spaceAvailable (
            in octet priorityQueueID
            );

      /* This operation allows the client to turn the High
      Watermark Signal ON and OFF.
      @roseuid 39EF488402A5 */
      void enableFlowControlSignals (
            in boolean enable
            );
```

```
        /* This operation allows the client to turn theEmpty Signal
        ON and OFF.
        @roseuid 39EF488402AE */
        void enableEmptySignal (
              in boolean enable
              );

        /*
        @roseuid 39EF488402B0 */
        void setNumOfPriorityQueues (
              in octet numOfPriorities
              );

};

/* Provide a pushPacket with an unsigned long payload */

interface UlongPacket {
        attribute unsigned short minPayloadSize;
        /* The maxPacketSize is a read only attribute set by the
        Packet Server and the get operation reports back the
        maximum number of traffic units allowed in one pushPacket
        call. */

        attribute unsigned short maxPayloadSize;

        /* This operation is used to push Client data to the Server
        with a Control element and a Payload element.
        @roseuid 39EF491B0107 */
        void pushPacket (
              in octet priority,
              in LOS::NullControl control,
              in PortTypes::UlongSequence payload
              );

        /* The operation returns the space available in the Servers
        queue(s) in terms of the implementers defined Traffic
        Units.
        @roseuid 39EF491B0113 */
        short spaceAvailable (
              in octet priorityQueueID
              );

        /* This operation allows the client to turn the High
        Watermark Signal ON and OFF.
        @roseuid 39EF491B011B */
        void enableFlowControlSignals (
              in boolean enable
              );

        /* This operation allows the client to turn theEmpty Signal
        ON and OFF.
        @roseuid 39EF491B011D */
        void enableEmptySignal (
              in boolean enable
              );
```

```
        /*
        @roseuid 39EF491B0126 */
        void setNumOfPriorityQueues (
              in octet numOfPriorities
              );

};

/* Provide a pushPacket with a unsigned short payload */

interface UshortPacket {
        /* The maxPacketSize is a read only attribute set by the
        Packet Server and the get operation reports back the
        maximum number of traffic units allowed in one pushPacket
        call. */

        attribute unsigned short maxPayloadSize;
        attribute unsigned short minPayloadSize;

        /* This operation is used to push Client data to the Server
        with a Control element and a Payload element.
        @roseuid 39EF46FA018F */
        void pushPacket (
              in octet priority,
              in LOS::NullControl control,
              in PortTypes::UshortSequence payload
              );

        /* The operation returns the space available in the Servers
        queue(s) in terms of the implementers defined Traffic
        Units.
        @roseuid 39EF46FA01A6 */
        short spaceAvailable (
              in octet priorityQueueID
              );

        /* This operation allows the client to turn the High
        Watermark Signal ON and OFF.
        @roseuid 39EF46FA01AE */
        void enableFlowControlSignals (
              in boolean enable
              );

        /* This operation allows the client to turn theEmpty Signal
        ON and OFF.
        @roseuid 39EF46FA01B8 */
        void enableEmptySignal (
              in boolean enable
              );

        /*
        @roseuid 39EF46FA01BA */
        void setNumOfPriorityQueues (
              in octet numOfPriorities
              );

};
```

```
interface User : UlongPacket, PacketBB::PacketSignals {
};

/* LOS modulation type */

enum ModulationType {
      FM,
      AM
};

struct RcvPacketControl {
      /* The frequency to tune for the duration of the hop.. */
      unsigned long frequencyInHz;
      /* Enumerated value that specifies the type of demodulation
      or modulation to use .  E.g. AM, FM, FSK, etc. */
      ModulationType modulation;
};

enum DataRateType {
      D2400,
      D4800,
      D16000,
      D300,
      D2400N,
      D75,
      D150,
      D4800N,
      D9600N,
      DOFF,
      TF,
      D1200,
      D1200N,
      RS232-9600,
      RS232-14400,
      RS232-38800,
      D600,
      PKT
};

enum OperationModeType {
      None,
      LNE_Off,
      LNE_On
};

enum COMSECType {
      PT,               /* Gen_Plain_Text_Mode_Msg #32 */
      CT                /* Gen_Cipher_ Text_Mode_Msg #31 */
};

enum DataModeConfigType {
      Audio,            /* Gen_Audio_Mode_Msg #27 */
      Data              /* Gen_Data_Mode_Msg #28 */
};

/* Gen_Channel_Num #86 */
```

```
struct ChannelModeType {
      DataRateType DataRate;
      COMSECType COMSEC;
      OperationModeType OperationMode;
      DataModeConfigType DataModeConfig;
      short ModemChannelNumber;
      boolean transmitInhibit;
};

interface CommonUtilities {
      /*
      @roseuid 3A0C079200DE */
      boolean setMode (
            in ChannelModeType ChannelMode
            );

      /*
      @roseuid 3A0C1839008E */
      boolean setXmitMode (
            in boolean PTTindicator //Gen_PTT_Msg #76
            );

};

/* DSP_Capabilities_Msg #33 */

struct CapabilityType {
      boolean receive;
      boolean transmit;
      boolean sendModeMsg;
      boolean CVSD;
};

struct TransmitterParamsType {
      unsigned short transceiverID;
      CapabilityType capability;
      unsigned long frequency;
      /* Gen_Set_Power_Msg #24 */
      short powerLevel;
};

struct ReceiverParamsType {
      unsigned short transceiverID;
      CapabilityType capability;
      /* Gen_Set_Frequency_Msg #23 */
      unsigned long frequency;
};

interface TransceiverSetup {
      /*
      @roseuid 39F461CF0066 */
      boolean setUpReceiverParams (
            in ReceiverParamsType RecvParams
            );

      /*
```

```
                @roseuid 39F461CF0068 */
                boolean setUpTransmitterParams (
                        in TransmitterParamsType TransParams
                        );

                /*
                @roseuid 3A1442AA0013 */
                boolean setDate (
                        in DateType date
                        );

        };

        interface Controller : CF::Port, CF::Resource, RadioMode,
        MediaSetup, TransceiverSetup, CommonUtilities {
        };

        interface ReceiveCommand {
                exception test {
                        long error_param;
                };

                /*
                @roseuid 39EC97430199 */
                oneway void Receive (
                        in RcvPacketControl control
                        )
                        raises (test);

        };

        interface Provider : ReceiveCommand, PacketBB::PacketSignals,
        UlongPacket {
        };

    };

};

#endif
```

10 UML.

UML class and component diagrams. Refer to Figure 10-1 through Figure 10-5 for UML class and component diagrams.

10.1 CONTROLLER RELATIONSHIP.

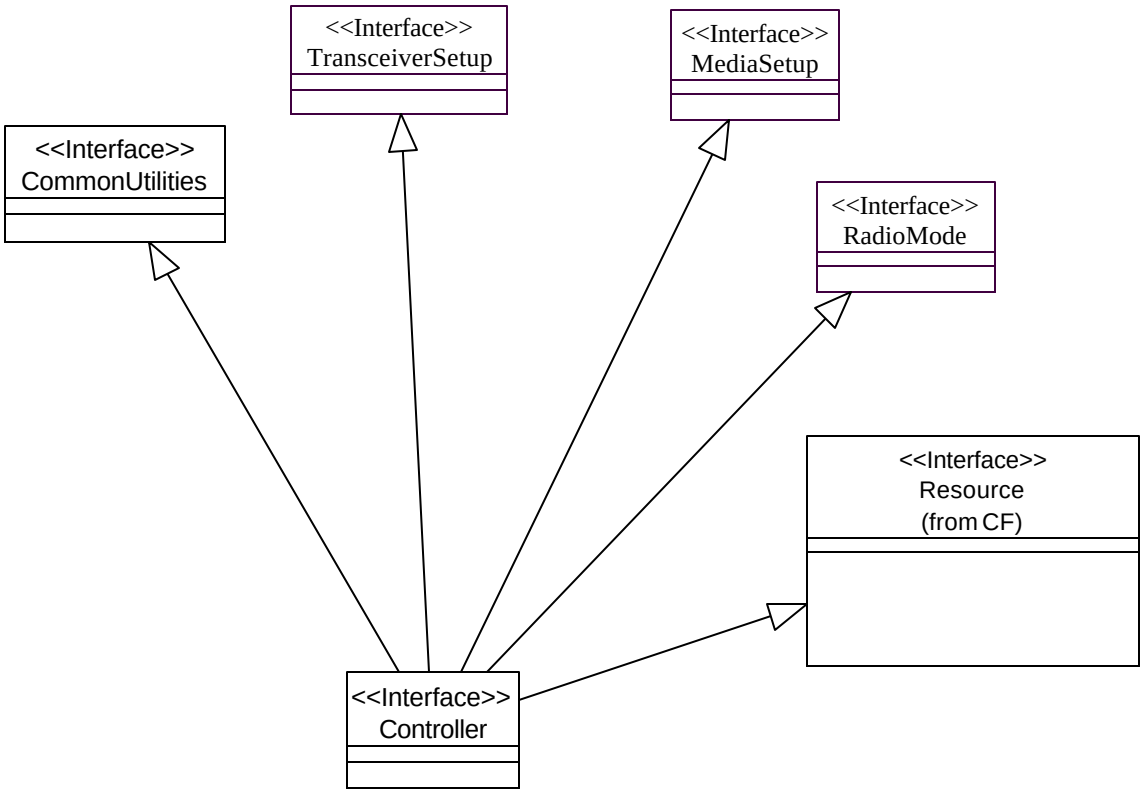

Figure 10-1. Controller Diagram

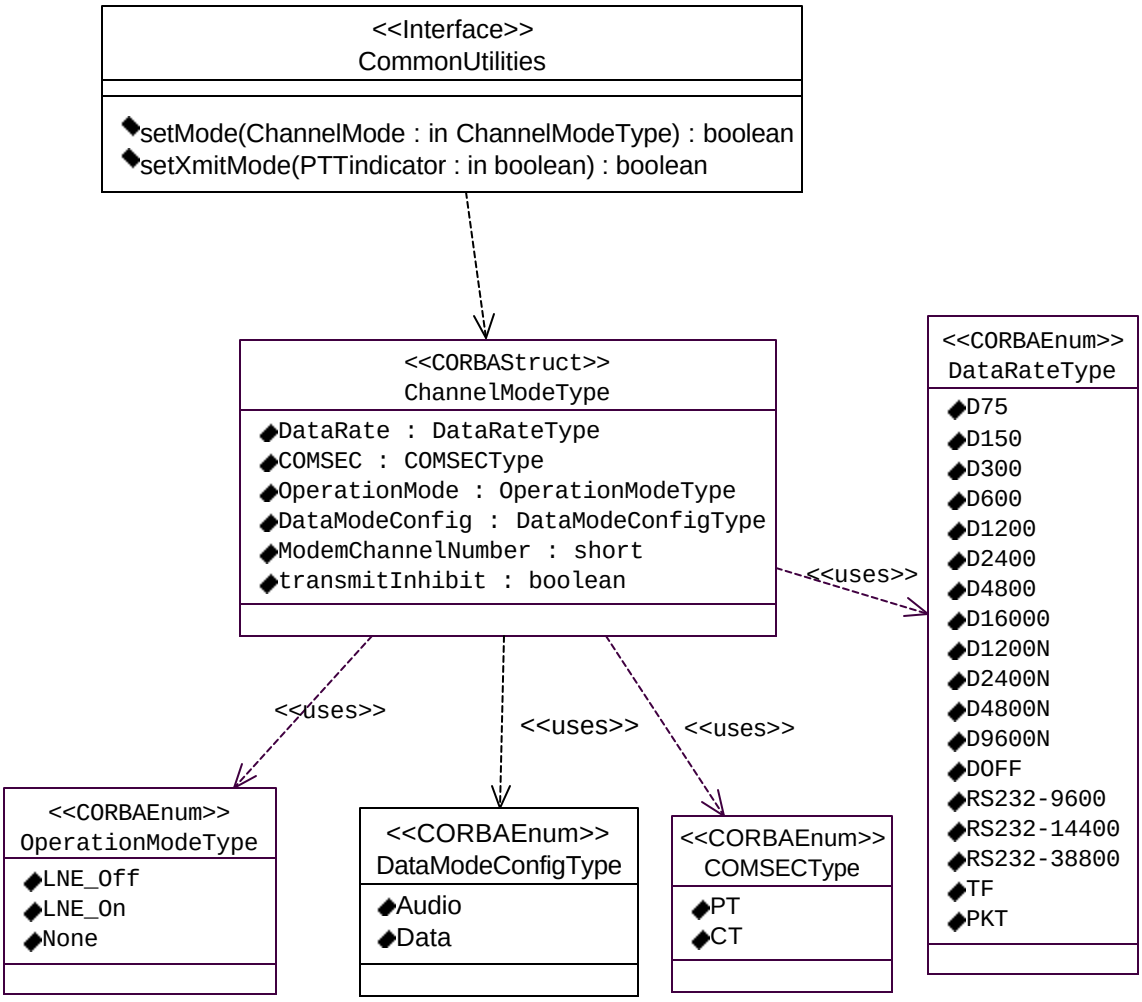

Figure 10-2. Common Utilities Diagram

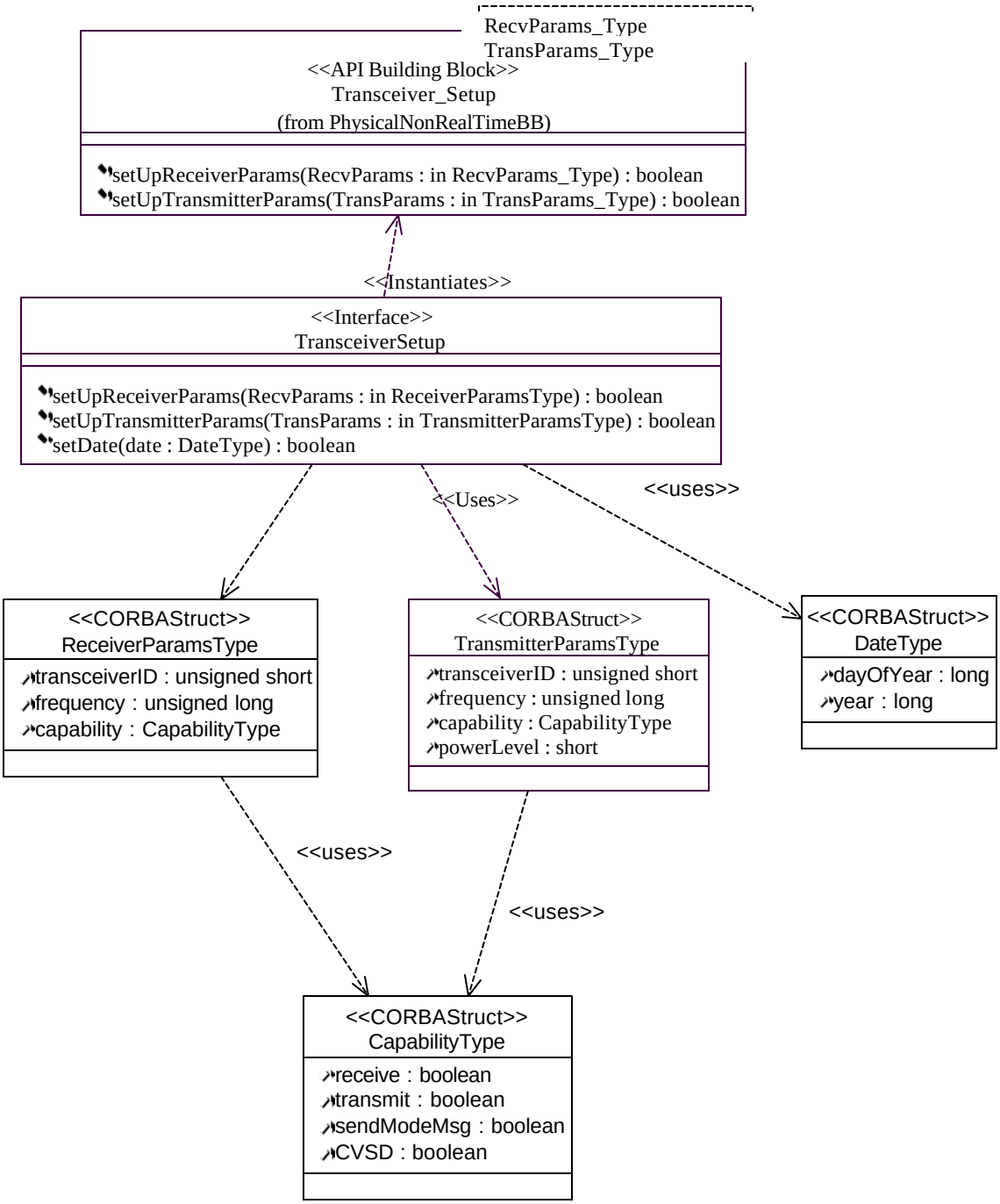

Figure 10-3. Transceiver Setup Diagram

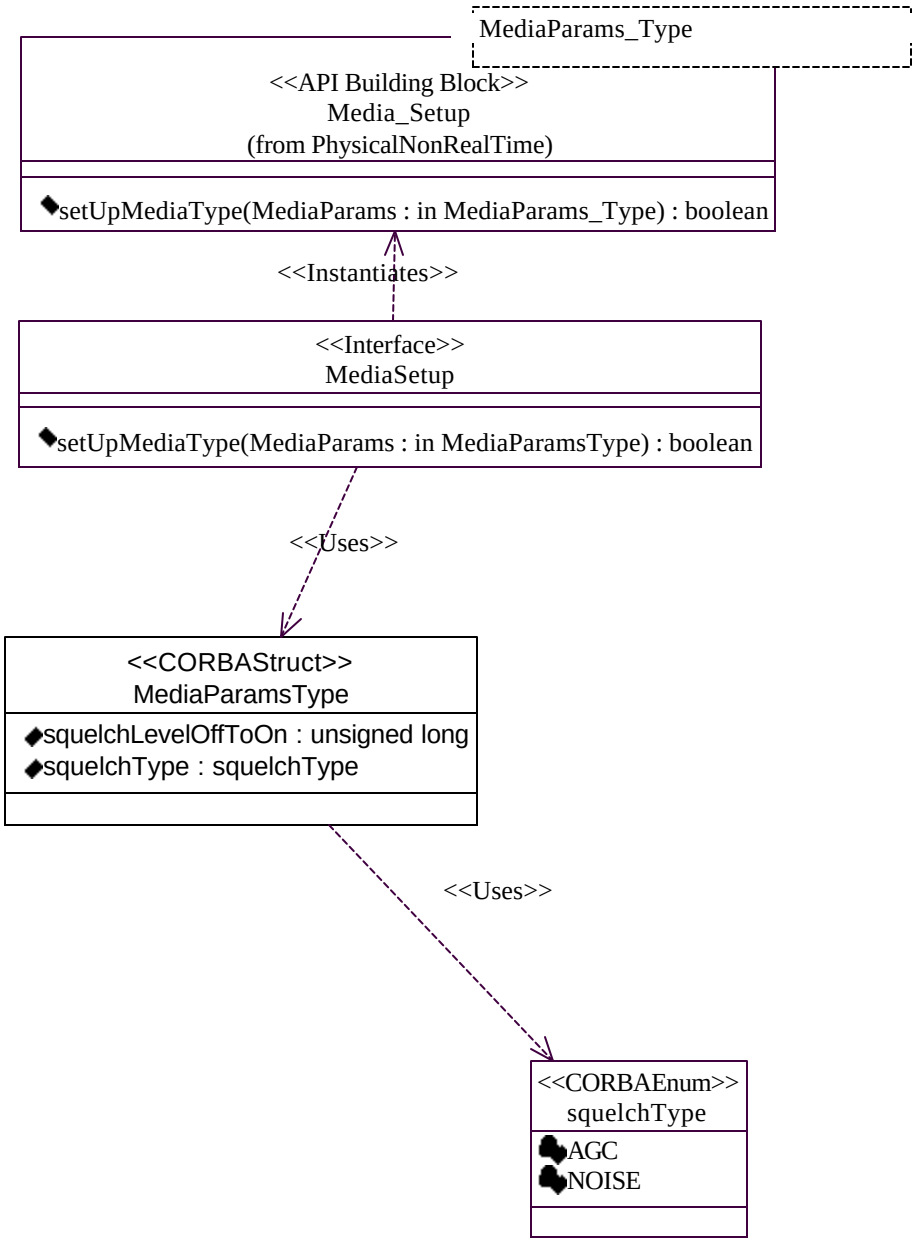

Figure 10-4. Media Setup Diagram

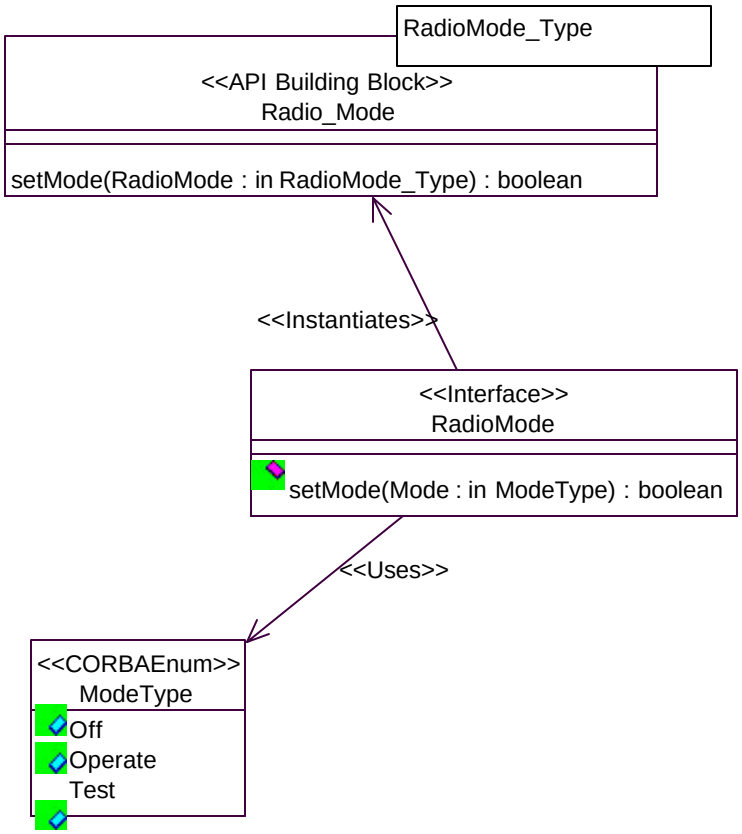

Figure 10-5. Radio Mode Diagram

10.2 USER RELATIONSHIP.

Refer to Figure 10-6 through Figure 10-8 for additional descriptions.

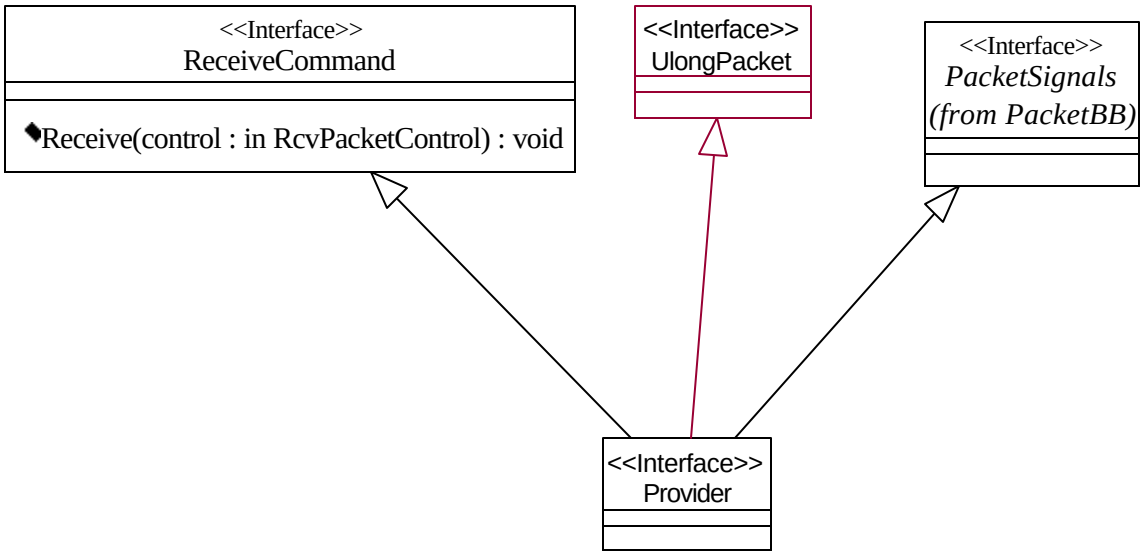

Figure 10-6. User Relationship Diagram

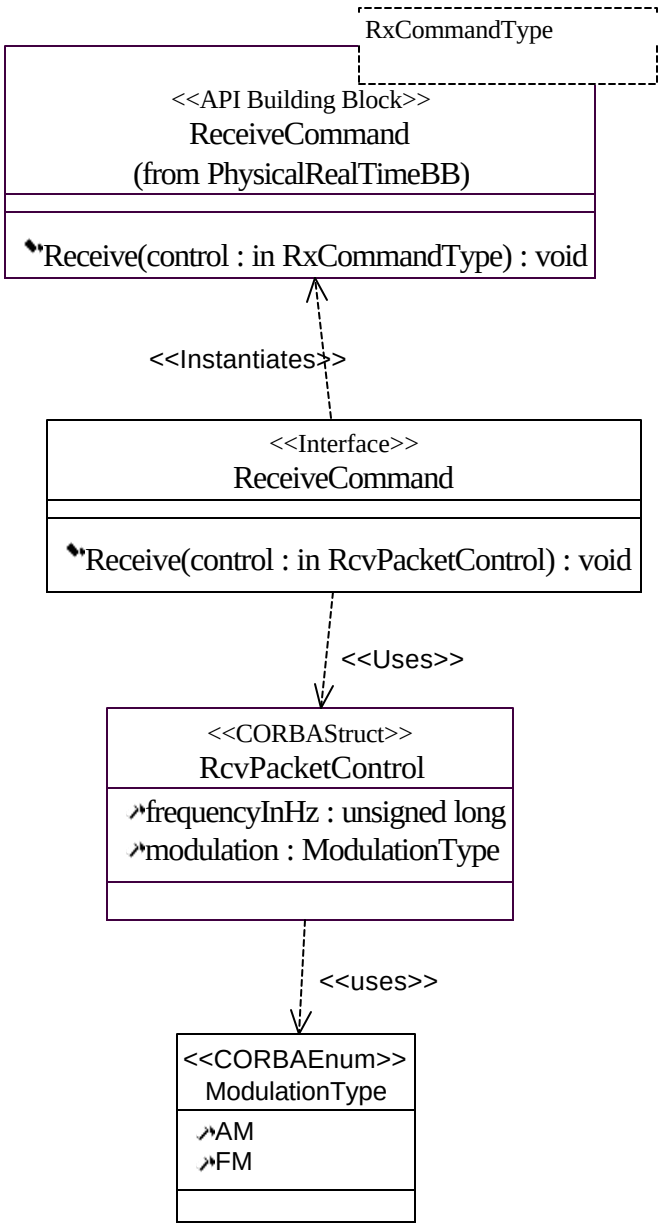

Figure 10-7. Receive Command Diagram

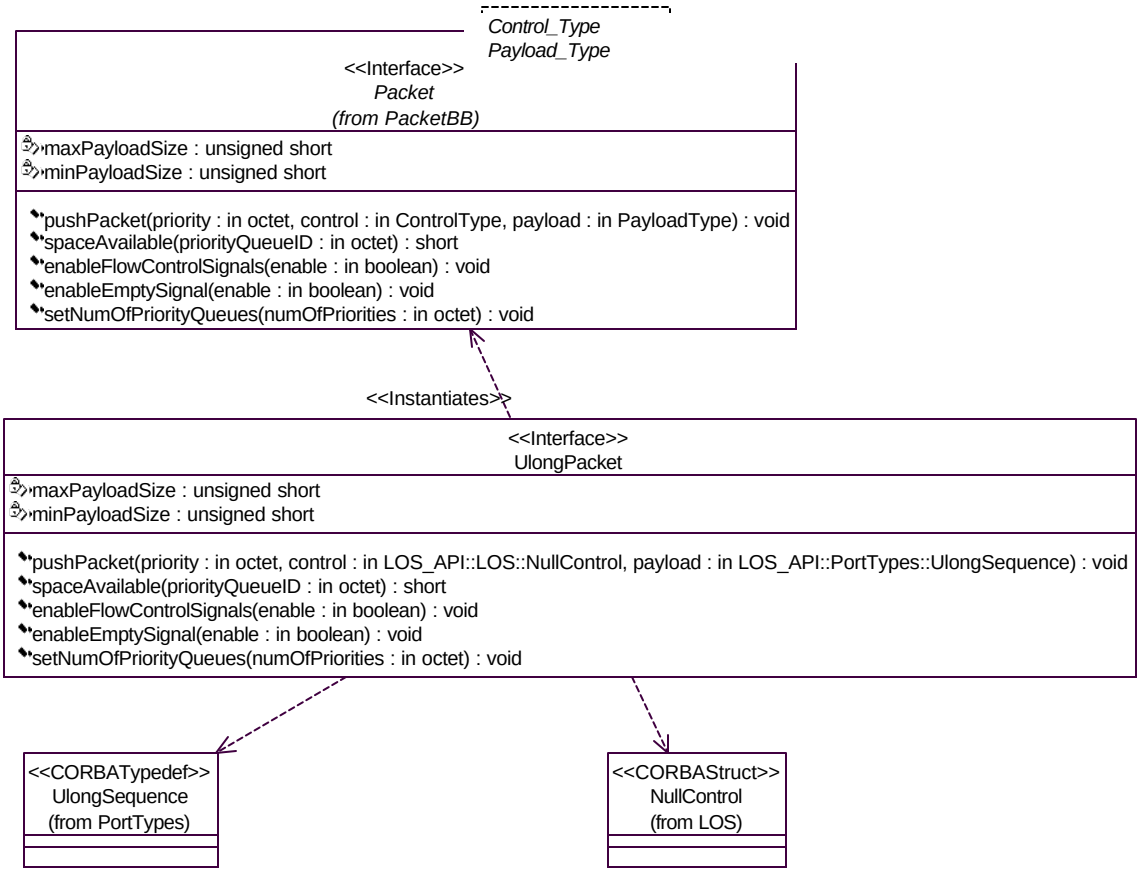

Figure 10-8. UlongPacket Diagram