Total Pages 43

Joint Tactical Radio System (JTRS) Standard
Audio Port Device
Application Program Interface (API)

Version: 1.3.4
29 July 2010

Statement A- Approved for public release; distribution is unlimited (29 July 2010)

REVISION HISTORY

Version	Authorization	Description	Last Modified Date
1.0		Initial release **ICWG Approved**	20-December-2005
1.1		Update outline format, add extensions **ICWG Approved**	02-February-2006
1.2		Removed Vocoder Extension.	22-February-2006
1.3		Updates based on Increment 4 ICWG. Added audio sample stream support, removed deprecated operations. **ICWG Approved**	09-May-2006
1.3.1		Updated Figure to be consistent with the Vocoder Service API. **ICWG Approved**	07-August-2006
1.3.1.1		Preparation for public release	29-March-2007
1.3.2		Errata: In AudioSampleStreamExt.idl corrected #ifndef/#define, i.e. changed `__AUDIO_SAMPLE_STREAM_DEFINED to __AUDIO_SAMPLE_STREAM_EXT_DEFINED`	02-April-2008
1.3.3		Correction: Section A.5.4.2: Clarified complex tone profile description **ICWG Approved**	29-June-2010
1.3.4		Preparation for public release	29-July-2010

Table of Contents

Table of Contents

Lists of Figures

List of Tables

A. AUDIO PORT DEVICE

A.1 INTRODUCTION

The *Audio Port Device* supports methods and attributes that are specific to the Audio Port hardware (HW) device it represents. The *Audio Port Device* provides the ability to control alert and alarm tones and to notify the device user of a Push-To-Talk (PTT) signal.

The *Audio Port Device* also provides a base configuration interface. It should be noted that this base *Audio Port Device* may be extended with the use of the extension (see B.1).

This document defines a common set of *Audio Port Device* provide services and interfaces required by most Joint Tactical Radio (JTR) Sets.

The *Audio Port Device* acts as "device adapter". It is used by Common Object Request Broker Architecture (CORBA) components (e.g., waveform application components) to access JTR Set Audio Port HW.

A.1.1 Overview

a. Section A.1, *Introduction*, contains the introductory material regarding the overview, service layer description, modes, states, and referenced documents of this document.

b. Section A.2, *Services*, specifies the interfaces for the component, port connections, and sequence diagrams.

c. Section A.3, *Service Primitives and Attributes*, specifies the operations that are provided by the *Audio Port Device*.

d. Section A.4, *IDL*.

e. Section A.5, UML.

f. Appendix A.A, *Abbreviations and Acronyms.*

g. Appendix A.B, *Performance Specification.*

A.1.2 Service Layer Description

A.1.2.1 Audio Port Device Port Connections

Figure 1 shows the port connections for the *Audio Port Device*.

Note: All port names are for reference only.

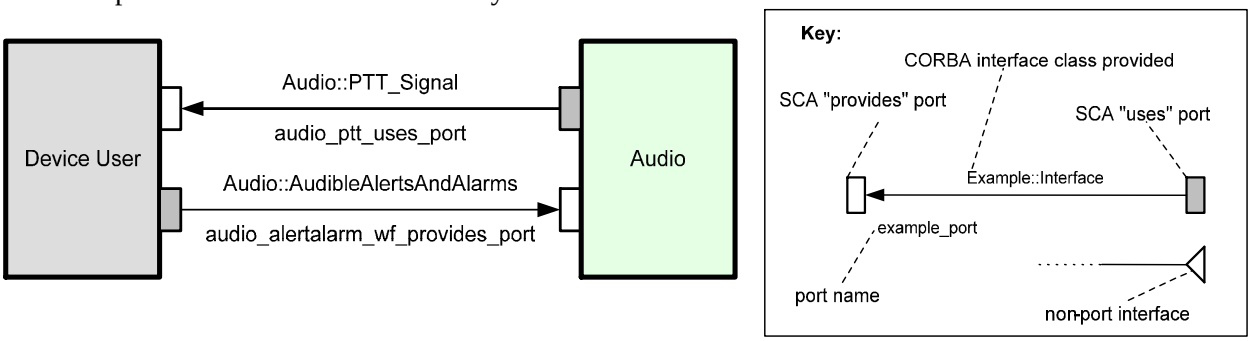

Figure 1 – Audio Port Device Port Diagram

***Audio Port Device* Provides Ports Definitions**

audio_alertalarm_wf_provides_port is provided by the *Audio Port Device* to control the alert and alarm tones by waveform.

***Audio Port Device* Uses Ports Definitions**

audio_ptt_uses_port is used to by the *Audio Port Device* to notify the Device User of PTT signal.

A.1.3 Modes of Service

Not applicable.

A.1.4 Service States

A.1.4.1.1 Audio Port Device State Diagram

The *Audio Port Device* states are illustrated in Figure 2. The *Audio Port Device* states ensure that received operations are only executed when the *Audio Port Device* is in the proper state. The five states of the *Audio Port Device* are as follow:

- CONSTRUCTED - The state transitioned to upon successful creation.
- INITIALIZED - The state transitioned to upon successful initialization.
- ENABLED - The state transitioned to upon successful start.
- DISABLED - The state transitioned to upon successful stop.
- RELEASED - The state transitioned to upon successful release.

The *Audio Port Device* transitions between states in response to the initialize, start, stop and releaseObject operations.

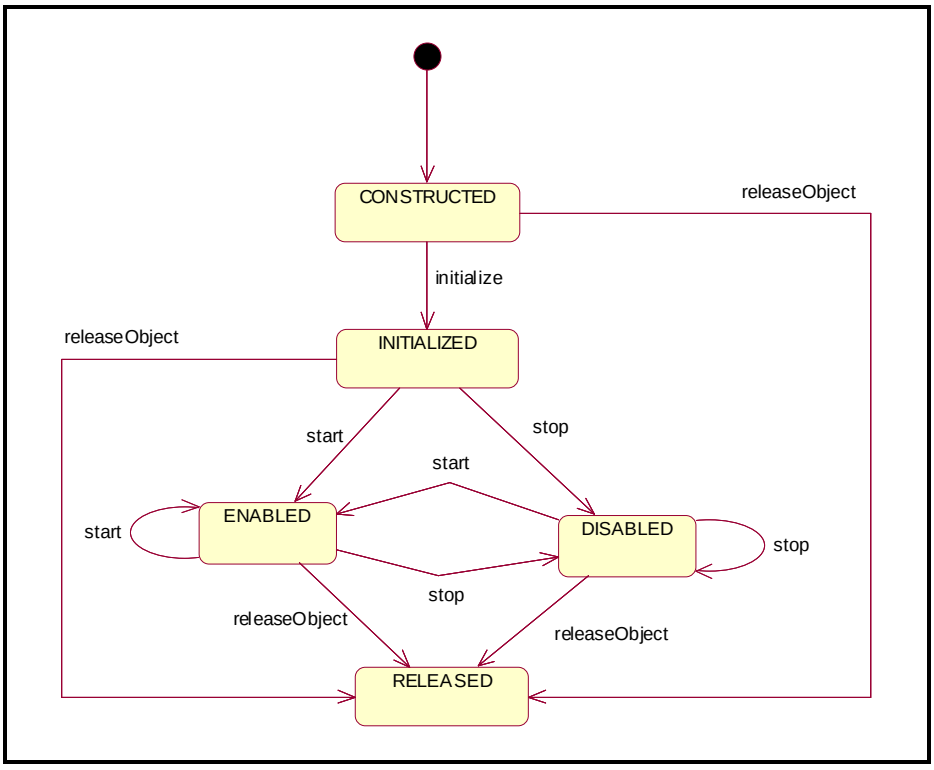

Figure 2 – Audio Port Device State Diagram

A.1.5 Referenced Documents

The following documents of the exact issue shown form a part of this specification to the extent specified herein.

A.1.5.1 Government Documents

A.1.5.1.1 Specifications

A.1.5.1.1.1 Federal Specifications

None

A.1.5.1.1.2 Military Specifications

None

A.1.5.1.2 Other Government Agency Documents

[1] JTRS Standard, “JTRS CORBA Types,” JPEO, Version 1.0.2.
[2] JTRS Standard, “Software Communications Architecture (SCA),” JPEO, Version 2.2.2.

A.1.5.2 Commercial Standards

None

A.2 SERVICES

A.2.1 Provide Services

The *Audio Port Device* provides service consists of the Table 1 service ports, interfaces, and primitives, which can be called by other client components.

Table 1 – Audio Port Device Provide Service Interface

Service Group (Port Name)	Service (Interface Provided)	Primitives (Provided)
audio_alertalarm_wf_in_port	Audio::AudibleAlertsAndAlarms	createTone()
		startTone()
		stopTone()
		stopAllTones()
		destroyTone()

A.2.2 Use Services

The *Audio Port Device* use service set consists of the Table 2 service ports, interfaces, and primitives. Since the *Audio Port Device* acts as a client with respect to these services from other components, it is required to connect these ports with corresponding service ports applied by the server component. The *Audio Port Device* uses the port name as the connectionID for the connection.

Table 2 – Audio Port Device Use Service Interface

Service Group (Port Name)	Service (Interface Used)	Primitives (Used)
audio_ptt_out_port	Audio::AudioPTT_Signal	setPTT()

A.2.3 Interface Modules

A.2.3.1 Audio Port Device

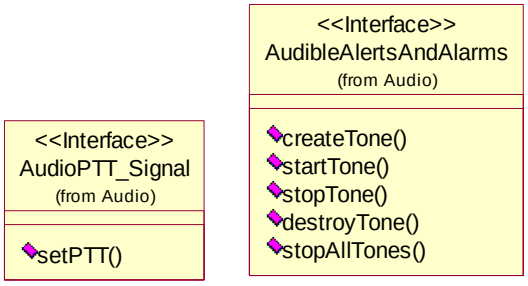

Figure 3 – Audio Port Device Interface Class Diagram

A.2.3.1.1 AudibleAlertsAndAlarms Interface Description

The *AudibleAlertsAndAlarms* interface provides tone and beep creation and their storage capability to the device user.

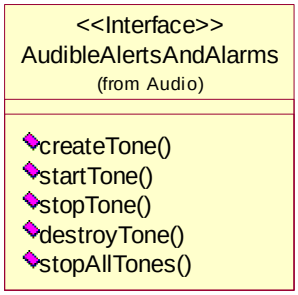

Figure 4 – AudioAlertsAndAlarms Class Diagram

A.2.3.1.2 AudioPTT_Signal Interface Description

The *AudioPTT_Signal* interface is defined in the *Audio Port Device* for the use by the waveforms, so that the *Audio Port Device* can signal an event to the waveform when a Push-To-Talk (PTT) event is received from the Audio Port HW.

Figure 5 – AudioPTT_Signal Class Diagram

A.2.4 Sequence Diagrams

None

A.3 SERVICE PRIMITIVES AND ATTRIBUTES

To enhance the readability of this API document and to avoid duplication of data, the type definitions of all structured types (i.e., data types, enumerations, exceptions, and structures) used by the Service Primitives and Attributes have been co-located in Section A.5. This cross-reference of types also includes any nested structures in the event of a structure of structures or an array of structures.

A.3.1 Audio::AudibleAlertsAndAlarms

A.3.1.1 *createTone* Operation

The *createTone* operation provides the capability of creating a tone or beep with the specified profile, for future use by the device user.

A.3.1.1.1 Synopsis

unsigned short createTone(in ToneProfileType toneProfile) raises(InvalidToneProfile);

A.3.1.1.2 Parameters

Parameter Name	Description	Type
toneProfile	A structure containing the elements used to create the tone/beep for the *Audio Port Device.*	ToneProfileType (see A.5.3)

A.3.1.1.3 State

ENABLED CF::Device::operationalState.

A.3.1.1.4 New State

This operation does not cause a state change.

A.3.1.1.5 Return Value

Type	Description	Valid Range
unsigned short	An identification number associated with the tone/beep.	0 – 64k

A.3.1.1.6 Originator

Service User

A.3.1.1.7 Exceptions

Exception	Description
InvalidToneProfile (see A.5.3.1)	A CORBA exception is raised when the tone/beep cannot be generated due to invalid attributes in *toneProfileType* structure.

A.3.1.2 *startTone* Operation

The *startTone* operation provides the user the ability to start the generation of a previously created tone/beep to the device user.

A.3.1.2.1 Synopsis

void startTone(in unsigned short toneId) raises(InvalidToneId);

A.3.1.2.2 Parameters

Parameter Name	Description	Type	Units	Valid Range
toneId	A tone ID associated with the tone /beep for sending to the Audio Port HW.	unsigned short	ID number	1 – 65535

A.3.1.2.3 State

ENABLED CF::Device::operationalState.

A.3.1.2.4 New State

This operation does not cause a state change.

A.3.1.2.5 Return Value

None

A.3.1.2.6 Originator

Service User

A.3.1.2.7 Exceptions

Exception	Description
InvalidToneId (see A.5.3.2)	A CORBA exception is raised when the tone/beep identification number is invalid.

A.3.1.3 *stopTone* Operation

The *stopTone* operation provides the device user the ability to stop generation of a previously started tone.

A.3.1.3.1 Synopsis

void stopTone(in unsigned short toneId) raises(InvalidToneId);

A.3.1.3.2 Parameters

Parameter Name	Description	Type	Units	Valid Range
toneId	A tone ID associated with the tone which needs to be stopped.	unsigned short	ID number	1 – 65535

A.3.1.3.3 State

ENABLED CF::Device::operationalState.

A.3.1.3.4 New State

This operation does not cause a state change.

A.3.1.3.5 Return Value

None

A.3.1.3.6 Originator

Service User

A.3.1.3.7 Exceptions

Exception	Description
InvalidToneId (see A.5.3.2)	A CORBA exception is raised when the tone/beep identification number is invalid.

A.3.1.4 *stopAllTones* Operation

The *stopAllTones* operation provides the device user the ability to stop generation of all previously started tones.

A.3.1.4.1 Synopsis

void stopAllTones();

A.3.1.4.2 Parameters

None

A.3.1.4.3 State

ENABLED CF::Device::operationalState.

A.3.1.4.4 New State

This operation does not cause a state change.

A.3.1.4.5 Return Value

None

A.3.1.4.6 Originator

Service User

A.3.1.4.7 Exceptions

None

A.3.1.5 *destroyTone* Operation

The *destroyTone* operation provides the device user the ability to destroy the previously created tone/beep to prevent the future use.

A.3.1.5.1 Synopsis

void destroyTone(in unsigned short toneId) raises(InvalidToneId);

A.3.1.5.2 Parameters

Parameter Name	Description	Type	Units	Valid Range
toneId	A tone/beep ID associated with the tone/beep which needs to be destroyed.	unsigned short	ID number	1 – 65535

A.3.1.5.3 State

ENABLED CF::Device::operationalState.

A.3.1.5.4 New State

This operation does not cause a state change.

A.3.1.5.5 Return Value

None

A.3.1.5.6 Originator

Service User

A.3.1.5.7 Exceptions

Exception	Description
InvalidToneId (see A.5.3.2)	A CORBA exception is raised when the tone/beep identification number is invalid.

A.3.2 Audio::AudioPTT_Signal

A.3.2.1 *setPTT* Operation

The *setPTT* operation is used to inform the downstream components of the push to talk signal.

A.3.2.1.1 Synopsis

void setPTT(in boolean PTT);

A.3.2.1.2 Parameters

Parameter Name	Description	Type	Valid Ranges
PTT	Indicates whether the push to talk signal has been received from the Audio Port HW.	boolean	TRUE = push to talk signal has been received from Audio Port HW; FALSE = push to talk signal has not been received from Audio Port HW.

A.3.2.1.3 State

ENABLED CF::Device::operationalState.

A.3.2.1.4 New State

This operation does not cause a state change.

A.3.2.1.5 Return Value

None

A.3.2.1.6 Originator

Service Provider

A.3.2.1.7 Exceptions

None

A.4 IDL

A.4.1 Audio IDL

```
/*
** Audio.idl - JTRS Base Audio Service Set
*/
#ifndef __AUDIO_DEFINED
#define __AUDIO_DEFINED

#ifndef __JTRSCORBATYPES_DEFINED
#include "JtrsCorbaTypes.idl"
#endif

module Audio
{
	// Push to Talk Control
	interface AudioPTT_Signal
	{
		void setPTT( in boolean PTT );
	};

	interface AudibleAlertsAndAlarms
	{
		exception InvalidToneProfile
		{
			boolean complexTone;		// changed to FALSE ComplexToneProfile structure invalid
			boolean simpleTone;		// changed to FALSE SimpleToneProfile structure invalid
			string msg;			// message exception location
		};

		exception InvalidToneId
		{
			string msg;			// message exception location
		};

		struct SimpleToneProfile
		{
			unsigned short frequencyInHz;	// frequency in Hz
```

```
                unsigned short durationPerBurstInMs;        // duration of tone per burst in
            milliseconds
                unsigned short repeatIntervalInMs;          // The repeat interval in milliseconds
            };

            enum ToneDescriminator
            {
                COMPLEX_TONE,                               // Select ComplexToneProfile
                SIMPLE_TONE                                 // Select SimpleToneProfile
            };

            struct ComplexToneProfile
            {
                JTRS::ShortSequence   toneSamples;          // tone samples
                unsigned short        numberOfRepeats;      // number times to repeat samples
            };

            union ToneProfileType switch ( ToneDescriminator )
            {
                case COMPLEX_TONE:
                    ComplexToneProfile complexTone;         // tone described by ComplexToneProfile
                case SIMPLE_TONE:
                    SimpleToneProfile simpleTone;           // tone described by SimpleToneProfile
            };

            unsigned short createTone( in ToneProfileType toneProfile )
            raises (InvalidToneProfile);

            void startTone( in unsigned short toneId )
            raises (InvalidToneId);

            void stopTone( in unsigned short toneId )
            raises (InvalidToneId);

            void destroyTone( in unsigned short toneId )
            raises (InvalidToneId);

            void stopAllTones();
        };

};
#endif
```

A.5 UML

This section contains the device component UML diagram and the definitions of all data types referenced (directly or indirectly) by A.3 Service Primitives and Attributes.

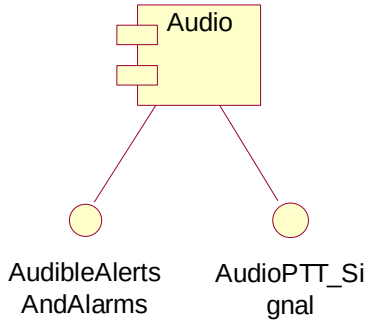

Figure 6 – Audio Port Device Component Diagram

A.5.1 Data Types

None

A.5.2 Enumerations

A.5.2.1 Audio::AudibleAlertsAndAlarms::ToneDescriminator

The *ToneDescriminator* enumeration type defines attributes to determine if the tone/beep selection is the *ComplexToneProfile* or a *SimpleToneProfile*. This enumeration type is used by *ToneProfileType* to make a selection between the two profiles.

```
enum ToneDescriminator
{
        COMPLEX_TONE,
        SIMPLE_TONE
};
```

Enum	Attributes	Description
ToneDescriminator	COMPLEX_TONE	Select ComplexToneProfile.
	SIMPLE_TONE	Select SimpleToneProfile.

A.5.3 Exceptions

A.5.3.1 Audio::AudibleAlertsAndAlarms::InvalidToneProfile

The *InvalidToneProfileType* exception is raised when any of the attributes in either *ComplexToneProfile* structure or *SimpleToneProfile* structure are out of range.

```
exception InvalidToneProfile
{
        boolean complexTone;
        boolean simpleTone;
        string msg;
};
```

Exception	Attributes	Description	Type
InvalidToneProfile	complexTone	This attribute will be changed to FALSE if the elements of the ComplexToneProfile structure are either invalid or out of range.	boolean
	simpleTone	This attribute will be changed to FALSE if the elements of the SimpleToneProfile structure are either invalid or out of range.*	boolean
	msg	A message of type string indicating that the exception has occurred.	string

Note: (*) The valid range for the frequencyInHz attribute of the SimpleToneProfile will be specified by the platform.

A.5.3.2 Audio::AudibleAlertsAndAlarms::InvalidToneId

The *InvalidToneId* exception is used by the *startTone*, *stopTone*, and *destroyTone* operations to indicate that the tone cannot be started, stopped, or destroyed due to an invalid *toneId*.

```
exception InvalidToneId
{
        string msg;
};
```

Exception	Attributes	Description	Type
InvalidToneId	Msg	A message of type string indicating that the exception has occurred.	string

A.5.4 Structures

A.5.4.1 Audio::AudibleAlertsAndAlarms::SimpleToneProfile

The *SimpleToneProfileType* structure defines attributes, which describes the tone or beep for the *Audio Port Device*. A beep is created by setting the repeatIntervalInMs attribute of the SimpleToneProfile to 0.

```
struct SimpleToneProfile
{
        unsigned short frequencyInHz;
        unsigned short durationPerBurstInMs;
        unsigned short repeatIntervalInMs;
};
```

Struct	Attributes	Description	Type	Units	Valid Range
SimpleToneProfile	frequencyInHz	The frequency in Hz.	unsigned short	Hz	50 - 4000
	durationPerBurstInMs	The duration of tone per burst in milliseconds.	unsigned short	ms	1 - 65535
	repeatIntervalInMs	The repeat interval in milliseconds.	unsigned short	ms	0* – 65535

Note: (*) The repeatInternvalInMs of range 1-65535 will indicate the repeat interval for the tone in milliseconds. A repeatIntervalInMs of value 0 will indicate the creation of a beep.

A.5.4.2 Audio::AudibleAlertsAndAlarms::ComplexToneProfile

The *ComplexToneProfileType* structure defines attributes to generate complex tones for the *Audio Port Device*.

```
struct ComplexToneProfile
{
        JTRS::ShortSequence   toneSamples;
        unsigned short        numberOfRepeats;
};
```

Struct	Attributes	Description	Type

ComplexToneProfile*	toneSamples	The tone samples of type ShortSequence (sequence of unsigned short). Tone samples are provided as a sequence of 16 bit linear Pulse Code Modulation (PCM) sampled at 8 kHz.	JTRS::ShortSeq (See *JTRS CORBA Types* [1])
	numberOfRepeats	The number of repeats for the tone samples.	unsigned short

Note: (*) All Complex tones combined can only occupy 3MB or 196608 samples (24 seconds).

A.5.5 Unions

A.5.5.1 Audio::AudibleAlertsAndAlarms::ToneProfileType

The *ToneProfileType* CORBA::Union type defines attributes, which describes the tone/beep for the *Audio Port Device*.

```
union ToneProfileType switch ( ToneDescriminator )
{
        case COMPLEX_TONE:
                ComplexToneProfile complexTone;
        case SIMPLE_TONE:
                SimpleToneProfile simpleTone;
};
```

Union	Attributes	Description	Type
ToneProfileType	COMPLEX_TONE	The tone described by the ComplexToneProfileType.	ComplexToneProfile (See A.5.4.2)
	SIMPLE_TONE	The tone/beep described by the SimpleToneProfileType.	SimpleToneProfile (See A.5.4.1)
	ToneDescriminator	Used to make a selection between the ComplexToneProfile or SimpleToneProfile types.	ToneDescriminator (See A.5.2.1)

Appendix A.A Abbreviations and Acronyms

API	Application Program Interface
CF	Core Framework
CORBA	Common Object Request Broker Architecture
Hz	Hertz
ICWG	Interface Control Working Group
ID	Identification
IDL	Interface Definition Language
JPEO	Joint Program Executive Office
JTRS	Joint Tactical Radio System
ms	Millisecond
PCM	Pulse Code Modulation
PTT	Push to Talk
SCA	Software Communications Architecture
UML	Unified Modeling Language
WF	Waveform

Appendix A.B Performance Specification

Table 3 provides a template for the generic performance specification for the *Audio Port Device* API which will be documented in the service or device using the interface. This performance specification corresponds to the port diagram in Figure 1.

Table 3 – Audio Port Device Performance Specification

Specification	Description	Units	Value
Worst Case Command Execution Time for audio_alertalarm_wf_provides_port	*	*	*
Worst Case Command Execution Time for audio_ptt_uses_port	*	*	*

Note: (*) These values should be filled in by individual developers.

B. AUDIO SAMPLE STREAM EXTENSION

B.1 INTRODUCTION

The *Audio Sample Stream* Extension is based upon the *Audio Port Device* API (see A.1). It extends the functionality of the common JTRS audio device to provide, consume, and control audio samples to/from the Audio Port Harware. It retains the methods and attributes defined in the base *Audio Port Device* API.

B.1.1 Overview

a. Section B.1, *Introduction*, contains the introductory material regarding the overview, service layer description, modes, states, and referenced documents of this document.

b. Section B.2, *Services*, specifies the interfaces for the component, port connections, and sequence diagrams.

c. Section B.3, *Service Primitives and Attributes*, specifies the operations that are provided by the *Audio Sample Stream* Extension.

d. Section B.4, *IDL*.

e. Section B.5, *UML*.

f. Appendix B.A, *Abbreviations and Acronyms.*

g. Appendix B.B, *Performance Specification.*

B.1.2 Service Layer Description

B.1.2.1 Audio Sample Stream Extension Port Connections

Figure 7 shows the port connections for the *Audio Sample Stream* Extension.

Note: All port names are for reference only. Ports in black are defined in the base *Audio Port Device* API (see Figure 1 – Audio Port Device Port Diagram).

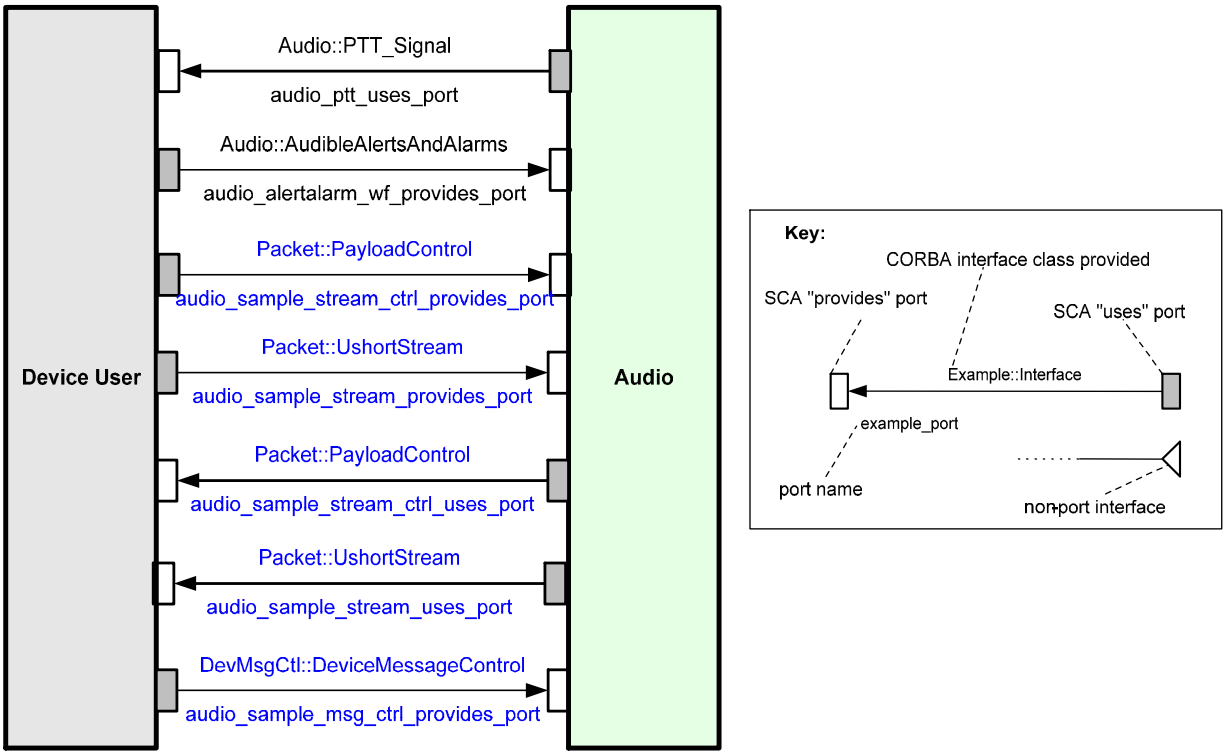

Figure 7 – Audio Sample Stream Extension Port Diagram

***Audio Sample Stream Extension* Provides Ports Definitions**

audio_sample_stream_provides_port is provided by the *Audio Port Device* to consume packets through the *pushPacket* operation.
audio_sample_stream_ctrl_provides_port is provided by the *Audio Port Device* to set the payload size by the Device User.
audio_sample_msg_ctrl_provides_port is provided by the *Audio Port Device* to manage the message flows.

***Audio Sample Stream* Extension Uses Ports Definitions**

audio_sample_stream_uses_port is used by the *Audio Port Device* to set the payload size of the incoming packets from the Device User.
audio_sample_stream_ctrl_uses_port is used by the *Audio Port Device* to push packets to the Device User.

B.1.3 Modes of Service

Not applicable.

B.1.4 Service States

B.1.4.1.1 Audio Sample Stream Extension Streaming State Diagram

The *AudioPort Device* streaming states are illustrated in Figure 8.

The two streaming states of the *AudioPort Device* are as follow:

- STREAMING - The state transitioned to when the *endOfStream* indicator in the *Packet::StreamControlType* of the pushPacket operation is set to FALSE.
- NOT STREAMING - The state transitioned to upon successful startup and when the *endOfStream* indicator in the *Packet::StreamControlType* of the pushPacket operation is set to TRUE.

See *Packet API* [2] for the definintion of *Packet::StreamControlType*.

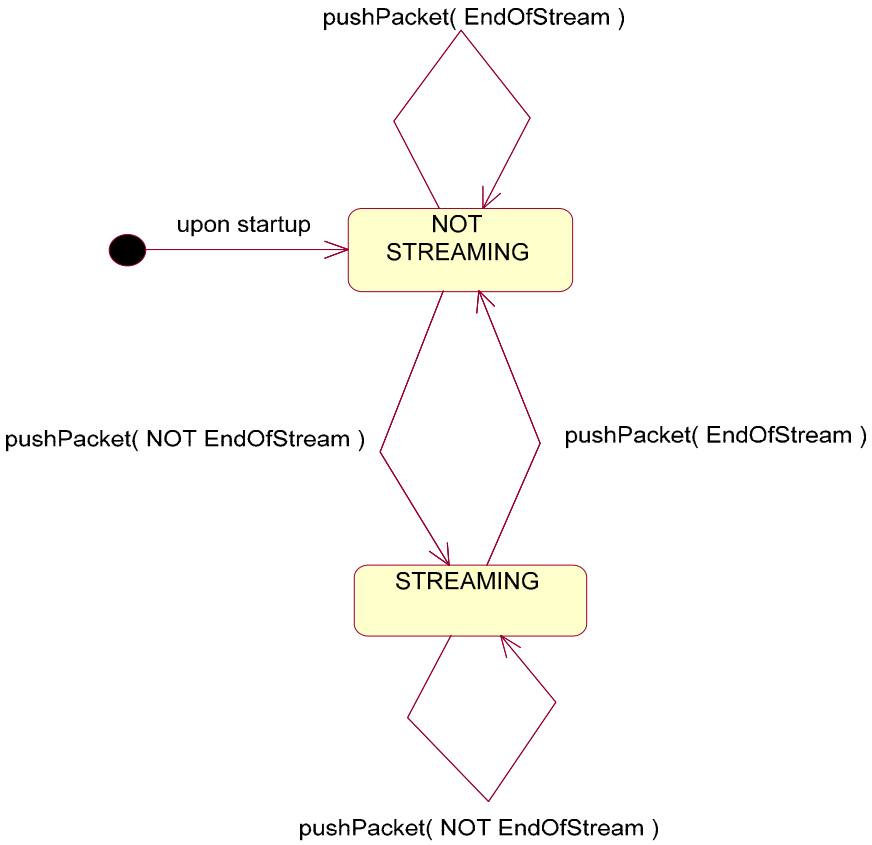

Figure 8 – Audio Sample Stream Extension Streaming State Diagram

B.1.5 Referenced Documents

The following documents are additional references not already defined in the base API.

B.1.5.1 Government Documents

B.1.5.1.1 Specifications

B.1.5.1.1.1 Federal Specifications

None

B.1.5.1.1.2 Military Specifications

None

B.1.5.1.2 Other Government Agency Documents

[1] JTRS Standard, “Device Message Control API,” JPEO, Version 1.1.1.
[2] JTRS Standard, “Packet API,” JPEO, Version 2.0.2.

B.1.5.2 Commercial Standards

None

B.2 SERVICES

B.2.1 Provide Services

The *Audio Sample Stream* Extension provides service consists of the Table 4 service ports, interfaces, and primitives, which can be called by other client components. Detailed definition of the interfaces and services shaded in grey is provided by separate documentation identified in the table.

Table 4 – Audio Sample Stream Extension Provide Service Interface

Service Group (Port Name)	Service (Interface Provided)		Primitives (Provided)	Parameter Name or Return Value	Valid Range
audio_sample_ stream_ provides_port	Audio:: SampleStream	Packet::UshortStream [2]	pushPacket()	*See Packet API* [2]	*See Packet API* [2]
		Packet:: PayloadStatus [2]	getMaxPayloadSize()	*Return Value*	1 to 16383
			getMinPayloadSize()	*Return Value*	0 to 512
			getDesiredPayloadSize()	*Return Value*	1 to 16383
			getMinOverrideTimeout()	*Return Value*	0 to 50
audio_sample_ stream_ctrl_ provides_port	Audio:: StreamControl	Packet:: PayloadControl [2]	setMaxPayloadSize()	maxPaylaodSize	1 to 16383
			setMinPayloadSize()	minPayloadSize	0 to 512
			setDesiredPayloadSize()	desiredPayloadSize	1 to 16383
			setMinOverrideTimeout()	minOverrideTimeout	0 to 50
audio_sample_ msg_ctrl_ provides_port	Audio:: SampleMessage Control	DevMsgCtl:: DeviceMessageControl [1]	txActive()	*Return Value*	TRUE or FALSE
			rxActive()	*Return Value*	TRUE or FALSE
			abortTx()	*Return Value*	void

B.2.2 Use Services

The *Audio Sample Stream Extension* use service set consists of the Table 5 service ports, interfaces, and primitives. Since the *Audio Port Device* acts as a client with respect to these services from other components, it is required to connect these ports with corresponding service ports applied by the server component. The *Audio Sample Stream Extension* uses the port name as connectionID for the connection. Detailed definition of the interfaces and services shaded in grey is provided by separate documentation specified in the table.

Table 5 – Audio Sample Stream Extension Use Service Interface

Service Group (Port Name)	**Service (Interface Provided)**		**Primitives (Provided)**	**Parameter Name or Return Value**	**Valid Range**
audio_sample_ stream_ uses_port	Audio:: SampleStream	Packet::UshortStream [2]	pushPacket()	*See Packet API* [2]	*See Packet API* [2]
		Packet:: PayloadStatus [2]	getMaxPayloadSize()	*Return Value*	1 to 16383
			getMinPayloadSize()	*Return Value*	0 to 512
			getDesiredPayloadSize()	*Return Value*	1 to 16383
			getMinOverrideTimeout()	*Return Value*	0 to 50
audio_sample_ stream_ctrl_ uses_port	Audio:: StreamControl	Packet:: PayloadControl [2]	setMaxPayloadSize()	maxPaylaodSize	1 to 16383
			setMinPayloadSize()	minPayloadSize	0 to 512
			setDesiredPayloadSize()	desiredPayloadSize	1 to 16383
			setMinOverrideTimeout()	minOverrideTimeout	0 to 50

B.2.3 Interface Modules

B.2.3.1 Audio Port Device

B.2.3.1.1 Audio Sample Stream Extension

The interface class diagram for the *Audio Sample Stream Extension* is provided in Figure 9. Interfaces defined in grey are specified in the *Device Message Control API* [1] or the *Packet API* [2].

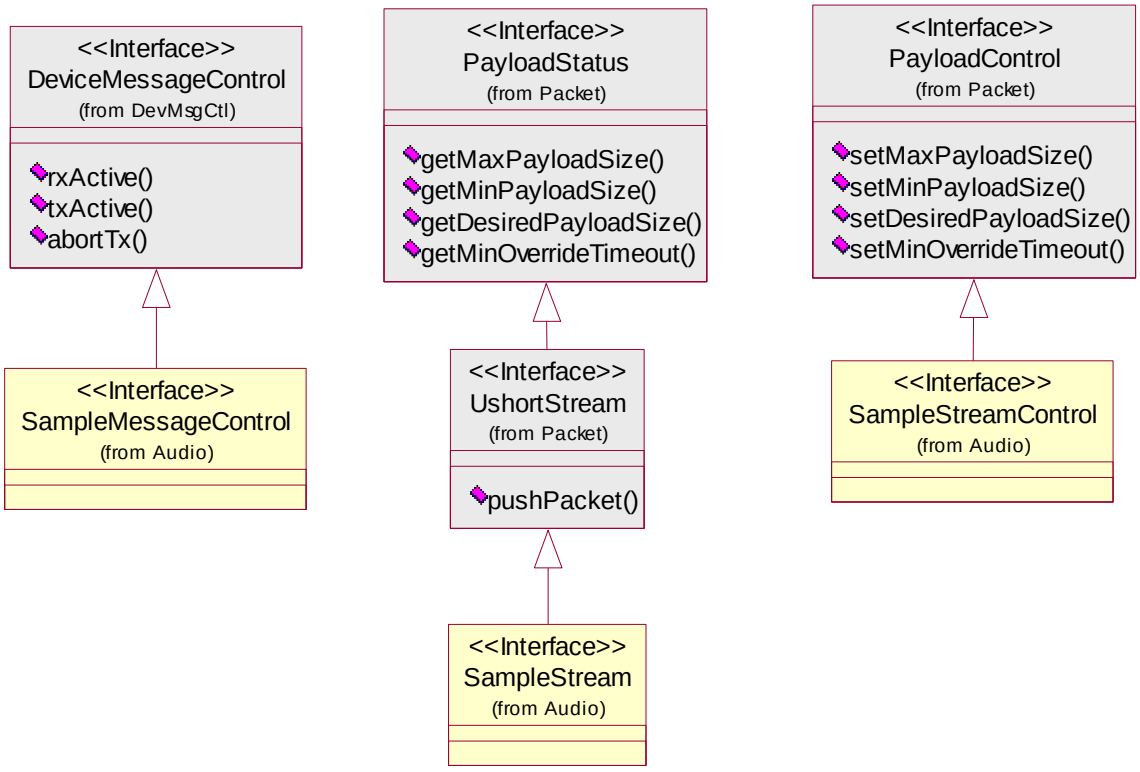

Figure 9 – Audio Sample Stream Extension Interface Class Diagram

B.2.3.1.2 SampleStream Interface Description

The interface design of *SampleStream* is shown in Figure 10. It extends the *Packet::UshortStream* interface defined in the *Packet API* [2] to provide the ability to status the audio sample packet sizes and to push audio sample packets to the Audio Port HW.

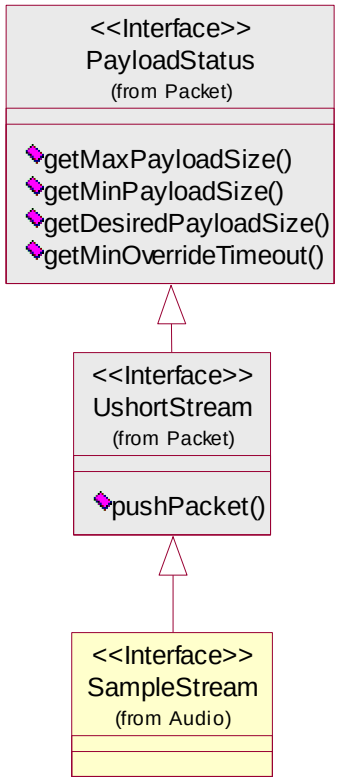

Figure 10 – SampleStream Interface Diagram

B.2.3.1.3 SampleStreamControl Interface Description

The interface design of *SampleStreamControl* is shown in Figure 11. It extends the *Packet::PayloadControl* interface defined in the *Packet API* [2] to provide the ability to configure the audio sample packet sizes.

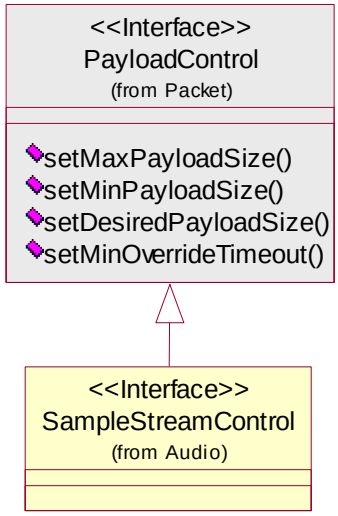

Figure 11 – SampleStreamControl Interface Diagram

B.2.3.1.4 SampleMessageControl Interface Description

The interface design of *SampleMessageControl* is shown in Figure 12. It extends the *DevMsgCtl::DeviceMessageControl* interface defined in the *DeviceMessageControl API* [1] to determine whether the supporting component is actively processing transmit or receive traffic. It also provides the ability to abort the transmission.

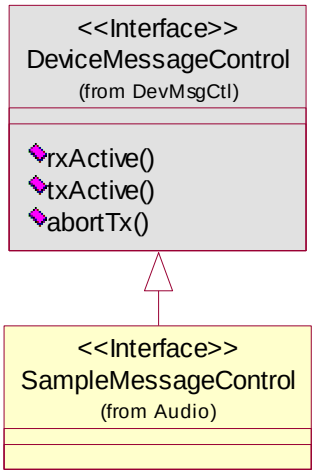

Figure 12 – SampleMessageControl Interface Diagram

B.2.4 Sequence Diagrams

None

B.3 SERVICE PRIMITIVES AND ATTRIBUTES

There are no additional service primitivites and attributes than those defined in the *Packet API* [2] and the *Device Message Control API* [1].

B.4 IDL

B.4.1 Audio Sample StreamExt IDL

```
/*
** AudioSampleStreamExt.idl - JTRS Audio Extension Service Set
*/
#ifndef __AUDIO_SAMPLE_STREAM_EXT_DEFINED
#define __AUDIO_SAMPLE_STREAM_EXT_DEFINED

#ifndef __PACKET_DEFINED
#include "Packet.idl"
#endif

#ifndef __DEVICEMESSAGECONTROL_DEFINED
#include "DeviceMessageControl.idl"
#endif

module Audio
{
    // Packet Consumer
       interface SampleStream : Packet::UshortStream
       {
       };

    // Packet Provider Control
    interface SampleStreamControl : Packet::PayloadControl
    {
    };

       // Abort Messaging
       interface SampleMessageControl : DevMsgCtl::DeviceMessageControl
       {
       };
};

#endif  //__AUDIO_SAMPLE_STREAM_EXT_DEFINED
```

B.5 UML

This section contains the *Audio Sample Stream* Extension component UML diagram.

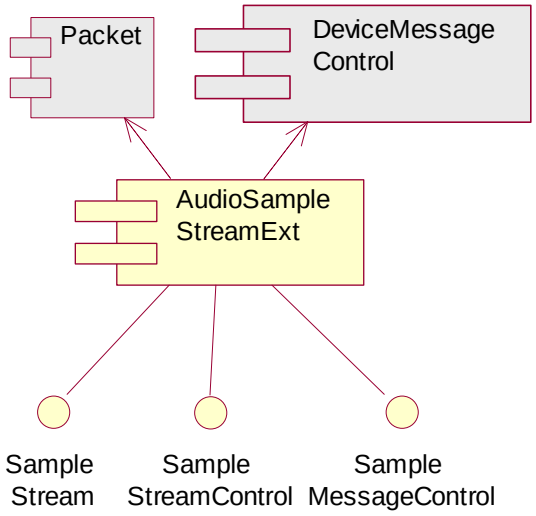

Figure 13 – Audio Sample Stream Extension Component Diagram

B.5.1 Data Types

None

B.5.2 Enumerations

There no are additions to the base API.

B.5.3 Exceptions

There no are additions to the base API.

B.5.4 Structures

There no are additions to the base API.

Appendix B.A Abbreviations and Acronyms

The following lists additional abbreviations and acronyms not defined in the base API Appendix A.A, Abbreviations and Acronyms.

Tx	Transmit
Rx	Receive

Appendix B.B Performance Specification

Table 6 provides a template for the generic performance specification for the *Audio Sample Stream* Extension API which will be documented in the service or device using the interface. This performance specification corresponds to the port diagram in Figure 7.

Table 6 – Audio Sample Stream Extension Performance Specification

Specification	Description	Units	Value
Worst Case Command Execution Time for pushPacket() on audio_sample_stream_provides_port	*	*	*
Worst Case Command Execution Time for pushPacket() on audio_sample_stream_uses_port	*	*	*
Worst Case Command Execution Time for audio_sample_stream_provides_port	*	*	*
Worst Case Command Execution Time for audio_sample_stream_uses_port	*	*	*
Worst Case Command Execution Time for audio_sample_stream_ctrl_provides_port	*	*	*
Worst Case Command Execution Time for audio_sample_stream_ctrl_uses_port	*	*	*
Worst Case Command Execution Time for audio_sample_msg_ctrl_provides_port	*	*	*

Note: (*) These values should be filled in by individual developers.