Joint Tactical Networking Center Standard
Vocoder Service
Application Program Interface

Version: 1.4
26 February 2015
# Revision History

<table>
<thead>
<tr>
<th>Version</th>
<th>Description</th>
<th>Last Modified Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1</td>
<td>Initial release</td>
<td>07-Apr-2006</td>
</tr>
<tr>
<td>0.2</td>
<td>Updates based on Increment 4 ICWG.</td>
<td>04-May-2006</td>
</tr>
<tr>
<td>0.3</td>
<td>Updates based on Increment 4c review comments.</td>
<td>19-May-2006</td>
</tr>
<tr>
<td>0.4</td>
<td>Updates based on Increment 4c ICWG.</td>
<td>08-Jun-2006</td>
</tr>
<tr>
<td>0.5</td>
<td>Updates based on Increment 4d comment.</td>
<td>11-Jul-2006</td>
</tr>
<tr>
<td>1.0</td>
<td>Updates based on Increment 4d ICWG.</td>
<td>07-Aug-2006</td>
</tr>
</tbody>
</table>

**ICWG Approved**

| 1.1     | Added SPEEX Extension | 08-Aug-2006 |

**ICWG Approved**

<table>
<thead>
<tr>
<th>1.1.1</th>
<th>Preparation for public release</th>
<th>29-Mar-2007</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1.1.1</td>
<td>Changed all references from Mode1 to KY57 in CVSD Extension text and VocoderCvsdExt.idl</td>
<td>02-Aug-2007</td>
</tr>
<tr>
<td></td>
<td>Corrected ifndef declaration in VocoderAudioStreamExt.idl</td>
<td></td>
</tr>
</tbody>
</table>

**ICWG Approved**

<table>
<thead>
<tr>
<th>1.2</th>
<th>Change Proposals:</th>
<th>10-Nov-2009</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>-Added Section G: G.729 Extension</td>
<td></td>
</tr>
<tr>
<td></td>
<td>-Added Section H: MELPe Extension</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>1.2</th>
<th>Community Review:</th>
<th>14-Jan-2010</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>-Section G.1 Introduction: Clarified the use of Annex A and B.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>-Section H: Renamed the extension and associated text/IDL to MELPe</td>
<td></td>
</tr>
</tbody>
</table>

**ICWG Approved**

| 1.2     | No further changes | 28-Jan-2010 |

<table>
<thead>
<tr>
<th>1.2.1</th>
<th>Preparation for public release</th>
<th>06-Apr-2010</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.2.1</td>
<td>No further changes</td>
<td>04-Mar-2011</td>
</tr>
</tbody>
</table>

**ICWG Approved**

<table>
<thead>
<tr>
<th>1.3</th>
<th>Change Proposals:</th>
<th>23-Mar-2011</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>-Added Section I: MELPe with DTX and VAD Extension:</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Added clarification text in H.1: Introduction</td>
<td></td>
</tr>
</tbody>
</table>

**ICWG Approved**

(Expedited Process)
<table>
<thead>
<tr>
<th>Version</th>
<th>Description</th>
<th>Last Modified Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.3.1</td>
<td>Errata:</td>
<td>15-Apr-2001</td>
</tr>
<tr>
<td></td>
<td>- Corrected case on boolean type for enableMDV field of the MelpeDtxVadConfig struct in VocoderMelpeDtxVadExt.idl</td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>ICWG Approved</strong></td>
<td></td>
</tr>
<tr>
<td>1.3.2</td>
<td>Change Proposals:</td>
<td>28-Jul-2011</td>
</tr>
<tr>
<td>&lt;Draft&gt;</td>
<td>- Deprecated Section H.MELPe Extension</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- Updated A.2.4.3 Switch Vocoder Algorithms sequence description</td>
<td></td>
</tr>
<tr>
<td>1.3.2</td>
<td>Misc. redlines.</td>
<td>24-Aug-2011</td>
</tr>
<tr>
<td>&lt;Final Draft&gt;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.3.2</td>
<td>- Updated A.2.4.3 Switch Vocoder Algorithms sequence description to clarify that the selected algorithm should not be reconfigured during the stream.</td>
<td>31-Aug-2011</td>
</tr>
<tr>
<td></td>
<td><strong>ICWG Approved</strong></td>
<td></td>
</tr>
<tr>
<td>1.3.3</td>
<td>Preparation for public release</td>
<td>26-Jun-2013</td>
</tr>
<tr>
<td></td>
<td><strong>ICWG Approved</strong></td>
<td></td>
</tr>
<tr>
<td>1.4</td>
<td>Added Section J G.711 vocoder extension</td>
<td>28-Jan-2015</td>
</tr>
<tr>
<td>&lt;DRAFT&gt;</td>
<td>Deleted content of Section H. MELPe Extension and associated IDL which was deprecated in version 1.3.2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Updated document to reflect program/organization changes Added Vocoder References</td>
<td></td>
</tr>
<tr>
<td>1.4</td>
<td><strong>ICWG Approved</strong></td>
<td>26 Feb 2015</td>
</tr>
<tr>
<td></td>
<td>Updated Table 3 to remove references to Worst Case Command Execution Time for non-pushPacket ports</td>
<td></td>
</tr>
</tbody>
</table>
Table of Contents

A. VOCODER SERVICE ............................................................................................................ 14
B. MELP EXTENSION ............................................................................................................. 41
C. LPC EXTENSION .................................................................................................................. 50
D. CVSD EXTENSION ............................................................................................................. 60
E. SPEEX EXTENSION ............................................................................................................. 69
F. VOCODER AUDIO STREAM EXTENSION ......................................................................... 74
G. G.729 EXTENSION ............................................................................................................. 83
H. MELPE EXTENSION <DEPRECATED> ............................................................................... 92
I. MELPE WITH DTX AND VAD EXTENSION ..................................................................... 93
J. G.711 EXTENSION ........................................................................................................... 106
# Table of Contents

A. VOCODER SERVICE .................................................................................................................. 14

A.1 Introduction............................................................................................................................. 14
  A.1.1 Overview ............................................................................................................................ 15
  A.1.2 Service Layer Description ............................................................................................... 16
    A.1.2.1 Vocoder Service API Port Connections ..................................................................... 16
  A.1.3 Modes of Service .............................................................................................................. 16
  A.1.4 Service States ................................................................................................................... 17
    A.1.4.1 Vocoder Service Streaming State Diagram ................................................................. 17
  A.1.5 Referenced Documents ................................................................................................... 17
    A.1.5.1 Government Documents ............................................................................................ 17
    A.1.5.2 Commercial Standards ............................................................................................... 18

A.2 Services ................................................................................................................................ 19
  A.2.1 Provide Services ............................................................................................................... 19
  A.2.2 Use Services .................................................................................................................... 20
  A.2.3 Interface Modules .......................................................................................................... 22
    A.2.3.1 Vocoder ....................................................................................................................... 22
  A.2.4 Sequence Diagrams ........................................................................................................ 23
    A.2.4.1 Vocoder Service Produces Data Sequence ................................................................. 23
    A.2.4.2 Vocoder Service Consumes Data Sequence ............................................................... 24
    A.2.4.3 Switch Vocoder Algorithms ....................................................................................... 25
    A.2.4.4 Abort Transmit Data ................................................................................................... 26

A.3 Service Primitives and Attributes ........................................................................................ 28
  A.3.1 Vocoder::Ctrl .................................................................................................................... 28
    A.3.1.1 getAlgorithmsSupported Operation ......................................................................... 28
    A.3.1.2 getLoopback Operation .............................................................................................. 29
    A.3.1.3 setLoopback Operation ............................................................................................... 30
    A.3.1.4 getTxAlgorithm Operation ......................................................................................... 31
    A.3.1.5 getRxAlgorithm Operation ......................................................................................... 32
    A.3.1.6 setTxAlgorithm Operation ......................................................................................... 33
    A.3.1.7 setRxAlgorithm Operation ......................................................................................... 34
    A.3.1.8 abortTx Operation ....................................................................................................... 35
  A.3.2 Vocoder::VocoderPacketConsumer .................................................................................. 35
  A.3.3 Vocoder::VocoderPacketProducer ................................................................................... 35

A.4 IDL ......................................................................................................................................... 36
  A.4.1 Vocoder ............................................................................................................................ 36

A.5 UML ....................................................................................................................................... 38
  A.5.1 Data Types ....................................................................................................................... 39
  A.5.2 Enumerations ................................................................................................................... 39
    A.5.2.1 Vocoder::Algorithm .................................................................................................... 39
    A.5.2.2 Vocoder::AlgorithmSequence .................................................................................... 39
  A.5.3 Exceptions ....................................................................................................................... 39
  A.5.4 Structures ........................................................................................................................ 39

Appendix A.A – Abbreviations and Acronyms ........................................................................... 40
Appendix A.B – Performance Specification .................................................................................. 40
B. MELP EXTENSION ........................................................................................................... 41
B.1 Introduction ..................................................................................................................... 41
   B.1.1 Overview .................................................................................................................. 41
   B.1.2 Service Layer Description ....................................................................................... 41
      B.1.2.1 Vocoder Service API MELP Extension Port Connections ......................... 41
   B.1.3 Modes of Service ....................................................................................................... 42
   B.1.4 Service States .......................................................................................................... 42
   B.1.5 Referenced Documents ........................................................................................... 42
B.2 Services .......................................................................................................................... 43
   B.2.1 Provide Services ....................................................................................................... 43
   B.2.2 Use Services ............................................................................................................ 43
   B.2.3 Interface Modules .................................................................................................... 43
      B.2.3.1 Vocoder .............................................................................................................. 43
   B.2.4 Sequence Diagrams ................................................................................................. 43
B.3 Service Primitives and Attributes .................................................................................. 44
   B.3.1 Vocoder::MELP ....................................................................................................... 44
      B.3.1.1 configureMelp Operation ............................................................................... 44
      B.3.1.2 getMelpConfig Operation .............................................................................. 45
B.4 IDL ................................................................................................................................... 46
   B.4.1 VocoderMelpExt ....................................................................................................... 46
B.5 UML .................................................................................................................................. 47
   B.5.1 Data Types ............................................................................................................... 48
   B.5.2 Enumerations .......................................................................................................... 48
      B.5.2.1 Vocoder::Algorithm ......................................................................................... 48
   B.5.3 Exceptions ................................................................................................................. 48
   B.5.4 Structures ................................................................................................................. 48
      B.5.4.1 MELP::Config .................................................................................................. 48
Appendix B.A – Abbreviations and Acronyms ................................................................. 49
Appendix B.B – Performance Specification ........................................................................ 49
C. LPC EXTENSION .............................................................................................................. 50
C.1 Introduction .................................................................................................................... 50
   C.1.1 Overview .................................................................................................................. 50
   C.1.2 Service Layer Description ....................................................................................... 50
      C.1.2.1 Vocoder Service API LPC Extension Port Connections .............................. 50
   C.1.3 Modes of Service ..................................................................................................... 51
   C.1.4 Service States .......................................................................................................... 51
   C.1.5 Referenced Documents ........................................................................................... 51
C.2 Services .......................................................................................................................... 52
   C.2.1 Provide Services ....................................................................................................... 52
   C.2.2 Use Services ............................................................................................................ 52
   C.2.3 Interface Modules .................................................................................................... 52
      C.2.3.1 Vocoder .............................................................................................................. 52
   C.2.4 Sequence Diagrams ................................................................................................. 52
C.3 Service Primitives and Attributes .................................................................................. 53
   C.3.1 Vocoder::LPC .......................................................................................................... 53
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>G.1.1</td>
<td>Overview</td>
<td>83</td>
</tr>
<tr>
<td>G.1.2</td>
<td>Service Layer Description</td>
<td>83</td>
</tr>
<tr>
<td>G.1.2.1</td>
<td>Vocoder Service API G.729 Extension Port Connections</td>
<td>83</td>
</tr>
<tr>
<td>G.1.3</td>
<td>Modes of Service</td>
<td>84</td>
</tr>
<tr>
<td>G.1.4</td>
<td>Service States</td>
<td>84</td>
</tr>
<tr>
<td>G.1.5</td>
<td>Referenced Documents</td>
<td>84</td>
</tr>
<tr>
<td>G.2</td>
<td>Services</td>
<td>85</td>
</tr>
<tr>
<td>G.2.1</td>
<td>Provide Services</td>
<td>85</td>
</tr>
<tr>
<td>G.2.2</td>
<td>Use Services</td>
<td>85</td>
</tr>
<tr>
<td>G.2.3</td>
<td>Interface Modules</td>
<td>85</td>
</tr>
<tr>
<td>G.2.3.1</td>
<td>Vocoder</td>
<td>85</td>
</tr>
<tr>
<td>G.2.4</td>
<td>Sequence Diagrams</td>
<td>85</td>
</tr>
<tr>
<td>G.3</td>
<td>Service Primitives and Attributes</td>
<td>86</td>
</tr>
<tr>
<td>G.3.1</td>
<td>Vocoder::G729</td>
<td>86</td>
</tr>
<tr>
<td>G.3.1.1</td>
<td>configureG729 Operation</td>
<td>86</td>
</tr>
<tr>
<td>G.3.1.2</td>
<td>getG729Config Operation</td>
<td>87</td>
</tr>
<tr>
<td>G.4</td>
<td>IDL</td>
<td>88</td>
</tr>
<tr>
<td>G.4.1</td>
<td>VocoderG729Ext</td>
<td>88</td>
</tr>
<tr>
<td>G.5</td>
<td>UML</td>
<td>89</td>
</tr>
<tr>
<td>G.5.1</td>
<td>Data Types</td>
<td>90</td>
</tr>
<tr>
<td>G.5.2</td>
<td>Enumerations</td>
<td>90</td>
</tr>
<tr>
<td>G.5.2.1</td>
<td>Vocoder::Algorithm</td>
<td>90</td>
</tr>
<tr>
<td>G.5.3</td>
<td>Exceptions</td>
<td>90</td>
</tr>
<tr>
<td>G.5.4</td>
<td>Structures</td>
<td>90</td>
</tr>
<tr>
<td>G.5.4.1</td>
<td>G729::Config</td>
<td>90</td>
</tr>
<tr>
<td>Appendix G.A</td>
<td>Abbreviations and Acronyms</td>
<td>91</td>
</tr>
<tr>
<td>Appendix G.B</td>
<td>– Performance Specification</td>
<td>91</td>
</tr>
<tr>
<td>H.</td>
<td>MELPE EXTENSION &lt;DEPRECATED&gt;</td>
<td>92</td>
</tr>
<tr>
<td>I.</td>
<td>MELPE WITH DTX AND VAD EXTENSION</td>
<td>93</td>
</tr>
<tr>
<td>I.1</td>
<td>Introduction</td>
<td>93</td>
</tr>
<tr>
<td>I.1.1</td>
<td>Overview</td>
<td>93</td>
</tr>
<tr>
<td>I.1.2</td>
<td>Service Layer Description</td>
<td>94</td>
</tr>
<tr>
<td>I.1.2.1</td>
<td>Vocoder Service API MELPe Extension Port Connections</td>
<td>94</td>
</tr>
<tr>
<td>I.1.3</td>
<td>Modes of Service</td>
<td>94</td>
</tr>
<tr>
<td>I.1.4</td>
<td>Service States</td>
<td>94</td>
</tr>
<tr>
<td>I.1.5</td>
<td>Referenced Documents</td>
<td>94</td>
</tr>
<tr>
<td>I.2</td>
<td>Services</td>
<td>95</td>
</tr>
<tr>
<td>I.2.1</td>
<td>Provide Services</td>
<td>95</td>
</tr>
<tr>
<td>I.2.2</td>
<td>Use Services</td>
<td>95</td>
</tr>
<tr>
<td>I.2.3</td>
<td>Interface Modules</td>
<td>95</td>
</tr>
<tr>
<td>I.2.3.1</td>
<td>Vocoder</td>
<td>95</td>
</tr>
<tr>
<td>I.2.4</td>
<td>Sequence Diagrams</td>
<td>95</td>
</tr>
<tr>
<td>I.3</td>
<td>Service Primitives and Attributes</td>
<td>96</td>
</tr>
<tr>
<td>I.3.1</td>
<td>Vocoder::Melpe</td>
<td>96</td>
</tr>
<tr>
<td>I.3.1.1</td>
<td>configureMelpe Operation</td>
<td>96</td>
</tr>
</tbody>
</table>
I.5 UML .......................................................................................................................... 102
  I.5.1 Data Types ........................................................................................................ 103
  I.5.2 Enumerations ..................................................................................................... 103
    I.5.2.1 Vocoder::Algorithm .................................................................................... 103
    I.5.2.2 Vocoder::Melpe::FrameType ................................................................. 103
    I.5.2.3 Vocoder::Melpe::RestartType ................................................................. 103
  I.5.3 Exceptions .......................................................................................................... 103
  I.5.4 Structures .......................................................................................................... 104
    I.5.4.1 Melpe::Config ........................................................................................... 104
    I.5.4.2 Melpe::MelpeDtxVadConfig ................................................................. 104
Appendix I.A – Abbreviations and Acronyms .............................................................. 105
Appendix I.B – Performance Specification ................................................................. 105

J. G.711 EXTENSION ..................................................................................................... 106
  J.1 Introduction .......................................................................................................... 106
    J.1.1 Overview ......................................................................................................... 106
    J.1.2 Service Layer Description ............................................................................ 107
      J.1.2.1 Vocoder Service API G.711 Extension Port Connections ................. 107
    J.1.3 Modes of Service .......................................................................................... 107
    J.1.4 Service States ................................................................................................. 107
    J.1.5 Referenced Documents ............................................................................... 107
  J.2 Services .................................................................................................................. 108
    J.2.1 Provide Services ............................................................................................ 108
    J.2.2 Use Services ................................................................................................... 108
    J.2.3 Interface Modules ........................................................................................ 108
      J.2.3.1 Vocoder .................................................................................................... 108
      J.2.4 Sequence Diagram ..................................................................................... 109
  J.3 Service Primitives and attributes .......................................................................... 109
    J.3.1 Vocoder::G711 ............................................................................................... 109
      J.3.1.1 configureG711 Operation ...................................................................... 109
      J.3.1.2 getG711Config Operation ................................................................... 110
  J.4 IDL ......................................................................................................................... 111
    J.4.1 VocoderG711Ext ............................................................................................. 111
  J.5 UML ....................................................................................................................... 112
    J.5.1 Data Types ..................................................................................................... 112
    J.5.2 Enumerations .................................................................................................. 112
      J.5.2.1 Vocoder::Algorithm .............................................................................. 112
      J.5.2.2 Vocoder::G711::EncType .................................................................... 112
    J.5.3 Exceptions ...................................................................................................... 114
    J.5.4 Structures .................................................................................................... 114
      J.5.4.1 G711::Config ......................................................................................... 114
Lists of Figures

FIGURE 1 – IMPLICIT CONNECTIONS TO THE AUDIO PORT DEVICE ........................................14
FIGURE 2 – EXPLICIT CONNECTIONS TO THE AUDIO PORT DEVICE ...............................15
FIGURE 3 – VOCODER SERVICE API PORT DIAGRAM ..........................................................16
FIGURE 4 – VOCODER SERVICE STREAMING STATE DIAGRAM ...........................................17
FIGURE 5 – VOCODER SERVICE API CLASS DIAGRAM ...........................................................22
FIGURE 6 – CTRL INTERFACE CLASS DIAGRAM ...............................................................22
FIGURE 7 – VOCODER_PACKET_PRODUCER INTERFACE CLASS DIAGRAM ......................23
FIGURE 8 – VOCODER_PACKET_CONSUMER INTERFACE CLASS DIAGRAM .......................23
FIGURE 9 – VOCODER SERVICE PRODUCES DATA SEQUENCE DIAGRAM ...........................24
FIGURE 10 – VOCODER SERVICE CONSUMES DATA ............................................................24
FIGURE 11 – SWITCH VOCODER ALGORITHMS SEQUENCE DIAGRAM ...............................26
FIGURE 12 – ABORT TRANSMIT DATA SEQUENCE DIAGRAM .............................................27
FIGURE 13 – VOCODER SERVICE API COMPONENT DIAGRAM .............................................38
FIGURE 14 – VOCODER SERVICE API MELP EXTENSION PORT DIAGRAM ..........................42
FIGURE 15 – MELP EXTENSION CLASS DIAGRAM .............................................................43
FIGURE 16 – MELP EXTENSION COMPONENT DIAGRAM .....................................................47
FIGURE 17 – VOCODER SERVICE API LPC EXTENSION PORT DIAGRAM ............................51
FIGURE 18 – LPC EXTENSION CLASS DIAGRAM ..................................................................52
FIGURE 19 – LPC EXTENSION COMPONENT DIAGRAM ..........................................................56
FIGURE 20 – VOCODER SERVICE API CVSD EXTENSION PORT DIAGRAM ...........................61
FIGURE 21 – CVSD EXTENSION CLASS DIAGRAM ...............................................................62
FIGURE 22 – CVSD EXTENSION COMPONENT DIAGRAM ......................................................66
FIGURE 23 – SPEEX EXTENSION COMPONENT DIAGRAM ......................................................72
FIGURE 24 – VOCODER SERVICE API VOCODER AUDIO STREAM EXTENSION PORT DIAGRAM ....75
FIGURE 25 – VOCODER_AUDIO_STREAM EXTENSION INTERFACE CLASS DIAGRAM .............77
FIGURE 26 – AUDIO_SAMPLE_STREAM INTERFACE DIAGRAM ...........................................78
FIGURE 27 – AUDIO_SAMPLE_CTRL INTERFACE DIAGRAM .................................................78
FIGURE 28 – VOCODER AUDIO STREAM EXTENSION COMPONENT DIAGRAM ........................80
FIGURE 29 – VOCODER SERVICE API G.729 EXTENSION PORT DIAGRAM ............................84
FIGURE 30 – G.729 EXTENSION CLASS DIAGRAM ..................................................................85
FIGURE 31 – G.729 EXTENSION COMPONENT DIAGRAM ......................................................89
FIGURE 32 – VOCODER SERVICE MELP WITH DTX AND VAD EXTENSION PORT DIAGRAM ....94
FIGURE 33 – MELP with DTX AND VAD EXTENSION CLASS DIAGRAM ..................................95
FIGURE 34 – MELP WITH DTX AND VAD EXTENSION COMPONENT DIAGRAM ........................102
FIGURE 35 – VOCODER SERVICE API G.711 EXTENSION PORT DIAGRAM ............................107
FIGURE 36 – G.711 EXTENSION CLASS DIAGRAM ..................................................................108
FIGURE 37 – G.711 EXTENSION COMPONENT DIAGRAM ......................................................112
List of Tables

Table 1 – Vocoder Service API Provides Service Interface .................................................. 19
Table 2 – Vocoder Service API Uses Service Interface .......................................................... 20
Table 3 – Vocoder Service Performance Specification ........................................................... 40
Table 4 – MELP Extension Provide Service Interface ............................................................. 43
Table 5 – MELP Extension Performance Specification ............................................................ 49
Table 6 – LPC Extension Provide Service Interface ................................................................. 52
Table 7 – LPC Extension Performance Specification .............................................................. 59
Table 8 – CVSD Extension Provide Service Interface .............................................................. 62
Table 9 – CVSD Extension Performance Specification ............................................................ 68
Table 10 – SPEEX Extension Performance Specification ....................................................... 73
Table 11 – Vocoder Audio Stream Extension Provide Service Interface .................................. 76
Table 12 – Vocoder Audio Stream Extension Use Service Interface ....................................... 76
Table 13 – Vocoder Audio Stream Extension Performance Specification .............................. 82
Table 14 – G.729 Extension Provide Service Interface ............................................................ 85
Table 15 – G.729 Extension Performance Specification ........................................................... 91
Table 16 – MELPe with DTX and VAD Extension Provide Service Interface ......................... 95
Table 17 – MELPe with DTX and VAD Extension Performance Specification .................... 105
Table 18 – G.711 Extension Provide Service Interface ............................................................ 108
Table 19 – G.711 Extension Performance Specification .......................................................... 115
A. VOCODER SERVICE

A.1 INTRODUCTION

The Vocoder Service API encapsulates vocoding capabilities that are common across all waveforms and applications. The Vocoder Service API supports loopback operations, provides for the transfer of encoded/decoded bit streams to and from the service user, and defines operations to select algorithms supplied by the vocoder.

It should be noted that the use of the base Vocoder Service API and codec extensions require an implicit or explicit connection with the Audio Port Device API [3]. Implicit connections are defined by the platform implementation (see Figure 1). Explicit connections are defined using the Vocoder Audio Stream API Extension defined in section F.1 (see Figure 2).

![Figure 1 – Implicit Connections to the Audio Port Device](image-url)
The provided IDL (see A.4.1) makes use of the CORBA IDL ‘module’ construct to allow the aggregation of required API interfaces within a common namespace. This may require the client to narrow to an inherited interface rather than the interface name. The data transport and control interfaces should be advertised at the common data exchange interface (e.g. Packet::OctetStream) to promote typeless binding to the component. Implementations may elect to provide the optional port bindings at the component interface (e.g. Vocoder::VocoderPacketConsumer) as specified.

This document defines a common set of Vocoder Service API provide services and interfaces required by most Tactical Radio (TR) sets.

Using the design principle of “least privilege”, device ports may be limited in the number of interfaces supported by that port. Each port will have an IDL file to specify its interfaces, but multiple ports could have identical IDL files.

In general, a design guideline is to isolate the interfaces used by the waveform from the interfaces accessed by the TR set.

**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 Vocoder Service API.

d. Section A.4, *IDL*. 
A.1.2 Service Layer Description

A.1.2.1 Vocoder Service API Port Connections

The following figure shows the port connections for the Vocoder Service API.

Note: All port names are for reference only.

Figure 3 – Vocoder Service API Port Diagram

Vocoder Service API Provides Ports Definitions

vocoder_packet_consumer_provides_port is provided by the Vocoder Service to consume packets through the pushPacket operation.

vocoder_packet_producer_provides_port is provided by the Vocoder Service to set the payload size by the Service User.

vocoder_ctrl_provides_port is provided by the Vocoder Service to support loopback operations, get the codec algorithms supported, and set/get the codec algorithm.

Vocoder Service API Uses Ports Definitions

vocoder_packet_producer_uses_port is used by the Vocoder Service to set the payload size of the incoming packets from the Device User.

vocoder_packet_consumer_uses_port is used by the Vocoder Service to push packets to the Service User.

A.1.3 Modes of Service

Not applicable
A.1.4 Service States

A.1.4.1 Vocoder Service Streaming State Diagram

The Vocoder Service streaming states are illustrated in the following diagram. The two streaming states of the Vocoder Service are as follow:

- **STREAMING** - The state transitioned to when the `endOfStream` indicator in the ::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 definition of `Packet::StreamControlType`.

![Vocoder Service Streaming State Diagram](image)

Figure 4 – Vocoder Service Streaming 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

A.1.5.2 Commercial Standards

None
A.2 SERVICES

A.2.1 Provide Services

The Vocoder Service API interface provides services consisting of the following service ports, interfaces, and primitives, which can be called by other client components. Detailed definition of the interfaces and services shaded in gray is provided by the Packet API [2].

Table 1 – Vocoder Service API Provides Service Interface

<table>
<thead>
<tr>
<th>Service Group (Port Name)</th>
<th>Service (Interface Provided)</th>
<th>Primitives (Provided)</th>
<th>Parameter Name or Return Value</th>
<th>Valid Range</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>getMaxPayloadSize()</td>
<td>Return Value</td>
<td>1 to 16383</td>
</tr>
<tr>
<td></td>
<td></td>
<td>getMinPayloadSize()</td>
<td>Return Value</td>
<td>0 to 512</td>
</tr>
<tr>
<td></td>
<td></td>
<td>getDesiredPayloadSize()*</td>
<td>Return Value</td>
<td>1 to 16383</td>
</tr>
<tr>
<td></td>
<td></td>
<td>getMinOverrideTimeout()*</td>
<td>Return Value</td>
<td>0 to 50</td>
</tr>
<tr>
<td>vocoder_packet_producer_provides_port</td>
<td>Vocoder::PacketProducer Packet::PayloadControl</td>
<td>setMaxPayloadSize()</td>
<td>maxPayloadSize</td>
<td>1 to 16383</td>
</tr>
<tr>
<td></td>
<td></td>
<td>setMinPayloadSize()</td>
<td>minPayloadSize</td>
<td>0 to 512</td>
</tr>
<tr>
<td></td>
<td></td>
<td>setDesiredPayloadSize()*</td>
<td>desiredPayloadSize</td>
<td>1 to 16383</td>
</tr>
<tr>
<td></td>
<td></td>
<td>setMinOverrideTimeout()*</td>
<td>minOverrideTimeout</td>
<td>0 to 50</td>
</tr>
<tr>
<td>vocoder_ctrl_provides_port</td>
<td>Vocoder::Ctrl</td>
<td>getLoopback()</td>
<td>see A.3, Service Primitives and Attributes</td>
<td>see A.3, Service Primitives and Attributes</td>
</tr>
<tr>
<td></td>
<td></td>
<td>setLoopback()</td>
<td>see A.3, Service Primitives and Attributes</td>
<td>see A.3, Service Primitives and Attributes</td>
</tr>
</tbody>
</table>
## A.2.2 Use Services

The Vocoder Service API Use Service set consists of the following service ports, interfaces, and primitives. Since the Vocoder Service API acts as a client with respect to these services from other components, the corresponding service ports that are required to be connected are applied by the server component. The Vocoder Service API uses the Port Name as connectionId for the connection. Detailed definition of the interfaces and services shaded in gray is provided by the Packet API [2].

### Table 2 – Vocoder Service API Uses Service Interface

<table>
<thead>
<tr>
<th>Service Group (Port Name)</th>
<th>Service (Interface Provided)</th>
<th>Primitives (Provided)</th>
<th>Parameter Name or Return Value</th>
<th>Valid Range</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Packet::OctetStream</td>
<td>getMaxPayloadSize()</td>
<td>Return Value</td>
<td>1 to 16383</td>
</tr>
<tr>
<td>Service Group (Port Name)</td>
<td>Service (Interface Provided)</td>
<td>Primitives (Provided)</td>
<td>Parameter Name or Return Value</td>
<td>Valid Range</td>
</tr>
<tr>
<td>--------------------------</td>
<td>------------------------------</td>
<td>-----------------------</td>
<td>-------------------------------</td>
<td>-------------</td>
</tr>
<tr>
<td></td>
<td></td>
<td>getMinPayloadSize()</td>
<td>Return Value</td>
<td>0 to 512</td>
</tr>
<tr>
<td></td>
<td></td>
<td>getDesiredPayloadSize()</td>
<td>Return Value</td>
<td>1 to 16383</td>
</tr>
<tr>
<td></td>
<td></td>
<td>getMinOverrideTimeout()</td>
<td>Return Value</td>
<td>0 to 50</td>
</tr>
<tr>
<td>vocoder_packet_</td>
<td>Vocoder::PacketProducer</td>
<td>setMaxPayloadSize()</td>
<td>maxPayloadSize</td>
<td>1 to 16383</td>
</tr>
<tr>
<td>producer_uses_port</td>
<td>Packet::PayloadControl</td>
<td>setMinPayloadSize()</td>
<td>minPayloadSize</td>
<td>0 to 512</td>
</tr>
<tr>
<td></td>
<td></td>
<td>setDesiredPayloadSize()</td>
<td>desiredPayloadSize</td>
<td>1 to 16383</td>
</tr>
<tr>
<td></td>
<td></td>
<td>setMinOverrideTimeout()</td>
<td>minOverrideTimeout</td>
<td>0 to 50</td>
</tr>
</tbody>
</table>
A.2.3 Interface Modules

A.2.3.1 Vocoder

The Vocoder Service API interface class diagram is shown below. Interfaces shaded in gray are defined in the Packet API [2].

Figure 5 – Vocoder Service API Class Diagram

A.2.3.1.1 Ctrl Interface Description

The Ctrl interface is shown in the interface class diagram below. The Ctrl interface provides the user the ability to set/get loopback, obtain the list of codec algorithms supported, set/get the receive/transmit algorithms, and abort the transmit stream.

Figure 6 – Ctrl Interface Class Diagram

A.2.3.1.2 VocoderPacketProducer Interface Description

The interface design of the VocoderPacketProducer is shown in the interface class diagram below. It extends the Packet::PayloadControl interface defined in the Packet API [2] to configure the packet sizes. Interfaces shaded in gray are defined in the Packet API [2].
A.2.3.1.3 VocoderPacketConsumer Interface Description

The interface design of VocoderPacketConsumer is shown in the interface class diagram below. It extends the Packet::OctetStream interface defined in the Packet API [2] to provide the ability to transfer codec streams and status the packet sizes. Interfaces shaded in gray are defined in the Packet API [2].

A.2.4 Sequence Diagrams

A.2.4.1 Vocoder Service Produces Data Sequence

Description

The Vocoder Service produces data as shown in the Vocoder Service Produces Data Sequence Diagram. When data is available, the Vocoder Service will encode the data, if necessary, and push it to the Service User using the pushPacket method. The vocoder algorithm used when encoding the data can be configured by the Service User.

Pre-conditions
The Vocoder Service is ENABLED.

**Post-conditions**

The Vocoder Service has pushed the encoded data to the Service User.

![Diagram of Vocoder Service Produces Data Sequence](image)

**Figure 9 – Vocoder Service Produces Data Sequence Diagram**

### A.2.4.2 Vocoder Service Consumes Data Sequence

**Description**

The Vocoder Service consumes data as shown in the Vocoder Service Consumes Data Sequence Diagram. The Service User transfers data to the Vocoder Service through the pushPacket method. The Vocoder Service decodes the outgoing data, if necessary, and sends it to the Data Sink. The vocoder algorithm used when decoding the data can be configured by the Service User.

**Pre-conditions**

The Vocoder Service is ENABLED.

**Post-conditions**

The decoded data has been sent to the Data Sink.

![Diagram of Vocoder Service Consumes Data](image)

**Figure 10 – Vocoder Service Consumes Data**
A.2.4.3 Switch Vocoder Algorithms

Description

A Service User switches between multiple vocoder algorithms as shown in the Switch Vocoder Algorithm Sequence Diagram (see Figure 11). Upon startup, the getTxAlgorithm() will return ALG_NONE indicating that no algorithm has been selected for the transmit path. The Service User initially sets the transmit algorithm to LPC by calling setTxAlgorithm. When the Vocoder Service pushes packets on the transmit path the Vocoder Service encodes the packets using the LPC algorithm configuration last provided by the Service User via the configureLpc Operation.

When required, the Service User will call setTxAlgorithm again to switch the transmit algorithm from LPC to CVSD. A call to getTxAlgorithm will confirm that the transmit algorithm is now CVSD. When the Vocoder Service pushes packets on the transmit path the Vocoder Service encodes the packets using the CVSD algorithm configuration last provided by the Service User via the configureCvsd Operation.

Note: LPC and CVSD have been used as example algorithms. The same sequence would apply for the receive path. Algorithms should not be switched during the stream, i.e. when the endOfStream indicator in the Packet::StreamControlType of the pushPacket operation is set to FALSE (see section A.1.4.1 Vocoder Service Streaming State Diagram). The selected algorithm should not be reconfigured during the stream.

Pre-conditions

The Vocoder Service is ENABLED. The LPC and CVSD algorithms are configured on the Vocoder Service via the respective configure operations (see section C.3.1.1 configureLpc Operation and section D.3.1.1 configureCvsd Operation).

Post-conditions

The Vocoder Service is configured to apply the CVSD algorithm on the transmit path.
A.2.4.4 Abort Transmit Data

Description

A Service User aborts the transmit data stream as shown in the Abort Transmit Data Sequence Diagram. After the abortTx is called, all data is purged from the device. The Vocoder Service will return a 0 length pushPacket operation with the “purge” and “endOfStream” flag set to TRUE.

Pre-conditions

The Device is in the ENABLED state.

Post-conditions

The transmit data is aborted.
All data is purged from the Vocoder Service

Vocoder Service sends a 0 length pushPacket with the "purge" and "endOfStream" flag set to TRUE

Figure 12 – Abort Transmit Data Sequence Diagram
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 Vocoder::Ctrl

A.3.1.1 getAlgorithmsSupported Operation

The `getAlgorithmsSupported` operation provides the ability to get all algorithms supported by the vocoder.

A.3.1.1.1 Synopsis

```
AlgorithmSequence getAlgorithmsSupported();
```

A.3.1.1.2 Parameters

None

A.3.1.1.3 State

Not applicable

A.3.1.1.4 New State

Not applicable

A.3.1.1.5 Return Value

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AlgorithmSequence</td>
<td>A sequence containing all algorithms supported by the vocoder.</td>
</tr>
</tbody>
</table>

(see A.5.2.2)

A.3.1.1.6 Originator

Service User

A.3.1.1.7 Exceptions

None
A.3.1.2 getLoopback Operation
The getLoopback operation provides the ability to status the loopback operation.

A.3.1.2.1 Synopsis

boolean getLoopback()

A.3.1.2.2 Parameters
None

A.3.1.2.3 State
Not applicable

A.3.1.2.4 New State
Not applicable

A.3.1.2.5 Return Value

<table>
<thead>
<tr>
<th>Type</th>
<th>Values</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>boolean</td>
<td>TRUE</td>
<td>Loopback enabled.</td>
</tr>
<tr>
<td></td>
<td>FALSE</td>
<td>Loopback disabled.</td>
</tr>
</tbody>
</table>

A.3.1.2.6 Originator
Service User

A.3.1.2.7 Exceptions
None
A.3.1.3 *setLoopback* Operation

The *setLoopback* operation provides the ability to enable or disable loopback.

A.3.1.3.1 Synopsis

\[\text{void setLoopback}(\text{in boolean loopback});\]

A.3.1.3.2 Parameters

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Type</th>
<th>Values</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>loopback</td>
<td>boolean</td>
<td>TRUE</td>
<td>Loopback enabled.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>FALSE</td>
<td>Loopback disabled.</td>
</tr>
</tbody>
</table>

A.3.1.3.3 State

Not applicable

A.3.1.3.4 New State

Not applicable

A.3.1.3.5 Return Value

None

A.3.1.3.6 Originator

Service User

A.3.1.3.7 Exceptions

None
A.3.1.4 getTxAlgorithm Operation
The getTxAlgorithm operation returns the current algorithm applied to the transmit path.

A.3.1.4.1 Synopsis
Algorithm getTxAlgorithm();

A.3.1.4.2 Parameters
None

A.3.1.4.3 State
Not applicable

A.3.1.4.4 New State
Not applicable

A.3.1.4.5 Return Value

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Algorithm (see A.5.2.1)</td>
<td>The algorithm of the transmit path.</td>
</tr>
</tbody>
</table>

A.3.1.4.6 Originator
Service User

A.3.1.4.7 Exceptions
None
A.3.1.5 getRxAlgorithm Operation
The getRxAlgorithm operation returns the current algorithm applied to the receive path.

A.3.1.5.1 Synopsis
Algorithm getRxAlgorithm();

A.3.1.5.2 Parameters
None

A.3.1.5.3 State
Not applicable

A.3.1.5.4 New State
Not applicable

A.3.1.5.5 Return Value

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Algorithm</td>
<td>The algorithm of the receive path.</td>
</tr>
<tr>
<td>(see A.5.2.1)</td>
<td></td>
</tr>
</tbody>
</table>

A.3.1.5.6 Originator
Service User

A.3.1.5.7 Exceptions
None
A.3.1.6 setTxAlgorithm Operation

The setTxAlgorithm operation provides the ability to set the algorithm of the transmit path. In a Vocoder Service supporting multiple algorithms, the setTxAlgorithm also deselects the current “active” algorithm. This allows the user to switch between multiple algorithms. Note: Algorithms should not be set during the stream.

A.3.1.6.1 Synopsis

void setTxAlgorithm( in Algorithm txAlgorithm ) raises(JTRS::Unsupported);

A.3.1.6.2 Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>txAlgorithm</td>
<td>Algorithm</td>
<td>The algorithm of the transmit path.</td>
</tr>
</tbody>
</table>

(see A.5.2.1)

A.3.1.6.3 State

Not applicable

A.3.1.6.4 New State

Not applicable

A.3.1.6.5 Return Value

None

A.3.1.6.6 Originator

Service User

A.3.1.6.7 Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>JTRS::Unsupported</td>
<td>The Algorithm selected is not supported by the Vocoder.</td>
</tr>
</tbody>
</table>

(see JTRS CORBA Types [1])
A.3.1.7 setRxAlgorithm Operation

The setRxAlgorithm operation provides the ability to set the algorithm of the receive path. In a Vocoder Service supporting multiple algorithms, the setRxAlgorithm also deselects the current “active” algorithm. This allows the user to switch between multiple algorithms. Note: Algorithms should not be set during the stream.

A.3.1.7.1 Synopsis

void setRxAlgorithm( in Algorithm rxAlgorithm ) raises(JTRS::Unsupported);

A.3.1.7.2 Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rxAlgorithm</td>
<td>Algorithm</td>
<td>The algorithm of the receive path.</td>
</tr>
<tr>
<td></td>
<td>(see A.5.2.1)</td>
<td></td>
</tr>
</tbody>
</table>

A.3.1.7.3 State

Not applicable

A.3.1.7.4 New State

Not applicable

A.3.1.7.5 Return Value

None

A.3.1.7.6 Originator

Service User

A.3.1.7.7 Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>JTRS::Unsupported</td>
<td>The Algorithm selected is not supported by the Vocoder.</td>
</tr>
<tr>
<td>(see JTRS CORBA Types [1])</td>
<td></td>
</tr>
</tbody>
</table>
A.3.1.8 abortTx Operation
The abortTx operation instructs the transmit stream to terminate and purge.

A.3.1.8.1 Synopsis
void abortTx( );

A.3.1.8.2 Parameters
None

A.3.1.8.3 State
Not applicable

A.3.1.8.4 New State
Not applicable

A.3.1.8.5 Return Value
None

A.3.1.8.6 Originator
Service User

A.3.1.8.7 Exceptions
None

A.3.2 Vocoder::VocoderPacketConsumer
There are no changes from the Packet API [2].

A.3.3 Vocoder::VocoderPacketProducer
There are no changes from the Packet API [2].
A.4 IDL

A.4.1 Vocoder

/*
** Vocoder.idl
*/

#ifndef __VOCODER_DEFINED
#define __VOCODER_DEFINED

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

// Base Interface
module Vocoder
{

typedef JTRS::ExtEnum Algorithm;
typedef JTRS::ExtEnumSequence AlgorithmSequence;

// Base algorithm constants
// ALG_NONE de-selects other algorithms
// (see: CODEC extension for service extension descriptions)

const Algorithm ALG_NONE = 0;

// Producer/Consumer Port Descriptions
// Resource
interface VocoderPacketProducer : Packet::PayloadControl
{
};

interface VocoderPacketConsumer : Packet::OctetStream
{
};

interface Ctrl
{
    boolean getLoopback();
    void setLoopback( in boolean loopback );

    // Algorithm (CODEC) Controls
    AlgorithmSequence getAlgorithmsSupported();
    Algorithm getTxAlgorithm();
    Algorithm getRxAlgorithm();
    void setTxAlgorithm( in Algorithm txAlgorithm )
    raises(JTRS::Unsupported);
    void setRxAlgorithm( in Algorithm rxAlgorithm )
    raises(JTRS::Unsupported);

    // Instruct Tx service to stop & purge
    void abortTx( );
};
// Raw Audio Samples
// Default Vocoder Consumer/Producer Control, may be overridden by
// extension CODEC services. No specialization interface for the
service
// or configuration is currently defined.
//
const Algorithm ALG_RAW = ALG_NONE+1;

};

#endif // VOCODER_DEFINED
A.5 UML

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

![UML Diagram]

Figure 13 – Vocoder Service API Component Diagram
A.5.1 Data Types
None

A.5.2 Enumerations

A.5.2.1 Vocoder::Algorithm
The Algorithm type definition is an extension enumeration (see JTRS CORBA Types [1]). It enumerates the algorithms supported by the Vocoder Service. Additional algorithms supported by the Vocoder Service will be defined in their respective extensions.

```cpp
typedef JTRS::ExtEnum Algorithm;
const Algorithm ALG_NONE = 0;
const Algorithm ALG_RAW = ALG_NONE+1;
```

<table>
<thead>
<tr>
<th>JTRS::ExtEnum Algorithm</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALG_NONE</td>
<td>0</td>
<td>No algorithms.</td>
</tr>
<tr>
<td>ALG_RAW</td>
<td>ALG_NONE+1</td>
<td>Raw voice data.</td>
</tr>
</tbody>
</table>

A.5.2.2 Vocoder::AlgorithmSequence
The AlgorithmSequence type definition is an extension enumeration sequence (see JTRS CORBA Types [1]). It is used to identify the set of algorithms supported by the Vocoder Service.

```cpp
typedef JTRS::ExtEnumSequence AlgorithmSequence;
```

A.5.3 Exceptions
None

A.5.4 Structures
None
APPENDIX A.A – ABBREVIATIONS AND ACRONYMS

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALG</td>
<td>Algorithm</td>
</tr>
<tr>
<td>API</td>
<td>Application Program Interface</td>
</tr>
<tr>
<td>bps</td>
<td>bits per second</td>
</tr>
<tr>
<td>CORBA</td>
<td>Common Object Request Broker Architecture</td>
</tr>
<tr>
<td>ICWG</td>
<td>Interface Control Working Group</td>
</tr>
<tr>
<td>ID</td>
<td>Identification</td>
</tr>
<tr>
<td>IDL</td>
<td>Interface Definition Language</td>
</tr>
<tr>
<td>JPEO</td>
<td>Joint Program Executive Office</td>
</tr>
<tr>
<td>JTNC</td>
<td>Joint Tactical Networking Center</td>
</tr>
<tr>
<td>JTR</td>
<td>Joint Tactical Radio</td>
</tr>
<tr>
<td>JTRS</td>
<td>Joint Tactical Radio System</td>
</tr>
<tr>
<td>LPC</td>
<td>Linear Predictive Coding</td>
</tr>
<tr>
<td>MAX</td>
<td>Maximum</td>
</tr>
<tr>
<td>rx</td>
<td>Receive</td>
</tr>
<tr>
<td>TR</td>
<td>Tactical Radio</td>
</tr>
<tr>
<td>tx</td>
<td>Transmit</td>
</tr>
<tr>
<td>UML</td>
<td>Unified Modeling Language</td>
</tr>
</tbody>
</table>

APPENDIX A.B – PERFORMANCE SPECIFICATION

The following table provides a template for the generic performance specification for the Vocoder Service. This performance specification corresponds to the port diagram in Figure 3.

<table>
<thead>
<tr>
<th>Specification</th>
<th>Description</th>
<th>Units</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Worst Case Command Execution Time for pushPacket() on</td>
<td>*</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>vocoder_packet_consumer_provides_port</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Worst Case Command Execution Time for</td>
<td>*</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>pushPacket() on</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>vocoder_packet_consumer_uses_port</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*Note this template will be filled in by individual developers.*
B. MELP EXTENSION

B.1 INTRODUCTION

The Mixed Excitation Linear Predictive (MELP) Coding Extension is based upon the Vocoder Service API. It extends the functionality of the Vocoder Service API [A] to include MELP vocoding capabilities as specified in [4] and [5].

It should be noted that the use of the base Vocoder Service API [A] and codec extensions require an implicit or explicit connection with the Audio Port Device API [3]. Implicit connections are defined by the platform implementation (see Figure 1). Explicit connections are defined using the Vocoder Audio Stream Extension defined in section F.1 (see Figure 2).

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, provides summary of service interface uses, interface for each device component, port connections, and sequence diagrams.

c. Section B.3, Service Primitives and Attributes, specifies the operations that are provided by the MELP Extension.

d. Section B.4, IDL.

e. Section B.5, UML.

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


B.1.2 Service Layer Description

B.1.2.1 Vocoder Service API MELP Extension Port Connections

The following figure shows the port connections for the Vocoder Service API [A] with extensions to support MELP.

Note: All port names are for reference only. Ports identified in black are provided in section A.1.2.1.
Vocoder Service API MELP Extension Provides Ports Definitions

*melp_config_provides_port* is provided by the *Vocoder Service* to support the configuration of MELP.

Vocoder Service API MELP Extension Uses Ports Definitions

None

B.1.3 Modes of Service

Not applicable

B.1.4 Service States

There are no changes from section A.1.4.

B.1.5 Referenced Documents

There are additions to section A.1.5.


[5] STANAG 4591 Ed. 1 The 600 Bit/s, 1200 Bit/s and 2400 Bit/s NATO Interoperable Narrow Band Voice Coder, 2008
B.2 SERVICES

B.2.1 Provide Services

The MELP Extension interface provides services consisting of the following service ports, interfaces, and primitives, which can be called by other client components.

<table>
<thead>
<tr>
<th>Service Group (Port Name)</th>
<th>Service (Interface Provided)</th>
<th>Primitives (Provided)</th>
</tr>
</thead>
<tbody>
<tr>
<td>melp_config_provides_port</td>
<td>Vocoder::MELP</td>
<td>configureMelp()</td>
</tr>
<tr>
<td></td>
<td></td>
<td>getMelpConfig()</td>
</tr>
</tbody>
</table>

B.2.2 Use Services

None

B.2.3 Interface Modules

B.2.3.1 Vocoder

B.2.3.1.1 MELP Interface Description

The MELP interface is shown in the interface class diagram below. The MELP interface provides the capability to configure and status the MELP algorithm.

![MELP Extension Class Diagram](image)

Figure 15 – MELP Extension Class Diagram

B.2.4 Sequence Diagrams

None
B.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 B.5, UML. This cross-reference of types also includes any nested structures in the event of a structure of structures or an array of structures.

B.3.1 Vocoder::MELP

B.3.1.1 configureMelp Operation

The configureMelp operation provides the ability to configure the MELP algorithm.

B.3.1.1.1 Synopsis

void configureMelp( in Config melpCfg) raises (JTRS::InvalidParameter)

B.3.1.2 Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>melpCfg</td>
<td>Config</td>
<td>A structure containing the elements used for the initial configuration of the MELP algorithm.</td>
</tr>
</tbody>
</table>

B.3.1.3 State

Not applicable

B.3.1.4 New State

Not applicable

B.3.1.5 Return Value

None

B.3.1.6 Originator

Service User

B.3.1.7 Exceptions

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>JTRS::InvalidParameter</td>
<td>The configuration selected is invalid.</td>
</tr>
<tr>
<td>(see JTRS CORBA Types [1])</td>
<td></td>
</tr>
</tbody>
</table>
B.3.1.2 getMelpConfig Operation
The getMelpConfig operation provides the ability to status the MELP algorithm configuration.

B.3.1.2.1 Synopsis

Config getMelpConfig();

B.3.1.2.2 Parameters
None

B.3.1.2.3 State
Not applicable

B.3.1.2.4 New State
Not applicable

B.3.1.2.5 Return Value

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Config</td>
<td>A structure containing the elements used for the configuration of the MELP algorithm.</td>
</tr>
<tr>
<td>(see B.5.4.1)</td>
<td></td>
</tr>
</tbody>
</table>

B.3.1.2.6 Originator
Service User

B.3.1.2.7 Exceptions
None
B.4 IDL

B.4.1 VocoderMelpExt

/*! VocoderMelpExt.idl */

#ifndef __VOCODER_MELP_EXT_DEFINED
#define __VOCODER_MELP_EXT_DEFINED

#ifndef __VOCODER_DEFINED
#include "Vocoder.idl"
#endif

module Vocoder
{
    const Algorithm ALG_MELP = ALG_NONE + 2;
    // No specialization required

    interface MELP
    {
        struct Config
        {
            unsigned short bitRate;
        }

        void configureMelp( in Config melpCfg ) raises (JTRS::InvalidParameter);
        Config getMelpConfig();
    }
};
#endif //__VOCODER_MELP_EXT_DEFINED
B.5 UML

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

Figure 16 – MELP Extension Component Diagram
B.5.1 Data Types

None

B.5.2 Enumerations

B.5.2.1 Vocoder::Algorithm

The following specifies the additional algorithms supported by the Vocoder Service that is defined in this extension.

\[ \text{const Algorithm ALG_MELP} = \text{ALG_NONE} + 2; \]

<table>
<thead>
<tr>
<th>JTRS::ExtEnum</th>
<th>Element</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Algorithm (see A.5.2.1)</td>
<td>ALG_MELP</td>
<td>ALG_NONE + 2;</td>
<td>Identifies the Mixed Excitation Linear Predictive Coding (MELP) Algorithm</td>
</tr>
</tbody>
</table>

B.5.3 Exceptions

None

B.5.4 Structures

B.5.4.1 MELP::Config

The Config structure defines the attributes which define the properties of MELP.

\[ \text{struct Config} \]
\[ \{ \]
\[ \quad \text{unsigned short bitRate;} \]
\[ \}; \]

<table>
<thead>
<tr>
<th>Struct</th>
<th>Attributes</th>
<th>Type</th>
<th>Units</th>
<th>Valid Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>Config</td>
<td>bitRate</td>
<td>unsigned short</td>
<td>bits per second</td>
<td>1200, 2400, 4800, 9600</td>
</tr>
</tbody>
</table>
APPENDIX B.A  – ABBREVIATIONS AND ACRONYMS

The following list additional abbreviations and acronyms not specified in Appendix A.A.

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>MELP</td>
<td>Mixed Excitation Linear Predictive Coding</td>
</tr>
</tbody>
</table>

APPENDIX B.B  – PERFORMANCE SPECIFICATION

The following table provides a template for the generic performance specification for the MELP Extension. This performance specification corresponds to the port diagram in Figure 14.

<table>
<thead>
<tr>
<th>Specification</th>
<th>Description</th>
<th>Units</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Worst Case Command Execution Time for pushPacket () vocoder_packet_consumer_provides_port</td>
<td>*</td>
<td>*</td>
<td>*</td>
</tr>
</tbody>
</table>

*Note this template will be filled in by individual developers.
C. LPC EXTENSION

C.1 INTRODUCTION

The Linear Predictive Coding (LPC) Extension is based upon the Vocoder Service API [A]. It extends the functionality of the Vocoder Service to include LPC10 vocoding capabilities [7].

It should be noted that the use of the base Vocoder Service API [A] and codec extensions require an implicit or explicit connection with the Audio Port Device API [3]. Implicit connections are defined by the platform implementation (see Figure 1). Explicit connections are defined using the Vocoder Audio Stream Extension defined in section F.1 (see Figure 2).

C.1.1 Overview

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

b. Section C.2, Services, provides summary of service interface uses, interface for each device component, port connections, and sequence diagrams.

c. Section C.3, Service Primitives and Attributes, specifies the operations that are provided by the LPC Extension.

d. Section C.4, IDL.

e. Section C.5, UML.

f. Appendix C.A, – Abbreviations and Acronyms.

g. Appendix C.B, – Performance Specification.

C.1.2 Service Layer Description

C.1.2.1 Vocoder Service API LPC Extension Port Connections

The following figure shows the port connections for the Vocoder Service API [A] with extensions to support LPC.

Note: All port names are for reference only. Ports identified in black are provided in section A.1.2.1.
Vocoder Service API LPC Extension Port Diagram

**Vocoder Service API LPC Extension Provides Ports Definitions**

`lpc_config_provides_port` is provided by the Vocoder Service to support the configuration of LPC.

**Vocoder Service API LPC Extension Uses Ports Definitions**

None

**C.1.3 Modes of Service**

Not applicable

**C.1.4 Service States**

There are no changes from section A.1.4.

**C.1.5 Referenced Documents**

There are additions to section A.1.5

C.2 SERVICES

C.2.1 Provide Services
The LPC Extension interface provides services consisting of the following service ports, interfaces, and primitives, which can be called by other client components.

<table>
<thead>
<tr>
<th>Service Group (Port Name)</th>
<th>Service (Interface Provided)</th>
<th>Primitives (Provided)</th>
</tr>
</thead>
<tbody>
<tr>
<td>lpc_config_provides_port</td>
<td>Vocoder::LPC</td>
<td>configureLpc()</td>
</tr>
<tr>
<td></td>
<td></td>
<td>getLpcConfig()</td>
</tr>
</tbody>
</table>

C.2.2 Use Services
None

C.2.3 Interface Modules

C.2.3.1 Vocoder

C.2.3.1.1 LPC Interface Description
The LPC interface is shown in the interface class diagram below. The LPC interface provides the capability to configure and status the LPC algorithm.

Figure 18 – LPC Extension Class Diagram

C.2.4 Sequence Diagrams
None
C.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 C.5, UML. This cross-reference of types also includes any nested structures in the event of a structure of structures or an array of structures.

C.3.1 Vocoder::LPC

C.3.1.1 configureLpc Operation

The configureLpc operation provides the ability to configure the LPC algorithm.

C.3.1.1.1 Synopsis

void configureLpc( in Config lpcCfg) raises (JTRS::InvalidParameter);

C.3.1.1.2 Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>lpcCfg</td>
<td>Config</td>
<td>A structure containing the elements used for the initial configuration of the LPC algorithm.</td>
</tr>
</tbody>
</table>

C.3.1.1.3 State

Not applicable

C.3.1.1.4 New State

Not applicable

C.3.1.1.5 Return Value

None

C.3.1.1.6 Originator

Service User

C.3.1.1.7 Exceptions

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>JTRS::InvalidParameter</td>
<td>The configuration selected is invalid.</td>
</tr>
<tr>
<td>(see JTRS CORBA Types [1])</td>
<td></td>
</tr>
</tbody>
</table>
C.3.1.2 getLpcConfig Operation
The getLpcConfig operation provides the ability to status the LPC algorithm configuration.

C.3.1.2.1 Synopsis

```c
Config getLpcConfig( );
```

C.3.1.2.2 Parameters
None

C.3.1.2.3 State
Not applicable

C.3.1.2.4 New State
Not applicable

C.3.1.2.5 Return Value

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Config</td>
<td>A structure containing the elements used for the configuration of the LPC</td>
</tr>
<tr>
<td>(see C.5.4.1)</td>
<td>algorithm.</td>
</tr>
</tbody>
</table>

C.3.1.2.6 Originator
Service User

C.3.1.2.7 Exceptions
None
C.4 IDL

C.4.1 VocoderLpcExt

/*
** VocoderLpcExt.idl
*/

#ifndef __VOCODER_LPC_EXT_DEFINED
#define __VOCODER_LPC_EXT_DEFINED
#endif

#ifndef __VOCODER_DEFINED
#include "Vocoder.idl"
#endif

module Vocoder
{
  const Algorithm ALG_LPC = ALG_NONE + 3;

  interface LPC
  {
    typedef JTRS::ExtEnum   CodeBook;
    // Known Code Book Types
    const CodeBook CB_NONE = 0;
    const CodeBook CB_ENGLISH = CB_NONE + 1;
    const CodeBook CB_DUTCH = CB_ENGLISH + 1;
    const CodeBook CB_SPANISH = CB_DUTCH + 1;
    const CodeBook CB_FRENCH = CB_SPANISH + 1;
    const CodeBook CB_ARABIC = CB_FRENCH + 1;
    const CodeBook CB_RUSSIAN = CB_ARABIC + 1;
    const CodeBook CB_MAX = CB_RUSSIAN + 1;

    struct Config
    {
      CodeBook cfgCodeBook;
      unsigned short bitrate;
    };

    void configureLpc( in Config lpcCfg ) raises (JTRS::InvalidParameter);
    Config getLpcConfig();
  }
};

#endif //__VOCODER_LPC_EXT_DEFINED
C.5 UML

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

![UML Diagram](image-url)

**Figure 19 – LPC Extension Component Diagram**
C.5.1 Data Types
None

C.5.2 Enumerations

C.5.2.1 LPC::CodeBook
The CodeBook type definition is an extension enumeration (see JTRS CORBA Types [1]). It enumerates the language code books to be used in the LPC::Config (see C.5.4.1).

typedef JTRS::ExtEnum  CodeBook;

// Known Code Book Types  
const CodeBook  CB_NONE     = 0;
const CodeBook  CB_ENGLISH  = CB_NONE+1;
const CodeBook  CB_DUTCH    = CB_ENGLISH+1;
const CodeBook  CB_SPANISH  = CB_DUTCH+1;
const CodeBook  CB_FRENCH   = CB_SPANISH+1;
const CodeBook  CB_ARABIC   = CB_FRENCH+1;
const CodeBook  CB_RUSSIAN  = CB_ARABIC+1;
const CodeBook  CB_MAX      = CB_RUSSIAN+1;

<table>
<thead>
<tr>
<th>JTRS::ExtEnum</th>
<th>Element</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CodeBook</td>
<td>CB_NONE</td>
<td>0</td>
<td>No code books</td>
</tr>
<tr>
<td></td>
<td>CB_ENGLISH</td>
<td>CB_NONE + 1</td>
<td>English code book</td>
</tr>
<tr>
<td></td>
<td>CB_DUTCH</td>
<td>CB_ENGLISH + 1</td>
<td>Dutch code book</td>
</tr>
<tr>
<td></td>
<td>CB_SPANISH</td>
<td>CB_DUTCH + 1</td>
<td>Spanish code book</td>
</tr>
<tr>
<td></td>
<td>CB_FRENCH</td>
<td>CB_SPANISH + 1</td>
<td>French code book</td>
</tr>
<tr>
<td></td>
<td>CB_ARABIC</td>
<td>CB_FRENCH + 1</td>
<td>Arabic code book</td>
</tr>
<tr>
<td></td>
<td>CB_RUSSIAN</td>
<td>CB_ARABIC + 1</td>
<td>Russian code book</td>
</tr>
<tr>
<td></td>
<td>CB_MAX</td>
<td>CB_RUSSIAN + 1</td>
<td>Maximum number of code books plus 1</td>
</tr>
</tbody>
</table>

C.5.2.2 Vocoder::Algorithm
The following specifies the additional algorithms supported by the Vocoder Service that is defined in this extension.

    const Algorithm  ALG_LPC     = ALG_NONE + 3;

<table>
<thead>
<tr>
<th>JTRS::ExtEnum</th>
<th>Element</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Algorithm (see A.5.2.1)</td>
<td>ALG_LPC</td>
<td>ALG_NONE + 3;</td>
<td>Identifies the LPC Algorithm</td>
</tr>
</tbody>
</table>

C.5.3 Exceptions
None


C.5.4 Structures

C.5.4.1 LPC::Config

The *Config* structure defines the attributes which define the properties of LPC.

```c
struct Config {
    CodeBook cfgCodeBook;
    unsigned short bitRate;
};
```

<table>
<thead>
<tr>
<th>Struct</th>
<th>Attributes</th>
<th>Type</th>
<th>Units</th>
<th>Valid Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>Config</td>
<td>cfgCodeBook</td>
<td>CodeBook (see C.5.2.1)</td>
<td>Not Applicable</td>
<td>Not Applicable</td>
</tr>
<tr>
<td></td>
<td>bitRate</td>
<td>unsigned short</td>
<td>bps</td>
<td>2400</td>
</tr>
</tbody>
</table>
APPENDIX C.A  — ABBREVIATIONS AND ACRONYMS
There are no changes from Appendix A.A.

APPENDIX C.B  — PERFORMANCE SPECIFICATION
The following table provides a template for the generic performance specification for the *LPC Extension*. This performance specification corresponds to the port diagram in Figure 20.

<table>
<thead>
<tr>
<th>Specification</th>
<th>Description</th>
<th>Units</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Worst Case Command Execution Time for pushPacket()</td>
<td>vocoder_packet_consumer_provides_port</td>
<td>*</td>
<td>*</td>
</tr>
</tbody>
</table>

*Note this template will be filled in by individual developers.*
D. CVSD EXTENSION

D.1 INTRODUCTION

The Continuously Variable Slope Delta (CVSD)[8] Extension is based upon the Vocoder Service API [A]. It extends the functionality of the Vocoder Service API to include CVSD vocoding capabilities.

It should be noted that the use of the base Vocoder Service API [A] and codec extensions require an implicit or explicit connection with the Audio Port Device API [3]. Implicit connections are defined by the platform implementation (see Figure 1). Explicit connections are defined using the Vocoder Audio Stream Extension defined in section F.1 (see Figure 2).

D.1.1 Overview

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

b. Section D.2, Services provides summary of service interface uses, interface for each device component, port connections, and sequence diagrams.

c. Section D.3, Service Primitives and Attributes specifies the operations that are provided by the CVSD Extension.

d. Section D.4, IDL.

e. Section D.5, UML.

f. Appendix D.A, – Abbreviations and Acronyms.


D.1.2 Service Layer Description

D.1.2.1 Vocoder Service API CVSD Extension Port Connections

The following figure shows the port connections for the Vocoder Service API [A] with extensions to support CVSD.

Note: All port names are for reference only. Ports identified in black are provided in section A.1.2.1.
Vocoder Service API

Figure 20 – Vocoder Service API CVSD Extension Port Diagram

Vocoder Service API CVSD Extension Provides Ports Definitions

cvsd_config_provides_port is provided by the Vocoder Service to support CVSD configuration.

Vocoder Service API CVSD Extension Uses Ports Definitions

None

D.1.3 Modes of Service

Not applicable

D.1.4 Service States

There are no changes from section A.1.4.

D.1.5 Referenced Documents

There are additions to section A.1.5.

D.2 SERVICES

D.2.1 Provide Services

The CVSD Extension interface provides services consisting of the following service ports, interfaces, and primitives, which can be called by other client components.

Table 8 – CVSD Extension Provide Service Interface

<table>
<thead>
<tr>
<th>Service Group (Port Name)</th>
<th>Service (Interface Provided)</th>
<th>Primitives (Provided)</th>
</tr>
</thead>
<tbody>
<tr>
<td>cvsd_config_provides_port</td>
<td>Vocoder::CVSD</td>
<td>configureCvsd()</td>
</tr>
<tr>
<td></td>
<td></td>
<td>getCvsdConfig()</td>
</tr>
</tbody>
</table>

D.2.2 Use Services

None

D.2.3 Interface Modules

D.2.3.1 Vocoder

D.2.3.1.1 CVSD Interface Description

The CVSD interface is shown in the interface class diagram below. The CVSD interface provides the capability to configure and status the CVSD algorithm.

Figure 21 – CVSD Extension Class Diagram

D.2.4 Sequence Diagrams

None
D.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 D.5 UML. This cross-reference of types also includes any nested structures in the event of a structure of structures or an array of structures.

D.3.1 Vocoder::CVSD

D.3.1.1 configureCvsd Operation
The configureCvsd operation provides the ability to configure the CVSD algorithm to be used in vocoding.

D.3.1.1.1 Synopsis
void configureCvsd(in Config cvsdCfg) raises(JTRS::InvalidParameter);

D.3.1.1.2 Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cvsdCfg</td>
<td>Config</td>
<td>A structure containing the elements used for the configuration of the CVSD algorithm.</td>
</tr>
</tbody>
</table>

D.3.1.1.3 State
Not applicable

D.3.1.1.4 New State
Not applicable

D.3.1.1.5 Return Value
None

D.3.1.1.6 Originator
Service User

D.3.1.1.7 Exceptions

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>JTRS::InvalidParameter</td>
<td>The configuration selected is invalid.</td>
</tr>
</tbody>
</table>

(see JTRS CORBA Types[1])
D.3.1.2 getCVSDConfig Operation
The getCVSDConfig operation provides the ability to status the CVSD algorithm configuration.

D.3.1.2.1 Synopsis

```c
Config getCVSDConfig( );
```

D.3.1.2.2 Parameters
None

D.3.1.2.3 State
Not applicable

D.3.1.2.4 New State
Not applicable

D.3.1.2.5 Return Value

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Config (see D.5.4.1)</td>
<td>A structure containing the elements used for the configuration of the CVSD algorithm.</td>
</tr>
</tbody>
</table>

D.3.1.2.6 Originator
Service User

D.3.1.2.7 Exceptions
None
D.4 IDL

D.4.1 VocoderCvsdExt

/*
 ** VocoderCvsdExt.idl
 */

#ifndef __VOCODERCVSDEXT_DEFINED
#define __VOCODERCVSDEXT_DEFINED

#ifndef __VOCODER_DEFINED
#include "Vocoder.idl"
#endif

module Vocoder
{
    const Algorithm ALG_CVSD = ALG_NONE + 4;

    interface CVSD
    {
        typedef JTRS::ExtEnum Mode;

        // Known Operating Modes
        const Mode Mode_NONE = 0;
        const Mode KY57 = Mode_NONE+1;
        const Mode Mode_MAX = KY57+1;

        struct Config
        {
            Mode cfgMode;
            unsigned short bitRate;
        };

        // Configuration
        void configureCvsd( in Config cvsdCfg ) raises ( JTRS::InvalidParameter );
        Config getCvsdConfig();
    }
};
#endif //__VOCODERCVSDEXT_DEFINED
D.5 UML
This section contains the Device component UML diagram and the definitions of all data types referenced (directly or indirectly) by the Service Primitives and Attributes in section A.3.

![Vocoder UML Diagram](image)

**Figure 22 – CVSD Extension Component Diagram**

D.5.1 Data Types
None

D.5.2 Enumerations
None

D.5.2.1 CVSD::Mode
The `Mode` type definition is an extension enumeration (see *JTRS CORBA Types* [1]). It enumerates the CVSD modes.

```c
typedef JTRS::ExtEnum Mode;

// Known Operating Modes
const Mode Mode_NONE = 0;
const Mode KY57 = Mode_NONE + 1;
const Mode Mode_MAX = KY57 + 1;
```

<table>
<thead>
<tr>
<th>JTRS::ExtEnum</th>
<th>Element</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mode</td>
<td>Mode_NONE</td>
<td>0</td>
<td>No mode</td>
</tr>
<tr>
<td></td>
<td>KY57</td>
<td>Mode_NONE+1</td>
<td>KY57</td>
</tr>
<tr>
<td></td>
<td>Mode_MAX</td>
<td>KY57+1</td>
<td>Maximum number of modes plus 1</td>
</tr>
</tbody>
</table>

D.5.2.2 Vocoder::Algorithm
The following specifies the additional algorithms supported by the *Vocoder Service* that is defined in this extension.

```c
const Algorithm ALG_CVSD = ALG_NONE + 4;
```

<table>
<thead>
<tr>
<th>JTRS::ExtEnum</th>
<th>Element</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Algorithm (see A.5.2.1)</td>
<td>ALG_CVSD</td>
<td>ALG_NONE + 4;</td>
<td>Identifies the Continuously Variable Slope Delta (CVSD) Modulation Algorithm</td>
</tr>
</tbody>
</table>
D.5.3 Exceptions
None

D.5.4 Structures

D.5.4.1 CVSD::Config
The Config structure defines the attributes which define the properties of CVSD.

```c
struct Config
{
    Mode cfgMode;
    unsigned short bitRate;
};
```

<table>
<thead>
<tr>
<th>Struct</th>
<th>Attributes</th>
<th>Type</th>
<th>Units</th>
<th>Valid Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>Config</td>
<td>cfgMode</td>
<td>Mode</td>
<td>Not Applicable</td>
<td>Not Applicable</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(see D.5.2.1)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>bitRate</td>
<td>unsigned short</td>
<td>bps</td>
<td>12K, 16K</td>
<td></td>
</tr>
</tbody>
</table>
APPENDIX D.A – ABBREVIATIONS AND ACRONYMS

The following list additional abbreviations and acronyms not specified in Appendix A.A.

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVSD</td>
<td>Continuously Variable Slope Delta</td>
</tr>
</tbody>
</table>

APPENDIX D.B – PERFORMANCE SPECIFICATION

The following table provides a template for the generic performance specification for the CVSD Extension. This performance specification corresponds to the port diagram in Figure 20.

<table>
<thead>
<tr>
<th>Specification</th>
<th>Description</th>
<th>Units</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Worst Case Command Execution Time for pushPacket() vocoder_packet_consumer_provides_port</td>
<td>*</td>
<td>*</td>
<td>*</td>
</tr>
</tbody>
</table>

*Note this template will be filled in by individual developers.*
E. SPEEX EXTENSION

E.1 INTRODUCTION

The Speex Coding [9] Extension is based upon the Vocoder Service API [A]. It extends the functionality of the Vocoder Service to include Speex vocoding capabilities. Speex is an open source/free software patent-free audio compression format designed for speech and is available at http://www.speex.org. It is based on Code Excited Linear Prediction (CELP) and is designed to compress voice at bit rates ranging from 2 to 44 kbps.

It should be noted that the use of the base Vocoder Service API [A] and codec extensions require an implicit or explicit connection with the Audio Port Device API [3]. Implicit connections are defined by the platform implementation (see Figure 1). Explicit connections are defined using the Vocoder Audio Stream Extension defined in section F.1 (see Figure 2).

E.1.1 Overview

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

b. Section E.2, Services, provides summary of service interface uses, interface for each device component, port connections, and sequence diagrams.

c. Section E.3, Service Primitives and Attributes, specifies the operations that are provided by the Speex Extension.

d. Section E.4, IDL.

e. Section E.5, UML.

f. Appendix E.A, – Abbreviations and Acronyms.

g. Appendix E.B, – Performance Specification.

E.1.2 Service Layer Description

There are no changes from section A.1.2.

E.1.3 Modes of Service

Not applicable

E.1.4 Service States

There are no changes from section A.1.4.

E.1.5 Referenced Documents

There are additions to section A.1.5.

E.2 SERVICES
There are no changes from section A.2.

E.3 SERVICE PRIMITIVES AND ATTRIBUTES
There are no changes from section A.3.
E.4 IDL

E.4.1 VocoderSpeexExt

/*
** VocoderSpeexExt.idl
*/

#ifndef __VOCODER_SPEEX_EXT_DEFINED
#define __VOCODER_SPEEX_EXT_DEFINED

#ifndef __VOCODER_DEFINED
    #include "Vocoder.idl"
#endif

module Vocoder
{
    const Algorithm ALG_SPEEX = ALG_NONE + 5;

    // The Speex CODEC has no specialized configuration, and hence no
    // interface is required.
};
#endif //__VOCODER_SPEEX_EXT_DEFINED
E.5 UML

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

![UML Diagram](image)

Figure 23 – Speex Extension Component Diagram

**E.5.1 Data Types**

None

**E.5.2 Enumerations**

**E.5.2.1 Vocoder::Algorithm**

The following specifies the additional algorithms supported by the Vocoder Service that is defined in this extension.

\[
\text{const Algorithm } \ ALG\_SPEEX \ = \ ALG\_NONE \ + \ 5;
\]

<table>
<thead>
<tr>
<th>JTRS::ExtEnum</th>
<th>Element</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Algorithm (see A.5.2.1)</td>
<td>ALG_SPEEX</td>
<td>ALG_NONE + 5;</td>
<td>Speex Algorithm</td>
</tr>
</tbody>
</table>

**E.5.3 Exceptions**

None

**E.5.4 Structures**

None
APPENDIX E.A – ABBREVIATIONS AND ACRONYMS
The following list additional abbreviations and acronyms not specified in Appendix A.A.

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>kbps</td>
<td>kilobits per second</td>
</tr>
<tr>
<td>CELP</td>
<td>Code Excited Linear Prediction</td>
</tr>
</tbody>
</table>

APPENDIX E.B – PERFORMANCE SPECIFICATION
The following table provides a template for the generic performance specification for the SPEEX Extension. This performance specification corresponds to the port diagram in Figure 3.

Table 10 – SPEEX Extension Performance Specification

<table>
<thead>
<tr>
<th>Specification</th>
<th>Description</th>
<th>Units</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Worst Case Command Execution Time for pushPacket()</td>
<td>*</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>vocoder_packet_consumer_provides_port</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
F. VOCODER AUDIO STREAM EXTENSION

F.1 INTRODUCTION

The Vocoder Audio Stream Extension is based upon the Vocoder Service API [A]. It extends the functionality of the common Vocoder Service to provide an explicit connection to a Service User to produce and consume audio samples. It retains the methods and attributes defined in the base Vocoder Service API [A].

F.1.1 Overview

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

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

c. Section F.3, Service Primitives and Attributes, specifies the operations that are provided by the Vocoder Audio Stream Extension.

d. Section F.4, IDL.

e. Section F.5, UML.

f. Appendix F.A, – Abbreviations and Acronyms.


F.1.2 Service Layer Description

F.1.2.1 Vocoder Service API Vocoder Audio Stream Extension Port Connections

The following figure shows the port connections for the Vocoder Service API Vocoder Audio Stream Extension.

Note: All port names are for reference only. Ports in black are defined in section A.1.2.1.
Figure 24 – Vocoder Service API Vocoder Audio Stream Extension Port Diagram

**Vocoder Service API Vocoder Audio Stream Extension Provides Ports Definitions**

- **vocoder_audio_sample_provides_port** is provided by the *Vocoder Service* to consume packets through the `pushPacket` operation.
- **vocoder_audio_sample_ctrl_provides_port** is provided by the *Vocoder Service* to set the payload size by the Service User.

**Vocoder Service API Vocoder Audio Stream Extension Uses Ports Definitions**

- **vocoder_audio_sample_uses_port** is used by the *Vocoder Service* to push packets to the Service User.
- **vocoder_audio_sample_ctrl_uses_port** is used by the *Vocoder Service* to set the payload size of the incoming packets from the Service User.

**F.1.3 Modes of Service**

Not applicable

**F.1.4 Service States**

There are no changes from section A.1.4.

**F.1.5 Referenced Documents**

There are changes from the section A.1.5.
F.2 SERVICES

F.2.1 Provide Services
The VocoderAudioStream Extension provides service consists of the following service ports, interfaces, and primitives, which can be called by other client components. Detailed definition of the interfaces and services shaded in gray is provided by separate documentation identified in the table.

<table>
<thead>
<tr>
<th>Service Group (Port Name)</th>
<th>Service (Interface Provided)</th>
<th>Primitives (Provided)</th>
</tr>
</thead>
<tbody>
<tr>
<td>vocoder_audio_stream_provides_port</td>
<td>Vocoder::AudioSample Stream</td>
<td>pushPacket()</td>
</tr>
<tr>
<td></td>
<td>Audio::SampleStream [3]</td>
<td>getMaxPayloadSize()</td>
</tr>
<tr>
<td></td>
<td>Packet::UshortStream [2]</td>
<td>getMinPayloadSize()</td>
</tr>
<tr>
<td></td>
<td></td>
<td>getDesiredPayloadSize()</td>
</tr>
<tr>
<td></td>
<td></td>
<td>getMinOverrideTimeout()</td>
</tr>
<tr>
<td>vocoder_audio_stream_control_provides_port</td>
<td>Vocoder::AudioSample Control</td>
<td>setMaxPayloadSize()</td>
</tr>
<tr>
<td></td>
<td>Audio::SampleStream Control [3]</td>
<td>setMinPayloadSize()</td>
</tr>
<tr>
<td></td>
<td>Packet::PayloadControl [2]</td>
<td>setDesiredPayloadSize()</td>
</tr>
<tr>
<td></td>
<td></td>
<td>setMinOverrideTimeout()</td>
</tr>
</tbody>
</table>

F.2.2 Use Services
The Vocoder Audio Stream Extension use service set consists of the following service ports, interfaces, and primitives. Since the Vocoder Audio Stream Extension 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 Vocoder Audio Stream Extension uses the port name as connectionId for the connection. Detailed definition of the interfaces and services shaded in gray is provided by separate documentation identified in the table.

<table>
<thead>
<tr>
<th>Service Group (Port Name)</th>
<th>Service (Interface Provided)</th>
<th>Primitives (Provided)</th>
</tr>
</thead>
<tbody>
<tr>
<td>vocoder_audio_stream_uses_port</td>
<td>Vocoder::AudioSample Stream</td>
<td>pushPacket()</td>
</tr>
<tr>
<td></td>
<td>Audio::SampleStream [3]</td>
<td>getMaxPayloadSize()</td>
</tr>
<tr>
<td></td>
<td>Packet::UshortStream [2]</td>
<td>getMinPayloadSize()</td>
</tr>
</tbody>
</table>
### F.2.3 Interface Modules

#### F.2.3.1 Vocoder

**F.2.3.1.1 Vocoder Audio Stream Extension Interface Description**

The following is the interface class diagram for the `VocoderAudioStreamExtension`. Interfaces in gray are specified in the `Packet API` [2] and `Audio Port Device API` [3].

![Class Diagram]

*Figure 25 – VocoderAudioStream Extension Interface Class Diagram*
F.2.3.1.2 AudioSampleStream Interface Description

The interface design of AudioSampleStream is shown in the interface class diagram below. It extends the Audio::SampleStream interface defined in the Audio Port Device API [3] to provide the ability to transfer audio samples and status the audio sample packet sizes.

![AudioSampleStream Interface Diagram](image)

Figure 26 – AudioSampleStream Interface Diagram

F.2.3.1.3 AudioSampleControl Interface Description

The interface design of AudioSampleControl is shown in the interface class diagram below. It extends the Audio::SampleStreamControl interface defined in the Audio Port Device API [3] to provide the ability to configure the audio sample packet sizes.

![AudioSampleCtrl Interface Diagram](image)

Figure 27 – AudioSampleCtrl Interface Diagram

F.2.4 Sequence Diagrams

None
F.3 SERVICE PRIMITIVES AND ATTRIBUTES

There are no changes from the Vocoder Service API [A] and Audio Port Device API: Audio Sample Stream Extension [3].
F.4 IDL

F.4.1 VocoderAudioStreamExt

/*
** VocoderAudioStreamExt.idl
*/

#ifndef __VOCODERAUDIOSTREAMEXT_DEFINED
#define __VOCODERAUDIOSTREAMEXT_DEFINED

#ifndef __AUDIO_SAMPLE_STREAM_EXT_DEFINED
    #include "AudioSampleStreamExt.idl"
#endif

module Vocoder
{
    interface AudioSampleStream : Audio::SampleStream
    {
    
    }

    interface AudioSampleControl : Audio::SampleStreamControl
    {
    
    }

};
#endif // __VOCODERAUDIOSTREAMEXT_DEFINED

F.5 UML

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

![Vocoder Audio Stream Extension Component Diagram]

Figure 28 – Vocoder Audio Stream Extension Component Diagram
F.5.1 Data Types
None

F.5.2 Enumerations
None

F.5.3 Exceptions
None

F.5.4 Structures
None
APPENDIX F.A – ABBREVIATIONS AND ACRONYMS

There are no changes from Appendix A.A.

APPENDIX F.B – PERFORMANCE SPECIFICATION

The following table provides a template for the generic performance specification for the Vocoder Audio Stream Extension. This performance specification corresponds to the port diagram in Figure 24.

Table 13 – Vocoder Audio Stream Extension Performance Specification

<table>
<thead>
<tr>
<th>Specification</th>
<th>Description</th>
<th>Units</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Worst Case Command Execution Time for pushPacket() on vocoder_audio_sample_provides_port</td>
<td>*</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>Worst Case Command Execution Time for pushPacket() on vocoder_audio_sample_uses_port</td>
<td>*</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>Worst Case Command Execution Time for vocoder_audio_sample_provides_port</td>
<td>*</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>Worst Case Command Execution Time for vocoder_audio_sample_uses_port</td>
<td>*</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>Worst Case Command Execution Time for vocoder_audio_sample_ctrl_provides_port</td>
<td>*</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>Worst Case Command Execution Time for vocoder_audio_sample_ctrl_uses_port</td>
<td>*</td>
<td>*</td>
<td>*</td>
</tr>
</tbody>
</table>

*Note this template will be filled in by individual developers.
G. G.729 EXTENSION

G.1 INTRODUCTION

The G.729 Extension is based upon the Vocoder Service API [A]. It extends the functionality of the Vocoder Service API [A] to include G.729 vocoding capabilities. G.729 is an International Telecommunications Union (ITU) recommendation also known as Conjugate Structure-Algebraic Code Excited Linear Prediction (CS-ACELP) coding [10]. The G.729 Extension includes Annex A [11] of the specification which provides reduced complexity. Annex B [12] is optional and is configurable based on whether the G729::Configure structure enableVad value is enabled or disabled.

It should be noted that the use of the base Vocoder Service API [A] and codec extensions require an implicit or explicit connection with the Audio Port Device API [3]. Implicit connections are defined by the platform implementation (see Figure 1). Explicit connections are defined using the Vocoder Audio Stream Extension defined in section F.1 (see Figure 2).

G.1.1 Overview

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

b. Section G.2, Services, provides summary of service interface uses, interface for each device component, port connections, and sequence diagrams.

c. Section G.3, Service Primitives and Attributes, specifies the operations that are provided by the G.729 Extension.

d. Section G.4, IDL.

e. Section G.5, UML.

f. Appendix G.A. – Abbreviations and Acronyms.


G.1.2 Service Layer Description

G.1.2.1 Vocoder Service API G.729 Extension Port Connections

The following figure shows the port connections for the Vocoder Service API [A] with extensions to support G.729.

Note: All port names are for reference only. Ports identified in black are provided in section A.1.2.1.
Vocoder Service API G.729 Extension Provides Ports Definitions

_\texttt{g729\_config\_provides\_port}\_ is provided by the \textit{Vocoder Service} to support the configuration of G.729.

Vocoder Service API G.729 Extension Uses Ports Definitions

None

G.1.3 Modes of Service

Not applicable

G.1.4 Service States

There are no changes from section A.1.4.

G.1.5 Referenced Documents

There are additions to section A.1.5.

G.2 SERVICES

G.2.1 Provide Services
The G.729 Extension interface provides services consisting of the following service ports, interfaces, and primitives, which can be called by other client components.

Table 14 – G.729 Extension Provide Service Interface

<table>
<thead>
<tr>
<th>Service Group (Port Name)</th>
<th>Service (Interface Provided)</th>
<th>Primitives (Provided)</th>
</tr>
</thead>
<tbody>
<tr>
<td>g729_config_provides_port</td>
<td>Vocoder::G729</td>
<td>configureG729()</td>
</tr>
<tr>
<td></td>
<td></td>
<td>getG729Config()</td>
</tr>
</tbody>
</table>

G.2.2 Use Services
None

G.2.3 Interface Modules

G.2.3.1 Vocoder

G.2.3.1.1 G729 Interface Description
The G729 interface is shown in the interface class diagram below. The G729 interface provides the capability to configure and status the G.729 algorithm.

Figure 30 – G.729 Extension Class Diagram

G.2.4 Sequence Diagrams
None
G.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 G.5, UML. This cross-reference of types also includes any nested structures in the event of a structure of structures or an array of structures.

G.3.1 Vocoder::G729

G.3.1.1 configureG729 Operation
The configureG729 operation provides the ability to configure the G.729 algorithm.

G.3.1.1.1 Synopsis
void configureG729(in Config g729Cfg ) raises (JTRS::InvalidParameter);

G.3.1.1.2 Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>g729Cfg</td>
<td>Config (see G.5.4.1)</td>
<td>A structure containing the elements used for the initial configuration of the G.729 algorithm.</td>
</tr>
</tbody>
</table>

G.3.1.1.3 State
Not applicable

G.3.1.1.4 New State
Not applicable

G.3.1.1.5 Return Value
None

G.3.1.1.6 Originator
Service User

G.3.1.1.7 Exceptions

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>JTRS::InvalidParameter</td>
<td>The configuration selected is invalid.</td>
</tr>
<tr>
<td>(see JTRS CORBA Types[1])</td>
<td></td>
</tr>
</tbody>
</table>
G.3.1.2 getG729Config Operation
The getG729Config operation provides the ability to status the G.729 algorithm configuration.

G.3.1.2.1 Synopsis
Config getG729Config();

G.3.1.2.2 Parameters
None

G.3.1.2.3 State
Not applicable

G.3.1.2.4 New State
Not applicable

G.3.1.2.5 Return Value

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Config (see G.5.4.1)</td>
<td>A structure containing the elements used for the configuration of the G.729 algorithm.</td>
</tr>
</tbody>
</table>

G.3.1.2.6 Originator
Service User

G.3.1.2.7 Exceptions
None
G.4 IDL

G.4.1 VocoderG729Ext

/*
** VocoderG729Ext.idl
*/

#ifndef __VOCODER_G729_EXT_DEFINED
#define __VOCODER_G729_EXT_DEFINED

#ifndef __VOCODER_DEFINED
#include "Vocoder.idl"
#endif

module Vocoder
{
    const Algorithm ALG_G729 = ALG_NONE + 6;

    interface G729
    {
        struct Config
        {
            boolean enableVad; /* Voice Activity Detector */
        };

        void configureG729(in Config g729Cfg ) raises (JTRS::InvalidParameter);

        Config getG729Config();
    };
};
#endif //__VOCODER_G729_EXT_DEFINED
**G.5 UML**

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

![Figure 31 – G.729 Extension Component Diagram](image-url)
G.5.1 Data Types
None

G.5.2 Enumerations

G.5.2.1 Vocoder::Algorithm
The following specifies the additional algorithms supported by the Vocoder Service that is defined in this extension.

\[
\text{const Algorithm ALG\_G729 = ALG\_NONE + 6;}
\]

<table>
<thead>
<tr>
<th>JTRS::ExtEnum</th>
<th>Element</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Algorithm</td>
<td>ALG_G729</td>
<td>ALG_NONE + 6;</td>
<td>Identifies the G.729 Algorithm</td>
</tr>
</tbody>
</table>

G.5.3 Exceptions
None

G.5.4 Structures

G.5.4.1 G729::Config
The Config structure defines the attributes which define the properties of G.729.

\[
\text{struct Config}
\]

\[
\text{//}
\]

\[
\text{boolean enableVad; /* Voice Activity Detector */}
\]

\[
\text{;}
\]

<table>
<thead>
<tr>
<th>Struct</th>
<th>Attributes</th>
<th>Type</th>
<th>Valid Range</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Config</td>
<td>enableVad</td>
<td>boolean</td>
<td>TRUE = enable</td>
<td>Determines whether Annex B of the</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>FALSE = disable</td>
<td>G.729 specification for Voice Activity Detector (VAD) is enabled/disabled.</td>
</tr>
</tbody>
</table>
APPENDIX G.A – ABBREVIATIONS AND ACRONYMS
The following list additional abbreviations and acronyms not specified in Appendix A.A.

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>CS-ACELP</td>
<td>Conjugate Structure-Algebraic Code Excited Linear Prediction</td>
</tr>
<tr>
<td>ITU</td>
<td>International Telecommunications Union</td>
</tr>
<tr>
<td>VAD</td>
<td>Voice Activity Detector</td>
</tr>
</tbody>
</table>

APPENDIX G.B – PERFORMANCE SPECIFICATION
The following table provides a template for the generic performance specification for the G.729 Extension. This performance specification corresponds to the port diagram in Figure 29.

Table 15 – G.729 Extension Performance Specification

<table>
<thead>
<tr>
<th>Specification</th>
<th>Description</th>
<th>Units</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Worst Case Command Execution Time for g729_config_provides_port</td>
<td>*</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>vocoder_packet_consumer_provides_port</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*Note this template will be filled in by individual developers.
H. MELPE EXTENSION <DEPRECATED>

Deprecated in API version 1.3.2, dated 01 August 2011.
I. MELPE WITH DTX AND VAD EXTENSION

I.1 INTRODUCTION

The Mixed Excitation Linear Predictive enhanced (MELPe) with Discontinuous Transmission (DTX) [6] and Voice Activity Detection (VAD) [6] Extension, extends the original MELPe extension otherwise known as STANAG 4591 [5] is based upon the Vocoder Service API [A]. It extends the functionality of the Vocoder Service to include MELPe vocoding capabilities.

The DTX/VAD capabilities extend the functionality of the MELPe Extension to include VAD\(^1\) to automatically initiate DTX\(^1\). A TR platform supporting this extension transmits MELPe voice frames when the Vocoder’s VAD detects voice is present and does not transmit MELPe frames during speech gaps and pauses.

It should be noted that the use of the base Vocoder Service API [A] and codec extensions require an implicit or explicit connection with the Audio Port Device API [3]. Implicit connections are defined by the platform implementation (see Figure 1). Explicit connections are defined using the Vocoder Audio Stream Extension defined in section F.1 (see Figure 2).

**Important:** The MELPe Extension documented in section H was deprecated in a previous release of the Vocoder Service API. The MELPe with DTX and VAD Extension documented in section I is a superset of the MELPe Extension and recommended for new designs.

I.1.1 Overview

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

b. Section I.2, Services, provides summary of service interface uses, interface for each device component, port connections, and sequence diagrams.

c. Section I.3, Service Primitives and Attributes, specifies the operations that are provided by the MELPe Extension.

d. Section I.4, IDL

e. Section I.5, UML.

f. Appendix I.A, – Abbreviations and Acronyms.


\(^1\)For this extension, DTX refers to the ability to suspend transmission of MELPe packets over the air. This reduces transmission power and battery drain. In shared spectrum systems such as CDMA, the reduced transmission power also reduces in-channel congestion caused by mutual access interference (MAI). This behavior reduces per-connection usage of the shared-channel, enabling a corresponding increase in system capacity.
I.1.2 Service Layer Description

I.1.2.1 Vocoder Service API MELPe Extension Port Connections
The following figure shows the port connections for the Vocoder Service API [A] with extensions to support MELPe.

Note: All port names are for reference only. Ports identified in black are provided in section A.1.2.1.

![Vocoder Service API MELPe Extension Port Connections Diagram]

**Figure 32 – Vocoder Service MELPe with DTX and VAD Extension Port Diagram**

*Vocoder Service* MELPe Extension Provides Ports Definitions

*melpe_config_provides_port* is provided by the *Vocoder Service* to support the configuration of MELPe with DTX and VAD.

*Vocoder Service* MELPe Extension Uses Ports Definitions

None

I.1.3 Modes of Service
Not applicable

I.1.4 Service States
There are no changes from section A.1.4.

I.1.5 Referenced Documents
There are additions to section A.1.5.

[5] STANAG 4591 Ed. 1 The 600 Bit/s, 1200 Bit/s and 2400 Bit/s NATO Interoperable Narrow Band Voice Coder, 2008
I.2 SERVICES

I.2.1 Provide Services
The MELPe with DTX and VAD Extension interface provides services consisting of the following service ports, interfaces, and primitives, which can be called by other client components.

Table 16 – MELPe with DTX and VAD Extension Provide Service Interface

<table>
<thead>
<tr>
<th>Service Group (Port Name)</th>
<th>Service (Interface Provided)</th>
<th>Primitives (Provided)</th>
</tr>
</thead>
<tbody>
<tr>
<td>melpe_config_provides_port</td>
<td>Vocoder::Melpe</td>
<td>configureMelpe()</td>
</tr>
<tr>
<td></td>
<td></td>
<td>getMelpeConfig()</td>
</tr>
<tr>
<td></td>
<td></td>
<td>configureMelpeDtxVad()</td>
</tr>
<tr>
<td></td>
<td></td>
<td>getMelpeDtxVadConfig()</td>
</tr>
</tbody>
</table>

I.2.2 Use Services
None

I.2.3 Interface Modules

I.2.3.1 Vocoder

I.2.3.1.1 Melpe Interface Description
The Melpe interface is shown in the interface class diagram below. The Melpe interface provides the capability to configure and status the MELPe algorithm.

Figure 33 – MELPe with DTX and VAD Extension Class Diagram

I.2.4 Sequence Diagrams
None
I.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 I.5, UML. This cross-reference of types also includes any nested structures in the event of a structure of structures or an array of structures.

I.3.1 Vocoder::Melpe

I.3.1.1 configureMelpe Operation

The configureMelpe operation provides the ability to configure the MELPe algorithm.

I.3.1.1.1 Synopsis

void configureMelpe( in Config melpeCfg ) raises (JTRS::InvalidParameter);

I.3.1.1.2 Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>melpeCfg</td>
<td>Config</td>
<td>A structure containing the elements used for the initial configuration of the MELPe algorithm.</td>
</tr>
<tr>
<td></td>
<td>(see I.5.4.1)</td>
<td></td>
</tr>
</tbody>
</table>

I.3.1.1.3 State

Not applicable

I.3.1.1.4 New State

Not applicable

I.3.1.1.5 Return Value

None

I.3.1.1.6 Originator

Service User

I.3.1.1.7 Exceptions

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>JTRS::InvalidParameter</td>
<td>The configuration selected is invalid.</td>
</tr>
<tr>
<td>(see JTRS CORBA Types [1])</td>
<td></td>
</tr>
</tbody>
</table>
I.3.1.2 getMelpeConfig Operation
The getMelpeConfig operation provides the ability to status the MELPe algorithm configuration.

I.3.1.2.1 Synopsis

```
Config getMelpeConfig();
```

I.3.1.2.2 Parameters

None

I.3.1.2.3 State

Not applicable

I.3.1.2.4 New State

Not applicable

I.3.1.2.5 Return Value

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Config</td>
<td>A structure containing the elements used for the configuration of the MELPe algorithm.</td>
</tr>
</tbody>
</table>

I.3.1.2.6 Originator

Service User

I.3.1.2.7 Exceptions

None
I.3.1.3 configureMelpeDtxVad Operation
The configureMelpeDtxVad operation provides the ability to configure the MELPe algorithm and the DTX and VAD attributes.

I.3.1.3.1 Synopsis

void configureMelpeDtxVad( in MelpeDtxVadConfig melpeDtxVadCfg ) raises (JTRS::InvalidParameter, JTRS::Unsupported);

I.3.1.3.2 Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>melpeDtxVadCfg</td>
<td>MelpeDtxVadConfig</td>
<td>A structure containing the elements used for the initial configuration of the MELPe algorithm and the DTX and VAD functionality.</td>
</tr>
</tbody>
</table>

I.3.1.3.3 State
Not applicable

I.3.1.3.4 New State
Not applicable

I.3.1.3.5 Return Value
None

I.3.1.3.6 Originator
Service User

I.3.1.3.7 Exceptions

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>JTRS::InvalidParameter</td>
<td>The parameter selected is invalid for the TR platform’s Vocoder implementation.</td>
</tr>
<tr>
<td>(see JTRS CORBA Types [1])</td>
<td></td>
</tr>
<tr>
<td>JTRS::Unsupported</td>
<td>The TR platform does not support the MelpeDtxVad extension.</td>
</tr>
<tr>
<td>(see JTRS CORBA Types [1])</td>
<td></td>
</tr>
</tbody>
</table>
I.3.1.4 *getMelpeDtxVadConfig* Operation

The *getMelpeDtxVadConfig* operation provides the ability to status the MELPe algorithm and the DTX and VAD attributes.

I.3.1.4.1 Synopsis

```
MelpeDtxVadConfig getMelpeDtxVadConfig() raises (JTRS::Unsupported);
```

I.3.1.4.2 Parameters

None

I.3.1.4.3 State

Not applicable

I.3.1.4.4 New State

Not applicable

I.3.1.4.5 Return Value

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MelpeDtxVadConfig</td>
<td>A structure containing the current configuration of the MELPe algorithm</td>
</tr>
<tr>
<td>(see I.5.4.2)</td>
<td>and the DTX and VAD functionality.</td>
</tr>
</tbody>
</table>

I.3.1.4.6 Originator

Service User

I.3.1.4.7 Exceptions

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>JTRS::Unsupported</td>
<td>The TR platform does not support the MelpeDtxVad extension.</td>
</tr>
<tr>
<td>(see <em>JTRS CORBA Types</em> [1])</td>
<td></td>
</tr>
</tbody>
</table>
I.4 IDL

I.4.1 VocoderMelpeDtxVadExt

/*
** VocoderMelpeDtxVadExt.idl
*/

#ifndef __VOCODER_MELPEDTXVAD_EXT_DEFINED
#define __VOCODER_MELPEDTXVAD_EXT_DEFINED

#ifndef __VOCODER_DEFINED
#include "Vocoder.idl"
#endif

module Vocoder {
    const Algorithm ALG_MELPE = ALG_NONE + 7;

    interface Melpe {
        struct Config {
            unsigned short bitRat;
        }

        void configureMelpe( in Config melpeCfg ) raises (JTRS::InvalidParameter);
        Config getMelpeConfig();

        // Frame count type definition
        typedef unsigned short NumMelpeFrames;

        // Blank Period frame type generated for the listener at the
        // receiving end.
        typedef JTRS::ExtEnum FrameType;

        const FrameType GRACE_FRAME = 0; // Repeat last received Grace
        // Period frame.
        const FrameType SILENCE_FRAME = 1; // Generate silent frames.

        typedef JTRS::ExtEnum RestartType; // Restart Message behavior
        const RestartType RESTART_IMMEDIATE = 0; // Simply resume MELPe
        // frames.

        struct MelpeDtxVadConfig {
            unsigned short bitRate; // Requested MELPe bit rate.
            boolean enableMDV; // true = Use DTX/VAD.
            // false = Melpe with no DTX/VAD.
            NumMelpeFrames numGraceFrames; // Number of Grace Period frames
            // prior to Blank Period.
            // Ignored if enableMDV == false.
        }
    }
}

See cover page for distribution statement.
```c
NumMelpeFrames minBlankPerFrames;  // Minimum # of Blank Period
   // frames.  
   // Ignored if
   // enableMDV == false.
FrameType frameTypeToRepeat;       // Blank Period frame type to
   // generate at the receiving
   // end.  
   // Ignored if
   // enableMDV == false.
RestartType restartModeConfig;     // Selects restart behavior. 
   // Ignored if
   // enableMDV == false.
}
```

// The Tactical Radio (TR) platform raises the Unsupported exception
// if the MelpeDtxVad extension isn’t supported.
// The TR platform raises the Invalid Parameter exception for
// unsupported parameter values.
void configureMelpeDtxVad( in MelpeDtxVadConfig melpeDtxVadCfg )
raises (JTRS::InvalidParameter, JTRS::Unsupported);

    // The TR platform raises the Unsupported exception if the
    // MelpeDtxVad extension isn’t supported.
MelpeDtxVadConfig getMelpeDtxVadConfig() raises (JTRS::Unsupported);
```
I.5 UML
This section contains the Device component UML diagram and the definitions of all data types referenced (directly or indirectly) by the Service Primitives and Attributes in section I.3.

Figure 34 – MELPe with DTX and VAD Extension Component Diagram
I.5.1 Data Types
None

I.5.2 Enumerations

I.5.2.1 Vocoder::Algorithm
The following specifies the additional algorithms supported by the Vocoder Service that is defined in this extension.

\[
\text{const Algorithm ALG_MELPE = ALG_NONE + 7;}
\]

<table>
<thead>
<tr>
<th>JTRS::ExtEnum</th>
<th>Element</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Algorithm</td>
<td>ALG_MELPE</td>
<td>ALG_NONE + 7;</td>
<td>Identifies the MELPe Coding Algorithm</td>
</tr>
</tbody>
</table>

I.5.2.2 Vocoder::Melpe::FrameType
The Frame Type specifies how the receiving end generates MELPe frames for the listener during the Blank Period after the Grace Period ends.

\[
\text{typedef JTRS::ExtEnum FrameType;}
\]

<table>
<thead>
<tr>
<th>JTRS::ExtEnum</th>
<th>Element</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FrameType</td>
<td>GRACE_FRAME</td>
<td>0</td>
<td>Receiving end replicates the last-received Grace Period frame and sends it repeatedly to the listener as Comfort Noise until the end of the Blank Period.</td>
</tr>
<tr>
<td></td>
<td>SILENCE_FRAME</td>
<td>1</td>
<td>Receiving end generates silent frames and sends them to the listener until the end of the Blank Period.</td>
</tr>
</tbody>
</table>

I.5.2.3 Vocoder::Melpe::RestartType
The Restart Message type specifies how the Vocoder resumes transmission of MELPe packets following a Blank Period.

\[
\text{typedef JTRS::ExtEnum RestartType;}
\]

<table>
<thead>
<tr>
<th>JTRS::ExtEnum</th>
<th>Element</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RestartType</td>
<td>RESTART_IMMEDIATE</td>
<td>0</td>
<td>Resume transmission of MELPe packets immediately</td>
</tr>
</tbody>
</table>

I.5.3 Exceptions
None
I.5.4 Structures

I.5.4.1 Melpe::Config

The Config structure defines the attributes which define the properties of MELPe.

```cpp
struct Config {
    unsigned short bitRate;
};
```

<table>
<thead>
<tr>
<th>Struct</th>
<th>Attributes</th>
<th>Type</th>
<th>Units</th>
<th>Valid Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>Config</td>
<td>bitRate</td>
<td>unsigned short</td>
<td>bps</td>
<td>600, 1200, 2400</td>
</tr>
</tbody>
</table>

I.5.4.2 Melpe::MelpeDtxVadConfig

The Config structure defines the attributes which define the properties of MELPe.

```cpp
struct MelpeDtxVadConfig {
    unsigned short bitRate; // Requested MELPe bit rate.
    boolean enableMDV; // true = Use DTX/VAD.
    // false = Melpe with no DTX/VAD.
    NumMelpeFrames numGraceFrames; // Number of Grace Period frames
    // prior to Blank Period.
    // Ignored if enableMDV == false.
    NumMelpeFrames minBlankPerFrames; // Minimum # of Blank Period
    // frames.
    // Ignored if
    // enableMDV == false.
    FrameType frameTypeToRepeat; // Blank Period frame type to
    // generate at the receiving
    // end.
    // Ignored if
    // enableMDV == false.
    RestartType restartModeConfig; // Selects restart behavior.
    // Ignored if
    // enableMDV == false.
};
```

<table>
<thead>
<tr>
<th>Struct</th>
<th>Attributes</th>
<th>Type</th>
<th>Units</th>
<th>Valid Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>MelpeDtxVadConfig</td>
<td>bitRate</td>
<td>unsigned short</td>
<td>bps</td>
<td>600, 1200, 2400</td>
</tr>
<tr>
<td></td>
<td>enableMDV</td>
<td>boolean</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td></td>
<td>numGraceFrames</td>
<td>NumMelpeFrames</td>
<td>Frames</td>
<td>0-unbounded</td>
</tr>
<tr>
<td></td>
<td>minBlankPerFrames</td>
<td>NumMelpeFrames</td>
<td>Frames</td>
<td>1-unbounded</td>
</tr>
<tr>
<td></td>
<td>frameTypeToRepeat</td>
<td>FrameType</td>
<td>N/A</td>
<td>Per FrameType (I.5.2.2)</td>
</tr>
<tr>
<td></td>
<td>restartModeConfig</td>
<td>RestartType</td>
<td>N/A</td>
<td>Per RestartType (I.5.2.3)</td>
</tr>
</tbody>
</table>
APPENDIX I.A – ABBREVIATIONS AND ACRONYMS

The following list additional abbreviations and acronyms not specified in Appendix A.A.

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>CDMA</td>
<td>Code Divided Multiple Access</td>
</tr>
<tr>
<td>DTX</td>
<td>Discontinuous Transmission</td>
</tr>
<tr>
<td>MAI</td>
<td>Mutual Access Interference</td>
</tr>
<tr>
<td>MELPe</td>
<td>Mixed Excitation Linear Predictive enhanced</td>
</tr>
<tr>
<td>STANAG</td>
<td>NATO abbreviation for Standardization Agreement</td>
</tr>
<tr>
<td>VAD</td>
<td>Voice Activity Detection</td>
</tr>
</tbody>
</table>

APPENDIX I.B – PERFORMANCE SPECIFICATION

The following table provides a template for the generic performance specification for the MELPe Extension. This performance specification corresponds to the port diagram in Figure 32.

Table 17 – MELPe with DTX and VAD Extension Performance Specification

<table>
<thead>
<tr>
<th>Specification</th>
<th>Description</th>
<th>Units</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Worst Case Command Execution Time for vocoder_packet_consumer_provides_port</td>
<td>*</td>
<td>*</td>
<td>*</td>
</tr>
</tbody>
</table>

*Note this template will be filled in by individual developers.*
J. G.711 EXTENSION

J.1 INTRODUCTION

The G.711 Extension is based upon the Vocoder Service API [A]. It extends the functionality of the Vocoder Service API [A] to include G.711 vocoding capabilities. G.711 is an International Telecommunications Union (ITU) recommendation for audio companding. This specification includes the capabilities in the base G.711 specification [13] only, and does not include the additional capabilities included in G.711 Appendix I (packet loss concealment) [14] or G.711 Appendix II (comfort noise / VAD) [15] or (G.711 Amendment 1 (Lossless compression) [16] or G.711 Amendment 2 (toolbox enhancements)[17].

G.711 Amendment 2 (toolbox enhancements) [17] and G.711 Appendix I (packet loss concealment) [14] specify encoder/decoder enhancements that are interoperable with the base specification. Such use of these enhancements is implementation-dependent.

It should be noted that the use of the base Vocoder Service API [A] and codec extensions require an implicit or explicit connection with the Audio Port Device API [3]. Implicit connections are defined by the platform implementation (see Figure 1). Explicit connections are defined using the Vocoder Audio Stream Extension defined in section F.1 (see Figure 2).

J.1.1 Overview

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

b. Section J.2, Services, provides summary of service interface uses, interface for each device component, port connections, and sequence diagrams.

c. Section J.3, Service Primitives and Attributes, specifies the operations that are provided by the G.711 Extension.

d. Section J.4, IDL.

e. Section J.5, UML.

f. Appendix J.A, – Abbreviations and Acronyms.

J.1.2 Service Layer Description

J.1.2.1 Vocoder Service API G.711 Extension Port Connections

The following figure shows the port connections for the Vocoder Service API [A] with extensions to support G.711.

Note: All port names are for reference only. Ports identified in black are provided in Section A.1.2.1.

Figure 35 – Vocoder Service API G.711 Extension Port Diagram

Vocoder Service API G.711 Extension Provides Ports Definitions

G711_config_provides_port is provided by the Vocoder Service to support the configuration of G.711

Vocoder Service API G.711 Extension Uses Ports Definitions

None

J.1.3 Modes of Service

Not applicable

J.1.4 Service States

There are no changes from Section A.1.4

J.1.5 Referenced Documents

There are additions to section A.1.5
J.2 SERVICES

J.2.1 Provide Services

The G.711 Extension interface provides services consisting of the following service ports, interfaces, and primitives, which can be called by other client components.

Table 18 – G.711 Extension Provide Service Interface

<table>
<thead>
<tr>
<th>Service Group (Port Name)</th>
<th>Service (Interface Provided)</th>
<th>Primitives (Provided)</th>
</tr>
</thead>
<tbody>
<tr>
<td>g729_config_provides_port</td>
<td>Vocoder::G711</td>
<td>configureG711()</td>
</tr>
<tr>
<td></td>
<td></td>
<td>getG711Config()</td>
</tr>
</tbody>
</table>

J.2.2 Use Services

None

J.2.3 Interface Modules

J.2.3.1 Vocoder

J.2.3.1.1 G.711 Interface Description

The G711 interface is shown in the interface class diagram below. The G711 interface provides the capability to configure and status the G.711 algorithm.

Figure 36 – G.711 Extension Class Diagram
J.2.4 Sequence Diagram
None

J.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 J.5, UML. This cross-reference of types also includes any nested structures in the event of a structure of structures or an array of structures.

J.3.1 Vocoder::G711

J.3.1.1 configureG711 Operation
The configureG711 operation provides the ability to configure the G.711 algorithm.

J.3.1.1.1 Synopsis
void configureG711(in Config g711Cfg ) raises (JTRS::InvalidParameter);

J.3.1.1.2 Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>g711Cfg</td>
<td>Config</td>
<td>A structure containing the elements used for the initial configuration of the G.711 algorithm.</td>
</tr>
</tbody>
</table>

J.3.1.1.3 State
Not applicable

J.3.1.1.4 New State
Not applicable

J.3.1.1.5 Return Value
None

J.3.1.1.6 Originator
Service User

J.3.1.1.7 Exceptions

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>JTRS::InvalidParameter</td>
<td>The configuration selected is invalid.</td>
</tr>
</tbody>
</table>

(see JTRS CORBA Type [1])
J.3.1.2 getG711Config Operation
The getG711Config operation provides the ability to status the G.711 algorithm configuration.

J.3.1.2.1 Synopsis

```
Config getG711Config();
```

J.3.1.2.2 Parameters
None

J.3.1.2.3 State
Not applicable

J.3.1.2.4 New State
Not applicable

J.3.1.2.5 Return

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Config (see G711::Config)</td>
<td>A structure containing the elements used for the configuration of the G.711 algorithm.</td>
</tr>
</tbody>
</table>

J.3.1.2.6 Originator
Service User

J.3.1.2.7 Exceptions
None
J.4 IDL

J.4.1 VocoderG711Ext

/*
** VocoderG711Ext.idl
*/
#ifndef __VOCODER_G711_EXT_DEFINED
#define __VOCODER_G711_EXT_DEFINED
#ifndef __VOCODER_DEFINED
#include "Vocoder.idl"
#endif
module Vocoder
{
    const Algorithm ALG_G711 = ALG_NONE + 8;

    interface G711
    {
        typedef JTRS::ExtEnum EncType;

        //Known Configurations Types
        const EncType Enc_NONE = 0;
        const EncType U_LAW = Enc_NONE + 1;
        const EncType A_LAW = U_LAW + 1;
        const EncType Enc_MAX = A_LAW + 1;

        struct Config
        {
            EncType encodingType; // a-Law or u-Law
        };

        void configureG711(in Config g711Cfg ) raises (JTRS::InvalidParameter);
        Config getG711Config();
    };
#endif //__VOCODER_G711_EXT_DEFINED
J.5 UML

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

![UML Diagram](image)

Figure 37 – G.711 Extension Component Diagram

J.5.1 Data Types

None

J.5.2 Enumerations

J.5.2.1 Vocoder::Algorithm

The following specifies the additional algorithms supported by the Vocoder Service that is defined in this extension.

```
const Algorithm ALG_G711 = ALG_NONE + 8;
```

<table>
<thead>
<tr>
<th>JTRS::ExtEnum</th>
<th>Element</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Algorithm</td>
<td>ALG_G711</td>
<td>ALG_NONE + 8;</td>
<td>Identifies the G.711 Algorithm</td>
</tr>
</tbody>
</table>

J.5.2.2 Vocoder::G711::EncType

The EncType type definition is an Extension enumeration. It enumerates the G.711 Encoding Types.

```
typedef JTRS::ExtEnum   EncType;

//Known Configurations Types
const EncType Enc_NONE = 0;
const EncType U_LAW = Enc_NONE + 1;
const EncType A_LAW = U_LAW + 1;
const EncType Enc_MAX = A_LAW + 1;
```
Note that Enc_NONE and Enc_MAX are invalid configuration types. Should they be selected, a JTRS::InvalidParameter exception will be raised.

<table>
<thead>
<tr>
<th>JTRS::ExtEnum</th>
<th>Element</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EncType</td>
<td>Enc_NONE</td>
<td>0</td>
<td>No encoding</td>
</tr>
<tr>
<td></td>
<td>U_LAW</td>
<td>Enc_NONE+1</td>
<td>U Law</td>
</tr>
<tr>
<td></td>
<td>A_LAW</td>
<td>U_LAW+1</td>
<td>A Law</td>
</tr>
<tr>
<td></td>
<td>Enc_MAX</td>
<td>A_LAW+1</td>
<td>Maximum number of encoding types plus 1</td>
</tr>
</tbody>
</table>
J.5.3 Exceptions
None

J.5.4 Structures

J.5.4.1 G711::Config
The Config structure defines the attributes which define the properties of G.711.

```
struct Config
{
    EncType encodingType; // a-Law or u-Law
};
```

<table>
<thead>
<tr>
<th>Struct</th>
<th>Attributes</th>
<th>Type</th>
<th>Valid Range</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Config</td>
<td>encodingType</td>
<td>EncType</td>
<td>U_LAW or A_LAW</td>
<td>Determines whether to use U_LAW or A_LAW encoding.</td>
</tr>
</tbody>
</table>
APPENDIX J.A  ABBREVIATIONS AND ACRONYMS
The following list additional abbreviations and acronyms not specified in Appendix A.A.
ITU International Telecommunications Union

APPENDIX J.B  PERFORMANCE SPECIFICATION
The following table provides a template for the generic performance specification for the G.711 Extension. This performance specification corresponds to the port diagram in Figure 29.

<table>
<thead>
<tr>
<th>Specification</th>
<th>Description</th>
<th>Units</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Worst Case Command Execution Time for g729_config_provides_port vocoder_packet_consumer_provides_port</td>
<td>*</td>
<td>*</td>
<td>*</td>
</tr>
</tbody>
</table>

*Note this template will be filled in by individual developers.