INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH INTEL® PRODUCTS. NO LICENSE, EXPRESS OR IMPLIED, BY
ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. EXCEPT AS PROVIDED IN
INTEL’S TERMS AND CONDITIONS OF SALE FOR SUCH PRODUCTS, INTEL ASSUMES NO LIABILITY WHATSOEVER, AND INTEL DISCLAIMS
ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO SALE AND/OR USE OF INTEL PRODUCTS INCLUDING LIABILITY OR WARRANTIES
RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER
INTELLECTUAL PROPERTY RIGHT. Intel products are not intended for use in medical, life saving, life sustaining applications.

Intel may make changes to specifications and product descriptions at any time, without notice.

Designers must not rely on the absence or characteristics of any features or instructions marked “reserved” or “undefined.” Intel reserves these for
future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them.

The Intel® IXP400 DSP Software v.2.4 may contain design defects or errors known as errata which may cause the product to deviate from published
specifications. Current characterized errata are available on request.

Contact your local Intel sales office or your distributor to obtain the latest specifications and before placing your product o rder.

Copies of documents which have an ordering number and are referenced in this document, or other Intel literature may be obtained by calling

AlertView, AnyPoint, AppChoice, BoardWatch, BunnyPeople, CablePort, Celeron, Chips, CT Connect, CT Media, Dialogic, DM3, EtherExpress,
ETOX, FlashFile, i386, i486, i960, iCOMP, InstantIP, Intel, Intel logo, Intel®386, Intel486, Intel740, IntelDX2, IntelDX4, IntelSX2, Intel Create & Share,
SingleDriver, Intel SpeedStep, Intel StrataFlash, Intel TeamStation, Intel Xeon, Intel XScale, I/Link, Itanium, LANDesk, LanRover, MCS, MMX, MMX
logo, Optimizer logo, OverDrive, Paragon, PC Dads, PC Parents, PCDcharm, Pentium, Pentium II Xeon, Pentium III Xeon, Performance at Your
Command, RemoteExpress, Shiva, SmartDie, Solutions960, Sound Mark, StorageExpress, The Computer Inside., The Journey Inside,
TokenExpress, Trillium, VoiceBrick, Vtune, and Xircom are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United
States and other countries.

Copyright © Intel Corporation, 2004

*Other names and brands may be claimed as the property of others.
## Contents

1.0 Introduction ............................................................................................................ 7
   1.1 General ............................................................................................................ 7
   1.2 Scope .......................................................................................................... 7
   1.3 Audience ...................................................................................................... 7
   1.4 Acronyms ..................................................................................................... 7
   1.5 Related Documentation .............................................................................. 8

2.0 Architecture Overview .......................................................................................... 9

3.0 Media-Processing Resource Components ............................................................. 10
   3.1 Network-Endpoint Resource Component ..................................................... 11
   3.2 Decoder Resource Component ..................................................................... 12
   3.3 Encoder Resource Component ..................................................................... 13
   3.4 Tone-Generation Resource Component ....................................................... 13
   3.5 Tone-Detection Resource Component .......................................................... 14
   3.6 Audio Player Resource Component .............................................................. 15
   3.7 Audio Mixer Resource Component ............................................................... 16
   3.8 T.38 Fax Resource Component ..................................................................... 16
   3.9 Message Agent Resource Component .......................................................... 17

4.0 Message Format and Delivery Mechanism ........................................................... 18
   4.1 Message Functions ....................................................................................... 18
   4.2 Message Header Format ............................................................................... 19
   4.3 Message Type List ........................................................................................ 20

5.0 Common Control Message ..................................................................................... 21
   5.1 Reset Message ............................................................................................. 21
   5.2 Start Message ............................................................................................... 21
   5.3 Stop Message ............................................................................................... 22
   5.4 Ping Message ................................................................................................ 22
   5.5 Set-Parameter Message ............................................................................... 22
   5.6 Set-Multiple-Parameter Message .................................................................. 23
   5.7 Get-Parameter Message ............................................................................... 23
   5.8 Get-Parameter-Acknowledge Message ........................................................ 24
   5.9 Get-All-Parameters Message ....................................................................... 24
   5.10 Get-All-Parameters-Acknowledge Message ............................................... 24
   5.11 General-Acknowledge Message .................................................................. 25
   5.12 Error Message ............................................................................................. 25
   5.13 Event Message ............................................................................................. 26

6.0 Resource-Specific Control Message ..................................................................... 27
   6.1 CODEC Start Message ............................................................................... 27
   6.2 CODEC Stop-Acknowledgement Message ................................................... 27
   6.3 Tone-Generator-Play Message ..................................................................... 28
   6.4 Tone-Generator-Play-FSK Message ............................................................. 28
   6.5 Tone-Generator-Play-Completed Message .................................................. 29
   6.6 Tone-Detector-Receive-Digit Message .......................................................... 29
Contents

6.7 Tone-Detector-Receive-Completed Message ................................................. 30
6.8 Tone-Detector-Receive-FSK Message ............................................................. 30
6.9 Tone-Detector-FSK-Receive-Completed Message .......................................... 31
6.10 Player-Start Message .................................................................................. 31
6.11 Player-Play-Completed Message ................................................................ 32
6.12 Get-Jitter-Buffer-Statistics Message ............................................................ 33
6.13 Complete Message of Getting Jitter Buffer Statistics ................................... 33

7.0 Packet Data Interface .................................................................................... 34
7.1 Packet Formats .............................................................................................. 34
7.2 Packet Delivery Mechanism .......................................................................... 35

8.0 Configuration and Initialization ..................................................................... 36
8.1 System Configuration .................................................................................... 36
8.2 Adding Tones to Tone Generator ................................................................... 37
8.3 Adding Tones to Tone Detector ..................................................................... 38
8.4 Getting DSP Resource Configuration and Routing Information .................... 39

9.0 Complementary Functions ............................................................................. 41
9.1 Direct Parameter Access ............................................................................... 41
9.2 Flash Hook Detection ................................................................................... 41
9.3 Cache Prompt Registration ........................................................................... 42

10.0 Constant Data ............................................................................................... 43
10.1 Error Codes .................................................................................................. 43
10.2 Event Codes ................................................................................................ 43
10.3 Tone IDs ....................................................................................................... 44
10.3.1 DTMF Tone IDs .................................................................................... 44
10.3.2 Fax-Tone IDs ....................................................................................... 44
10.3.3 Call-Progression IDs ............................................................................ 45
10.4 Other Constants ............................................................................................ 46

Figures
1 Intel® IXP400 DSP Software v.2.4 Architecture ................................................. 9
2 Resource-Component Identifiers ...................................................................... 10

Tables
None.
## Revision History

<table>
<thead>
<tr>
<th>Date</th>
<th>Revision</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>January 2004</td>
<td>005</td>
<td>Updates for the release of Intel® IXP400 DSP Software Version 2.4</td>
</tr>
<tr>
<td>September 2003</td>
<td>004</td>
<td>Clarified input for XStatus_t xMsgReceive message function.</td>
</tr>
<tr>
<td>September 2003</td>
<td>003</td>
<td>Updates for the release of Intel® IXP400 DSP Software Version 2.3</td>
</tr>
<tr>
<td>March 2003</td>
<td>002</td>
<td>Added minor updates to represent features of Intel® IXP400 DSP Software Version 1.1.</td>
</tr>
<tr>
<td>January 2003</td>
<td>001</td>
<td>First release of this document.</td>
</tr>
</tbody>
</table>
1.0 Introduction

Intel® IXP400 DSP Software is a software module that provides the basic voice-processing functionalities for Voice-over-Internet-Protocol (VoIP) residential gateway applications. It can be viewed as a complete, media-processing layer with control and data interfaces as its API.

This document defines the API specifications.

1.1 General

Intel® IXP400 DSP Software is a software module for media processing, targeted for next-generation Integrated Access Devices (IADs) — such as Consumer Premise Equipment (CPE), specifically, to perform audio encoding/decoding, echo cancellation, tone processing and jitter control — as required in any IP media gateway or real-time, media-streaming functionalities.

This document is intended to describe the control and data interfaces for a third-party developer to incorporate the module into a media gateway or server system. It provides sufficient details about the interfaces so that users can fully configure and control the operations and services.

This document also describes the data interface and format as well as message and data-delivery mechanisms.

1.2 Scope

The interface of DSP software is a set of functions, macros, messages, and packet formats that determines how the applications access the media-processing resource components.

1.3 Audience

This document is intended for the following audiences

- Firmware engineers who are responsible for the development of DSP resources
- Third-party software engineers who are building gateway or server applications
- System architects and engineers
- Project development managers

1.4 Acronyms

AGC Automatic Gain Control for voice data towards IP network
ALC Automatic Level Control
CPE Consumer Premise Equipment
EC Echo Cancellation
FSK Frequency Shift Keying
IP Internet Protocol
ISR    Interrupt Service Routine
NLP    Non-linear Processing (for EC)
SP     Signal Processing
VAD    Voice Activity Detection

1.5 Related Documentation

<table>
<thead>
<tr>
<th>Title</th>
<th>Document Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intel® IXP400 Digital Signal Processing (DSP) Software Version 2.4 Release Notes</td>
<td>N/A</td>
</tr>
<tr>
<td>Intel® IXP400 Software Programmer’s Guide</td>
<td>252539</td>
</tr>
</tbody>
</table>
2.0 Architecture Overview

Intel® IXP400 DSP Software is implemented as an independent module having its own tasks and runtime environment. The software architecture is of a two-layer hierarchy – a control layer that provides the control interface and control logic and a data processing layer where the media data streams are processed by appropriate algorithms. Figure 1 shows the decomposition of the module.

In this architecture, a group of media resource (MPR) components forms a channel for full-duplex media processing. They are the addressable entities that can be controlled individually by the applications.

Figure 1. Intel® IXP400 DSP Software v.2.4 Architecture
3.0 Media-Processing Resource Components

As shown in Figure 1, the addressable control entities of the DSP software are media-processing resource (MPR) components similar to those defined in many Intel® Dialogic® computer-telephony system architecture.

There are nine resource components, working together to provide all media processing needed by a gateway or server channel. Each resource component has a unique identifier as shown below. In the following, we will refer to each of these eight media-processing entities as either a resource or a resource component.

Figure 2. Resource-Component Identifiers

```c
typedef enum{
    XMPR_ANY=0, /* Any resource, not supported in */
    XMPR_NET,   /* Network Endpoint resource */
    XMPR_DEC,   /* Decoder resource */
    XMPR_ENC,   /* Encoder resource */
    XMPR_TNGEN, /* Tone generator resource */
    XMPR_TNDET, /* Tone detector resource */
    XMPR_PLY,   /* Audio player resource */
    XMPR_MIX,   /* Audio mixer resource */
    XMPR_T38,   /* T38 fax resource */
    XMPR_MA     /* Message Agent resource */
} XMPResource_t;
```

Each resource contains a particular set of algorithms to perform a specific set of media-processing functions. For example, the Network Endpoint resource consists of echo cancellation, high-pass filter and PCM data conversion algorithms to perform TDM front-end processing. Each resource, therefore, has a unique set of parameters associated with the particular set of algorithms it contains.

Communications of control information to these resource components are through messages defined in this document. Some messages are common to all resources while others are unique only to a particular resource.

The following sections describe each resource in terms of their identifiers, media-processing functions, parameters, and control messages. The resource parameters can be read or modified by the messages or direct function calls. Some of the parameters can only be set through the messages because they can only be updated by the internal control task.
## 3.1 Network-Endpoint Resource Component

### Media Processing Functions
- A-law or µ-law compression and decompression
- High-Pass Filter
- Echo Cancellation (EC)
- Supplementary functions (timer and flash hook detection)

### Resource-Specific Control Messages
None

### Parameters

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description, Values</th>
<th>Attr.</th>
<th>Direct Write</th>
</tr>
</thead>
<tbody>
<tr>
<td>XPARMID_RES_STATE</td>
<td>Current state (0: idle, 1: active)</td>
<td>R</td>
<td>N</td>
</tr>
<tr>
<td>XPARMID_NET_LP_STREAM</td>
<td>The L-Port stream ID. Default: the stream assigned to the IP termination’s T-Port of the same channel if exist, otherwise −1.</td>
<td>R/W</td>
<td>N</td>
</tr>
<tr>
<td>XPARMID_NET_LAW</td>
<td>PCM data format on HSS TDM bus. XPARM_NET_ALAW or XPARM_NET_MULAW. Default: XPARM_NET_MULAW</td>
<td>R/W</td>
<td>N</td>
</tr>
<tr>
<td>XPARMID_NET_ECENABLE</td>
<td>EC enabling flag. XPARM_ON or XPARM_OFF. Default: XPARM_ON</td>
<td>R/W</td>
<td>Y</td>
</tr>
<tr>
<td>XPARMID_NET_ECTAIL</td>
<td>EC tail length (2, 4, 6, 8, ... up to 64 in 1 ms unit). Default: 6. The resource must be reset after setting the parameter.</td>
<td>R/W</td>
<td>N</td>
</tr>
<tr>
<td>XPARMID_NET_ECNLP</td>
<td>EC NLP and suppress flag, XPARM_OFF, XPARM_EC_NLP_ON or XPARM_EC_NLP_SUP_ON. Default: XPARM_OFF</td>
<td>R/W</td>
<td>N</td>
</tr>
<tr>
<td>XPARMID_NET_ECFREEZE</td>
<td>EC freezing flag. XPARM_ON (freeze) or XPARM_OFF (adaptive). Typically, freeze is used only in debug situations. Default: XPARM_OFF</td>
<td>R/W</td>
<td>N</td>
</tr>
<tr>
<td>XPARMID_NET_DELAYCOMP</td>
<td>EC delay compensation (0 ~ 240 in 0.125 ms unit). Default: 20 (or 2.5 ms delay compensation)</td>
<td>R/W</td>
<td>Y</td>
</tr>
<tr>
<td>XPARMID_NET_FLASH_HK</td>
<td>The window of flash hook detection (in 10-ms unit) Default: 100</td>
<td>R/W</td>
<td>Y</td>
</tr>
<tr>
<td>XPARMID_NET_TIMER</td>
<td>Timer counter (in 10-ms units). This timer can be used for timing that is synchronized to the TDM clock. Default: 0</td>
<td>R/W</td>
<td>Y</td>
</tr>
<tr>
<td>XPARMID_NET_GAIN_RX</td>
<td>Input gain of HSS interface (+15 ~ −40 in 1-dB units) Default: 0</td>
<td>R/W</td>
<td>N</td>
</tr>
<tr>
<td>XPARMID_NET_GAIN_TX</td>
<td>Output gain of HSS interface (+15 ~ −40 in 1-dB units) Default: 0</td>
<td>R/W</td>
<td>N</td>
</tr>
<tr>
<td>XPARMID_NET_HSS_BYPASS</td>
<td>TDM short bypass flag. XPARM_ON or XPARM_OFF. Default: XPARM_OFF</td>
<td>R/W</td>
<td>N</td>
</tr>
</tbody>
</table>

### Events
- XEVT_NET_HOOK_STATE – Hook state change detected.
- EVT_NET_TIMER – Timer expired.
### 3.2 Decoder Resource Component

<table>
<thead>
<tr>
<th>Resource Type</th>
<th>XMPR_DEC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Media Processing Functions</td>
<td></td>
</tr>
<tr>
<td>• Decoding</td>
<td></td>
</tr>
<tr>
<td>• Automatic level control and/or volume control</td>
<td></td>
</tr>
<tr>
<td>• Comfort noise generation</td>
<td></td>
</tr>
<tr>
<td>• Jitter compensation</td>
<td></td>
</tr>
<tr>
<td>Resource-Specific Control Messages</td>
<td></td>
</tr>
<tr>
<td>• XMSG_CODER_START (inbound)</td>
<td></td>
</tr>
<tr>
<td>• XMSG_CODER_STOP_ACK (outbound)</td>
<td></td>
</tr>
</tbody>
</table>

#### Parameters

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description, Values</th>
<th>Attr.</th>
<th>Direct Write</th>
</tr>
</thead>
<tbody>
<tr>
<td>XPARMID_RES_STATE</td>
<td>Current state (0: idle, 1: active)</td>
<td>R</td>
<td>N</td>
</tr>
<tr>
<td>XPARMID_DEC_VOL</td>
<td>Decoder volume adjustment; +15 ~ –30 in 1 dB unit. Default: 0</td>
<td>R/W</td>
<td>N</td>
</tr>
<tr>
<td>XPARMID_DEC_ALC</td>
<td>ALC enable flag. XPARM_ON or XPARM_OFF. Default: XPARM_ON</td>
<td>R/W</td>
<td>N</td>
</tr>
<tr>
<td>XPARMID_DEC_CNG</td>
<td>CNG enable flag. XPARM_ON or XPARM_OFF. Default: XPARM_OFF</td>
<td>R/W</td>
<td>Y</td>
</tr>
<tr>
<td>XPARMID_DEC_CTYPE</td>
<td>Coder type. Currently supported types are XCODER_TYPE_G711MU_10MS,</td>
<td>R/W</td>
<td>N</td>
</tr>
<tr>
<td></td>
<td>XCODER_TYPE_G711A_10MS, XCODER_TYPE_G729A, or XCODE_TYPE_G723. Default: XCODER_TYPE_G711MU_10MS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>XPARMID_DEC_EVT_PKT</td>
<td>Report bad and lost packet, caused by the jitter buffer unable to provide packets to the decoder. XPARM_ON or XPARM_OFF. Default: XPARM_OFF</td>
<td>R/W</td>
<td>Y</td>
</tr>
<tr>
<td>XPARMID_DEC_EVT_PKTCHNG</td>
<td>Report RTP payload type change. XPARM_ON or XPARM_OFF. Default: XPARM_ON.</td>
<td>R/W</td>
<td>Y</td>
</tr>
<tr>
<td>XPARMID_DEC_AUTOSW</td>
<td>Auto-Switch mask bits. This specifies which coder types are allowed to be auto-switched based on input RTP payload type. Default: XPARM_DEC_AUTOSW_ALL</td>
<td></td>
<td>Y</td>
</tr>
<tr>
<td>XPARMID_DEC_JB_MAXDLY</td>
<td>Jitter buffer maximum delay (0 ~ 500 in 1-ms unit). Default: 200.</td>
<td>R/W</td>
<td>N</td>
</tr>
<tr>
<td>XPARMID_DEC_JB_PLR</td>
<td>Jitter buffer packet loss rate in 0.1% unit. Default: 1</td>
<td>R/W</td>
<td>N</td>
</tr>
</tbody>
</table>

#### Events

- XEVET_LOST_PACKET – Bad or lost packet.
- XEVET_DEC_PACKET_CHNG – RTP payload type changed.
### 3.3 Encoder Resource Component

<table>
<thead>
<tr>
<th>Resource Type</th>
<th>Media Processing Functions</th>
<th>Resource-Specific Control Messages</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>• Encoding</td>
<td>• XMSG_CODER_START (inbound)</td>
</tr>
<tr>
<td></td>
<td>• Automatic Gain Control</td>
<td>• XMSG_CODER_STOP_ACK (outbound)</td>
</tr>
<tr>
<td></td>
<td>• Voice-Activity Detection</td>
<td></td>
</tr>
</tbody>
</table>

#### Parameters

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description, Values</th>
<th>Attr.</th>
<th>Direct Write</th>
</tr>
</thead>
<tbody>
<tr>
<td>XPARMID_RES_STATE</td>
<td>Current state (0: idle, 1: active)</td>
<td>R</td>
<td>N</td>
</tr>
<tr>
<td>XPARMID_ENC_LP_STREAM</td>
<td>L-Port stream ID. Default: the stream assigned to the TDM termination's T-Port of the same channel if exist, otherwise –1.</td>
<td>R/W</td>
<td>N</td>
</tr>
<tr>
<td>XPARMID_ENC_AGC</td>
<td>AGC enable flag. XPARM_ON or XPARM_OFF. Default: XPARM_OFF</td>
<td>R/W</td>
<td>N</td>
</tr>
<tr>
<td>XPARMID_ENC_VAD</td>
<td>VAD enable flag. XPARM_ON or XPARM_OFF. Default: XPARM_OFF</td>
<td>R/W</td>
<td>N</td>
</tr>
<tr>
<td>XPARMID_ENC_CTYPE</td>
<td>Coder type. Currently supported types are XCODER_TYPE_G711MU_10MS, XCODER_TYPE_G711A_10MS, XCODER_TYPE_G729A or XCODE_TYPE_G723. Default: XCODER_TYPE_G711MU_10MS</td>
<td>R/W</td>
<td>N</td>
</tr>
<tr>
<td>XPARMID_ENC_MFPP</td>
<td>Number of frames per packet. Supported range is 1<del>6 for G.711, 1</del>8 for G.723 and 1~24 for G.729. Default: 1.</td>
<td>R/W</td>
<td>N</td>
</tr>
<tr>
<td>XPARMID_ENC_EVT_PKT</td>
<td>Enable packet lost event. XPARM_ON or XPARM_OFF. Default: XPARM_OFF</td>
<td>R/W</td>
<td>Y</td>
</tr>
</tbody>
</table>

#### Events

- XEV_T_GIT_PEER_DISCONNECT – Peer disconnected
- XEV_GIT_PEER_DISCONNECT – Gateway disconnect
- XEV_T_GIT_CONNECT – Peer connected
- XEV_GIT_CONNECT – Gateway connect
- XEV_T_GIT потерять – Peer lost
- XEV_GIT потерять – Gateway lost
- XEV_T_GIT потерять – Lost packet

### 3.4 Tone-Generation Resource Component

<table>
<thead>
<tr>
<th>Resource Type</th>
<th>Media Processing Functions</th>
<th>Resource-Specific Control Messages</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>• Generating multiple-frequency tone signals</td>
<td>• XMSG_TG_PLAY (inbound)</td>
</tr>
<tr>
<td></td>
<td>• Generating call-progress tones</td>
<td>• XMSG_TG_PLAY_FSK (inbound)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• XMSG_TG_PLAY_CMPLT (outbound)</td>
</tr>
</tbody>
</table>

#### Parameters

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description, Values</th>
<th>Attr.</th>
<th>Direct Write</th>
</tr>
</thead>
<tbody>
<tr>
<td>XPARMID_RES_STATE</td>
<td>Current state (0: idle, 1: active)</td>
<td>R/W</td>
<td>N</td>
</tr>
<tr>
<td>XPARMID_TNGEN_VOL</td>
<td>Tone Generator's volume adjustment, +15 ~ –20 in dB unit. Default: 0</td>
<td>R/W</td>
<td>Y</td>
</tr>
</tbody>
</table>
3.5 Tone-Detection Resource Component

<table>
<thead>
<tr>
<th>Resource Type</th>
<th>Description</th>
<th>Attr.</th>
<th>Direct Write</th>
</tr>
</thead>
<tbody>
<tr>
<td>XPARMID_TNGEN_FSK_CS</td>
<td>CS bit length of FSK modulator (in bit unit). Default: 300 if country code set to COUNTRY_CODE_US or COUNTRY_CODE_PRC, otherwise 0.</td>
<td>R/W Y</td>
<td></td>
</tr>
<tr>
<td>XPARMID_TNGEN_FSK_MARK</td>
<td>Mark bit length of FSK modulator (in bit unit). Default: 180 if country code set to COUNTRY_CODE_US or COUNTRY_CODE_PRC, otherwise 100.</td>
<td>R/W Y</td>
<td></td>
</tr>
<tr>
<td>XPARMID_TNGEN_FSK_RATE</td>
<td>FSK modulator baud rate (XPARM_TNGEN_FSK_R1200, XPARM_TNGEN_FSK_R600, XPARM_TNGEN_FSK_R300, XPARM_TNGEN_FSK_R150, or XPARM_TNGEN_FSK_R75). Default: XPARM_TNGEN_FSK_R1200, i.e., 1200 bps</td>
<td>R/W N</td>
<td></td>
</tr>
<tr>
<td>XPARMID_TNGEN_RFC2833</td>
<td>RFC2833 enable flag. XPARM_ON or XPARM_OFF. Default: XPARM_ON</td>
<td>R/W N</td>
<td></td>
</tr>
</tbody>
</table>

Events
None
### 3.6 Audio Player Resource Component

<table>
<thead>
<tr>
<th>Resource Type</th>
<th>Description</th>
<th>Attr.</th>
<th>Direct Write</th>
</tr>
</thead>
<tbody>
<tr>
<td>XPARMID_RES_STATE</td>
<td>Current state (0: idle, 1: active)</td>
<td>R</td>
<td>N</td>
</tr>
<tr>
<td>XPARMID_PLY_VOL</td>
<td>Volume adjustment (+15 ~ –30 in 1dB unit), Default: 0</td>
<td>R/W</td>
<td>N</td>
</tr>
</tbody>
</table>

Events

None

---

### XPARMID_TD_RFC2833E_ENABLE

RFC2833 event enable flag, XPARM_ON or XPARM_OFF. Default: XPARM_OFF

- **Type:** R/W
- **Visibility:** Y

### XPARMID_TD_RFC2833E_UPDATERATE

RFC2833 packet rate in 10 ms unit, i.e., the period between the packets generated when a tone event is detected. Default: 5

- **Type:** R/W
- **Visibility:** N

### XPARMID_TD_RFC2833E_NUMEOE

Redundancy of end-of-event packet. Range 0-255. Default: 3

- **Type:** R/W
- **Visibility:** Y

### XPARMID_TD_RFC2833E_NUMBOE

Redundancy of begin-of-event packet. Range 0-255. Default: 0

- **Type:** R/W
- **Visibility:** Y

### XPARMID_TD_RFC2833E_AUDIOSUPRESS

Flag of audio encoding suppression when event detected. XPARM_ON or XPARM_OFF. Default: XPARM_ON

- **Type:** R/W
- **Visibility:** N

### XPARMID_TD_RFC2833E_PAYLOADTYPE

RFC2833 Payload type, Range is in the RTP dynamic payload type range of 96 to 127. Default: 0x65.

- **Type:** R/W
- **Visibility:** Y

### XPARMID_TD_FSK_CS

Minimum CS-bit length required by FSK receiver. Default: 200 if country code set to COUNTRY_CODE_US or COUNTRY_CODE_PRC, otherwise 0.

- **Type:** R/W
- **Visibility:** Y

### XPARMID_TD_FSK_MARK

Minimum mark-bit length required by FSK receiver. Default: 100 if country code set to COUNTRY_CODE_US or COUNTRY_CODE_PRC, otherwise 60.

- **Type:** R/W
- **Visibility:** Y

### XPARMID_TD_FSK_STOP

Extra stop bits allowed between data. Default: 20

- **Type:** R/W
- **Visibility:** Y

### XPARMID_TD_FSK_RATE

Baud rate of FSK receiver. (Reserved for future, currently only support 1200 bps rate).

- **Type:** R/W
- **Visibility:** Y

### Events

- XEVT_CODE_TD_TONEON – Tone-on event for an individual tone
- XEVT_CODE_TD_TONEOFF – Tone-off event for an individual tone

**NOTE:** Event data1 gives the tone ID and data2 gives the time stamp in 10-ms unit.
### 3.7 Audio Mixer Resource Component

**Resource Type**

<table>
<thead>
<tr>
<th>Media Processing Functions</th>
<th>Resource-Specific Control Messages</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Mixing multiple audio streams for 3-way call or small audio conference. The maximum number of parties to the mixer is currently 5.</td>
<td>• None.</td>
</tr>
</tbody>
</table>

**Parameters**

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description, Values</th>
<th>Attr.</th>
<th>Direct Write</th>
</tr>
</thead>
<tbody>
<tr>
<td>XPARMID_RES_STATE</td>
<td>Current state (0: idle, 1: active)</td>
<td>R</td>
<td>N</td>
</tr>
<tr>
<td>XPARMID_MIX_LP_STREAM</td>
<td>The first L-Port stream ID. Default: –1</td>
<td>R/W</td>
<td>N</td>
</tr>
<tr>
<td>XPARMID_MIX_LP_STREAM+1</td>
<td>The 2nd L-Port stream ID. Default: –1</td>
<td>R/W</td>
<td>N</td>
</tr>
<tr>
<td>XPARMID_MIX_LP_STREAM+n-1</td>
<td>The nth L-Port stream ID. Default: –1</td>
<td>R/W</td>
<td>N</td>
</tr>
</tbody>
</table>

**Events**

None.

### 3.8 T.38 Fax Resource Component

**Resource Type**

<table>
<thead>
<tr>
<th>Media Processing Functions</th>
<th>Resource-Specific Control Messages</th>
</tr>
</thead>
<tbody>
<tr>
<td>• time fax gateway between TDM interface and IP network.</td>
<td>• None.</td>
</tr>
</tbody>
</table>

**Parameters**

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description, Values</th>
<th>Attr.</th>
<th>Direct Write</th>
</tr>
</thead>
<tbody>
<tr>
<td>XPARMID_RES_STATE</td>
<td>Current state (0: idle, 1: active)</td>
<td>R</td>
<td>N</td>
</tr>
<tr>
<td>XPARMID_T38_ELLIPSIS</td>
<td>Flag of enabling ellipsis. XPARM_ON or XPARM_OFF. Default: XPARM_OFF</td>
<td>R/W</td>
<td>N</td>
</tr>
<tr>
<td>XPARMID_T38_FEC</td>
<td>Flag of enabling FEC. XPARM_ON or XPARM_OFF. Default: XPARM_OFF</td>
<td>R/W</td>
<td>N</td>
</tr>
<tr>
<td>XPARMID_T38_REDUNDANCY</td>
<td>Redundancy level, (0 ~ 7). Default: 0</td>
<td>R/W</td>
<td>N</td>
</tr>
<tr>
<td>XPARMID_T38_RATE_NEG</td>
<td>Method of modem rate negotiation. XPARM_T38_RATE_NEG_LOCAL or XPARM_T38_RATE_NEG_REMOTE. Default: XPARM_T38_RATE_NEG_REMOTE if packet transferred over UDP, otherwise XPARM_T38_RATE_NEG_LOCAL</td>
<td>R/W</td>
<td>N</td>
</tr>
<tr>
<td>XPARAID_T38_TCF_THRSHLD</td>
<td>TCF error threshold (in percentage). Default: 5</td>
<td>R/W</td>
<td>N</td>
</tr>
</tbody>
</table>
3.9 Message Agent Resource Component

<table>
<thead>
<tr>
<th>Resource Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>XPARMID_T38_TRANSPORT</td>
<td>Protocol used to transfer T.38 packets over IP network. XPARMID_T38_TRANS_UDP or XPARMID_T38_TRANS_TCP. Default: XPARMID_T38_TRANS_UDP</td>
</tr>
<tr>
<td>XPARMID_T38_MODE</td>
<td>Special mode, XPARM_T38_MODE_ITU or XPARM_T38_MODE_CHINA. Default: XPARM_T38_MODE_ITU</td>
</tr>
</tbody>
</table>

Events

XEVT_T38_END – End of the T.38 session. Event Data1 gives the reason of the termination
4.0 Message Format and Delivery Mechanism

There are two message queues (in-bound and out-bound) for the user application to send control messages and to receive response and event messages respectively. The message queues are created from pre-allocated memory buffers in consideration of maximum message size and total number of messages.

The entire message header and content are copied to/from the buffers in the message queue during message transmitting and receiving. The memory used for messaging is not shared between the message sender and the receiver.

4.1 Message Functions

Three functions are provided to send and receive messages.

<table>
<thead>
<tr>
<th>Function Name</th>
<th>Description</th>
<th>Input/Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>xMsgSend</td>
<td>Sends a control message to the in-bound message queue</td>
<td><strong>Input</strong>: pMsgBuf – Pointer to the message buffer. <strong>Output</strong>: None</td>
</tr>
<tr>
<td>xMsgReceive</td>
<td>Receives acknowledgement or event from the outbound message queue.</td>
<td><strong>Input</strong>: pMsgBuf – Pointer to the message buffer, channel – Channel number. (Reserved for future extension), timeout – waiting flag (Cannot be called from ISR)</td>
</tr>
<tr>
<td>xMsgWrite</td>
<td>post a message (e.g. an user defined external event message) to the out-bound queue so that it can be retrieved by XMsgReceive().</td>
<td><strong>Input</strong>: pMsgBuf — Pointer to the message buffer.</td>
</tr>
</tbody>
</table>
4.2 Message Header Format

```c
typedef struct {
    UINT32     transactionId; /* used by apps to track the message */
    UINT16     instance;    /* instance ID (1-0xffff), 0:reserved */
    UINT8      resource;    /* MPR resource type */
    UINT8      reserved;    /* reserved for future */
    UINT16     size;       /* total size in bytes */
    UINT8      type;       /* message type */
    UINT8      attribute;  /* attribute, reserved for future */
} XMsgHdr_t, *XMsgRef_t_t;

#define XMSG_MAKE_HEAD(pMsg, trans, res, inst, sz, typ, attr) \
    ((XMsgRef_t)(pMsg))->transactionId   = trans;\ 
    ((XMsgRef_t)(pMsg))->instance      = inst;\ 
    ((XMsgRef_t)(pMsg))->resource      = res;\ 
    ((XMsgRef_t)(pMsg))->reserved      = 0;\ 
    ((XMsgRef_t)(pMsg))->size          = sz;\ 
    ((XMsgRef_t)(pMsg))->type          = typ;\ 
    ((XMsgRef_t)(pMsg))->attribute     = attr;
```

Caution Message buffer requires 4-byte alignment.

Note The message buffer can be used for any other purpose, after posting.
4.3 Message Type List

All message types are pre-defined as:

```c
typedef enum {
    XMSG_BEGIN = 0, /* Begin list */
    XMSG_RESET, /* reset a SP resource */
    XMSG_START, /* start media processing a SP resource */
    XMSG_STOP, /* stop a current action on a SP resource */
    XMSG_PING, /* ping a SP resource */
    XMSG_SET_PARM, /* set a parameter on a SP resource */
    XMSG_SET_MPARMS, /* set multiple parameters on a SP resource */
    XMSG_GET_PARM, /* get a parameter from a SP resource */
    XMSG_GET_PARM_ACK, /* acknowledgement to get parameter message */
    XMSG_GET_ALLPARMS, /* get all parameters from a SP resource */
    XMSG_GET_ALLPARMS_ACK, /* acknowledgement to get all parameter message */
    XMSG_ACK, /* general acknowledgement message */
    XMSG_ERROR, /* error message from SP resource */
    XMSG_EVENT, /* event message from SP resource */
    XMSG_CODER_START, /* start a codec resource */
    XMSG_CODER_STOP_ACK, /* acknowledgement to stop message */
    XMSG_TD_PLAY, /* play a digit string on a TG instance */
    XMSG_TD_PLAY_FSK, /* play FSK modulated data */
    XMSG_TD_PLAY_CMPLT, /* play-completed message from a TG instance */
    XMSG_TD_RCV, /* receive a digit string on a TD instance */
    XMSG_TD_RCV_CMPLT, /* receive-completed message from a channel */
    XMSG_TD_RCV_FSK, /* receive a FSK signal on a TD instance */
    XMSG_TD_RCV_FSK_CMPLT, /* receive-completed message from TD instance */
    XMSG_PLY_START, /* start playing audio on a Player instance */
    XMSG_GET_JBSTAT, /* get jitter buffer statistics from Dec */
    XMSG_GET_JBSTAT_CMPLT, /* response to the get-JB-statistics msg */
    XMSG_PLY_CMPLT, /* play-completed message from Player */
    XMSG_END /* end of list */
} XMsgType_t;
```
5.0 Common Control Message

This section defines the control messages that can be applied to all the resources.

5.1 Reset Message

<table>
<thead>
<tr>
<th>Type</th>
<th>XMSG_RESET</th>
</tr>
</thead>
<tbody>
<tr>
<td>Direction</td>
<td>Inbound</td>
</tr>
<tr>
<td>Description</td>
<td>Stops the current action and resets the resource to idle state.</td>
</tr>
<tr>
<td>Format</td>
<td>typedef struct{</td>
</tr>
<tr>
<td></td>
<td>XMsgHdr_t    head; /* message header */</td>
</tr>
<tr>
<td></td>
<td>) XMsgReset_t;</td>
</tr>
<tr>
<td>Macro</td>
<td>#define XMSG_MAKE_RESET(pMsg, trans, res, inst)</td>
</tr>
<tr>
<td></td>
<td>{</td>
</tr>
<tr>
<td></td>
<td>XMSG_MAKE_HEAD(pMsg, trans, res, inst, sizeof(XMsgReset_t),</td>
</tr>
<tr>
<td></td>
<td>XMSG_RESET, 0)\</td>
</tr>
<tr>
<td></td>
<td>}</td>
</tr>
<tr>
<td>Response</td>
<td>• General acknowledgement message (XMSG_ACK)</td>
</tr>
<tr>
<td></td>
<td>• Error message (XMSG_ERROR) if error.</td>
</tr>
<tr>
<td>Caution</td>
<td>Any intermediate results are discarded.</td>
</tr>
</tbody>
</table>

5.2 Start Message

<table>
<thead>
<tr>
<th>Type</th>
<th>XMSG_Start</th>
</tr>
</thead>
<tbody>
<tr>
<td>Direction</td>
<td>Inbound</td>
</tr>
<tr>
<td>Description</td>
<td>Generic start message. Starts the media-processing functions on a resource.</td>
</tr>
<tr>
<td>Format</td>
<td>typedef struct{</td>
</tr>
<tr>
<td></td>
<td>XMsgHdr_t    head; /* message header */</td>
</tr>
<tr>
<td></td>
<td>) XMsgStart_t;</td>
</tr>
<tr>
<td>Macro</td>
<td>#define XMSG_MAKE_START(pMsg, trans, res, inst)</td>
</tr>
<tr>
<td></td>
<td>{</td>
</tr>
<tr>
<td></td>
<td>XMSG_MAKE_HEAD(pMsg, trans, res, inst, sizeof(XMsgStart_t),</td>
</tr>
<tr>
<td></td>
<td>XMSG_START, 0)\</td>
</tr>
<tr>
<td></td>
<td>}</td>
</tr>
<tr>
<td>Response</td>
<td>• General acknowledgement message (XMSG_ACK)</td>
</tr>
<tr>
<td></td>
<td>• Error message (XMSG_ERROR) if error.</td>
</tr>
<tr>
<td>Caution</td>
<td>Currently only the Network Endpoint and Tone Detector resources support the start message.</td>
</tr>
</tbody>
</table>
## 5.3 Stop Message

<table>
<thead>
<tr>
<th>Type</th>
<th>XMSG_STOP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Direction</td>
<td>Inbound</td>
</tr>
<tr>
<td>Description</td>
<td>Stops the current action.</td>
</tr>
</tbody>
</table>

### Format
```c
typedef struct{
    XMsgHdr_t head; /* message header */
} XMsgStop_t;
```

### Macro
```c
#define XMSG_MAKE_STOP(pMsg, trans, res, inst) {
    XMSG_MAKE_HEAD(pMsg, trans, res, inst, sizeof(XMsgStop_t),
                    XMSG_STOP, 0)
}
```

### Response
Resource returns the processing results or states, if any, depending on the resources and current actions.

## 5.4 Ping Message

<table>
<thead>
<tr>
<th>Type</th>
<th>XMSG_PING</th>
</tr>
</thead>
<tbody>
<tr>
<td>Direction</td>
<td>Inbound</td>
</tr>
<tr>
<td>Description</td>
<td>Verifies if the resource is alive.</td>
</tr>
</tbody>
</table>

### Format
```c
typedef struct{
    XMsgHdr_t head; /* message header */
} XMsgPing_t;
```

### Macro
```c
#define XMSG_MAKE_PING(pMsg, trans, res, inst) {
    XMSG_MAKE_HEAD(pMsg, trans, res, inst, sizeof(XMsgPing_t),
                    XMSG_PING, 0)
}
```

### Response
- General acknowledgement message (XMSG_ACK)
- Error message (XMSG_ERROR) if error.

## 5.5 Set-Parameter Message

<table>
<thead>
<tr>
<th>Type</th>
<th>XMSG_SET_PARM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Direction</td>
<td>Inbound</td>
</tr>
<tr>
<td>Description</td>
<td>Sets a parameter to a resource.</td>
</tr>
</tbody>
</table>

(Sheet 1 of 2)
### 5.6 Set-Multiple-Parameter Message

<table>
<thead>
<tr>
<th>Type</th>
<th>XMSG_SET_MPARMS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Direction</td>
<td>Inbound</td>
</tr>
<tr>
<td>Description</td>
<td>Set multiple parameters to a resource</td>
</tr>
</tbody>
</table>

#### Format

```c
typedef struct{
    XMsgHdr_t head; /* message header */
    UINT16 numParms; /* number of parameters */
    UINT16 parmIDs[XMAX_PARMS]; /* parameter id */
    UINT16 values[XMAX_PARMS]; /* parameter value */
} XMsgSetxParms_t;
```

#### Macro

```c
#define XMSG_MAKE_SET_MPARMS(pMsg, trans, res, inst, num) 
{
    XMSG_MAKE_HEAD(pMsg, trans, res, inst, sizeof(XMsgSetxParms_t),
    XMSG_SET_MPARMS, 0)
    ((XMsgSetxParms_t *)(pMsg))->numParms = num;
}
#define XMSG_FIELD_SET_MPARMS(pMsg, pIDs, pVals) 
{
    pIDs = ((XMsgSetxParms_t *)(pMsg))->parmIDs;
    pVals = ((XMsgSetxParms_t *)(pMsg))->values;
}
```

#### Response

- General acknowledgement message (XMSG_ACK)
- Error message (XMSG_ERROR) if error.
5.7  Get-Parameter Message

<table>
<thead>
<tr>
<th>Type</th>
<th>XMSG_GET_PARM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Direction</td>
<td>Inbound</td>
</tr>
<tr>
<td>Description</td>
<td>Gets a parameter from a resource.</td>
</tr>
</tbody>
</table>

```
typedef struct{
    XMsgHdr_t head; /* message header */
    UINT16 parmId; /* parameter id */
} XMsgGetParm_t;
```

```
#define XMSG_MAKE_GET_PARM(pMsg, trans, res, inst, id) \
{\n    XMSG_MAKE_HEAD(pMsg, trans, res, inst, sizeof(XMsgGetParm_t),\n                     XMSG_GET_PARM, 0)\n    ((XMsgGetParm_t *)(pMsg))->parmId= id;\n}
```

Response
- Specific acknowledgement message (XMSG_GET_PARM_ACK)
- Error message (XMSG_ERROR) if error.

5.8  Get-Parameter-Acknowledge Message

<table>
<thead>
<tr>
<th>Type</th>
<th>XMSG_GET_PARM_ACK</th>
</tr>
</thead>
<tbody>
<tr>
<td>Direction</td>
<td>Outbound</td>
</tr>
<tr>
<td>Description</td>
<td>Resource returns the parameter enquired.</td>
</tr>
</tbody>
</table>

```
typedef struct{
    XMsgHdr_t head; /* message header */
    UINT16 parmId; /* parameter id */
    UINT16 value; /* parameter value */
} XMsgGetParmAck_t;
```

```
#define XMSG_FIELD_GET_PARM_ACK(pMsg, id, val)\n{\n    id = ((XMsgGetParmAck_t *)(pMsg))->parmId;\n    val = ((XMsgGetParmAck_t *)(pMsg))->value;\n}
```

5.9  Get-All-Parameters Message

<table>
<thead>
<tr>
<th>Type</th>
<th>XMSG_GET_ALLPARMS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Direction</td>
<td>Inbound</td>
</tr>
<tr>
<td>Description</td>
<td>Gets all parameters from a resource.</td>
</tr>
</tbody>
</table>
5.10 Get-All-Parameters-Acknowledge Message

<table>
<thead>
<tr>
<th>Type</th>
<th>XMSG_GET_ALLPARMS_ACK</th>
</tr>
</thead>
<tbody>
<tr>
<td>Direction</td>
<td>Outbound</td>
</tr>
<tr>
<td>Description</td>
<td>Resource returns the parameter inquired.</td>
</tr>
</tbody>
</table>

```c
typedef struct{    
    XMsgHdr_t head;    /* message header */
    UINT16 numParms;  /* number of parameters */
    UINT16 parmIDs[XMAX_PARMS_GET];/* array of parameter IDs */
    UINT16 values[XMAX_PARMS_GET];  /* array of parameter values */
} XMsgGetAllParmsAck_t;
```

```
#define XMSG_FIELD_GET_ALLPARMS_ACK(pMsg, num, pIDs, pVals)\
{\
    num = ((XMsgGetAllParmsAck_t *)(pMsg))->numParms;\
    pIDs = ((XMsgGetAllParmsAck_t *)(pMsg))->parmIDs;\
    pVals = ((XMsgGetAllParmsAck_t *)(pMsg))->values;\
}
```

5.11 General-Acknowledge Message

<table>
<thead>
<tr>
<th>Type</th>
<th>XMSG_ACK</th>
</tr>
</thead>
<tbody>
<tr>
<td>Direction</td>
<td>Outbound</td>
</tr>
<tr>
<td>Description</td>
<td>Resource indicates the control message has been processed successfully.</td>
</tr>
</tbody>
</table>

```c
typedef struct{    
    XMsgHdr_t head;    /* message header */
} XMsgAck_t;
```
5.12 Error Message

**Type**: XMSG_ERROR

**Direction**: Outbound

**Description**: Resource reports an error condition. (See constant data section for error codes.)

```c
typedef struct{
    XMsgHdr_t head; /* message header */
    UINT32 code; /* error code */
    UINT32 data1; /* error data1 */
    UINT32 data2; /* error data2 */
} XMsgError_t;
```

**Macro**

```c
#define XMSG_FIELD_ERROR(pMsg, c, d1, d2) 
{ 
    c = ((XMsgError_t *)(pMsg))->code;
    d1 = ((XMsgError_t *)(pMsg))->data1;
    d2 = ((XMsgError_t *)(pMsg))->data2;
} 
```

5.13 Event Message

**Type**: XMSG_EVENT

**Direction**: Outbound

**Description**: Resource reports an event condition. (See constant data section for error codes.)

```c
typedef struct{
    XMsgHdr_t head; /* message header */
    UINT32 code; /* event code */
    UINT32 data1; /* event data1 */
    UINT32 data2; /* event data2 */
} XMsgEvent_t;
```

**Macro**

```c
#define XMSG_FIELD_EVENT(pMsg, c, d1, d2) 
{ 
    c = ((XMsgEvent_t *)(pMsg))->code;
    d1 = ((XMsgEvent_t *)(pMsg))->data1;
    d2 = ((XMsgEvent_t *)(pMsg))->data2;
} 
```
6.0 Resource-Specific Control Message

This section defines the resource-specific messages.

6.1 CODEC Start Message

<table>
<thead>
<tr>
<th>Type</th>
<th>XMSG_CODER_START</th>
</tr>
</thead>
<tbody>
<tr>
<td>Direction</td>
<td>Inbound</td>
</tr>
<tr>
<td>Description</td>
<td>Starts a decoder or encoder.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Format</th>
</tr>
</thead>
</table>
| typedef struct{
  XMsgHdr_t head; /* message header */
  UINT16 codecType; /* codec type */
  UINT16 frmsPerPkt; /* number of frames per packet */
} XMsgCoderStart_t; |

<table>
<thead>
<tr>
<th>Macro</th>
</tr>
</thead>
</table>
| #define XMSG_MAKE_CODER_START(pMsg, trans, res, inst, cType, fpp){
  XMSG_MAKE_HEAD(pMsg, trans, res, inst, sizeof(XMsgCoderStart_t),
  XMSG_CODER_START, 0)
  ((XMsgCoderStart_t *)(pMsg))->codecType = cType;
  ((XMsgCoderStart_t *)(pMsg))->frmsPerPkt = fpp;
} |

<table>
<thead>
<tr>
<th>Response</th>
</tr>
</thead>
<tbody>
<tr>
<td>• General acknowledgement message (XMSG_ACK)</td>
</tr>
<tr>
<td>• Error message (XMSG_ERROR) if error.</td>
</tr>
</tbody>
</table>

6.2 CODEC Stop-Acknowledgement Message

<table>
<thead>
<tr>
<th>Type</th>
<th>XMSG_CODER_STOP_ACK</th>
</tr>
</thead>
<tbody>
<tr>
<td>Direction</td>
<td>Outbound</td>
</tr>
<tr>
<td>Description</td>
<td>Decoder or encoder resource acknowledges the XMSG_STOP message</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Format</th>
</tr>
</thead>
</table>
| typedef struct{
  XMsgHdr_t head; /* message header */
  UINT32 numFrames; /* total number of frames processed */
  UINT32 numBadFrames; /* number of bad frames */
} XMsgCoderStopAck_t; |

<table>
<thead>
<tr>
<th>Macro</th>
</tr>
</thead>
</table>
| #define XMSG_FIELD_EVENT(pMsg, num, numBad){
  num = ((XMsgCoderStopAck_t *)(pMsg))->numFrames;
  numBad = ((XMsgCoderStopAck_t *)(pMsg))->numBadFrames;
} |
6.3 Tone-Generator-Play Message

<table>
<thead>
<tr>
<th>Type</th>
<th>XMSG_TG_PLAY</th>
</tr>
</thead>
<tbody>
<tr>
<td>Direction</td>
<td>Inbound</td>
</tr>
<tr>
<td>Description</td>
<td>Requires Tone Generator to play a tone string. (Tone ID's are listed in the constant data section.)</td>
</tr>
</tbody>
</table>

Format:
```c
typedef struct{
    XMsgHdr_t head;              /* message header */
    UINT8 numTones;            /* number of tones to play */
    UINT8 toneId[XMAX_TONEBUFSIZE];  /* tone ID string */
} XMsgTGPlay_t;
```

Macro:
```c
#define XMSG_MAKE_TG_PLAY(pMsg, trans, inst, num)
{                  
    XMSG_MAKE_HEAD(pMsg, trans, XMPR_TNGEN, inst, sizeof(XMsgTGPlay_t),
                    XMSG_TG_PLAY, 0)
    ((XMsgTGPlay_t *)(pMsg))->numTones = num;
}
#define XMSG_FIELD_TG_PLAY(pMsg, pToneID) 
{                  
    pToneID = ((XMsgTGPlay_t *)(pMsg))->toneId;
}
```

6.4 Tone-Generator-Play-FSK Message

<table>
<thead>
<tr>
<th>Type</th>
<th>MSG_TG_PLAY_FSK</th>
</tr>
</thead>
<tbody>
<tr>
<td>Direction</td>
<td>Inbound</td>
</tr>
<tr>
<td>Description</td>
<td>Require Tone Generator to play a FSK modulated data</td>
</tr>
</tbody>
</table>

Format:
```c
typedef struct{
    XMsgHdr_t head;    /* message header */
    UINT8 numBytes; /* number of bytes to play */
    INT8 data[XMAX_FSKDATASIZE]; /* data string */
} XMsgTGPlayFSK_t;
```

Macro:
```c
#define XMSG_MAKE_TG_PLAY_FSK(pMsg, trans, inst, num)
{                  
    XMSG_MAKE_HEAD(pMsg, trans, XMPR_TNGEN, inst, sizeof(XMsgTGPlayFSK_t),
                    XMSG_TG_PLAY_FSK, 0)
    ((XMsgTGPlayFSK_t *)(pMsg))->numBytes = num;
}
#define XMSG_FIELD_TG_PLAY_FSK(pMsg, pData) 
{                  
    pData = ((XMsgTGPlayFSK_t *)(pMsg))->data;
}
```

Response: • Tone Generator Play-Completed message (XMSG_TG_PLAY_CMPLT)
6.5 Tone-Generator-Play-Completed Message

Type: XMSG_TG_PLAY_CMPLT
Direction: Outbound
Description: Tone Generator indicates the completion of playing tones.

Format:
```c
typedef struct{
    XMsgHdr_t  head; /* message header */
    UINT16   reason; /* the reason of completion: */
                      /* XMSG_STOP_REASON_USER (1) */
                      /* XMSG_STOP_REASON_EOD (2) */
    UINT8   numTones; /* number of tones played. 0 if FSK data */
} XMsgTGPlayCmplt_t;
```

Macro:
```c
#define XMSG_FIELD_TG_PLAY_CMPLT(pMsg, rsn, num)\  {\  reason = ((XMsgTGPlayCmplt_t *)(pMsg))->reason;\  num = ((XMsgTGPlayCmplt_t *)(pMsg))->numTones;\ }\```

6.6 Tone-Detector-Receive-Digit Message

Type: XMSG_TD_RCV
Direction: Inbound
Description: Require Tone Detector to receive a tone string.

Format:
```c
typedef struct{
    XMsgHdr_t  head;               /* message header */
    UINT16     totalTimeout;       /* total time out (in 10 ms unit) */
    UINT16     firstDigitTimeout;  /* first digit time out (10 ms unit)*/
    UINT16     interDigitTimeout;  /* inter digit time out (10 ms unit)*/
    UINT16     termDigit;          /* OR'd terminate digit bits */
    UINT8      numDigits;          /* number of digits to receive */
} XMsgTDRcv_t;
```

Macro:
```c
#define XMSG_MAKE_TD_RCV(pMsg, trans, inst, num, term, tm, fstTm, intTm)\  {\  XMSG_MAKE_HEAD(pMsg, trans, XMPR_TNDET, inst,\      sizeof(XMsgTDRcv_t), XMSG_TD_RCV, 0)\  ((XMsgTDRcv_t *)(pMsg))->numDigits = num;\  ((XMsgTDRcv_t *)(pMsg))->termDigit = term;\  ((XMsgTDRcv_t *)(pMsg))->totalTimeout = tm;\  ((XMsgTDRcv_t *)(pMsg))->firstDigitTimeout = fstTm;\  ((XMsgTDRcv_t *)(pMsg))->interDigitTimeout = intTm;\ }\```

Response: Tone detector receives completed message (XMSG_TD_RCV_CMPLT)
### 6.7 Tone-Detector-Receive-Completed Message

<table>
<thead>
<tr>
<th>Type</th>
<th>XMSG_TD_RCV_CMPLT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Direction</td>
<td>Outbound</td>
</tr>
<tr>
<td>Description</td>
<td>Tone detector indicates the completion of receiving DTMF tones.</td>
</tr>
</tbody>
</table>

**Format**

```c
typedef struct{
    XMsgHdr_t   head;           /* message header */
    UINT16      reason;         /* the reason of completion */
    UINT8       numDigits;      /* number of tones received */
    UINT8       digits[XMAX_DIGITBUFSIZE];  /* received tone IDs */
} XMsgTDRcvCmplt_t;
```

where the reason may be:

```c
#define XMSG_STOP_REASON_EOD        2
#define XMSG_STOP_REASON_TERM       3
#define XMSG_STOP_REASON_TIMEOUT    4
```

**Macro**

```c
#define XMSG_FIELD_TD_RCV_CMPLT(pMsg, rsn, num, pBuf)\  {\  rsn = ((XMsgTDRcvCmplt_t *)(pMsg))->reason;\  num = ((XMsgTDRcvCmplt_t *)(pMsg))->numDigits;\  pBuf= ((XMsgTDRcvCmplt_t *)(pMsg))->digits;\ }
```

### 6.8 Tone-Detector-Receive-FSK Message

<table>
<thead>
<tr>
<th>Type</th>
<th>MSG_TD_RCV_FSK</th>
</tr>
</thead>
<tbody>
<tr>
<td>Direction</td>
<td>Inbound</td>
</tr>
<tr>
<td>Description</td>
<td>Require Tone Detector to receive FSK data</td>
</tr>
</tbody>
</table>

**Format**

```c
typedef struct{
    XMsgHdr_t   head; /* message header */
    UINT16      timeout; /* total time out (in 10 ms unit) */
} XMsgTDRcvFSK_t;
```

**Macro**

```c
#define XMSG_MAKE_TD_RCV_FSK(pMsg, trans, inst, tmout)\  {\  XMSG_MAKE_HEAD(pMsg, trans, XMPR_TNDET, inst,\  Csizeof(XMsgTDRcvFSK_t), XMSG_TD_RCV_FSK, 0)\  ((XMsgTDRcvFSK_t *)(pMsg))->timeout = tmout;\ }
```

**Response**

Tone Detector FSK receive-completed message (XMSG_TD_RCV_FSK_CMPLT)
### 6.9 Tone-Detector-FSK-Receive-Completed Message

<table>
<thead>
<tr>
<th>Type</th>
<th>XMSG_TD_RCV_FSK_CMPLT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Direction</td>
<td>Outbound</td>
</tr>
<tr>
<td>Description</td>
<td>Tone Detector indicates the completion of receiving FSK data</td>
</tr>
</tbody>
</table>

```c
typedef struct{
    XMsgHdr_t   head;           /* message header */
    UINT16      reason;         /* the reason of completion */
    UINT8       numBytes;       /* number of bytes received */
    UINT8       data[XMAX_FSKDATASIZE];  /* received data */
} XMsgTDRcvFskCmplt_t;
```

where the reason may be:

- `#define XMSG_STOP_REASON_EOD 2`
- `#define XMSG_STOP_REASON_TIMEOUT 4`

```c
#define XMSG_FIELD_TD_RCV_FSK_CMPLT(pMsg, rsn, num, pBuf)\
{\
    rsn = ((XMsgTDRcvFskCmplt_t *)(pMsg))->reason;\
    num = ((XMsgTDRcvFskCmplt_t *)(pMsg))->numBytes;\
    pBuf = ((XMsgTDRcvFskCmplt_t *)(pMsg))->data;\
}
```

### 6.10 Player-Start Message

<table>
<thead>
<tr>
<th>Type</th>
<th>XMSG_PLY_START</th>
</tr>
</thead>
<tbody>
<tr>
<td>Direction</td>
<td>Inbound</td>
</tr>
<tr>
<td>Description</td>
<td>Start Player to play back pre-recorded audio data</td>
</tr>
</tbody>
</table>

(Sheet 1 of 2)
### 6.11 Player-Play-Completed Message

<table>
<thead>
<tr>
<th>Type</th>
<th>XMSG_PLY_CMPLT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Direction</td>
<td>Outbound</td>
</tr>
<tr>
<td>Description</td>
<td>Player indicates the completion of playing audio data.</td>
</tr>
</tbody>
</table>

```c
typedef struct{
    XMshdr_t head; /* message header */
    UINT16 reason; /* the reason of completion */
} XmsgPlyCmplt_t;
```

Where the reason may be:

- `#define XMSG_STOP_REASON_USER 1`
- `#define XMSG_STOP_REASON_EOD 2`

```c
#define XMSG_FIELD_PLY_CMPLT(pMsg, rsn) \{
    rsn = ((XmsgPlyCmplt_t *)(pMsg))->reason;
}
```

---

**Player-Play-Completed Message**

<table>
<thead>
<tr>
<th>Type</th>
<th>XMSG_PLY_CMPLT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Format</td>
<td></td>
</tr>
<tr>
<td>Macro</td>
<td></td>
</tr>
</tbody>
</table>

**Response**

Player play-completed message (XMSG_PLY_CMPLT)
### 6.12 Get-Jitter-Buffer-Statistics Message

<table>
<thead>
<tr>
<th>Type</th>
<th>XMSG_GET_JBSTAT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Direction</td>
<td>Inbound</td>
</tr>
<tr>
<td>Description</td>
<td>Get the jitter buffer statistics from a Decoder instance.</td>
</tr>
</tbody>
</table>

**Format**

```c
typedef struct {
    XMsgHdr_t       head;       /* message header */
    UINT16          reset;      /* reset flag, 1: reset statistics after retrieve the information */
} XMsgGetJBStat_t;
```

**Macro**

```c
#define XMSG_MAKE_GET_JBSTAT(pMsg, trans, inst, clr)
{
    XMSG_MAKE_HEAD(pMsg, trans, XMPR_DEC, inst, sizeof(XMsgGetJBStat_t), XMSG_GET_JBSTAT, 0)
    ((XMsgGetJBStat_t *)(pMsg))->reset = clr;
}
```

**Response**

complete message of getting jitter buffer statistics (XMSG_GET_JBSTAT_CMPLT)

### 6.13 Complete Message of Getting Jitter Buffer Statistics

<table>
<thead>
<tr>
<th>Type</th>
<th>XMSG_GET_JBSTAT_CMPLT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Direction</td>
<td>Outbound</td>
</tr>
<tr>
<td>Description</td>
<td>Response to the message of getting the jitter buffer statistics.</td>
</tr>
</tbody>
</table>

**Format**

```c
typedef struct {
    XMsgHdr_t           head;   /* message header */
    XJBStatistics_t     stat;   /* jitter buffer statistics */
} XMsgGetJBStatCmplt_t;
```

where the XMsgGetJBStatCmplt_t data structure of jitter buffer statistics is defined as

```c
typedef struct {
    UINT32      rcvdPackets;        /* total packets received */
    UINT32      lostPackets;        /* lost packets */
    UINT32      badFrames;          /* decoder bad frames */
    UINT32      rcvdTonePackets;    /* RFC2833 packets received */
} XJBStatistics_t;
```

**Macro**

```c
#define XMSG_FIELD_GET_JBSTAT_CMPLT(pMsg, pStat)
{
    pStat = &(((XMsgGetJBStatCmplt_t *)(pMsg))->stat);
}
```
7.0 Packet Data Interface

The packet data interface is a protocol for the DSP software to exchange the encoded data packets with IP stack. This interface is defined as a packet format and two callback functions — one is provided by DSP software release and another is provided by the user (IP stack).

7.1 Packet Formats

The ingress packet from the IP stack to the DSP software has an 8-byte header as shown below:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Channel ID</td>
<td>Packet arrival time as measured by a local clock.</td>
</tr>
<tr>
<td>M</td>
<td>Packet data sampling time measured by a remote clock.</td>
</tr>
<tr>
<td>Payload Type</td>
<td>Payload length in bytes.</td>
</tr>
<tr>
<td>Media</td>
<td>4-bit media type field is defined as:</td>
</tr>
<tr>
<td></td>
<td>• 0x01 – Audio</td>
</tr>
<tr>
<td></td>
<td>• 0x02 – Tone (RFC2833 event type)</td>
</tr>
<tr>
<td></td>
<td>• 0x04 – Tone (RFC2833 tone type)</td>
</tr>
<tr>
<td></td>
<td>• 0x08 – T.38 UDP</td>
</tr>
<tr>
<td></td>
<td>• 0x09 – T.38 TCP</td>
</tr>
<tr>
<td>Remote Time Stamp</td>
<td>Payload length in bytes.</td>
</tr>
<tr>
<td>Payload</td>
<td></td>
</tr>
</tbody>
</table>

Similarly, the egress packet from the DSP software to the IP stack has an 8-byte header as shown below:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Channel ID</td>
<td>Packet arrival time as measured by a local clock.</td>
</tr>
<tr>
<td>M</td>
<td>Packet data sampling time measured by a remote clock.</td>
</tr>
<tr>
<td>Payload Type</td>
<td>Payload length in bytes.</td>
</tr>
<tr>
<td>Media</td>
<td>4-bit media type field is defined as:</td>
</tr>
<tr>
<td></td>
<td>• 0x01 – Audio</td>
</tr>
<tr>
<td></td>
<td>• 0x02 – Tone (RFC2833 event type)</td>
</tr>
<tr>
<td></td>
<td>• 0x04 – Tone (RFC2833 tone type)</td>
</tr>
<tr>
<td></td>
<td>• 0x08 – T.38 UDP</td>
</tr>
<tr>
<td></td>
<td>• 0x09 – T.38 TCP</td>
</tr>
<tr>
<td>Local Time Stamp</td>
<td>Payload length in bytes.</td>
</tr>
<tr>
<td>Payload</td>
<td></td>
</tr>
</tbody>
</table>

The fields of the packet header and the payload are described as:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Local Time Stamp</td>
<td>Packet arrival time as measured by a local clock.</td>
</tr>
<tr>
<td>Remote Time Stamp</td>
<td>Packet data sampling time measured by a remote clock.</td>
</tr>
<tr>
<td>Payload Length</td>
<td>Payload length in bytes.</td>
</tr>
<tr>
<td>Media</td>
<td>4-bit media type field is defined as:</td>
</tr>
<tr>
<td></td>
<td>• 0x01 – Audio</td>
</tr>
<tr>
<td></td>
<td>• 0x02 – Tone (RFC2833 event type)</td>
</tr>
<tr>
<td></td>
<td>• 0x04 – Tone (RFC2833 tone type)</td>
</tr>
<tr>
<td></td>
<td>• 0x08 – T.38 UDP</td>
</tr>
<tr>
<td></td>
<td>• 0x09 – T.38 TCP</td>
</tr>
<tr>
<td>M</td>
<td>Marker bit for the RTP packet. This bit set indicates the first speech packet after a silence period or the first packet of a RFC-2833 tone event, otherwise 0.</td>
</tr>
<tr>
<td>Payload type</td>
<td>RTP payload type as defined in RFC 1990.</td>
</tr>
<tr>
<td>Payload</td>
<td>Encoded audio data or RFC-2838 tone event information.</td>
</tr>
</tbody>
</table>
The corresponding data structure is defined as:

```c
typedef struct{
    UINT8     channelID;    /* channel ID */
    UINT8     payloadType;  /* bit[0-6] payload type, 
                              bit[7] SID mark bit */
    unsigned int mediaType:4; /* media type */
    unsigned int payloadLen:12; /* payload length */
    UINT32     timeStamp;   /* local or remote time stamp */
} __attribute__ ((packed)) XPacketHeader_t;
```

In ingress, the header information of Remote Time Stamp, Payload Type and Marker bit is directly copied from a RTP packet. In egress, the header information is filled by DSP software except for the Payload Type of RFC-2833 event packets. The RTP processing module is responsible to determine the payload type if media type indicates a RFC-2833 tone-event packet.

### 7.2 Packet Delivery Mechanism

Packets are transferred between the DSP software and IP stack via callback functions. The packet delivery module calls the function and passes the packet each time when a packet is produced. The rules of using the callback function to deliver the packets include:

- The packet receiver registers a callback function with the packet deliverer.
- The packet deliverer is responsible to prepare the memory for the packet.
- The packet receiver has to copy the data to its internal buffer immediately in the callback function because the deliverer may reuse the same memory for the next packet (i.e., the packet data may not be valid any more after the callback function returns).
- The packet receiver may perform some data processing in the callback function provided the execution of such processing is predictable (i.e., the processing must be guaranteed to complete within a certain short period of time).

The function that DSP software provides to receive the packets from IP stack is defined as follows:

```c
XStatus_t xPacketReceive (UNIT16 channel, XPacket_t *buffer);
```

<table>
<thead>
<tr>
<th>Description</th>
<th>Call-back function to receive packets.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
<td>Buffer – memory address of the packet</td>
</tr>
<tr>
<td></td>
<td>Channel – Channel numbers</td>
</tr>
<tr>
<td>Output</td>
<td>None</td>
</tr>
<tr>
<td>Return</td>
<td>XSUCC – If successful</td>
</tr>
<tr>
<td></td>
<td>XERROR – If the packet receptor is unable to process the packet.</td>
</tr>
</tbody>
</table>

IP stack has to build DSP software data packets from the IP packets it receives and deliver them to the DSP software by calling this function.

In egress direction, IP stack must provide a function to receive egress data packets from the DSP software. The DSP software will call the function each time when a packet is generated. That function must be registered during initialization.
8.0 Configuration and Initialization

The Intel® IXP400 DSP Software is configurable at initialization time, allowing the user to specify the HSS parameters, the number of resource instances to be created and the country-specific features. The user-supplied call back functions are also registered at that time.

8.1 System Configuration

### Description

This function performs the following procedures:

- Initialize and start HSS port.
- Create TDM termination channels (i.e., Network Endpoint resource instance) and link them to the HSS time slots sequentially. Error will occur if not enough time slots are enabled for all the TDM channels.
- Create the IP terminations (i.e., Decoder, Encoder, Tone Generator and Tone Detector resources).
- Create media service resources (i.e., Player and Mixer).
- Enable country-specific call progress tones and set country-specific default parameters to the resources.
- Register user-supplied call back functions.

<table>
<thead>
<tr>
<th>Prototype</th>
<th>void xDspSysInit(XDSPSysConfig_t *pSysConfig);</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
<td>pSysConfig – system configuration information</td>
</tr>
<tr>
<td>Output</td>
<td>None</td>
</tr>
<tr>
<td>Return</td>
<td>None</td>
</tr>
</tbody>
</table>
The configuration information in this function is defined as:

```c
typedef struct{
    int numChTDM;    /* number of channels of TDM termination (1-4) */
    int numChIP;     /* number of channels of IP termination (1-4) */
    int numPlayers;  /* number of Player instances (1-4) */
    int numMixers;   /* number of Audio Mixers (must be 1) */
    int numPortsPerMixer; /* number of ports per mixer (3-5) */
    int countryCode; /* country code */
    int taskPriBase; /* the base priority of DSP module */
    int taskPriOrder; /* the priority ordering of the OS */
    IxHssAccHssPort port;   /* HSS port (must be Port 0) */
    IxHssAccConfigParams *pHssCfgParms; /* HSS configuration parameters */
    IxHssAccTdmSlotUsage *pHssTDMSlots; /* HSS TDM time slot mapping */
    XPktRcvFxn_t pktRcvFxn; /* packet receiver function in egress */
    XMsgAgentDec_t msgDecoder; /* optional message decoder function of MA */
    XMsgAgentEnc_t msgEncoder; /* optional message encoder function of MA */
} XDSPSysConfig_t;
```

This function must be called after downloading HSS NPE. An assertion occurs if any fatal errors happen (e.g., memory exhausted) during the initialization. If the numbers of resources to be created are not specified correctly, the default ones are applied, which can be retrieved by the xDspGetResConfig() function.

### 8.2 Adding Tones to Tone Generator

**Prototype**

```c
XStatus_t xBuildToneTG(UINT16 toneId, UINT16 numSegs, XTGToneSeg_t *pToneSegs, UINT32 *pErrCode);
```

**Input**

- **toneId** — Tone ID, must be in the range of 16 ~ 255
- **numSegs** — Number of segments of the tone
- **pToneSegs** — Array of tone segment definition

**Output**

- **pErrCode** — Error code if errors

**Return**

- **XSUCC** if successful
- Otherwise **XERROR**
Description

This function adds a new tone which can be played by the Tone Generator resources. Each new tone can contains one or more segments which is defined as

```c
typedef struct {
    UINT16  repCount;          /* repetition number of the segment.
                                0 means to repeat forever */
    UINT16  segType;                  /* signal type (single or dual frequency
                                wave or AM wave ) */
    UINT32  durationOn; /* active duration in 1-ms unit. */
    UINT32  durationOff; /* silence duration in 1-ms unit. */
    INT16   freqA; /* 1st frequency if single or dual
                                frequency wave, or the modulated carry frequency if
                                AM wave, in Hz unit*/
    INT16   freqB; /* 2nd frequency if dual frequency
                                wave or the modulating frequency if AM
                                wave, ignored if single frequency wave */
    INT16   ampA; /* amplitude of frequency A above,
                                (0~ - 45 in 1dBm unit) */
    INT16   ampB; /* amplitude of frequency B if dual
                                frequency wave, or modulation rate if AM
                                wave (0~100 in 1% unit), ignored if single frequency wave */
    UINT16  mode; /* mode, overwrite or mix over the
                                Decoder output */
    INT16   nextSeg; /* the index of next segment relative
                                to the current segment. e.g., 1 means
                                to go the following segment, 0 means
                                repeat the current segment, -2 means
                                go back to previous 2 segments.
                                XTG_LASTSEG means end-of-tone */
} XTGToneSeg_t;
```

Warning: New tone definition must be added during the initialization after `xDspSysInit()`. The predefined country-specific call progress tone will be overwritten if a new tone is added with the same tone ID.

8.3 Adding Tones to Tone Detector

<table>
<thead>
<tr>
<th>Prototype</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Status_t xBuildToneTD(UINT8 toneId, XTDToneInfo_t *pToneInfo, UINT32 *pErrCode);</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Input</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>• <code>toneId</code> – Tone ID, must be in the range of 16 ~ 255</td>
<td></td>
</tr>
<tr>
<td>• <code>pToneInfo</code> — Tone detection criterion information</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>pErrCode</code> – Error code if errors</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Return</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>• <code>XSUCC</code> if successful</td>
<td></td>
</tr>
<tr>
<td>• Otherwise <code>XERROR</code></td>
<td></td>
</tr>
</tbody>
</table>
Description

This function adds a criterion for the Tone Detector to detect a new tone. The criterion specifies the qualification ranges in a set of parameters defined as:

```c
/* segment data for tone detection template. */
typedef struct {
  UINT16   type;    /* tone type (single or dual frequency tone) */
  UINT16   criteria; /* loose, medium or tight, use medium for normal case, use loose to get higher detection probability in poor SNR, use tight to get lower false detection probability in good SNR */
  UINT16   freqLowA; /* low bound of the 1st frequency in Hz */
  UINT16   freqHighA; /* high bound of the 1st frequency in Hz */
  UINT16   freqLowB; /* low bound of the 2nd frequency in Hz */
  UINT16   freqHighB; /* high bound of the 2nd frequency in Hz */
  INT16    ampLowA;  /* low level of the 1st frequency in dBm */
  INT16    ampHighA; /* high level of the 1st frequency in dBm */
  INT16    ampLowB;  /* low level of the 2nd frequency in dBm */
  INT16    ampHighB; /* high level of the 2nd frequency in dBm */
  UINT8    attributes; /* attribute (report the tone on, tone off or both on/off) */
} XTDToneInfo_t;
```

**Warning:** New tone detection criterion must be added during the initialization before `xDspSysInit()`.

### 8.4 Getting DSP Resource Configuration and Routing Information

<table>
<thead>
<tr>
<th>Prototype</th>
<th><code>void xDspGetResConfig(XDSPResConfig_t *pCfgInfo)</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
<td><code>pCfgInfo</code> – Pointer to DSP configuration data structure</td>
</tr>
<tr>
<td>Output</td>
<td>The resource configuration and the assignment of the routing streams</td>
</tr>
<tr>
<td>Return</td>
<td>None</td>
</tr>
</tbody>
</table>

API Reference Manual 39
Description

The user’s applications can call this function any time after xDspSysInit() to obtain the DSP resource configuration and the stream IDs assigned to the T-Ports of each type of the resources. The data structure XDSPResConfig_t is defined as:

```c
typedef struct {
    int numChTDM;  /* number of TDM termination channels */
    int numChIP;   /* number of IP termination channels */
    int numPlayers; /* number of player instances */
    int numMixers; /* number of Audio Mixers */
    int numPortsPerMixer; /* number of ports per mixer */
    int numStreams; /* number of total streams in the router */
    int streamBaseTDM; /* T-Port stream ID of the first TMD termination channel */
    int streamBaseIP;  /* T-Port stream ID of the first IP termination channel */
    int streamBasePly; /* T-Port stream ID 1st port of the 1st Player instance */
    int streamBaseMix; /* T-Port stream ID of the first mixer port */
    int countryCode;  /* country code */
} XDSPResConfig_t;
```

The stream ID information is used for the application to connect the T-Ports and L-Ports of the resources.
9.0 Complementary Functions

9.1 Direct Parameter Access

The user's applications can bypass the messages and directly access the DSP parameters. This allows quicker access without having to send a message and receive a response. All parameters can be directly read, but only some of them can be directly modified.

The functions to access the parameters are:

<table>
<thead>
<tr>
<th>Prototype</th>
<th>XStatus_t xDspParmRead(UINT8 res, UINT16 inst, UINT16 parmId, UINT16 *pParmVal);</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
<td>• res – DSP resource ID &lt;br&gt;• inst – Instance ID of the resource &lt;br&gt;• parmId – Parameter ID &lt;br&gt;• pParmVal – Pointer to the variable that receives the returned parameter value</td>
</tr>
<tr>
<td>Output</td>
<td>Parameter value</td>
</tr>
<tr>
<td>Return</td>
<td>• XSUCC if successful &lt;br&gt;• Otherwise XERROR</td>
</tr>
<tr>
<td>Description</td>
<td>This function retrieves the specified parameter value.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Prototype</th>
<th>XStatus_t xDspParmWrite(UINT8 res, UINT16 inst, UINT16 parmId, UINT16 parmVal, UINT32 transId);</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
<td>• res – DSP resource ID &lt;br&gt;• inst – Instance ID of the resource &lt;br&gt;• parmId – Parameter ID &lt;br&gt;• parmVal – Parameter value to be set &lt;br&gt;• transId – Transaction ID</td>
</tr>
<tr>
<td>Output</td>
<td>None</td>
</tr>
<tr>
<td>Return</td>
<td>• XSUCC if successful &lt;br&gt;• Otherwise XERROR</td>
</tr>
<tr>
<td>Description</td>
<td>This function sets the value of the specified parameter.</td>
</tr>
</tbody>
</table>

9.2 Flash Hook Detection

<table>
<thead>
<tr>
<th>Prototype</th>
<th>Status_t xFlashHookDetect(UINT16 channel, XHookState_t hookState, XUINT32 transId);</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
<td>• channel – Channel number starting from 1 &lt;br&gt;• hookState – Hook state, XHOOK_STATE_ON or XHOOK_STATE_OFF &lt;br&gt;• transId – Transaction ID</td>
</tr>
</tbody>
</table>
### 9.3 Cache Prompt Registration

<table>
<thead>
<tr>
<th>Prototype</th>
<th>Status_t xFlashHookDetect(UINT16 channel, XHookState_t hookState, XUINT32 transId);</th>
</tr>
</thead>
<tbody>
<tr>
<td>Output</td>
<td>None</td>
</tr>
<tr>
<td>Return</td>
<td>• XSUC if successful • Otherwise XERROR</td>
</tr>
</tbody>
</table>

**Description**

This function is called by the SLIC driver to report the hook state changes via the event message. If an on-hook transition followed by an off-hook one within the time specified by the XPARMID_NET_FLASH_HK parameter, a flash hook event is reported. The hook states are defined as:

```c
typedef enum{
    XHOOK_STATE_ON = 0,
    XHOOK_STATE_OFF,
    XHOOK_STATE_FLASH
}XHookState_t;
```

---

**Prototype**

```c
XMediaHandle_t xDspRegCachePrompt(XCachePromptDesc_t *pDesc);
```

**Input**

- `pDesc` – The pointer to structure XCachePromptDesc_t.

**Output**

None

**Return**

- `XMediaHandle` — Returns XMEDIA_HANDLE_NULL in the error case.

**Description**

This function is called to register a cached prompt for playing at a later time. XCachePromptDesc_t describes the data required to register a cached prompt.

```c
typedef struct{
    UINT8 *pBuffer; /* Pointer to the play buffer. */
    INT32 size;    /* The size of play buffer. */
    XCoderType_t type; /* The type of data in play buffer. The valid types are XCODER_TYPE_G711MU_10MS, XCODER_TYPE_G711A_10MS and XCODER_TYPE_G729A */
} XCachePromptDesc_t;
```
10.0 Constant Data

This section lists up the definitions for constant data such as error codes and event codes.

10.1 Error Codes

Errors are reported via `XMSG_ERROR` message with an error code and two error data. The common error codes are defined as:

```c
#define XERR_SYSTEM 0x0001 /* system error */
#define XERR_HSSIF 0x0002 /* HSS interface error */
#define XERR_MEMORY 0x0003 /* memory error */
#define XERR_INVALID_RES_ID 0x0011 /* invalid resource id */
#define XERR_INVALID_CHAN_ID 0x0012 /* invalid channel id */
#define XERR_INVALID_PARM_ID 0x0013 /* invalid parameter id */
#define XERR_INVALID_STREAM_ID 0x0014 /* invalid stream id */
#define XERR_PARM_READONLY 0x0015 /* real only parameter */
#define XERR_PARM_GET_FAIL 0x0017 /* cannot get parameter */
#define XERR_PARM_SET_FAIL 0x0016 /* cannot set parameter */
#define XERR_UNEXPECTED_MSG 0x0018 /* unexpected message */
#define XERR_UNSUPPORTED_MSG 0x0019 /* unsupported message */
#define XERR_ALGORITHM 0x0041 /* algorithm related error */
#define XERR_OTHERS 0x00ff /* other errors */
```

The resource-specific error codes are defined as:

```c
#define XERR_INVALID_CODE_TYPE 0x401 /* invalid codec type */
#define XERR_INVALID_FPP 0x402 /* invalid # frms per pkt */
#define XERR_TG_INVALID_TONE_ID 0x403 /* invalid tone ID */
#define XERR_TG_INVALID_TID_NUM 0x404 /* too many tone IDs */
#define XERR_TG_INVALID_DATA_NUM 0x405 /* too many FSK data */
#define XERR_TD_INVALID_DIGIT_NUM 0x406 /* too many digits */
#define XERR_RESOURCE_BUSY 0x407 /* resource is busy */
#define XERR_RESOURCE_IDLE 0x408 /* resource is idle */
#define XERR_MA_DEEP_RECURSIVE 0x409 /* deep recursive msg decoder */
#define XERR_MA_MSG_DECORDER 0x40a /* message decoding fail */
```

10.2 Event Codes

Events are reported via `XMSG_EVENT` message with an event code and two event data. The resource specific event codes are defined as:

```c
#define XEVN_CODE_TD_TONEON 0x101 /* tone-on event */
#define XEVN_CODE_TD_TONEOFF 0x102 /* tone-off event */
#define XEVN_LOST_PACKET 0x103 /* lost packet */
#define XEVN_DEC_PACKET_CHNG 0x104 /* RTP payload type changed */
#define XEVN_NET_HOOK_STATE 0x105 /* hook state change detected */
#define XEVN_NET_TIMER 0x106 /* timer expired */
```
10.3 Tone IDs

10.3.1 DTMF Tone IDs

The DTMF tone IDs used by Tone Generator and Detector are defined as:

```
#define RFC_TID_DTMF_0 0
#define RFC_TID_DTMF_1 1
#define RFC_TID_DTMF_2 2
#define RFC_TID_DTMF_3 3
#define RFC_TID_DTMF_4 4
#define RFC_TID_DTMF_5 5
#define RFC_TID_DTMF_6 6
#define RFC_TID_DTMF_7 7
#define RFC_TID_DTMF_8 8
#define RFC_TID_DTMF_9 9
#define RFC_TID_DTMF_STAR 10
#define RFC_TID_DTMF_POUND 11
#define RFC_TID_DTMF_A 12
#define RFC_TID_DTMF_B 13
#define RFC_TID_DTMF_C 14
#define RFC_TID_DTMF_D 15
```

10.3.2 Fax-Tone IDs

Fax tone IDs reported by the Tone Detector for fax bypass applications. Not supported by the Tone Generator.

```
#define RFC_TID_FAX_CED 32
#define RFC_TID_FAX_CNG 36
#define RFC_TID_FAX_V21 40
```
### 10.3.3 Call-Progression IDs

The general call progress tone IDs used by the Tone Generator are defined as:

```c
#define RFC_TID_DIAL 66
#define RFC_TID_PBX_DIAL 67
#define RFC_TID_SP_DIAL 68
#define RFC_TID_2ND_DIAL 69
#define RFC_TID_RING 70
#define RFC_TID_SP_RING 71
#define RFC_TID_BUSY 72
#define RFC_TID_CONGESTION 73
#define RFC_TID_SP_INFO 74
#define RFC_TID_COMFORT 75
#define RFC_TID_HOLD 76
#define RFC_TID_REC 77
#define RFC_TID_CALLER_WT 78
#define RFC_TID_CALL_WAIT 79
#define RFC_TID_PAY 80
#define RFC_TID_POS_IND 81
#define RFC_TID_NEG_IND 82
#define RFC_TID_WARNING 83
#define RFC_TID_INSTRUCTION 84
#define RFC_TID_CAL_CARD 85
#define RFC_TID_PAYPHONE 86
```

Currently only the following specific call progress tones are supported for tone generation:

#### Japan Call-Progress Tones

```c
#define NTT_TID_DT RFC_TID_DIAL    /* dial tone */
#define NTT_TID_RBT RFC_TID_RING    /* ring back tone */
#define NTT_TID_BT RFC_TID_BUSY     /* busy tone */
#define NTT_TID_PDT RFC_TID_PBX_DIAL /* private dial tone */
#define NTT_TID_SDT RFC_TID_2ND_DIAL /* 2nd dial tone */
#define NTT_TID_CPT RFC_TID_POS_IND /* acceptance tone */
#define NTT_TID_HST RFC_TID_HOLD     /* hold service tone */
#define NTT_TID_IIT RFC_TID_CALL_WT /* incoming id tone */
#define NTT_TID_SIIT 110            /* special incoming id tone */
#define NTT_TID_HOW RFC_TID_OFFHK_WARN /* howler tone */
```

#### United States Call-Progress Tones

```c
#define US_TID_DIAL RFC_TID_DIAL     /* dial tone */
#define US_TID_RING RFC_TID_RING     /* ring back tone */
#define US_TID_BUSY RFC_TID_BUSY     /* busy tone */
#define US_TID_RC_DIAL RFC_TID_SP_DIAL /* recall dial tone */
#define US_TID_PBX_DIAL RFC_TID_PBX_DIAL /* PBX dial tone */
#define US_TID_CONGESTION RFC_TID_CONGESTION /* congestion tone */
#define US_TID_CALL_WT RFC_TID_CALL_WT /* call waiting tone */
#define US_TID_WARN_OPER 110         /* operator intervening tone */
```
China Call-Progress Tones

```
#define PRC_TID_DIAL RFC_TID_DIAL /* dial tone */
#define PRC_TID_RING RFC_TID_RING /* ring back tone */
#define PRC_TID_BUSY RFC_TID_BUSY /* busy tone */
#define PRC_TID_SP_DIAL RFC_TID_SP_DIAL /* special dial tone */
#define PRC_TID_CONGESTION RFC_TID_CONGESTION /* congestion tone */
#define PRC_TID_UNAVAILABLE RFC_TID_UNAVAILABLE /* number unavailable */
#define PRC_TID_TOLL RFC_TID_TOLL /* toll (long distance) */
#define PRC_TID_QUEUE RFC_TID_QUEUE /* queue tone */
#define PRC_TID_CALL_WAIT RFC_TID_CALL_WAIT /* call waiting tone */
#define PRC_TID_THR_PARTY RFC_TID_THR_PARTY /* 3 party remind tone */
#define PRC_TID_CONFIRMATION RFC_TID_CONFIRMATION /* confirmation tone */
#define PRC_TID_OFFHOOK_WARN RFC_TID_OFFHOOK_WARN /* off hook warning */
```

### 10.4 Other Constants

The coder types used in the XPARM_ID_DEC_CTYPE and XPARM_ID_ENC_CTYPE parameters and the XMSG_CODER_START message are defined as:

```
typedef enum{
    XCODER_TYPE_PASSTHRU = 0,
    XCODER_TYPE_G711MU_10MS,
    XCODER_TYPE_G711A_10MS,
    XCODER_TYPE_G729A,
    XCODER_TYPE_G723,
    XCODER_TYPE_G729 = 17,
    XCODER_TYPE_UNDEF = -1
} XCoderType_t;
```

Mask bits used to specify the coder type subset in Decoder auto-switch parameter are defined as:

```
#define XPARM_DEC_AUTOSW_OFF        0x0000
#define XPARM_DEC_AUTOSW_G711MU     0x0001
#define XPARM_DEC_AUTOSW_G711A      0x0002
#define XPARM_DEC_AUTOSW_G729A      0x0004
#define XPARM_DEC_AUTOSW_G723        0x0008
#define XPARM_DEC_AUTOSW_ALL        0xffff
```
Mask bits used to specify the termination digits in the XMSG_TG_PLAY_CMPLT and XMSG_TD_RCV_CMPLT messages are defined as:

```
#define XTD_TERM_DIGIT_NONE    0x0000
#define XTD_TERM_DIGIT_0       0x0001
#define XTD_TERM_DIGIT_1       0x0002
#define XTD_TERM_DIGIT_2       0x0004
#define XTD_TERM_DIGIT_3       0x0008
#define XTD_TERM_DIGIT_4       0x0010
#define XTD_TERM_DIGIT_5       0x0020
#define XTD_TERM_DIGIT_6       0x0040
#define XTD_TERM_DIGIT_7       0x0080
#define XTD_TERM_DIGIT_8       0x0100
#define XTD_TERM_DIGIT_9       0x0200
#define XTD_TERM_DIGIT_STAR    0x0400
#define XTD_TERM_DIGIT_POUND   0x0800
#define XTD_TERM_DIGIT_A       0x1000
#define XTD_TERM_DIGIT_B       0x2000
#define XTD_TERM_DIGIT_C       0x4000
#define XTD_TERM_DIGIT_D       0x8000
```

The stop-reasons in the XMSG_TG_PLAY_CMPLT and XMSG_TD_RCV_CMPLT messages are defined as:

```
#define XMSG_STOP_REASON_USER 1 /* stopped by XMSG_STOP message */
#define XMSG_STOP_REASON_EOD 2 /* end of data */
#define XMSG_STOP_REASON_TERM 3 /* stopped by the terminate digits */
#define XMSG_STOP_REASON_TIMEOUT 4 /* time out */
```