ENVISIONING • EMPOWERING • EXCELLING

This document has been made public and no NDA is needed.

The "Confidential" statement in the attached material is no longer valid and may be disregarded.

This document falls into one of these categories:

1. **Document has MaxLinear branding**
   The "MaxLinear Confidential" statement will be removed from the document upon its next revision
2. **Document has non-MaxLinear branding**
   In 2020, MaxLinear acquired the Connected Home Division business of Intel Corporation, including the former Intel® product/s referenced in the title of the attached material. The MaxLinear logo will be added to the attached material upon its next revision. MaxLinear is now the manufacturer of this product.

Direct any questions and product support requests to your MaxLinear sales contact, MaxLinear Sales Representative or Distributor, or login to your myMxL account and create a new support ticket.

MaxLinear

Corporate Headquarters:
5966 La Place Court
Suite 100
Carlsbad, CA 92008
Tel.:+1 (760) 692-0711
Fax: +1 (760) 444-8598
www.maxlinear.com

Telephony Chipset for CPE

DXS Series

DXS API Device Driver

for

DXS102 (PEF32002VTV12, PEF32002VTV13)
DXS101 (PEF32001VTV12, PEF32001VSV12, PEF32001VSV13)

Programmer's Reference

MaxLinear Confidential

Revision 3.0, 2022-03-31
Reference ID 617580

Legal Notice

The content of this document is furnished for informational use only, is subject to change without notice, and should not be construed as a commitment by MaxLinear, Inc. MaxLinear, Inc. assumes no responsibility or liability for any errors or inaccuracies that may appear in the informational content contained in this guide. Complying with all applicable copyright laws is the responsibility of the user. Without limiting the rights under copyright, no part of this document may be reproduced into, stored in, or introduced into a retrieval system, or transmitted in any form or by any means (electronic, mechanical, photocopying, recording, or otherwise), or for any purpose, without the express written permission of MaxLinear, Inc.

MaxLinear, Inc. does not recommend the use of any of its products in life support applications where the failure or malfunction of the product can reasonably be expected to cause failure of the life support system or to significantly affect its safety or effectiveness. Products are not authorized for use in such applications unless MaxLinear, Inc. receives, in writing, assurances to its satisfaction that: (a) the risk of injury or damage has been minimized; (b) the user assumes all such risks; (c) potential liability of MaxLinear, Inc. is adequately protected under the circumstances.

MaxLinear, Inc. may have patents, patent applications, trademarks, copyrights, or other intellectual property rights covering subject matter in this document. Except as expressly provided in any written license agreement from MaxLinear, Inc., the furnishing of this document does not give you any license to these patents, trademarks, copyrights, or other intellectual property.

MaxLinear, the MaxLinear logo, any MaxLinear trademarks (MxL, Full-Spectrum Capture, FSC, G.now, AirPHY, Puma, and AnyWAN), and the MaxLinear logo on the products sold are all property of MaxLinear, Inc. or one of MaxLinear's subsidiaries in the U.S.A. and other countries. All rights reserved.

*Other company trademarks and product names appearing herein are the property of their respective owners.

MaxLinear, Inc.
5966 La Place Court, Suite 100
Carlsbad, CA 92008
Tel.: +1 (760) 692-0711
Fax: +1 (760) 444-8598
www.maxlinear.com

Revision History

Current:	**Revision 3.0, 2022-03-31**
Previous:	**Revision 2.0, 2018-10-31**
Page	**Major changes since previous revision**
All	This document was rebranded from Intel to MaxLinear, including the logo and Legal Notice page. All product names were rebranded by removing Intel® from their names.
All	Removed the DXS101 (PEF32001VV12) device.
All	Added the DXS102 (PEF32002VTV13) and DXS101 (PEF32001VSV13) devices.
121	Updated the **Literature References**.

Table of Contents

List of Figures

List of Tables

Preface

This Programmer's Reference describes the structure and usage of MaxLinear's DXS API device driver.

To simplify matters, the following synonyms are used:

DXS Series, DXS

Synonyms used for these MaxLinear Telephony Chipset for CPE DXS Series devices:

- DXS102 (PEF32002VTV12, PEF32002VTV13)
- DXS101 (PEF32001VTV12, PEF32001VSV12, PEF32001VSV13)

Organization of this Document

This document is organized as follows:

- **Chapter 1, Introduction**
  Provides an overview of the DXS API device driver. This chapter describes how to compile the device driver and explains the configuration options.
- **Chapter 2, Description of the Feature Specific API Device Driver Interfaces**
  Describes the DXS Series specific API device driver interfaces.
- **Chapter 3, Module Reference**
  Provides a detailed reference description of the DXS API device driver interfaces.
- **Literature References**
  References to related documentation and standards.
- **Terminology**
  List of abbreviations and descriptions of symbols.

1 Introduction

This chapter provides an introduction to the DXS API Device Driver and how to compile it.

1.1 Introduction to the DXS API Device Driver

The DXS API Device Driver (referred to hereafter as “driver”) is a software module that contains the implementation of an API function set sufficient to control the DXS Series device. **Figure 1** provides an overview of the software architecture and software interaction with the hardware.

The driver is a library of functions for use by the application software. The functions configure different parameters of the DXS Series device; switch the line modes; get different measurements from the line; and transport line and system events from the DXS Series device to the application software.

The application software controls how these functions are called.

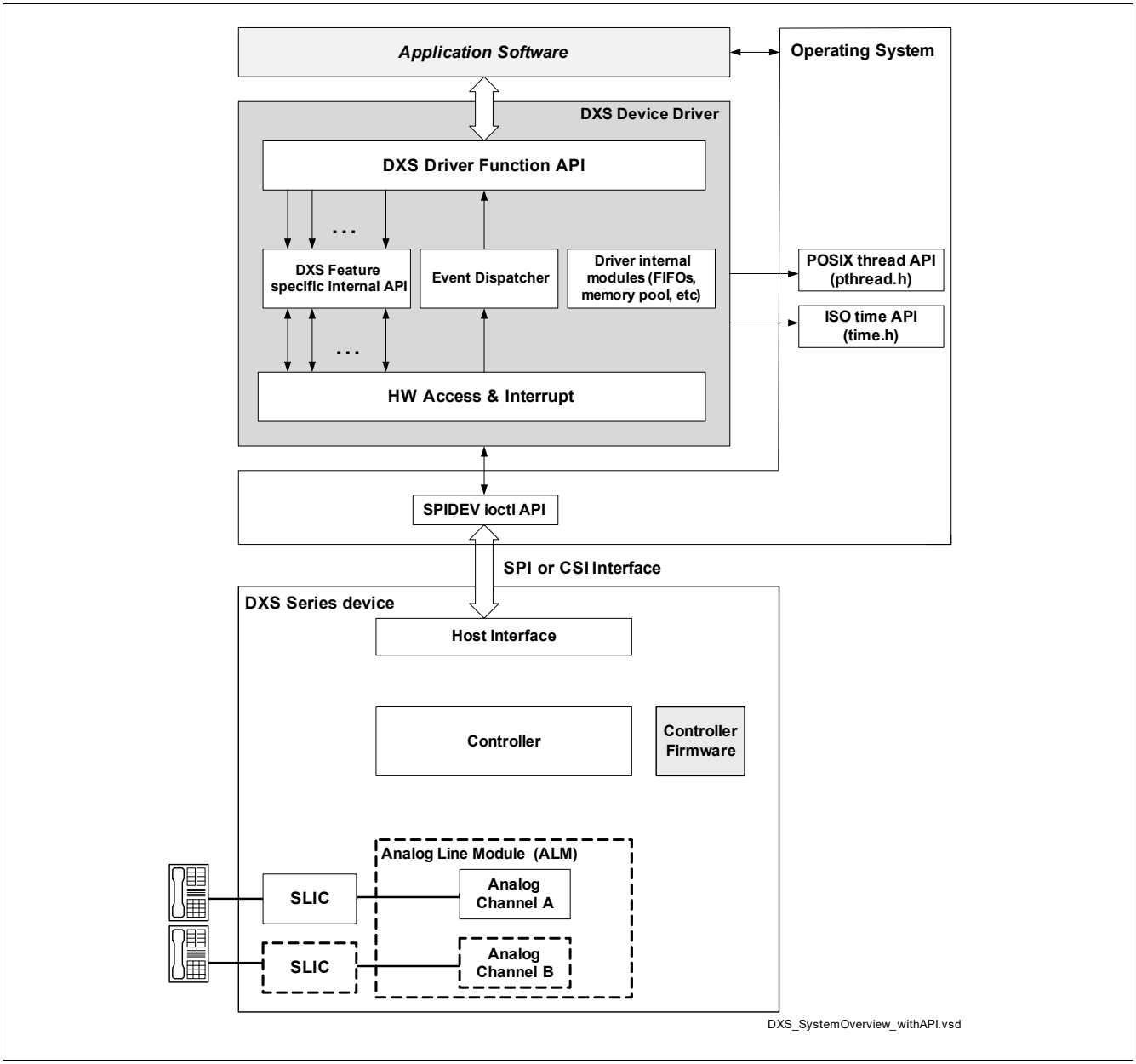

Figure 1 DXS API Device Driver Architecture and System Overview

The driver utilizes the Linux* SPI core user space API to access the DXS Series device. This means the driver's SPI access layer must be adapted when the driver is ported to any operating system other than Linux*.

The driver relies on POSIX thread and timer API functions exported by the **pthread.h** and **time.h** header files. These header files are provided by most state-of-the-art toolchains.

The driver accesses the DXS Series device in either interrupt mode or polling mode. When polling mode is chosen, it is not necessary to implement the interrupt hardware in the application design.

The driver operates on both big-endian and little-endian controllers.

1.2 Compilation of the DXS API Device Driver

This chapter describes the driver compilation for Linux*.

Table 1 DXS Driver Configuration Options

Description	Default[1)]	Required	Configuration Option
Path to target Linux* kernel sources include directory	–	Always	--with-kernel-incl[=DIR]
Path to target Linux* kernel build directory	–	Always	--with-kernel-build[=DIR]
DTMF Detector	Enabled	Optional	--disable-dtmfd
Tone Generator	Enabled	Optional	--disable-tg
Hook State Machine	Enabled	Optional	--disable-hook-sm
Network Line Testing GR-909	Enabled	Optional	--disable-nlt-gr909
Network Line Testing Capacitance Measurement	Enabled	Optional	--disable-nlt-capmeas
Interrupt Support	Enabled	Optional	--disable-interrupts
Universal Tone Detector	Disabled	Optional	--enable-utd
TTX Metering	Disabled	Optional	--enable-metering
AC Level Meter Measurements	Disabled	Optional	--enable-nlt-acmeter
FSK Generator	Disabled	Optional	--enable-fsk
CID Generator State Machine	Disabled	Optional	--enable-cid
GPIO	Disabled	Optional	--enable-gpio
Debug API	Disabled	Optional	--enable-debug-api
Maximum devices to support	1	Optional	--with-max-devices=val
Polling interval (ms)	10	Optional	--with-polling-interval-ms=val

1) Status when configuration option is omitted.

2 Description of the Feature Specific API Device Driver Interfaces

This chapter describes how to use the DXS API. To simplify the code examples, checking of API return codes was omitted. It is highly recommended that the end user application handles the API return codes.

2.1 Tone Generation

The driver provides an API for tone generation in the RX direction (towards telephone handset).

The **DxsToneConfig** function configures the tone to be played out. It is possible to configure the lower and higher frequency levels of the tone individually. Enabling amplitude modulation is optional.

The **DxsToneEnable** function starts tone playout. It is mandatory to configure the tone beforehand using the **DxsToneConfig** function.

The **DxsToneDisable** function stops the tone currently being played. Tone configuration is not reset.

For cadenced tones (e.g. busy tone), the configuration must be carried out once, then the timer is able to call **DxsToneEnable** and **DxsToneDisable** sequentially.

Example of Dial Tone Play

Configure and play out the US dial tone for device `dev` and channel `line`.

To achieve the required resulting level of -13 dBm, the following factors must taken into consideration:

- BBD coefficients match the load and -10 dBr relative gain is assumed for the RX path.
- For dual frequency tone, -6 dBm0 is configured by the API for each frequency.
  Assuming a relative gain of -10 dBr in the RX direction, the expected output level at the tip-ring wires for each tone is: -6 dBm0 - 10 dBr = -16 dBm
  Calculating the sum of the two applied tones of -16 dBm each, the resulting level is -13 dBm, as requested.

```
/* US dial tone: f1=350Hz, f2=440Hz, continuous, -13dBm */
/* configure level1, level2, f1, f2, no amplitude modulation, tone levels are
   given in 0.1 dBm0 units*/
DxsToneConfig(dev, line, -60, -60, 350, 440, 0);
/* start tone playout */
DxsToneEnable(dev, line);
```

Stop tone playout:

```
DxsToneDisable(dev, line);
```

2.2 FSK Generation

The driver provides an API for FSK signal generation for the Caller ID. The API described in this section only serves for low-level FSK generator access, but also allows implementation of a specific high-level Caller ID protocol.

Attention: The driver also provides a high-level Caller ID protocol API (see Chapter 2.3), which covers the most popular Caller ID protocol standards.

The **DxsFskConfig** function configures the FSK generator output level, the number of seizure bits and the number of mark bits.

The function pair **DxsFskEnable** and **DxsFskDisable** switches the FSK generator on and off.

For the **DxsFskEnable** function, two additional parameters must be specified in addition to the device and line selectors:

- FSK generator standard selector (Bell 202 or ITU-T V.23)
- Auto Deactivation flag (OFF or ON)

The Auto Deactivation flag defines the behavior of the **DxsFskDisable** function. When the flag is set to “OFF”, the FSK generator stops immediately when the **DxsFskDisable** function is called. In this case the remaining data in the DXS FSK buffer is not transmitted. When the flag is set to “ON”, the FSK generator continues transmitting the data from its buffer after the **DxsFskDisable** function is called until all the data is sent from the DXS FSK data buffer; only then does the FSK generator stop.

The **DxsFskData** function provides the next data buffer for the FSK generator. The maximum size of the buffer is 27 bytes. The next data buffer must be provided upon request from the driver. The request event is DXS_EVENT_CID_REQ_DATA. This event is generated when the FSK generator has less than 5 bytes remaining in the data buffer. The user application must supply the next data buffer by calling **DxsFskData** within the next 30 ms.

The user application must handle the entire data message fragmentation and the particular CID protocol implementation.

Example of FSK Generation

Configure and enable FSK generator for device `dev` and channel `line`:

```
/* Configure FSK generator output -6.9dBm0,
   300 seizure bits, 180 mark bits */
DxsFskConfig(dev, line, -69, 300, 180);
/* Enable FSK generator for V.23 Bell 202 specification,
   with auto-deactivation enabled */
DxsFskEnable(dev, line, 0, 1);
/* wait for DXS_EVENT_CID_REQ_DATA event */
```

2.3 Caller ID Generation

The driver provides a Caller ID generation macro API in addition to the FSK generator API described in **Chapter 2.2**. The following Caller ID protocols are supported:

- ETSI[1], refer to **[5]**, **[6]**
- Telcordia, refer to **[9]**
- SIN BT, refer to **[7]**, **[8]**
- NTT, refer to **[10]**

The Caller ID API supports both on-hook Caller ID transmission (Type 1) and off-hook Caller ID transmission (Type 2).

The first step for Caller ID generation is to call the **DxsCidInit** function. This function initializes the Caller ID instance for a given channel and must be called before any other Caller ID API function. The **DxsCidInit** function configures the Caller ID protocol and the alerting signal that will be used for Caller ID Type 1 sequence transmission. The alerting signal configuration is ignored for Caller ID Type 2 transmission. Possible alerting signals for different Caller ID protocols are summarized in **Table 2**.

Attention: The DxsCidInit function must be called at the Caller ID protocol configuration stage. It is not necessary to call the DxsCidInit function more than once unless the goal is to change the Caller ID protocol or alerting signal type. Calling the DxsCidInit function flushes all previously configured Caller ID messages.

Table 2 Alerting Signals for Caller ID Protocols

Caller ID Protocol	Possible Alerting Signals for Type 1 Caller ID
ETSI	None (Caller ID message between 1st and 2nd ring bursts), DTAS, DTAS after Line Reversal, RPAS
Telcordia	None (Caller ID message between 1st and 2nd ring bursts), OSI
SIN BT	DTAS after Line Reversal
NTT	CAR

The **DxsCidTimerConfig** function is aimed at configuring timers for supported Caller ID protocols. The use of this function is optional. The **DxsCidTimerConfig** function allows fine adjustment of timers for Caller ID protocols previously selected using the **DxsCidInit** function. When the **DxsCidTimerConfig** function call is omitted, the default timing configuration is applied.

The **DxsCidMsgSetup** function prepares the different Caller ID messages by adding one parameter to the parameter list of a given message type. Possible message types are defined by the **DXS_CID_MsgType_t** enumerator. Possible message parameters are defined by the **DXS_CID_MsgParam_t** enumerator type. When a particular message parameter was already added to the parameter list of the Caller ID message, only the parameter value is updated. The **DxsCidMsgSetup** function must be called for each parameter of the Caller ID message.

The **DxsCidMsgReset** function cleans up the previously configured Caller ID message entirely, deleting all previously added message parameters.

The **DxsCidTypeIMsgStart** and **DxsCidTypeIIMsgStart** functions are used to start the Caller ID sequence playout for Caller ID Type 1 and Type 2 respectively. The Caller ID protocol must be configured previously using the **DxsCidInit** function, and the Caller ID message must be set up previously using the **DxsCidMsgSetup** function. Both the **DxsCidTypeIMsgStart** and **DxsCidTypeIIMsgStart** function return immediately without waiting for Caller ID sequence completion. The driver spawns the DXS_EVENT_CID_SEQ_END event to indicate the completion of the Caller ID sequence.

1) Only FSK data transmission is supported. DTMF data transmission is not implemented.

When an error occurs during the Caller ID sequence playout, the DXS_EVENT_CID_SEQ_ERROR event is generated by the driver. Additional fields in the DXS_EVENT_CID_SEQ_ERROR event message indicate the cause of the error.

It is possible to use the **DxsCidSeqStop** function to abort the running Caller ID sequence at any moment.

Attention: An application must know the hook state of the line before starting a Type 1 or Type 2 Caller ID sequence.

Example of Caller ID Generation

Configure and play out a Caller ID Type 1 sequence according to the ETSI protocol, without an alerting signal (Caller ID message to be transmitted between the 1st and 2nd ring bursts), using default timing:

```
struct tm tms = {0};
time_t rawtime;
uint8_t i = 0;
char timedate_buffer[8];
char cid_cli[] = "123";
char cid_party_name[] = "CALLING";


/* Initialize Caller ID for device "dev", channel "line":
   1) ETSI protocol
   2) No alerting signal
   3) FSK data format
   4) DTMF acknowledgment signal 'D' (for type2 transmission) */
DxsCidInit(dev, line, DXS_CID_STD_ETSI, DXS_CID_AS_NONE, DXS_CID_DATA_FSK, 31);

/* Prepare Call Setup message for device "dev", channel "line":
   1) Add Calling Line ID parameter
   2) Add Calling Party Name parameter
   3) Add Date Time parameter */
DxsCidMsgSetup(dev, line, DXS_CID_MSG_TYPE_CS,
               DXS_CID_MSG_PARAM_CALLING_LINE_ID,
               cid_cli,
               strlen(cid_cli));
DxsCidMsgSetup(dev, line, DXS_CID_MSG_TYPE_CS,
               DXS_CID_MSG_PARAM_CALLING_PARTY_NAME,
               cid_party_name,
               strlen(cid_party_name));
/* Date and time */
rawtime = time(NULL);
localtime_r(&rawtime, &tms);
/* month */
timedate_buffer[i++] = (tms.tm_mon + 1) / 10 + '0';
timedate_buffer[i++] = (tms.tm_mon + 1) % 10 + '0';
/* day */
timedate_buffer[i++] = tms.tm_mday / 10 + '0';
timedate_buffer[i++] = tms.tm_mday % 10 + '0';
/* hour */
timedate_buffer[i++] = tms.tm_hour / 10 + '0';
timedate_buffer[i++] = tms.tm_hour % 10 + '0';
/* minute */
```

```
timedate_buffer[i++] = tms.tm_min / 10 + '0';
timedate_buffer[i++] = tms.tm_min % 10 + '0';
DxsCidMsgSetup(dev, line, DXS_CID_MSG_TYPE_CS,
               DXS_CID_MSG_PARAM_DATE_TIME,
               timedate_buffer,
               sizeof(timedate_buffer));

/* Transmit Type1 Call Setup message */
DxsCidTypeIMsgStart(dev, line, DXS_CID_MSG_TYPE_CS);

/* Wait for DXS_EVENT_CID_SEQ_END event */
```

2.4 Universal Tone Detection

The driver provides an API for Universal Tone Detection switching. The **DxsUtdEnable** function enables UTD, and the **DxsUtdDisable** function disables UTD.

The UTD must be configured via the **DxsBBD** function using a BBD file containing at least one block with number $0x1007_H$. The BBD file for UTD is generated by the XTCOS program. The UTD must be configured via **DxsBBD** before **DxsUtdEnable** is called. The BBD file possibly contains configurations for multiple tone indexes, but only one tone detector is allowed to be active at a time. The tone index table is global for all channels, but the **DxsBBD** function is called per channel. The tone index table is allowed a maximum of 5 entries.

The tone detector is activated either in the direction from the PCM bus or in the direction from the analog line. Simultaneous detection in both directions is not possible.

Sequential calling of **DxsUtdEnable** is allowed. In this case the next active configuration replaces the previous configuration.

A tone detection event is reported by the driver via the DXS_EVENT_TONE_DET_CPT event. On tone completion, the DXS_EVENT_TONE_DET_CPT_END event is reported.

Example of Universal Tone Detection

Configure the tone index 24 and activate UTD for device `dev` and channel `line`:

```
/* Download the UTD BBD file for device "dev" channel "line".
   Tone index 24 is configured within the BBD file block 0x1007. */
DxsBBD(dev, line, pData, nBytes);
/* Activate UTD for tone index 24, coming from analog line */
DxsUtdEnable(dev, line, 24, 1);
/* Wait for DXS_EVENT_TONE_DET_CPT event */
```

2.5 DTMF Detection

The driver provides an API for DTMF detector configuration, activation and deactivation. The **DxsDtmfConfig** function is used for DTMF detector configuration, the functions **DxsDtmfEnable** and **DxsDtmfDisable** for activation and deactivation respectively.

The minimum signal level and maximum allowed signal twist are configurable DTMF detector parameters.

A DTMF detection event is reported by the driver via a DXS_EVENT_DTMF_DIGIT event. The detected digit is reported as event data, see **DXS_Event_t**, **DXS_Event_data_t** and **DXS_EVENT_DATA_DTMF_t** type definitions.

Example of DTMF Detection

Configure -30.0 dBm0 as minimum DTMF signal level and 7.0 dB as maximum DTMF twist for device `dev` and channel `line`, then activate DTMF detector.

```
/* Minimum level -30.0 dBm0, maximum twist 7.0 dB */
DxsDtmfConfig(dev, line, -300, 70);
/* Activate DTMF detector for device "dev", channel "line" */
DxsDtmfEnable(dev, line);
/* The DXS_EVENT_DTMF_DIGIT event is generated
   when DTMF digit is detected */
```

2.6 TTX Metering

The driver provides an API for Teletax metering pulse playout. The physical parameters of a metering pulse (frequency, level, duration) are not configurable via the API but are provided in a BBD file generated by the XTCOS program.

The **DxsMeteringPulseEnable** function generates a single metering pulse. The generation of a pulse series is not supported. The DXS_EVENT_METERING_END event is generated at the end of a pulse playout. The **DxsMeteringPulseEnable** function, in conjunction with the DXS_EVENT_METERING_END event, generates a pulse series.

The line must be in either DXS_LINE_FEED_ACTIVE or DXS_LINE_FEED_ACTIVE_REVPOL mode for metering pulse playout to be started, otherwise an error code is returned.

Example of TTX Pulse Generation

Download BBD, to be performed once at system initialization. BBD download configures the TTX metering pulse parameters (e.g. frequency):

```
/* Download BBD */
DxsBBD(dev, line, pBbd, nBytes);
```

Set the line to ACTIVE mode and generate one TTX metering pulse:

```
/* Set active line mode */
DxsLineModeSet(dev, line, DXS_LINE_FEED_ACTIVE);
/* Generate the 1st pulse */
DxsMeteringPulseEnable(dev, line);
/* The DXS_EVENT_METERING_END event is generated
   when the pulse playout is complete*/
```

2.7 Continuous Measurement

The driver provides the **DxsContMeasurementResultGet** function to read the Continuous Measurement results at any time. The returned measurements are the output voltage of the DC regulation, the actual line DC current, the last ring burst current, and the last ring burst voltage. The function returns the measurements in the completed **DXS_ContMeasurementResult_t** structure.

Example of Continuous Measurement

To read the Continuous Measurement results:

```
DXS_ContMeasurementResult_t cm;

memset(&cm, 0, sizeof(cm));
DxsContMeasurementResultGet(dev, line, &cm);
```

2.8 GR-909 Line Testing

The driver provides an API for GR-909 line testing. Setting test limit values via the **DxsGr909LimitsSet** function is optional. Currently programmed limit values are read using the **DxsGr909LimitsGet** function.

The GR-909 line testing is initiated by setting the line operating mode to DXS_LINE_FEED_GR909 using the **DxsLineModeSet** function. The completion of GR-909 line testing is signaled by the event DXS_EVENT_NLT_END sent by the driver to the application. Then the application is able to call the **DxsGr909ResultGet** function to obtain the line testing results.

Attention: The line operating mode must be set to DXS_LINE_FEED_DISABLED before GR-909 line testing is executed.

Example of GR-909 Line Testing

Set GR-909 limit values (optional):

```
/* Set GR-909 limits */
DXS_GR909Config_t Gr909Limits;

memset(&Gr909Limits, 0, sizeof(Gr909Limits));
Gr909Limits.settle_time = 0;
Gr909Limits.HPT_W2G_AC_Lim = 50000;
Gr909Limits.HPT_W2W_AC_Lim = 50000;
Gr909Limits.HPT_W2G_DC_Lim = 135000;
Gr909Limits.HPT_W2W_DC_Lim = 135000;
Gr909Limits.FEMF_W2G_AC_Lim = 10000;
Gr909Limits.FEMF_W2W_AC_Lim = 10000;
Gr909Limits.FEMF_W2G_DC_Lim = 6000;
Gr909Limits.FEMF_W2W_DC_Lim = 6000;
Gr909Limits.RFT_Res_Lim = 150000;
Gr909Limits.ROH_Lin_Lim = 15;
Gr909Limits.RIT_Low_Lim = 1392;
Gr909Limits.RIT_High_Lim = 39600;
```

Read back GR-909 limit values (optional):

```
/* Read GR-909 limits */
DXS_GR909Config_t Gr909Limits;
```

```
memset(&Gr909Limits, 0, sizeof(DXS_GR909Config_t));
DxsGr909LimitsGet(dev, line, &Gr909Limits);
```

Start GR-909 testing, then read the results:

```
/* Start GR-909 and read the results */
DXS_GR909Result_t Gr909Result;

memset(&Gr909Result, 0, sizeof(DXS_GR909Result_t));
/* Set line mode to DISABLED */
DxsLineModeSet(dev, line, DXS_LINE_FEED_DISABLED);
/* Start GR-909 by setting the corresponding line mode */
DxsLineModeSet(dev, line, DXS_LINE_FEED_GR909);

/* Wait for DXS_EVENT_NLT_END event */

/* Read results */
DxsGr909ResultGet(dev, line, &Gr909Result);
```

2.9 Capacitance Measurement

The driver provides an API to measure line capacitances in nF. It measures the Ring to Ground, Tip to Ground and Tip to Ring capacitances. These capacitance values allow evaluation of the tip/ring line condition. The capacitance between tip/ring is used to detect the ringing circuitry of a connected POTS phone. The capacitance to ground is used to evaluate the wiring and a possible mismatch between tip and ring.

Capacitance measurement is initiated by setting the line operating mode to DXS_LINE_FEED_CAP_MEAS using the **DxsLineModeSet** function. Capacitance measurement completion is signaled by the event DXS_EVENT_NLT_END sent by the driver to the application.Then the application is able to read the capacitance measurement results using the **DxsCapMeasurementResultGet** function.

Attention: The line operating mode must be set to DXS_LINE_FEED_DISABLED before capacitance measurement is switched on.

Example of Capacitance Measurement

Start the capacitance measurement and read the results:

```
/* Capacitance measurement */
DXS_Capacitance_t cap;

memset(&cap, 0, sizeof(cap));
/* Set line mode to DISABLED */
DxsLineModeSet(dev, line, DXS_LINE_FEED_DISABLED);
/* Start capacitance measurement */
DxsLineModeSet(dev, line, DXS_LINE_FEED_CAP_MEAS);

/* Wait for DXS_EVENT_NLT_END event */

/* Read capacitance measurement results */
DxsCapMeasurementResultGet(dev, line, &cap);
```

2.10 Analog Line Calibration

The driver provides an API for analog line calibration. The calibration is initiated by setting the line operating mode to DXS_LINE_FEED_CALIBRATE using the **DxsLineModeSet** function. Completion of the calibration is signaled with the event DXS_EVENT_CALIBRATION_END generated by the driver. Then the application is able to read the calibration results via the **DxsCalibrationGet** function.

Writing previously stored calibration data to the device is possible at any time using the **DxsCalibrationSet** function. Calibration results are populated as opaque data within the **DXS_Calibration_t** structure.

Attention: The line operating mode must be set to DXS_LINE_FEED_DISABLED before analog line calibration is switched on.

Example of Analog Line Calibration

Start the calibration and read the results:

```
/* Analog line calibration */
DXS_Calibration_t calib;

memset(&calib, 0, sizeof(calib));
/* Set line mode to DISABLED */
DxsLineModeSet(dev, line, DXS_LINE_FEED_DISABLED);
/* Start analog line calibration */
DxsLineModeSet(dev, line, DXS_LINE_FEED_CALIBRATE);

/* Wait for DXS_EVENT_CALIBRATION_END event */

/* Read calibration results */
DxsCalibrationGet(dev, line, &calib);
```

Program the previously stored calibration data:

```
/* Program calibration data */
DXS_Calibration_t calib;

memset(&calib, 0, sizeof(calib));
calib.data[0] = 0xb6de3;
calib.data[1] = 0x5d0000;
calib.crc8ccitt = 0x2b;
DxsCalibrationSet(dev, line, &calib);
```

2.11 Open Loop Calibration

The device and the external board components (capacitors and resistors) have different manufacturing tolerances.

Each analog line has different tolerances that influence the line testing measurement results. The **DxsOpenLoopCalibration** function allows for open-loop calibration of an analog line. The calibration process returns conductance and capacitance measurements for the device and the external board components. This calibration process is executed once per analog line channel and per board. To achieve more precise measurement values, it is possible to configure a multiple number of loops for the calibration process. The results are stored by the customer application in a non-volatile memory location. They are to be used for future post-processing of actual field test measurements.

The customer application provides the results of the calibration process to the driver during system startup via the **DxsOpenLoopConfigSet** function before a line testing operation is executed. The driver then uses these

calibration values to correct the results of every new line test measurement. The **DxsOpenLoopConfigGet** function allows the current Open Loop Calibration configuration to be read.

It is also possible to execute line testing measurements without any calibration values. In this case, the driver functions are unable to correct the manufacturing tolerances.

Example of Open Loop Calibration

Run Open Loop Calibration and store the results:

```
DXS_OLCalibration_t OLCalibration, NonVolatil;

DxsOpenLoopCalibration(dev, line, loops, &OLCalibration);
memcopy(&NonVolatil, &OLCalibration, sizeof(DXS_OLCalibration_t));
```

Program the previously stored Open Loop Calibration results:

```
DXS_OLCalibration_t OLCalibration, NonVolatil;

OLCalibration.GRG = NonVolatil.GRG;
OLCalibration.GTG = NonVolatil.GTG;
OLCalibration.GTR = NonVolatil.GTR;
OLCalibration.CRG = NonVolatil.CRG;
OLCalibration.CTG = NonVolatil.CTG;
OLCalibration.CTR = NonVolatil.CTR;
DxsOpenLoopConfigSet(dev, line, &OLCalibration);
```

Read the Open Loop Calibration results:

```
DXS_OLCalibration_t OLCalibration;

DxsOpenLoopConfigGet(dev, line, loops, &OLCalibration);
```

2.12 Message Waiting Lamp

The driver provides an API to change the parameters affecting the sawtooth-shaped voltage signal for the Message Waiting Lamp, the **DxsMwlConfig** function.

Example of Configuring a Voltage Signal for the Message Waiting Lamp

Set the new parameters:

Voltage = 100 V, Hook Threshold = 15 mA, Waiting Slope = 200 V/s, On-time (lamp ON) = 1000 ms,

Off-time (lamp OFF) = 1000 ms:

```
DxsMwlConfig(dev, line, 100, 15, 200, 1000, 1000);
```

2.13 Hook State Machine

The driver has a built-in hook state machine that allows hook, flash-hook and pulse digit events to be discriminated. The default timing settings of the hook state machine are suitable for most telephones in the world. However, for additional flexibility the driver allows the timing settings to be adjusted, using the **DxsHookValTimeSet** function, so that it is possible to match the hook discriminator timing to the timing of a specific telephone.

The **DxsHookValTimeSet** function addresses the following parameters:

- Minimum off-hook validation time
- Minimum on-hook validation time
- Flash-hook validation interval
- Make validation interval
- Break validation interval
- Minimum interdigit validation time
- Minimum flash hold-off time

Example of Hook Validation Timer Setting

Apply the new hook validation timer configuration:

```
DXS_HookValTime_t hvt = {0};

/* set flash break between 80 and 400 ms */
hvt.nType = DXS_HOOK_VT_HOOKFLASH_TIME;
hvt.nMinTime = 80;
hvt.nMaxTime = 400;
DxsHookValTimeSet(dev, line, &hvt);

/* set flash hold-off 200 ms */
hvt.nType = DXS_HOOK_VT_FLASH_HOLDOFF_TIME;
hvt.nMinTime = 200;
hvt.nMaxTime = 0;
DxsHookValTimeSet(dev, line, &hvt);

/* set break interval between 30 and 80 ms */
hvt.nType = DXS_HOOK_VT_DIGITHIGH_TIME;
hvt.nMinTime = 30;
hvt.nMaxTime = 80;
DxsHookValTimeSet(dev, line, &hvt);
```

```
/* set make interval between 30 and 80 ms */
hvt.nType = DXS_HOOK_VT_DIGITLOW_TIME;
hvt.nMinTime = 30;
hvt.nMaxTime = 80;
DxsHookValTimeSet(dev, line, &hvt);

/* set minimum off-hook time 40 ms */
hvt.nType = DXS_HOOK_VT_HOOKOFF_TIME;
hvt.nMinTime = 40;
hvt.nMaxTime = 0;
DxsHookValTimeSet(dev, line, &hvt);

/* set minimum on-hook time 400 ms */
hvt.nType = DXS_HOOK_VT_HOOKON_TIME;
hvt.nMinTime = 400;
hvt.nMaxTime = 0;
DxsHookValTimeSet(dev, line, &hvt);

/* set minimum interdigit time 300 ms */
hvt.nType = DXS_HOOK_VT_INTERDIGIT_TIME;
hvt.nMinTime = 300;
hvt.nMaxTime = 0;
DxsHookValTimeSet(dev, line, &hvt);
```

2.14 AC Level Meter

The driver provides an API to initiate the AC Level Meter measurements and obtain the measurement results. Four measurement types are supported:

- Frequency Response measurement
- Transhybrid measurement
- Gain Tracking measurement
- Signal to Noise Ratio measurement

The measurement types are enumerated with the **DXS_ACLM_Measurement_t** type.

Attention: For all AC Level Meter measurements, the BBD file corresponding to the analog line termination, e.g. 600 Ω, must be loaded.

The **DxsAcLmMeasurementStart** function starts one of the four listed measurements on a given line. The function does not exit until the selected measurement is completed successfully or an error occurs during the measurement. After successful exit from **DxsAcLmMeasurementStart**, the measurement result is available and readable. In the case of an error, the measurement result is not available and is not readable. When the functions **DxsAcLmMeasResNumGet**, **DxsAcLmMeasResArgGet**, or **DxsAcLmMeasResValGet** are called after an unsuccessful measurement start, they return an error code DXS_statusAclmResultsNotAvail and zero return value.

The AC Level Meter measurement results are returned in a table of N rows and two columns (see **Figure 2**). N represents the number of measurement points. This table is available for reading after successful measurement is completed for each line individually.

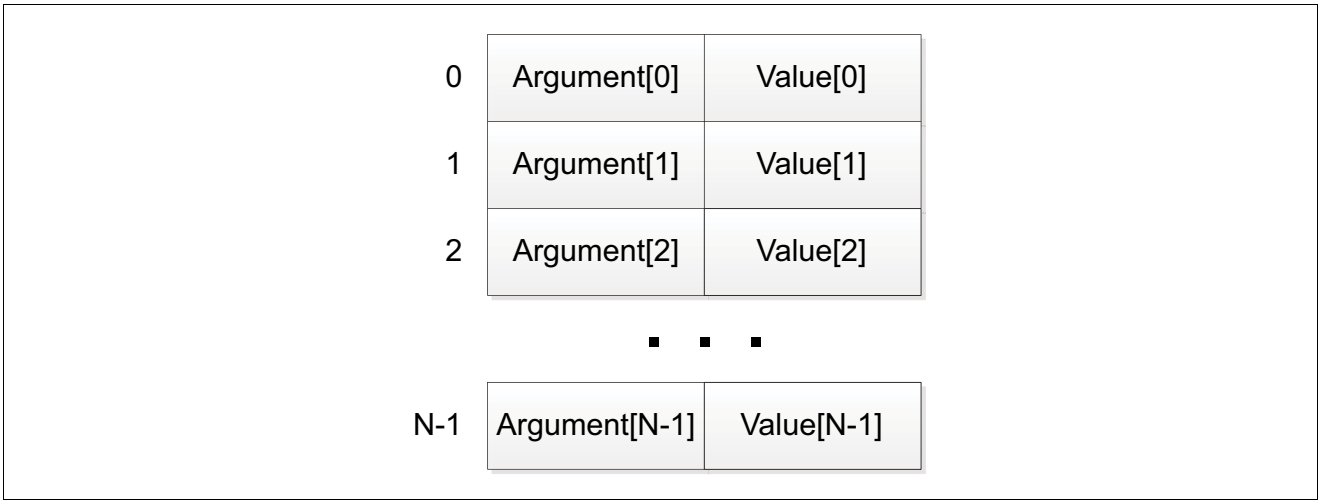

Figure 2 AC Level Meter Measurement Result Representation

The Argument field of a measurement point is an integer value that has a different meaning depending on the particular measurement:

- For Frequency Response and Transhybrid measurements, it is a frequency in Hz.
- For Gain Tracking and Signal to Noise Ratio measurements, it is the input level in dBm0, with a resolution of 0.1 dBm0. For example, '123' represents 12.3 dBm0.

The Value field of a measurement point is an integer representing the test result in dB, with a resolution of 0.1 dB.

After successful measurement completion, use the **DxsAcLmMeasResNumGet** function to obtain the number of measurement points (N). Based on the value of N, it is possible to read the Argument and Value fields separately for each measurement point using the functions **DxsAcLmMeasResArgGet** and **DxsAcLmMeasResValGet**, respectively.

Example of AC Level Meter Frequency Response Measurement

The code example below starts the ACLM Frequency Response measurement for device `dev` and channel `line` and displays the measurement results on the console. It is possible to use the measurement results as source data for a plotting program. Before calling the **DxsAcLmMeasurementStart** function, the line must be terminated with the load that corresponds to the BBD file used.

```
int32_t N = 0, arg = 0, val = 0, i, ret;

/* Start ACLM Frequency Response measurement for device "dev", channel "line" */
ret = DxsAcLmMeasurementStart(dev, line, DXS_ACLM_FR);
if (ret == DXS_statusOk)
{
   ret = DxsAcLmMeasResNumGet(dev, line, DXS_ACLM_FR, &N);
   if (ret == DXS_statusOk)
   {
      printf("Freq,[Hz]\tLevel,[dB]\n");
      for (i=0; i<N; i++)
      {
         DxsAcLmMeasResArgGet(dev, line, DXS_ACLM_FR, i, &arg);
         DxsAcLmMeasResValGet(dev, line, DXS_ACLM_FR, i, &val);
         printf("%d\t\t%.1f\n", arg, (float)val/10.0f);
      }
   }
}
```

Example output of the above code fragment:

```
Freq,[Hz]       Level,[dB]
100             -14.8
200             -1.1
300             -0.2
400             -0.1
500             -0.1
600             -0.1
700             -0.1
800             -0.1
900             0.0
1000            0.0
1100            0.0
1200            0.0
1300            0.0
1400            0.0
1500            0.0
1600            0.0
1700            0.0
1800            0.0
1900            0.0
2000            0.0
2100            0.0
2200            0.0
2300            0.0
2400            0.0
2500            0.0
2600            0.0
2700            0.0
2800            0.0
2900            0.0
3000            0.0
3100            0.0
3200            0.0
3300            -0.2
3400            -0.8
3500            -2.6
```

2.15 GPIO Pin Control

The driver provides an API to control the four DXS General Purpose IO (GPIO) lines. The **DxsGpioPortConfig** function sets up the direction of each line (input or output), and enables or disables the internal pull-up resistor of the line.

The **DxsGpioWrite** function sets the output of a given GPIO line. It is mandatory to configure the GPIO line direction for output before calling **DxsGpioWrite**.

The **DxsGpioRead** function gets the input value from a given GPIO line. It is mandatory to configure the GPIO line direction for input before calling **DxsGpioRead**.

Example of GPIO Pin Control

Configure GPIO0, GPIO1 and GPIO2 for input, GPIO3 for output. Enable pull-up resistors for all lines. Read the line status of the GPIO0-GPIO2 lines, then write 0 to the GPIO3 line.

```
DXS_GpioConfig_t gpio_ctrl;
uint8_t gpio_port_status = 0, tmp;

/* Configure GPIO0-GPIO2 for input, GPIO3 for output. Enable PU for all. */
memset(&gpio_ctrl, 0, sizeof(gpio_ctrl));
gpio_ctrl.gpio0_PU = 1;
gpio_ctrl.gpio1_PU = 1;
gpio_ctrl.gpio2_PU = 1;
gpio_ctrl.gpio3_PU = 1;
gpio_ctrl.gpio0_DIR = 1;
gpio_ctrl.gpio1_DIR = 1;
gpio_ctrl.gpio2_DIR = 1;
gpio_ctrl.gpio3_DIR = 0;
DxsGpioPortConfig(dev, &gpio_ctrl);
/* Read GPIO0-GPIO2 lines and store result in gpio_port_status */
DxsGpioRead(dev, 2, &tmp);
gpio_port_status |= (tmp << 2);
DxsGpioRead(dev, 1, &tmp);
gpio_port_status |= (tmp << 1);
DxsGpioRead(dev, 0, &tmp);
gpio_port_status |= tmp;
/* Write 0 to GPIO3 */
DxsGpioWrite(dev, 3, 0);
```

3 Module Reference

This chapter references all elements by modules.

3.1 DXS API Device Driver Interface Reference

This section describes the functional interfaces, also called DXS API device driver interfaces.

Table 3 Module Overview

Name	Description
Initialization	Device initialization API
Configuration	Device configuration API
Control	Line mode control API
Debug	Debug low level API
Tone Generation	Tone generator control API
Caller ID Generation	Caller ID generator control API
Ringing service	Ring cadence configuration API
FSK Generation	FSK generator control API
Tone Detection	Universal Tone Detector control API
DTMF Detection	DTMF detector control API
TTX Metering	Teletax Metering pulse generation control API
Continuous Measurement	Reading of Continuous Measurement API
GR-909 Network Line Testing	GR-909 Line Testing configuration and control API
Calibration	Analog Line Calibration and Open Loop Calibration control API
Hook State Machine	Hook State Machine configuration API
Capacitance Measurement	Capacitance Measurement API
AC Level Meter Measurement	AC Level Meter measurement control API
GPIO Pin Control Interface	GPIO/IO pin handling API
Clock Failure Handling	PCM clock failure handling
Shared DC/DC converter	Shared DC/DC converter configuration API
Event Reporting	Event reporting API

3.1.1 Initialization

Table 4 lists the device initialization API functions.

Table 4 Initialization Function Overview

Name	Description
DxsBBD	This function downloads the DXS BBD (Block Based Download) file.
DxsCapabilitiesRead	This function reads DXS Series device capabilities.
DxsExit	This function closes the DXS SPI access and performs deinitialization of driver internal objects.
DxsInit	This function initializes the DXS Series device.
DxsPatch	This function downloads DXS PRAM patch.

Table 4 Initialization Function Overview (cont'd)

Name	Description
DxsSleepDisable	This function disables the DXS sleep mode.
DxsSleepEnable	This function enables the DXS sleep mode to achieve the minimum power consumption, provided that both the channels are either in STANDBY or DISABLED mode (see DXS_LINE_MODE_t)
DxsVersionRead	This function reads the DXS Series device version.

3.1.1.1 DxsInit

Description

This function initializes the DXS Series device.

It performs these operations:

- Opens the low level access interface for the DXS Series device.
- Initializes the driver internal message queues, memory pool, device and channel resources.
- Initializes the DXS outbox data handling thread.
- In the case of polling, starts polling.

Prototype

```
int32_t DxsInit (
        uint8_t dxs,
        uint8_t chipSelect,
        int32_t irqNumber,
        uint8_t dcdcType,
        uint8_t acc_mode );
```

Parameters

Data Type	Name	Description	Dir
uint8_t	dxs	Device number.	-
uint8_t	chipSelect	Chip Select number, system specific	-
int32_t	irqNumber	IRQ line number, system specific, (-1) for polling mode	-
uint8_t	dcdcType	DC/DC converter handled by the DXS_DCDC_Var_t type	-
uint8_t	acc_mode	Access mode handled by the DXS_ACCESS_MODE_t type	-

Return Values

Data Type	Description
int32_t	Return value according to DXS_status_t.

3.1.1.2 DxsExit

Description

This function closes the DXS SPI access and deinitializes the driver internal objects.

Prototype

```
int32_t DxsExit (
        uint8_t dxs );
```

Parameters

Data Type	Name	Description	Dir
uint8_t	dxs	Device number.	-

Return Values

Data Type	Description
int32_t	DXS_statusOk

3.1.1.3 DxsBBD

Description

This function downloads the DXS Block Based Download (BBD) file.

The BBD file is a MaxLinear proprietary format file intended for adaptation to hardware design and country specific requirements. The contents of the BBD file must be calculated using the XTCOS program and are assumed to be in a byte array here.

Prototype

```
int32_t DxsBBD (
        uint8_t dxs,
        uint8_t line,
        uint8_t * pBbd,
        uint32_t nBytes );
```

Parameters

Data Type	Name	Description	Dir
uint8_t	dxs	Device number.	-
uint8_t	line	Line number.	-
uint8_t *	pBbd	Pointer to the beginning of BBD data	-
uint32_t	nBytes	BBD data size in bytes	-

Return Values

Data Type	Description
int32_t	Return value according to **DXS_status_t**.

3.1.1.4 DxsPatch

Description

This function downloads the DXS PRAM patch.

Prototype

```
int32_t DxsPatch (
        uint8_t dxs,
        uint8_t * pPatch,
        uint32_t nBytes );
```

Parameters

Data Type	Name	Description	Dir
uint8_t	dxs	Device number.	-
uint8_t *	pPatch	Pointer to PRAM patch data	-
uint32_t	nBytes	PRAM patch data size in bytes	-

Return Values

Data Type	Description
int32_t	Return value according to **DXS_status_t**.

3.1.1.5 DxsCapabilitiesRead

Description

This function reads the DXS Series device capabilities.

When the return code indicates success, the returned **DXS_Caps_t** type structure is filled with data.

Prototype

```
int32_t DxsCapabilitiesRead (
        uint8_t dxs,
        DXS_Caps_t * pCap );
```

Parameters

Data Type	Name	Description	Dir
uint8_t	dxs	Device number.	-
DXS_Caps_t *	pCap	Pointer to DXS_Caps_t structure	-

Return Values

Data Type	Description
int32_t	Return value according to DXS_status_t.

3.1.1.6 DxsVersionRead

Description

This function reads the DXS Series device version.

When the return code indicates success, the returned DXS_Version_t type structure is filled with data.

Prototype

```
int32_t DxsVersionRead (
        uint8_t dxs,
        DXS_Version_t * pVersion );
```

Parameters

Data Type	Name	Description	Dir
uint8_t	dxs	Device number.	-
DXS_Version_t *	pVersion	Pointer to DXS_Version_t structure	-

Return Values

Data Type	Description
int32_t	Return value according to DXS_status_t.

3.1.1.7 DxsSleepEnable

Description

This function enables the DXS sleep mode to achieve the minimum power consumption, provided that both the channels are in either STANDBY or DISABLED mode (see DXS_LINE_MODE_t).

Prototype

```
int32_t DxsSleepEnable (
        uint8_t dxs );
```

Parameters

Data Type	Name	Description	Dir
uint8_t	dxs	Device number.	-

Return Values

Data Type	Description
int32_t	Return value according to **DXS_status_t**.

3.1.1.8 DxsSleepDisable

Description

This function disables the DXS sleep mode.

Prototype

```
int32_t DxsSleepDisable (
        uint8_t dxs );
```

Parameters

Data Type	Name	Description	Dir
uint8_t	dxs	Device number.	-

Return Values

Data Type	Description
int32_t	Return value according to **DXS_status_t**.

3.1.2 Configuration

Table 5 lists the device configuration API functions.

Table 5 Configuration Function Overview

Name	Description
DxsGainsGet	This function gets the relative digital gain values for a given DXS line.
DxsGainsSet	This function configures the relative digital gain values for a given DXS line.
DxsMwlConfig	This function configures the Message Waiting Lamp of DXS line.
DxsPcmChannelDisable	This function disables the PCM channel (timeslot) for a given DXS line.
DxsPcmChannelEnable	This function configures and enables the PCM channel for a given DXS line.
DxsPcmChannelMute	This function mutes the PCM channel in the RX direction (D->A) for a given DXS line.
DxsPcmChannelUnmute	This function unmutes the PCM channel in the RX direction (D->A) for a given DXS line.

Table 5 Configuration Function Overview (cont'd)

Name	Description
DxsPcmInterfaceEnable	This function configures and enables the PCM interface of a given DXS Series device.
DxsRingConfig	This function configures the ringing of the DXS line.

3.1.2.1 DxsPcmInterfaceEnable

Description

This function configures and enables the PCM interface of a given DXS Series device.

Prototype

```
int32_t DxsPcmInterfaceEnable (
        uint8_t dxs,
        uint8_t xoff,
        uint8_t dbl,
        uint8_t xs,
        uint8_t rs,
        uint8_t drv0,
        uint8_t sh,
        uint8_t roff );
```

Parameters

Data Type	Name	Description	Dir
uint8_t	dxs	Device number.	-
uint8_t	xoff	Transmit Bit Offset: 0_D **No Offset** 1_D **1 Period** ... 7_D **7 Periods**	-
uint8_t	dbl	Double Bit Clock: 0=Single, 1=Double. 0_D **Single** 1_D **Double**	-
uint8_t	xs	Transmit Slope: 0_D **Rising Edge** 1_D **Falling Edge**	-
uint8_t	rs	Receive Slope: 0_D **Rising Edge** 1_D **Falling Edge**	-
uint8_t	drv0	Bit 0 Drive Length: 0_D **Entire Period** 1_D **First Half**	-

Data Type	Name	Description	Dir
uint8_t	sh	Shift Control: 0_D **No Shift** 1_D **Shift**	-
uint8_t	roff	Receive Bit Offset: 0_D **No Offset** 1_D **1 Period** ... 7_D **7 Periods**	-

Return Values

Data Type	Description
int32_t	Return value according to **DXS_status_t**.

3.1.2.2 DxsPcmChannelEnable

Description

This function configures and enables the PCM channel for a given DXS line.

Prototype

```
int32_t DxsPcmChannelEnable (
        uint8_t dxs,
        uint8_t line,
        uint8_t wb,
        uint8_t wbtsc,
        uint8_t codec,
        uint8_t xts,
        uint8_t rts );
```

Parameters

Data Type	Name	Description	Dir
uint8_t	dxs	Device number.	-
uint8_t	line	Line number.	-
uint8_t	wb	0_B **8 kHz** 1_B **16 kHz** Wideband	-
uint8_t	wbtsc	Wideband PCM Timeslot Configuration: 0_B **Consecutive** 1_B **16 kHz Spacing**	-
uint8_t	codec	Coder: 0_D **Linear** 2_D **G711 A-Law** 3_D **G711 µ-Law**	-

Data Type	Name	Description	Dir
uint8_t	xts	Transmit Highway Timeslot.	-
uint8_t	rts	Receive Highway Timeslot.	-

Return Values

Data Type	Description
int32_t	Return value according to **DXS_status_t**.

3.1.2.3 DxsPcmChannelDisable

Description

This function disables the PCM channel (timeslot) for a given DXS line.

Prototype

```
int32_t DxsPcmChannelDisable (
        uint8_t dxs,
        uint8_t line );
```

Parameters

Data Type	Name	Description	Dir
uint8_t	dxs	Device number.	-
uint8_t	line	Line number.	-

Return Values

Data Type	Description
int32_t	Return value according to **DXS_status_t**.

3.1.2.4 DxsPcmChannelMute

Description

This function mutes the PCM channel in the RX direction (D->A) for a given DXS line.

A possible use of this function for tone generators is to avoid disturbances from the PCM interface.

Prototype

```
int32_t DxsPcmChannelMute (
        uint8_t dxs,
        uint8_t line );
```

Parameters

Data Type	Name	Description	Dir
uint8_t	dxs	Device number.	-
uint8_t	line	Line number.	-

Return Values

Data Type	Description
int32_t	Return value according to **DXS_status_t**.

3.1.2.5 DxsPcmChannelUnmute

Description

This function unmutes the PCM channel in the RX direction (D->A) for a given DXS line.

This function does the opposite of **DxsPcmChannelMute**.

Prototype

```
int32_t DxsPcmChannelUnmute (
        uint8_t dxs,
        uint8_t line );
```

Parameters

Data Type	Name	Description	Dir
uint8_t	dxs	Device number.	-
uint8_t	line	Line number.	-

Return Values

Data Type	Description
int32_t	Return value according to **DXS_status_t**.

3.1.2.6 DxsRingConfig

Description

This function configures the ringing of the DXS line.

The ring configuration for a given country must be set by the BBD file provided with a system package and downloaded using the **DxsBBD** API. The **DxsRingConfig** function allows modification of some of the ring configuration parameters after the BBD download.

Prototype

```
int32_t DxsRingConfig (
```

```
        uint8_t dxs,
        uint8_t line,
        uint8_t waveForm,
        uint8_t crestFactor,
        uint8_t frequency,
        uint16_t amplitude,
        uint16_t dcOffset );
```

Parameters

Data Type	Name	Description	Dir
uint8_t	dxs	Device number.	-
uint8_t	line	Line number.	-
uint8_t	waveForm	Waveform of ringing signal: • 0: Sinewave. • 1: Trapez.	-
uint8_t	crestFactor	Ring Crest Factor. 0_D **1.0** Ring with a rectangular waveform. 1_D **1.10** Ring with trapezoidal waveform. 2_D **1.21** Ring with trapezoidal waveform. 3_D **1.31** Ring with trapezoidal waveform. 4_D **1.42** Ring with trapezoidal waveform, near sinewave. 5_D **1.52** Ring with trapezoidal waveform. 6_D **1.63** Ring with trapezoidal waveform. 7_D **1.73** Ring with triangular waveform.	-
uint8_t	frequency	Ringing Frequency, [Hz].	-
uint16_t	amplitude	Ringing Amplitude, [Vpeak].	-
uint16_t	dcOffset	Ringing DC Offset, [V].	-

Return Values

Data Type	Description
int32_t	Return value according to **DXS_status_t**.

3.1.2.7 DxsMwlConfig

Description

This function configures the Message Waiting Lamp of the DXS line.

The Message Waiting Lamp configuration must be set by the BBD file provided with a system package and downloaded using the **DxsBBD** API. The **DxsMwlConfig** function allows modification of the Message Waiting Lamp configuration parameters after the BBD download.

Prototype

```
int32_t DxsMwlConfig (
        uint8_t dxs,
        uint8_t line,
```

```
        uint8_t voltage,
        uint8_t thresh,
        uint16_t slope,
        uint16_t onTime,
        uint16_t offTime );
```

Parameters

Data Type	Name	Description	Dir
uint8_t	dxs	Device number.	-
uint8_t	line	Line number.	-
uint8_t	voltage	Message Waiting Lamp Voltage, [V], [0..144]	-
uint8_t	thresh	Message Waiting Hook Threshold, [mA], [0..100]	-
uint16_t	slope	Message Waiting Slope, [V/s], [100..1000]	-
uint16_t	onTime	Message Waiting On-time (Lamp ON), [ms], [0..6375]	-
uint16_t	offTime	Message Waiting Off-time (Lamp OFF), [ms], [0..6375]	-

Return Values

Data Type	Description
int32_t	Return value according to **DXS_status_t**.

3.1.2.8 DxsGainsSet

Description

This function configures the relative digital gain values for a given DXS line.

Prototype

```
int32_t DxsGainsSet (
        uint8_t dxs,
        uint8_t line,
        int16_t Txg,
        int16_t Rxg );
```

Parameters

Data Type	Name	Description	Dir
uint8_t	dxs	Device number.	-
uint8_t	line	Line number.	-
int16_t	Txg	TX relative level setting in units of 0.1 dB: [-100..30]	-
int16_t	Rxg	RX relative level setting in units of 0.1 dB: [-100..0]	-

Return Values

Data Type	Description
int32_t	Return value according to **DXS_status_t**.

3.1.2.9 DxsGainsGet

Description

This function gets the relative digital gain values for a given DXS line.

Prototype

```
int32_t DxsGainsGet (
        uint8_t dxs,
        uint8_t line,
        int16_t * pTxg,
        int16_t * pRxg );
```

Parameters

Data Type	Name	Description	Dir
uint8_t	dxs	Device number.	-
uint8_t	line	Line number.	-
int16_t *	pTxg	Output parameter - TX relative Level Setting, units of 0.1 dB.	-
int16_t *	pRxg	Output parameter - RX relative Level Setting, units of 0.1 dB.	-

Return Values

Data Type	Description
int32_t	Return value according to **DXS_status_t**.

3.1.3 Control

Table 6 lists the line mode control API functions.

Table 6 Control Function Overview

Name	Description
DxsLineModeGet	This function gets the current line feeding mode of a given DXS line.
DxsLineModeSet	This function sets the new line feeding mode for a given DXS line.

3.1.3.1 DxsLineModeSet

Description

This function sets the new line feeding mode for a given DXS line.

Possible line modes are enumerated by the **DXS_LINE_MODE_t** type. Not all line mode transitions are permitted. This function returns an error when a line feeding mode transition is not permitted and the current line feeding mode does not change.

Prototype

```
int32_t DxsLineModeSet (
        uint8_t dxs,
        uint8_t line,
        int mode );
```

Parameters

Data Type	Name	Description	Dir
uint8_t	dxs	Device number.	-
uint8_t	line	Line number.	-
int	mode	New line mode according to **DXS_LINE_MODE_t** type.	-

Return Values

Data Type	Description
int32_t	Return value according to **DXS_status_t**.

3.1.3.2 DxsLineModeGet

Description

This function gets the current line feeding mode of a given DXS line.

The output parameter is an enumerator type **DXS_LINE_MODE_t**.

Prototype

```
int32_t DxsLineModeGet (
        uint8_t dxs,
        uint8_t line,
        int * pMode );
```

Parameters

Data Type	Name	Description	Dir
uint8_t	dxs	Device number.	-

Data Type	Name	Description	Dir
uint8_t	line	Line number.	-
int *	pMode	Output parameter: returned value according to **DXS_LINE_MODE_t** type.	-

Return Values

Data Type	Description
int32_t	Return value according to **DXS_status_t**.

3.1.4 Debug

Table 7 lists the debug low level API functions.

Table 7 Debug Function Overview

Name	Description
DxsDebugCmdRead	This function is used for the DXS firmware command read.
DxsDebugCmdWrite	This function is used for the DXS firmware command write.
DxsDebugRegRead16	This function is used for a 16-bit read from the DXS internal register.
DxsDebugRegRead32	This function is used for a 32-bit read from the DXS internal register.
DxsDebugRegWrite16	This function is used for a 16-bit write to the DXS internal register.
DxsDebugRegWrite32	This function is used for a 32-bit write to the DXS internal register.

3.1.4.1 DxsDebugRegWrite16

Description

This function is used for a 16-bit write to the DXS internal register.

Prototype

```
int32_t DxsDebugRegWrite16 (
        uint8_t dxs,
        uint8_t offset,
        uint16_t value );
```

Parameters

Data Type	Name	Description	Dir
uint8_t	dxs	Device number.	-
uint8_t	offset	DXS register offset.	-
uint16_t	value	16-bit value to write.	-

Return Values

Data Type	Description
int32_t	Return value according to **DXS_status_t**.

3.1.4.2 DxsDebugRegRead16

Description

This function is used for a 16-bit read from the DXS internal register.

Prototype

```
int32_t DxsDebugRegRead16 (
        uint8_t dxs,
        uint8_t offset,
        uint16_t * pValue );
```

Parameters

Data Type	Name	Description	Dir
uint8_t	dxs	Device number.	-
uint8_t	offset	DXS register offset.	-
uint16_t *	pValue	Output parameter: read 16-bit value.	-

Return Values

Data Type	Description
int32_t	Return value according to **DXS_status_t**.

3.1.4.3 DxsDebugRegWrite32

Description

This function is used for a 32-bit write to the DXS internal register.

Prototype

```
int32_t DxsDebugRegWrite32 (
        uint8_t dxs,
        uint8_t offset,
        uint32_t value );
```

Parameters

Data Type	Name	Description	Dir
uint8_t	dxs	Device number.	-
uint8_t	offset	DXS register offset.	-
uint32_t	value	32-bit value to write.	-

Return Values

Data Type	Description
int32_t	Return value according to **DXS_status_t**.

3.1.4.4 DxsDebugRegRead32

Description

This function is used for a 32-bit read from the DXS internal register.

Prototype

```
int32_t DxsDebugRegRead32 (
        uint8_t dxs,
        uint8_t offset,
        uint32_t * pValue );
```

Parameters

Data Type	Name	Description	Dir
uint8_t	dxs	Device number.	-
uint8_t	offset	DXS register offset.	-
uint32_t *	pValue	Output parameter: read 32-bit value.	-

Return Values

Data Type	Description
int32_t	Return value according to **DXS_status_t**.

3.1.4.5 DxsDebugCmdWrite

Description

This function is used for the DXS firmware command write.

Prototype

```
int32_t DxsDebugCmdWrite (
        uint8_t dxs,
```

```
        uint8_t length,
        uint32_t * pCmd );
```

Parameters

Data Type	Name	Description	Dir
uint8_t	dxs	Device number.	-
uint8_t	length	The length of the buffer given in pCmd in bytes.	-
uint32_t *	pCmd	Pointer to buffer with command.	-

Return Values

Data Type	Description
int32_t	Return value according to **DXS_status_t**.

3.1.4.6 DxsDebugCmdRead

Description

This function is used for the DXS firmware command read.

Prototype

```
int32_t DxsDebugCmdRead (
        uint8_t dxs,
        uint32_t hdr,
        uint8_t * pLength,
        uint32_t * pData );
```

Parameters

Data Type	Name	Description	Dir
uint8_t	dxs	Device number.	-
uint32_t	hdr	Command header.	-
uint8_t *	pLength	On input, the length of the buffer given in pData. On return, the length of valid words in pData. The length counts the number of bytes.	-
uint32_t *	pData	Buffer for reading the data; to be prepared upfront.	-

Return Values

Data Type	Description
int32_t	Return value according to **DXS_status_t**.

3.1.5 Tone Generation

Table 8 lists the Tone Generator control API functions.

Table 8 Tone Generation Function Overview

Name	Description
DxsToneConfig	This function configures the Tone Generator output signal for a given DXS line.
DxsToneDisable	This function disables the Tone Generator output signal for a given DXS line.
DxsToneEnable	This function enables the Tone Generator output signal for a given DXS line.

3.1.5.1 DxsToneConfig

Description

This function configures the Tone Generator output signal for a given DXS line.

The Tone Generator output is a combination of two signals, signal one and signal two. This functions allows the frequency and level of signal one and signal two to be set independently. The optional amplitude modulation modulates signal one with signal two; otherwise, signal one is summed with signal two. Zero frequency means a signal is not generated.

Prototype

```
int32_t DxsToneConfig (
        uint8_t dxs,
        uint8_t line,
        int16_t level1,
        int16_t level2,
        uint16_t freq1,
        uint16_t freq2,
        uint8_t amModulated );
```

Parameters

Data Type	Name	Description	Dir
uint8_t	dxs	Device number.	-
uint8_t	line	Line number.	-
int16_t	level1	Level for Frequency 1 (Lower Frequency), [-169..31], in units of 0.1 dBm0.	-
int16_t	level2	Level for Frequency 2 (Higher Frequency). [-169..31], in units of 0.1 dBm0.	-
uint16_t	freq1	Frequency of signal one of the DTMF or Alert Tone generator. Possible range [0..4000], in Hz. Zero value means signal one is not generated.	-

Data Type	Name	Description	Dir
uint16_t	freq2	Frequency of signal two of the DTMF or Alert Tone generator. Possible range [0..4000], in Hz. Zero value means signal two is not generated.	-
uint8_t	amModulated	Amplitude Modulation: 0_D **OFF** Signal one of the DTMF or Alert Tone generator is summed with signal two of the DTMF or Alert Tone generator. 1_D **ON** Signal one of the DTMF or Alert Tone generator is modulated with signal two of the DTMF or Alert Tone generator.	-

Return Values

Data Type	Description
int32_t	Return value according to **DXS_status_t**.

3.1.5.2 DxsToneEnable

Description

This function enables the Tone Generator output signal for a given DXS line.

Prototype

```
int32_t DxsToneEnable (
        uint8_t dxs,
        uint8_t line );
```

Parameters

Data Type	Name	Description	Dir
uint8_t	dxs	Device number.	-
uint8_t	line	Line number.	-

Return Values

Data Type	Description
int32_t	Return value according to **DXS_status_t**.

3.1.5.3 DxsToneDisable

Description

This function disables the Tone Generator output signal for a given DXS line.

Prototype

```
int32_t DxsToneDisable (
        uint8_t dxs,
        uint8_t line );
```

Parameters

Data Type	Name	Description	Dir
uint8_t	dxs	Device number.	-
uint8_t	line	Line number.	-

Return Values

Data Type	Description
int32_t	Return value according to **DXS_status_t**.

3.1.6 Caller ID Generation

Table 9 lists the Caller ID generator control API.

Table 9 Caller ID Generation Function Overview

Name	Description
DxsCidInit	This function initializes and configures the CID interface for a given DXS line.
DxsCidMsgReset	This function clears the CID message on a given DXS line.
DxsCidMsgSetup	This function creates or updates the CID message for a given DXS line.
DxsCidSeqStop	This function stops a CID sequence on a given DXS line.
DxsCidTimerConfig	This function configures the CID timers for a given DXS line.
DxsCidTypeIIMsgStart	This function starts a CID Type 2 sequence on a given DXS line.
DxsCidTypeIMsgStart	This function starts a CID Type 1 sequence on a given DXS line.

3.1.6.1 DxsCidInit

Description

This function initializes and configures the CID interface for a given DXS line.

Valid CID Type 1 alert signals are:

- ETSI: **DXS_CID_AS_NONE**, **DXS_CID_AS_DTAS**, **DXS_CID_AS_LR_DTAS**, **DXS_CID_AS_RPAS**;
- Telcordia: **DXS_CID_AS_NONE**, **DXS_CID_AS_OSI**;
- SIN BT: **DXS_CID_AS_LR_DTAS**;
- NTT: **DXS_CID_AS_CAR**.

Prototype

```
int32_t DxsCidInit (
        uint8_t dxs,
        uint8_t line,
        DXS_CID_Std_t std,
        DXS_CID_AS_t as,
        DXS_CID_DATA_FMT_t fmt,
        uint8_t dtmf_ack_idx );
```

Parameters

Data Type	Name	Description	Dir
uint8_t	dxs	Device number.	-
uint8_t	line	Line number.	-
DXS_CID_Std_t	std	CID standard **DXS_CID_Std_t**	-
DXS_CID_AS_t	as	Alert signal (CID Type 1) **DXS_CID_AS_t**	-
DXS_CID_DATA_FMT_t	fmt	Data format **DXS_CID_DATA_FMT_t**	-
uint8_t	dtmf_ack_idx	DTMF index for acknowledgment (CID Type 2).	-

Return Values

Data Type	Description
int32_t	Return value according to **DXS_status_t**.

3.1.6.2 DxsCidTimerConfig

Description

This function configures the CID timers for a given DXS line.

Prototype

```
int32_t DxsCidTimerConfig (
        uint8_t dxs,
        uint8_t line,
        DXS_CID_Std_t std,
```

```
        DXS_CID_Timers_t * pTimers );
```

Parameters

Data Type	Name	Description	Dir
uint8_t	dxs	Device number.	-
uint8_t	line	Line number.	-
DXS_CID_Std_t	std	CID standard **DXS_CID_Std_t**	-
DXS_CID_Timers_t *	pTimers	Pointer to **DXS_CID_Timers_t** union	-

Return Values

Data Type	Description
int32_t	Return value according to **DXS_status_t**.

3.1.6.3 DxsCidMsgSetup

Description

This function creates or updates the CID message for a given DXS line.

This function extends the parameter list of a given message type. When a parameter was already added to the list, the value is updated. When parameters have a fixed length, the length is ignored.

Prototype

```
int32_t DxsCidMsgSetup (
        uint8_t dxs,
        uint8_t line,
        DXS_CID_MsgType_t msg_type,
        DXS_CID_MsgParam_t param,
        char * pData,
        uint8_t length );
```

Parameters

Data Type	Name	Description	Dir
uint8_t	dxs	Device number.	-
uint8_t	line	Line number.	-
DXS_CID_MsgType_t	msg_type	CID message type selector **DXS_CID_MsgType_t**.	-
DXS_CID_MsgParam_t	param	CID message parameter selector for a given message type **DXS_CID_MsgParam_t**.	-
char *	pData	Pointer to character data of parameter value.	-
uint8_t	length	Length of the data pointed by pData.	-

Return Values

Data Type	Description
int32_t	Return value according to **DXS_status_t**.

3.1.6.4 DxsCidMsgReset

Description

This function clears the CID message on a given DXS line.

Prototype

```
int32_t DxsCidMsgReset (
        uint8_t dxs,
        uint8_t line );
```

Parameters

Data Type	Name	Description	Dir
uint8_t	dxs	Device number.	-
uint8_t	line	Line number.	-

Return Values

Data Type	Description
int32_t	Return value according to **DXS_status_t**.

3.1.6.5 DxsCidTypeIMsgStart

Description

This function starts a CID Type 1 sequence on a given DXS line.

Prototype

```
int32_t DxsCidTypeIMsgStart (
        uint8_t dxs,
        uint8_t line,
        DXS_CID_MsgType_t msg_type );
```

Parameters

Data Type	Name	Description	Dir
uint8_t	dxs	Device number.	-
uint8_t	line	Line number.	-
DXS_CID_MsgType_t	msg_type	Message type selector DXS_CID_MsgType_t.	-

Return Values

Data Type	Description
int32_t	Return value according to DXS_status_t.

3.1.6.6 DxsCidSeqStop

Description

This function stops a CID sequence on a given DXS line.

Prototype

```
int32_t DxsCidSeqStop (
        uint8_t dxs,
        uint8_t line );
```

Parameters

Data Type	Name	Description	Dir
uint8_t	dxs	Device number.	-
uint8_t	line	Line number.	-

Return Values

Data Type	Description
int32_t	Return value according to DXS_status_t.

3.1.6.7 DxsCidTypeIIMsgStart

Description

This function starts a CID Type 2 sequence on a given DXS line.

Notes:

1. Enable the DTMF detector prior to a CID Type 2 start.
2. To enable OSI or SAS (Telcordia Type II optional signals), set the duration using **DxsCidTimerConfig**.
3. When SAS tone (Telcordia Type II optional signal) is enabled, configure its characteristics using the **DxsToneConfig** function every time before a CID Type 2 start.

Prototype

```
int32_t DxsCidTypeIIMsgStart (
        uint8_t dxs,
        uint8_t line,
        DXS_CID_MsgType_t msg_type );
```

Parameters

Data Type	Name	Description	Dir
uint8_t	dxs	Device number.	-
uint8_t	line	Line number.	-
DXS_CID_MsgType_t	msg_type	Message type selector **DXS_CID_MsgType_t**	-

Return Values

Data Type	Description
int32_t	Return value according to **DXS_status_t**.

3.1.7 Ringing service

Table 10 lists the ring cadence configuration API functions.

Table 10 Ringing service Function Overview

Name	Description
DxsRingCadenceInit	This function initializes the ringing cadence on a given DXS line.
DxsRingCadenceStart	This function starts the ringing cadence previously configured by the DxsRingCadenceInit function.
DxsRingCadenceStop	This function terminates a currently running ringing cadence.

3.1.7.1 DxsRingCadenceInit

Description

This function initializes the ringing cadence on a given DXS line.

It may be called multiple times to apply different configurations. This function only configures the ringing cadence. The execution of the configured cadence is triggered by the **DxsRingCadenceStart** function.

Notes:

1. The ringing cadence is a bit sequence stored in the **DXS_RingCadence_t** structure. Each bit represents a time interval of 50 ms. When a bit is set to 1, it means ringing is on. When a bit is set to 0, it means ringing is off off. The maximum length of a programmable bit sequence is 320 bits, which results in a maximum duration of 16 seconds for a ring cadence.
2. Different DXS lines are allowed to have different ringing cadence configurations.
3. Changing the configuration of a line on which the ring cadence is started results in an error. The configuration must only be applied when the ring state machine is in the idle state.
4. For designs where one DC/DC converter is shared between multiple chips, the number of simultaneous ring bursts might be limited, e.g., in the case of the **DXS_DCDC_IFB12CH8**. The API also accepts identical ringing cadence configurations for all lines in such cases. The driver internally shifts the programmed ringing cadences to fulfill hardware limitation requirements. It is not possible for the ringing cadence to be absolutely arbitrary when there are hardware limitations. The configuration may result in an error due to conflicts with cadence configurations on other lines.

Prototype

```
int32_t DxsRingCadenceInit (
        uint8_t dxs,
        uint8_t line,
        DXS_RingCadence_t * pRingCad );
```

Parameters

Data Type	Name	Description	Dir
uint8_t	dxs	Device number.	-
uint8_t	line	Line number.	-
DXS_RingCadence_t *	pRingCad	Ringing cadence configuration **DXS_RingCadence_t**	-

Return Values

Data Type	Description
int32_t	Return value according to **DXS_status_t**.

3.1.7.2 DxsRingCadenceStart

Description

This function starts the ringing cadence previously configured using the **DxsRingCadenceInit** function.

When a Caller ID message was previously configured using the **DxsCidInit** and **DxsCidMsgSetup** functions, this is executed within the ringing cadence according to the configured CID standard.

Prototype

```
int32_t DxsRingCadenceStart (
        uint8_t dxs,
        uint8_t line );
```

Parameters

Data Type	Name	Description	Dir
uint8_t	dxs	Device number.	-
uint8_t	line	Line number.	-

Return Values

Data Type	Description
int32_t	Return value according to **DXS_status_t**.

3.1.7.3 DxsRingCadenceStop

Description

This function terminates a currently running ringing cadence.

Prototype

```
int32_t DxsRingCadenceStop (
        uint8_t dxs,
        uint8_t line );
```

Parameters

Data Type	Name	Description	Dir
uint8_t	dxs	Device number.	-
uint8_t	line	Line number.	-

Return Values

Data Type	Description
int32_t	Return value according to **DXS_status_t**.

3.1.8 FSK Generation

Table 11 lists the FSK generator control API functions.

Table 11 FSK Generation Function Overview

Name	Description
DxsFskConfig	This function configures the FSK generator for a given DXS line.
DxsFskData	This function delivers a new data buffer for an FSK generator already started on a given DXS line.
DxsFskDisable	This function disables the FSK generator for a given DXS line.
DxsFskEnable	This function enables the FSK generator for a given DXS line.

3.1.8.1 DxsFskConfig

Description

This function configures the FSK generator for a given DXS line.

Prototype

```
int32_t DxsFskConfig (
        uint8_t dxs,
        uint8_t line,
        int16_t level,
        uint16_t seizure,
        uint16_t mark );
```

Parameters

Data Type	Name	Description	Dir
uint8_t	dxs	Device number.	-
uint8_t	line	Line number.	-
int16_t	level	CID Send Level: [-929..31], in units of 0.1 dBm0.	-
uint16_t	seizure	Number of Seizure Bits, [0..32767].	-
uint16_t	mark	Number of Mark Bits, [0..32767].	-

Return Values

Data Type	Description
int32_t	Return value according to DXS_status_t.

3.1.8.2 DxsFskEnable

Description

This function enables the FSK generator for a given DXS line.

Prototype

```
int32_t DxsFskEnable (
        uint8_t dxs,
        uint8_t line,
        uint8_t standard,
        uint8_t autodeact );
```

Parameters

Data Type	Name	Description	Dir
uint8_t	dxs	Device number.	-
uint8_t	line	Line number.	-
uint8_t	standard	CID Specification: 0_D **V23_BEL202** The Bell 202 specification is used for the FSK generator. 1_D **V23_ITU_T** The ITU-T V.23 specification is used for the FSK generator.	-
uint8_t	autodeact	Auto Deactivation: 0_D **OFF** 1_D **ON**	-

Return Values

Data Type	Description
int32_t	Return value according to **DXS_status_t**.

3.1.8.3 DxsFskDisable

Description

This function disables the FSK generator for a given DXS line.

Prototype

```
int32_t DxsFskDisable (
        uint8_t dxs,
        uint8_t line,
        uint8_t autodeact );
```

Parameters

Data Type	Name	Description	Dir
uint8_t	dxs	Device number.	-
uint8_t	line	Line number.	-
uint8_t	autodeact	Auto Deactivation: 0_D **OFF** 1_D **ON**	-

Return Values

Data Type	Description
int32_t	Return value according to **DXS_status_t**.

3.1.8.4 DxsFskData

Description

This function delivers a new data buffer for an FSK generator already started on a given DXS line.

Prototype

```
int32_t DxsFskData (
        uint8_t dxs,
        uint8_t line,
        uint8_t nByte,
        uint8_t * pByte );
```

Parameters

Data Type	Name	Description	Dir
uint8_t	dxs	Device number.	-
uint8_t	line	Line number.	-
uint8_t	nByte	Data buffer size in bytes.	-
uint8_t *	pByte	Pointer to the data buffer.	-

Return Values

Data Type	Description
int32_t	Return value according to **DXS_status_t**.

3.1.9 Tone Detection

Table 12 lists the Universal Tone Detector control API functions.

Table 12 Tone Detection Function Overview

Name	Description
DxsUtdDisable	This function disables the Universal Tone Detector (UTD) for a given DXS line.
DxsUtdEnable	This function enables the Universal Tone Detector (UTD) for a given DXS line.

3.1.9.1 DxsUtdEnable

Description

This function enables the Universal Tone Detector (UTD) for a given DXS line.

The configuration of the UTD is set by the BBD file using the DxsBBD function. It is not possible to configure the UTD via the API. This function only triggers the UTD operation start.

Prototype

```
int32_t DxsUtdEnable (
        uint8_t dxs,
        uint8_t line,
        uint16_t tone_idx,
        int direction );
```

Parameters

Data Type	Name	Description	Dir
uint8_t	dxs	Device number.	-
uint8_t	line	Line number.	-
uint16_t	tone_idx	Tone index	-
int	direction	Direction of detection: 0_D **PCM Interface** Detect tones coming from the PCM interface. 1_D **Analog Line** Detect tones coming from the analog line.	-

Return Values

Data Type	Description
int32_t	Return value according to DXS_status_t.

3.1.9.2 DxsUtdDisable

Description

This function disables the Universal Tone Detector (UTD) for a given DXS line.

Prototype

```
int32_t DxsUtdDisable (
        uint8_t dxs,
        uint8_t line );
```

Parameters

Data Type	Name	Description	Dir
uint8_t	dxs	Device number.	-
uint8_t	line	Line number.	-

Return Values

Data Type	Description
int32_t	Return value according to **DXS_status_t**.

3.1.10 DTMF Detection

Table 13 lists the DTMF detector control API functions.

Table 13 DTMF Detection Function Overview

Name	Description
DxsDtmfConfig	This function configures the DTMF Detector for a given DXS line.
DxsDtmfDisable	This function disables the DTMF Detector for a given DXS line.
DxsDtmfEnable	This function enables the DTMF Detector for a given DXS line.

3.1.10.1 DxsDtmfConfig

Description

This function configures the DTMF Detector for a given DXS line.

The configuration parameters are the minimum detection level and maximum allowed level twist.

Prototype

```
int32_t DxsDtmfConfig (
        uint8_t dxs,
        uint8_t line,
        int16_t level,
        int16_t twist );
```

Parameters

Data Type	Name	Description	Dir
uint8_t	dxs	Device number.	-
uint8_t	line	Line number.	-
int16_t	level	Minimum signal level in units of 0.1 dBm0, range [-369..-29]	-
int16_t	twist	Maximum allowed signal twist in units of 0.1 dB, range [10..120].	-

Return Values

Data Type	Description
int32_t	Return value according to **DXS_status_t**.

3.1.10.2 DxsDtmfEnable

Description

This function enables the DTMF Detector for a given DXS line.

Prototype

```
int32_t DxsDtmfEnable (
        uint8_t dxs,
        uint8_t line );
```

Parameters

Data Type	Name	Description	Dir
uint8_t	dxs	Device number.	-
uint8_t	line	Line number.	-

Return Values

Data Type	Description
int32_t	Return value according to **DXS_status_t**.

3.1.10.3 DxsDtmfDisable

Description

This function disables the DTMF Detector for a given DXS line.

Prototype

```
int32_t DxsDtmfDisable (
        uint8_t dxs,
        uint8_t line );
```

Parameters

Data Type	Name	Description	Dir
uint8_t	dxs	Device number.	-
uint8_t	line	Line number.	-

Return Values

Data Type	Description
int32_t	Return value according to **DXS_status_t**.

3.1.11 TTX Metering

Table 14 lists the Teletax Metering pulse generation control API functions.

Table 14 TTX Metering Function Overview

Name	Description
DxsMeteringPulseDisable	This function disables the TTX pulse generation for a given DXS line.
DxsMeteringPulseEnable	This function enables the TTX pulse generation for a given DXS line.

3.1.11.1 DxsMeteringPulseEnable

Description

This function enables the TTX pulse generation for a given DXS line.

The TTX parameters are configured by the BBD file using the **DxsBBD** function. It is not possible to configure the TTX, e.g., switch the TTX frequency, via this API. This function enables playout of a single TTX pulse. Generation of a pulse series is not supported by the driver API and must be carried out at the application software level.

Prototype

```
int32_t DxsMeteringPulseEnable (
        uint8_t dxs,
        uint8_t line );
```

Parameters

Data Type	Name	Description	Dir
uint8_t	dxs	Device number.	-
uint8_t	line	Line number.	-

Return Values

Data Type	Description
int32_t	Return value according to **DXS_status_t**.

3.1.11.2 DxsMeteringPulseDisable

Description

This function disables the TTX pulse generation for a given DXS line.

Prototype

```
int32_t DxsMeteringPulseDisable (
        uint8_t dxs,
        uint8_t line );
```

Parameters

Data Type	Name	Description	Dir
uint8_t	dxs	Device number.	-
uint8_t	line	Line number.	-

Return Values

Data Type	Description
int32_t	Return value according to **DXS_status_t**.

3.1.12 Continuous Measurement

Table 15 lists the Reading of Continuous Measurement API functions.

Table 15 Continuous Measurement Function Overview

Name	Description
DxsContMeasurementResultGet	This function gets DXS Continuous Measurement results for a given line.

3.1.12.1 DxsContMeasurementResultGet

Description

This function gets DXS Continuous Measurement results for a given line.

When the return code indicates success, the structure of **DXS_ContMeasurementResult_t** type is filled with data. It is possible to call this API function at any time.

Prototype

```
int32_t DxsContMeasurementResultGet (
        uint8_t dxs,
        uint8_t line,
        DXS_ContMeasurementResult_t * pResult );
```

Parameters

Data Type	Name	Description	Dir
uint8_t	dxs	Device number.	-
uint8_t	line	Line number.	-
DXS_ContMeasurementResult_t *	pResult	Pointer to structure DXS_ContMeasurementResult_t.	-

Return Values

Data Type	Description
int32_t	Return value according to DXS_status_t.

3.1.13 GR-909 Network Line Testing

Table 16 lists the GR-909 Line Testing configuration and control API functions.

Table 16 GR-909 Network Line Testing Function Overview

Name	Description
DxsGr909LimitsGet	This function gets the limit values for GR-909 measurements for a given DXS line.
DxsGr909LimitsSet	This function programs the limit values for GR-909 measurements for a given DXS line.
DxsGr909ResultGet	This function gets the GR-909 measurement results for a given DXS line.

3.1.13.1 DxsGr909LimitsSet

Description

This function programs the limit values for GR-909 measurements for a given DXS line.

The limit values are supplied via the DXS_GR909Config_t type structure.

Prototype

```
int32_t DxsGr909LimitsSet (
        uint8_t dxs,
        uint8_t line,
        DXS_GR909Config_t * pGR909Conf );
```

Parameters

Data Type	Name	Description	Dir
uint8_t	dxs	Device number.	-
uint8_t	line	Line number.	-
DXS_GR909Config_t *	pGR909Conf	Pointer to structure DXS_GR909Config_t.	-

Return Values

Data Type	Description
int32_t	Return value according to **DXS_status_t**.

3.1.13.2 DxsGr909LimitsGet

Description

This function gets the limit values for GR-909 measurements for a given DXS line.

When the return code indicates success, the limit values are read into the **DXS_GR909Config_t** type structure.

Prototype

```
int32_t DxsGr909LimitsGet (
        uint8_t dxs,
        uint8_t line,
        DXS_GR909Config_t * pGR909Conf );
```

Parameters

Data Type	Name	Description	Dir
uint8_t	dxs	Device number.	-
uint8_t	line	Line number.	-
DXS_GR909Config_t *	pGR909Conf	Pointer to structure **DXS_GR909Config_t**.	-

Return Values

Data Type	Description
int32_t	Return value according to **DXS_status_t**.

3.1.13.3 DxsGr909ResultGet

Description

This function gets the GR-909 measurement results for a given DXS line.

When the return code indicates success, the results are read into the **DXS_GR909Result_t** type structure.

Prototype

```
int32_t DxsGr909ResultGet (
        uint8_t dxs,
        uint8_t line,
        DXS_GR909Result_t * pGR909Result );
```

Parameters

Data Type	Name	Description	Dir
uint8_t	dxs	Device number.	-
uint8_t	line	Line number.	-
DXS_GR909Result_t *	pGR909Result	Pointer to result structure DXS_GR909Result_t.	-

Return Values

Data Type	Description
int32_t	Return value according to DXS_status_t.

3.1.14 Calibration

Table 17 lists the Analog Line Calibration and Open Loop Calibration control API functions.

Table 17 Calibration Function Overview

Name	Description
DxsCalibrationGet	This function gets the Analog Line Calibration data for a given DXS line.
DxsCalibrationSet	This function programs the Analog Line Calibration data for a given DXS line.
DxsOpenLoopCalibration	This function performs Open Loop Calibration for a given DXS line.
DxsOpenLoopConfigGet	This function gets the Open Loop Calibration results for a given DXS line.
DxsOpenLoopConfigSet	This function programs the Open Loop Calibration values for a given DXS line.

3.1.14.1 DxsCalibrationGet

Description

This function gets the Analog Line Calibration data for a given DXS line.

When the return code indicates success, the calibration data is provided in the DXS_Calibration_t type structure.

Prototype

```
int32_t DxsCalibrationGet (
        uint8_t dxs,
        uint8_t line,
        DXS_Calibration_t * pCalibration );
```

Parameters

Data Type	Name	Description	Dir
uint8_t	dxs	Device number.	-
uint8_t	line	Line number.	-
DXS_Calibration_t *	pCalibration	Pointer to structure DXS_Calibration_t.	-

Return Values

Data Type	Description
int32_t	Return value according to **DXS_status_t**.

3.1.14.2 DxsCalibrationSet

Description

This function programs the Analog Line Calibration data for a given DXS line.

The calibration data is supplied in the **DXS_Calibration_t** type structure.

Prototype

```
int32_t DxsCalibrationSet (
        uint8_t dxs,
        uint8_t line,
        DXS_Calibration_t * pCalibration );
```

Parameters

Data Type	Name	Description	Dir
uint8_t	dxs	Device number.	-
uint8_t	line	Line number.	-
DXS_Calibration_t *	pCalibration	Pointer to structure **DXS_Calibration_t**.	-

Return Values

Data Type	Description
int32_t	Return value according to **DXS_status_t**.

3.1.14.3 DxsOpenLoopCalibration

Description

This function performs Open Loop Calibration for a given DXS line.

The `loops` parameter defines the number of executions. The average is calculated and returned in the **DXS_OLCalibration_t** type structure.

Prototype

```
int32_t DxsOpenLoopCalibration (
        uint8_t dxs,
        uint8_t line,
        uint8_t loops,
        DXS_OLCalibration_t * pOLCalibrationResult );
```

Parameters

Data Type	Name	Description	Dir
uint8_t	dxs	Device number.	-
uint8_t	line	Line number.	-
uint8_t	loops	Number of loops.	-
DXS_OLCalibration_t *	pOLCalibrationResult	Pointer to structure **DXS_OLCalibration_t**.	-

Return Values

Data Type	Description
int32_t	Return value according to **DXS_status_t**.

3.1.14.4 DxsOpenLoopConfigSet

Description

This function programs the Open Loop Calibration values for a given DXS line.

The values are supplied in the **DXS_OLCalibration_t** type structure.

Prototype

```
int32_t DxsOpenLoopConfigSet (
        uint8_t dxs,
        uint8_t line,
        DXS_OLCalibration_t * pOLCalibration );
```

Parameters

Data Type	Name	Description	Dir
uint8_t	dxs	Device number.	-
uint8_t	line	Line number.	-
DXS_OLCalibration_t *	pOLCalibration	Pointer to structure DXS_OLCalibration_t.	-

Return Values

Data Type	Description
int32_t	Return value according to DXS_status_t.

3.1.14.5 DxsOpenLoopConfigGet

Description

This function gets the Open Loop Calibration results for a given DXS line.

When the return code indicates success, the results are stored in the DXS_OLCalibration_t type structure.

Prototype

```
int32_t DxsOpenLoopConfigGet (
        uint8_t dxs,
        uint8_t line,
        DXS_OLCalibration_t * pOLCalibration );
```

Parameters

Data Type	Name	Description	Dir
uint8_t	dxs	Device number.	-
uint8_t	line	Line number.	-
DXS_OLCalibration_t *	pOLCalibration	Pointer to structure DXS_OLCalibration_t.	-

Return Values

Data Type	Description
int32_t	Return value according to DXS_status_t.

3.1.15 Hook State Machine

Table 18 lists the Hook State Machine configuration API functions.

Table 18 Hook State Machine Function Overview

Name	Description
DxsHookValTimeSet	This function sets the hook validation windows for a given DXS line.

3.1.15.1 DxsHookValTimeSet

Description

This function sets the hook validation windows for a given DXS line.

Prototype

```
int32_t DxsHookValTimeSet (
        uint8_t dxs,
        uint8_t line,
        DXS_HookValTime_t * pTime );
```

Parameters

Data Type	Name	Description	Dir
uint8_t	dxs	Device number.	-
uint8_t	line	Line number.	-
DXS_HookValTime_t *	pTime	Pointer to structure DXS_HookValTime_t.	-

Return Values

Data Type	Description
int32_t	Return value according to DXS_status_t.

3.1.16 Capacitance Measurement

Table 19 lists the Capacitance Measurement API functions.

Table 19 Capacitance Measurement Function Overview

Name	Description
DxsCapMeasurementResultGet	This function gets the Capacitance Measurement results for a given DXS line.

3.1.16.1 DxsCapMeasurementResultGet

Description

This function gets the Capacitance Measurement results for a given DXS line.

Prototype

```
int32_t DxsCapMeasurementResultGet (
        uint8_t dxs,
        uint8_t line,
        DXS_Capacitance_t * pCapacitance );
```

Parameters

Data Type	Name	Description	Dir
uint8_t	dxs	Device number.	-
uint8_t	line	Line number.	-
DXS_Capacitance_t *	pCapacitance	Pointer to result structure DXS_Capacitance_t.	-

Return Values

Data Type	Description
int32_t	Return value according to DXS_status_t.

3.1.17 AC Level Meter Measurement

Table 20 lists the AC Level Meter measurement control API functions.

Table 20 AC Level Meter Measurement Function Overview

Name	Description
DxsAcLmMeasResArgGet	Each single measurement result consists of two coordinates, the argument and the value. These are used to display the results in an Output (value).
DxsAcLmMeasResNumGet	This function gets the available number of measurement results for a given line and measurement code.
DxsAcLmMeasResValGet	This function gets the result of the particular measurement point.
DxsAcLmMeasurementStart	This function starts the AC Level Meter measurement on a given DXS line.

3.1.17.1 DxsAcLmMeasurementStart

Description

This function starts the AC Level Meter measurement on a given DXS line.

The measurement code is defined by the DXS_ACLM_Measurement_t enumerator.

Prototype

```
int32_t DxsAcLmMeasurementStart (
        uint8_t dxs,
        uint8_t line,
        DXS_ACLM_Measurement_t aclm_mt );
```

Parameters

Data Type	Name	Description	Dir
uint8_t	dxs	Device number.	-

Data Type	Name	Description	Dir
uint8_t	line	Line number.	-
DXS_ACLM_Measurement_t	aclm_mt	Measurement selection: value of DXS_ACLM_Measurement_t type.	-

Return Values

Data Type	Description
int32_t	Return value according to DXS_status_t.

3.1.17.2 DxsAcLmMeasResNumGet

Description

This function gets the number of measurement results available for a given line and measurement code.

It is mandatory to call the DxsAcLmMeasurementStart function for the given code before calling DxsAcLmMeasResNumGet.

Prototype

```
int32_t DxsAcLmMeasResNumGet (
        uint8_t dxs,
        uint8_t line,
        DXS_ACLM_Measurement_t aclm_mt,
        int32_t * pResultNum );
```

Parameters

Data Type	Name	Description	Dir
uint8_t	dxs	Device number.	-
uint8_t	line	Line number.	-
DXS_ACLM_Measurement_t	aclm_mt	Measurement selection: value of DXS_ACLM_Measurement_t type.	-
int32_t *	pResultNum	Output parameter: number of available results for a selected measurement.	-

Return Values

Data Type	Description
int32_t	Return value according to DXS_status_t.

3.1.17.3 DxsAcLmMeasResArgGet

Description

Each single measurement result consists of two coordinates, the argument and the value. These are used to display the results in the output (value) vs. input (argument) plot.

This function gets the argument of the particular measurement point. It is mandatory to call the **DxsAcLmMeasurementStart** function before calling **DxsAcLmMeasResArgGet**. The `index` parameter value must be lower than the total number of measurement points returned by the **DxsAcLmMeasResNumGet** function. The output value is either a frequency in Hz in the case of Frequency Response or Transhybrid measurements, or a level in units of 0.1 dBm0 in the case of Gain Tracking or Signal to Noise measurements.

Prototype

```
int32_t DxsAcLmMeasResArgGet (
        uint8_t dxs,
        uint8_t line,
        DXS_ACLM_Measurement_t aclm_mt,
        int32_t index,
        int32_t * pMeasArg );
```

Parameters

Data Type	Name	Description	Dir
uint8_t	dxs	Device number.	-
uint8_t	line	Line number.	-
DXS_ACLM_Measurement_t	aclm_mt	Measurement selection: Value of DXS_ACLM_Measurement_t type.	-
int32_t	index	Measurement index.	-
int32_t *	pMeasArg	Output parameter: Measurement argument, [Hz] or units of 0.1 dBm0, depending on aclm_mt selection.	-

Return Values

Data Type	Description
int32_t	Return value according to **DXS_status_t**.

3.1.17.4 DxsAcLmMeasResValGet

Description

This function gets the result at the particular measurement point.

It is mandatory to call the **DxsAcLmMeasurementStart** function before calling **DxsAcLmMeasResValGet**. The `index` parameter value must be lower than total number of measurement points returned by **DxsAcLmMeasResNumGet**. The output value is a level in units of 0.1 dB.

Prototype

```
int32_t DxsAcLmMeasResValGet (
        uint8_t dxs,
        uint8_t line,
        DXS_ACLM_Measurement_t aclm_mt,
        int32_t index,
        int32_t * pResVal );
```

Parameters

Data Type	Name	Description	Dir
uint8_t	dxs	Device number.	-
uint8_t	line	Line number.	-
DXS_ACLM_Measurement_t	aclm_mt	Measurement selection: Value of **DXS_ACLM_Measurement_t** type.	-
int32_t	index	Measurement index.	-
int32_t *	pResVal	Output parameter: ACLM measurement result, units of 0.1 dB.	-

Return Values

Data Type	Description
int32_t	Return value according to **DXS_status_t**.

3.1.18 GPIO Pin Control Interface

Table 21 lists the GPIO/IO pin handling API functions.

Table 21 GPIO Pin Control Interface Function Overview

Name	Description
DxsGpioPortConfig	This function configures the GPIO port of the given DXS device.
DxsGpioRead	This function reads the input signal of a particular GPIO pin for a given DXS Series device.
DxsGpioWrite	This function sets the output signal of a particular GPIO pin for a given DXS Series device.

3.1.18.1 DxsGpioPortConfig

Description

This function configures the GPIO port of the given DXS device.

The configuration is provided by the DXS_GpioConfig_t type structure.

Prototype

```
int32_t DxsGpioPortConfig (
        uint8_t dxs,
        DXS_GpioConfig_t * pGpioConf );
```

Parameters

Data Type	Name	Description	Dir
uint8_t	dxs	Device number.	-
DXS_GpioConfig_t *	pGpioConf	Pointer to structure DXS_GpioConfig_t.	-

Return Values

Data Type	Description
int32_t	Return value according to DXS_status_t.

3.1.18.2 DxsGpioWrite

Description

This function sets the output signal of a particular GPIO pin for a given DXS Series device.

It is mandatory to call the DxsGpioPortConfig function to set the GPIO pin direction to `output` before calling DxsGpioWrite.

Prototype

```
int32_t DxsGpioWrite (
        uint8_t dxs,
```

```
        uint8_t pin,
        uint8_t val );
```

Parameters

Data Type	Name	Description	Dir
uint8_t	dxs	Device number.	-
uint8_t	pin	GPIO Pin Number [0..3].	-
uint8_t	val	Value to be written [0 or 1].	-

Return Values

Data Type	Description
int32_t	Return value according to **DXS_status_t**.

3.1.18.3 DxsGpioRead

Description

This function reads the input signal of a particular GPIO pin for a given DXS Series device.

It is mandatory to call the **DxsGpioPortConfig** function to set the GPIO pin direction to `input` before calling **DxsGpioRead**.

Prototype

```
int32_t DxsGpioRead (
        uint8_t dxs,
        uint8_t pin,
        uint8_t * value );
```

Parameters

Data Type	Name	Description	Dir
uint8_t	dxs	Device number.	-
uint8_t	pin	GPIO Pin Number [0..3].	-
uint8_t *	value		-

Return Values

Data Type	Description
int32_t	Return value according to **DXS_status_t**.

3.1.19 Clock Failure Handling

Table 22 lists the PCM clock failure handling functions.

Table 22 Clock Failure Handling Function Overview

Name	Description
DxsClockFailLineFeedFreeze	This function prevents Emergency Shutdown (ESD) line mode and line voltage drop when a PCM clock failure occurs.
DxsClockFailLineFeedUnFreeze	This function restores the automatic switching to Emergency Shutdown (ESD) line mode in the case of clock failure.

3.1.19.1 DxsClockFailLineFeedFreeze

Description

This function prevents Emergency Shutdown (ESD) line mode and line voltage drop when there is a PCM clock failure.

Prototype

```
int32_t DxsClockFailLineFeedFreeze (
        uint8_t dxs );
```

Parameters

Data Type	Name	Description	Dir
uint8_t	dxs	Device number.	-

Return Values

Data Type	Description
int32_t	Return value according to DXS_status_t.

3.1.19.2 DxsClockFailLineFeedUnFreeze

Description

This function restores the automatic switching to Emergency Shutdown (ESD) line mode in the case of clock failure.

Prototype

```
int32_t DxsClockFailLineFeedUnFreeze (
        uint8_t dxs );
```

Parameters

Data Type	Name	Description	Dir
uint8_t	dxs	Device number.	-

Return Values

Data Type	Description
int32_t	Return value according to **DXS_status_t**.

3.1.20 Shared DC/DC converter

Table 23 lists the shared DC/DC converter configuration API functions.

Table 23 Shared DC/DC converter Function Overview

Name	Description
DxsSharedDcDcMasterSet	This function configures the driver to know the master channel.

3.1.20.1 DxsSharedDcDcMasterSet

Description

This function configures the driver to know the master channel. This is the DXS Series device/line pair to which the DC/DC converter hardware is connected. Only this channel receives the hardware configuration from the BBD.

This API call is only allowed in multi-device, multi-channel designs where there is a single shared DC/DC converter.

The API driver must be previously initialized using the **DxsInit** API, and **DxsSharedDcDcMasterSet** must then be called before **DxsBBD**.

When the configuration is completed, it is not modifiable unless the affected device is reinitialized using the **DxsInit** function.

Prototype

```
int32_t DxsSharedDcDcMasterSet (
        uint8_t dxs,
        uint8_t line );
```

Parameters

Data Type	Name	Description	Dir
uint8_t	dxs	Device number, that addresses DC/DC HW, starts with 0	-
uint8_t	line	Line number, that addresses DC/DC HW, starts with 0	-

Return Values

Data Type	Description
int32_t	• **DXS_statusOk** - on success • **DXS_statusDevNotFound** - not existing device • **DXS_statusDevNotInitialized** - not initialized device • **DXS_statusChannelNotFound** - not existing line • **DXS_statusInvalidParam** - configuration is already done

3.1.21 Event Reporting

Table 24 lists the event reporting function type and Table 25 lists the event reporting API functions.

Table 24 Event Reporting Function_Type Overview

Name	Description
DXS_WaitList_t	Waiting list type.

Table 25 Event Reporting Function Overview

Name	Description
DxsEventDisable	This function disables dispatching of a particular event on a given line to the application.
DxsEventEnable	This function enables dispatching of a particular event on a given line to the application.
DxsEventGet	This function reads a single event from the given device.
DxsWaitForEvent	This function waits on a given waiting list.
DxsWaitListDestroy	This function deletes the waiting list.
DxsWaitListDevAdd	This function adds a device into a given waiting list.
DxsWaitListDevRemove	This function removes a device from a given waiting list.
DxsWaitListInit	This function initializes the waiting list.

3.1.21.1 DXS_WaitList_t

Prototype

```
typedef void * DXS_WaitList_t;
```

Parameters

Data Type	Name	Description
void *	DXS_WaitList_t	Waiting list type.

3.1.21.2 DxsWaitListInit

Description

This function initializes the waiting list.

Prototype

```
DXS_WaitList_t DxsWaitListInit (
       void  );
```

Parameters

Data Type	Name	Description	Dir
void			-

Return Values

Data Type	Description
DXS_WaitList_t	New waiting list instance or NULL in the case of an error.

3.1.21.3 DxsWaitListDestroy

Description

This function deletes the waiting list.

This is a reverse operation to the DxsWaitListInit function.

Prototype

```
void DxsWaitListDestroy (
       DXS_WaitList_t wl );
```

Parameters

Data Type	Name	Description	Dir
DXS_WaitList_t	wl	Waiting list instance, a variable of DXS_WaitList_t type.	-

3.1.21.4 DxsWaitListDevAdd

Description

This function adds the particular device into a given waiting list.

Prototype

```
int32_t DxsWaitListDevAdd (
       uint8_t dxs,
       DXS_WaitList_t wl );
```

Parameters

Data Type	Name	Description	Dir
uint8_t	dxs	Device number.	-
DXS_WaitList_t	wl	Waiting list instance	-

Return Values

Data Type	Description
int32_t	Return value according to DXS_status_t.

3.1.21.5 DxsWaitListDevRemove

Description

This function removes the particular device from a given waiting list.

This is a reverse operation to the **DxsWaitListDevAdd** function.

Prototype

```
int32_t DxsWaitListDevRemove (
        uint8_t dxs,
        DXS_WaitList_t wl );
```

Parameters

Data Type	Name	Description	Dir
uint8_t	dxs	Device number.	-
DXS_WaitList_t	wl	Waiting list instance.	-

Return Values

Data Type	Description
int32_t	Return value according to **DXS_status_t**.

3.1.21.6 DxsWaitForEvent

Description

This function waits on a given waiting list.

The function blocks the calling thread as long as all devices in the waiting list have empty event queues. The function returns when an event occurs on any device entered into the given waiting list. The returned positive value is treated as a device number with at least one queued event. It is then possible to read the event details using the **DxsEventGet** function.

Prototype

```
int32_t DxsWaitForEvent (
        DXS_WaitList_t wl );
```

Parameters

Data Type	Name	Description	Dir
DXS_WaitList_t	wl	Waiting list instance.	-

Return Values

Data Type	Description
int32_t	When this is a positive value, it is a device number for a device with a non-empty event queue.When this is a negative value, it indicates an incorrect parameter.

3.1.21.7 DxsEventGet

Description

This function reads a single event from the given device.

Example 1. Four devices in a single thread.
Example 2. Two devices in a separate thread each.

Prototype

```
int32_t DxsEventGet (
        uint8_t dxs,
        DXS_Event_t * pEvent );
```

Parameters

Data Type	Name	Description	Dir
uint8_t	dxs	Device number.	-
DXS_Event_t *	pEvent	Pointer to **DXS_Event_t** structure	-

Return Values

Data Type	Description
int32_t	The remaining event count in the device event queue or a negative value when there is an error.

Example

```
// Initialize waiting list
DXS_WaitList_t wlist = DxsWaitListInit();

// Add devices 0,1,2 and 3 to the waiting list
DxsWaitListDevAdd(0, wlist);
DxsWaitListDevAdd(1, wlist);
DxsWaitListDevAdd(2, wlist);
DxsWaitListDevAdd(3, wlist);

//Event processing loop
while(1)
{
   DXS_Event_t event = {0};
   int32_t have_more = 0, dev;
```

```
   //Wait for event from any device listed in wlist
   dev = DxsWaitForEvent(wlist);
   if (dev < 0)
   {
      //handle error
   }

   do
   {
      have_more = DxsGetEvent(dev, &event);
      ProcessEvent(&event);
   } while (have_more);
}
```

```
[THREAD 1:]

wl1 = DxsWaitListInit();
DxsWaitListDevAdd(0, wl1);
while(1)
{
   DXS_Event_t event = {0};
   int32_t have_more = 0, dev;

   dev = DxsWaitForEvent(wl1);
   if (dev < 0)
   {
      handle error
   }
   do
   {
      have_more = DxsGetEvent(dev, &event);
&event);
      ProcessEvent(&event);
   } while (have_more);
}
```

```
[THREAD 2:]

wl2 = DxsWaitListInit();
DxsWaitListDevAdd(1, wl2);
while(1)
{
  DXS_Event_t event = {0};
  int32_t have_more = 0, dev;

  dev = DxsWaitForEvent(wl2);
  if (dev < 0)
  {
     handle error
  }
  do
  {
     have_more = DxsGetEvent(dev,

     ProcessEvent(&event);
  } while (have_more);
}
```

3.1.21.8 DxsEventEnable

Description

This function enables the dispatching of a particular event on a given line to the application.

Notes

1. *Most events are enabled by default after device initialization. Some rarely used events must be enabled explicitly when they are required.*
2. *Events **DXS_EVENT_NONE** and **DXS_EVENT_MAX** serve only for internal range checking. Attempts to enable these events for the application result in an error.*

Prototype

```
int32_t DxsEventEnable (
       uint8_t dxs,
       uint8_t line,
```

```
        DXS_Event_id_t event );
```

Parameters

Data Type	Name	Description	Dir
uint8_t	dxs	Device number.	-
uint8_t	line	Line number.	-
DXS_Event_id_t	event	Event: value of **DXS_Event_id_t** type.	-

Return Values

Data Type	Description
int32_t	Return value according to **DXS_status_t**.

3.1.21.9 DxsEventDisable

Description

This function disables dispatching of a particular event on a given line to the application.

It is not possible to disable every type of event using this API function. Attempting to disable critical error events results in an error.

Prototype

```
int32_t DxsEventDisable (
        uint8_t dxs,
        uint8_t line,
        DXS_Event_id_t event );
```

Parameters

Data Type	Name	Description	Dir
uint8_t	dxs	Device number.	-
uint8_t	line	Line number.	-
DXS_Event_id_t	event	Event: value of **DXS_Event_id_t** type.	-

Return Values

Data Type	Description
int32_t	Return value according to **DXS_status_t**.

3.2 Data Types and Structure Reference

This sections contains the reference details for the data types and module structures.

3.2.1 Structure Reference

Table 26 contains the Structure reference.

Table 26 Structure Overview

Name	Description
DXS_Calibration_t	This structure contains data specific to the calibration.
DXS_Capacitance_t	This structure contains data specific to the capacitance measurement.
DXS_Caps_t	This structure contains data specific to the capabilities.
DXS_CID_BT_Timers_t	This structure is used to configure the timers for the CID SIN BT protocol.
DXS_CID_ETSI_Timers_t	This structure is used to configure the timers for the CID ETSI protocol.
DXS_CID_NTT_Timers_t	This structure is used to configure the timers for the CID NTT protocol.
DXS_CID_TELCORDIA_Timers_t	This structure is used to configure the timers for the CID Telcordia protocol.
DXS_ContMeasurementResult_t	This structure contains data specific to the continuous measurements.
DXS_EVENT_DATA_CERR_t	This structure contains data specific to the command error event.
DXS_EVENT_DATA_DTMF_t	This structure contains data specific to the DTMF event.
DXS_EVENT_DATA_PULSE_t	This structure contains data specific to the pulse dialing event.
DXS_EVENT_DATA_TONE_DET_t	This structure contains data specific to the tone detection event.
DXS_Event_t	This structure contains data about the DXS event.
DXS_GpioConfig_t	This structure contains data specific to the GPIO configuration.
DXS_GR909Config_t	This structure contains GR-909 measurement limit values.
DXS_GR909Result_t	This structure contains data specific to the GR-909 result.
DXS_HookValTime_t	This structure contains validation times of hook, hook flash, and pulse dialing.
DXS_OLCalibration_t	This structure contains data specific to the open loop calibration.
DXS_RingCadence_t	This structure is used to configure the ringing cadence.
DXS_Version_t	This structure contains data specific to the version.

3.2.1.1 DXS_Calibration_t

Description

This structure contains data specific to the calibration.

Prototype

```
struct
{
        uint8_t version;
        uint8_t crc8ccitt;
        uint32_t data[2];
} DXS_Calibration_t;
```

Parameters

Data Type	Name	Description
uint8_t	version	Version of the structure.
uint8_t	crc8ccitt	CRC-8-CCITT of data. POLYNOMIAL = 0x07
uint32_t	data[2]	Calibration data.

3.2.1.2 DXS_Capacitance_t

Description

This structure contains data specific to the capacitance measurement.

Prototype

```
struct
{
        uint16_t Cap_R2G;
        uint16_t Cap_T2G;
        uint16_t Cap_T2R;
} DXS_Capacitance_t;
```

Parameters

Data Type	Name	Description
uint16_t	Cap_R2G	Capacitance Ring to Ground [nF].
uint16_t	Cap_T2G	Capacitance Tip to Ground [nF].
uint16_t	Cap_T2R	Capacitance Tip to Ring [nF].

3.2.1.3 DXS_Caps_t

Description

This structure contains data specific to the capabilities.

Prototype

```
struct
{
        uint8_t nChannels;
        uint8_t nGR909;
        uint8_t nACLM;
        uint8_t nTTX;
        uint8_t nUTD;
        uint8_t nCID;
        uint8_t nHowler;
} DXS_Caps_t;
```

Parameters

Data Type	Name	Description
uint8_t	nChannels	Actual number of analog channels read from the device.
uint8_t	nGR909	GR-909 support: 0_D **No** GR-909 not supported. 1_D **Yes** GR-909 supported.
uint8_t	nACLM	AC Level Meter support: 0_D **No** AC Level Meter not supported. 1_D **Yes** AC Level Meter supported.
uint8_t	nTTX	Teletax support: 0_D **No** Teletax not supported. 1_D **Yes** Teletax supported.
uint8_t	nUTD	Universal Tone Detector support: 0_D **No** Universal Tone Detector not supported. 1_D **Yes** Universal Tone Detector supported.
uint8_t	nCID	FSK Caller ID support: 0_D **No** FSK Caller ID not supported. 1_D **Yes** FSK Caller ID supported.
uint8_t	nHowler	Howler Tone generation support: 0_D **No** Howler Tone generation not supported. 1_D **Yes** Howler Tone generation supported.

3.2.1.4 DXS_CID_BT_Timers_t

Description

This structure is used to configure timers for CID SIN BT protocol.

Prototype

```
struct
{
        uint16_t dtas;
        uint16_t T0;
        uint16_t T1;
        uint16_t T2;
        uint16_t t2_dtas;
        uint16_t t2_ack;
        uint16_t t2_T0;
        uint16_t t2_T1;
        uint16_t t2_T2;
} DXS_CID_BT_Timers_t;
```

Parameters

Data Type	Name	Description
uint16_t	dtas	DTAS duration.
uint16_t	T0	Silence period after line reversal.

Data Type	Name	Description
uint16_t	T1	Silence period prior to FSK message.
uint16_t	T2	Interval between FSK message and CID sequence end DXS_EVENT_CID_SEQ_END.
uint16_t	t2_dtas	DTAS duration (CID Type 2).
uint16_t	t2_ack	Timeout to detect acknowledgment signal (CID Type 2).
uint16_t	t2_T0	Silence period prior to DTAS (CID Type 2).
uint16_t	t2_T1	Transition time between acknowledgment detection and start of FSK generation (CID Type 2).
uint16_t	t2_T2	Interval between FSK message and CID sequence end DXS_EVENT_CID_SEQ_END (CID Type 2).

3.2.1.5 DXS_CID_ETSI_Timers_t

Description

This structure is used to configure timers for CID ETSI protocol.

Prototype

```
struct
{
        uint16_t lring;
        uint16_t dtas;
        uint16_t rpas;
        uint16_t T0;
        uint16_t T1;
        uint16_t T2;
        uint16_t T3;
        uint16_t T4;
        uint16_t T5;
        uint16_t T6;
        uint16_t t2_dtas;
        uint16_t t2_T14;
        uint16_t t2_T10;
        uint16_t t2_T12;
        uint16_t t2_T13;
        uint16_t t2_T9;
} DXS_CID_ETSI_Timers_t;
```

Parameters

Data Type	Name	Description
uint16_t	lring	First ring duration.
uint16_t	dtas	DTAS duration.
uint16_t	rpas	Short ring duration.
uint16_t	T0	Silence period after line reversal.

Data Type	Name	Description
uint16_t	T1	Silence period prior to FSK message (alert signal type - DXS_CID_AS_LR_DTAS)
uint16_t	T2	Interval between FSK message and CID sequence end DXS_EVENT_CID_SEQ_END.
uint16_t	T3	Silence period prior to FSK message (alert signal type - DXS_CID_AS_RPAS)
uint16_t	T4	Silence period prior to FSK message (alert signal type - DXS_CID_AS_DTAS)
uint16_t	T5	Silence period prior to FSK message (alert signal type - DXS_CID_AS_NONE)
uint16_t	T6	Interval between FSK message and CID sequence end DXS_EVENT_CID_SEQ_END (alert signal type - DXS_CID_AS_NONE)
uint16_t	t2_dtas	DTAS duration (CID Type 2)
uint16_t	t2_T14	Timeout to detect acknowledgment signal (CID Type 2)
uint16_t	t2_T10	Silence period prior to DTAS (CID Type 2)
uint16_t	t2_T12	Transition time between acknowledgment detection and start of FSK generation (CID Type 2)
uint16_t	t2_T13	Interval between FSK message and CID sequence end DXS_EVENT_CID_SEQ_END (CID Type 2)
uint16_t	t2_T9	Transition time after t2_T14 timeout elapsed and before CID sequence end DXS_EVENT_CID_SEQ_END (CID Type 2)

3.2.1.6 DXS_CID_NTT_Timers_t

Description

This structure is used to configure timers for the CID NTT protocol.

Prototype

```
struct
{
        uint16_t CARon;
        uint16_t CARoff;
        uint16_t T0;
        uint16_t T1;
        uint16_t T2;
        uint16_t T3;
        uint16_t T4;
        uint16_t t2_as_first;
        uint16_t t2_as_pause;
        uint16_t t2_as_second;
        uint16_t t2_T0;
        uint16_t t2_T1;
        uint16_t t2_T2;
} DXS_CID_NTT_Timers_t;
```

Parameters

Data Type	Name	Description
uint16_t	CARon	CAR ring on duration.
uint16_t	CARoff	CAR ring off duration.
uint16_t	T0	Silence period after line reversal.
uint16_t	T1	Primary answer signal arrival waiting time.
uint16_t	T2	Silence period prior to FSK message.
uint16_t	T3	Incoming successful signal arrival waiting timing.
uint16_t	T4	Secondary answer signal reception timing.
uint16_t	t2_as_first	First alert tone duration (CID Type 2)
uint16_t	t2_as_pause	Pause between alert tones (CID Type 2)
uint16_t	t2_as_second	Second alert tone duration (CID Type 2)
uint16_t	t2_T0	Silence period prior to alert signal (CID Type 2)
uint16_t	t2_T1	Transition time between alert tone and start of FSK generation (CID Type 2)
uint16_t	t2_T2	Interval between FSK message and CID sequence end DXS_EVENT_CID_SEQ_END (CID Type 2)

3.2.1.7 DXS_CID_TELCORDIA_Timers_t

Description

This structure is used to configure timers for the CID Telcordia protocol.

Prototype

```
struct
{
        uint16_t lring;
        uint16_t osi;
        uint16_t T0;
        uint16_t T1;
        uint16_t T2;
        uint16_t t2_osi;
        uint16_t t2_W;
        uint16_t t2_X;
        uint16_t t2_X1;
        uint16_t t2_Y;
        uint16_t t2_T1;
        uint16_t t2_Q;
        uint16_t t2_S;
} DXS_CID_TELCORDIA_Timers_t;
```

Parameters

Data Type	Name	Description
uint16_t	Iring	First ring duration.
uint16_t	osi	OSI duration.
uint16_t	T0	Silence period after the first ring.
uint16_t	T1	Silence period after OSI.
uint16_t	T2	Interval between FSK message and CID sequence end DXS_EVENT_CID_SEQ_END.
uint16_t	t2_osi	OSI duration (CID Type 2).
uint16_t	t2_W	Silence period prior to alerting sequence (CID Type 2).
uint16_t	t2_X	SAS duration (CID Type 2).
uint16_t	t2_X1	Transition time between alerting signals (CID Type 2).
uint16_t	t2_Y	Alerting signal (CID Type 2).
uint16_t	t2_T1	Acknowledgment timeout (CID Type 2).
uint16_t	t2_Q	Transition time between acknowledgment detection and start of FSK generation (CID Type 2).
uint16_t	t2_S	Interval between FSK message or second OSI and CID sequence end DXS_EVENT_CID_SEQ_END (CID Type 2).

3.2.1.8 DXS_ContMeasurementResult_t

Description

This structure contains data specific to the continuous measurements.

Prototype

```
struct
{
        int32_t Vline;
        int32_t Itrans;
        uint32_t Iring;
        uint32_t Vring;
} DXS_ContMeasurementResult_t;
```

Parameters

Data Type	Name	Description
int32_t	Vline	Output voltage of DC regulation, [mV].
int32_t	Itrans	Value of the actual line current, [uA].
uint32_t	Iring	Value of the last ring burst current. RMS value of ring current within the last ring period, [uArms]
uint32_t	Vring	Value of the last ring burst voltage. RMS value of VDAC sine, [mVrms]

3.2.1.9 DXS_EVENT_DATA_CERR_t

Description

This structure contains data specific to the command error event.

Prototype

```
struct
{
       uint16_t reason;
       uint32_t command;
} DXS_EVENT_DATA_CERR_t;
```

Parameters

Data Type	Name	Description
uint16_t	reason	Reason given by the firmware for the command error.
uint32_t	command	Header of error command.

3.2.1.10 DXS_EVENT_DATA_DTMF_t

Description

This structure contains data specific to the DTMF event.

Prototype

```
struct
{
       uint8_t digit;
       char ascii;
} DXS_EVENT_DATA_DTMF_t;
```

Parameters

Data Type	Name	Description
uint8_t	digit	DTMF digit number information. 0_D **No_Key** no key detected 11_D **Key_0** DTMF key '0' detected 1_D **Key_1** DTMF key '1' detected 2_D **Key_2** DTMF key '2' detected 3_D **Key_3** DTMF key '3' detected 4_D **Key_4** DTMF key '4' detected 5_D **Key_5** DTMF key '5' detected 6_D **Key_6** DTMF key '6' detected 7_D **Key_7** DTMF key '7' detected 8_D **Key_8** DTMF key '8' detected 9_D **Key_9** DTMF key '9' detected 10_D **Key_*** DTMF key '*' detected 12_D **Key_#** DTMF key '#' detected 28_D **Key_A** DTMF key 'A' detected 29_D **Key_B** DTMF key 'B' detected 30_D **Key_C** DTMF key 'C' detected 31_D **Key_D** DTMF key 'D' detected
char	ascii	DTMF digit in ASCII representation. 0_D **No_Key** no key detected 48_D **Key_0** DTMF key '0' detected 49_D **Key_1** DTMF key '1' detected 50_D **Key_2** DTMF key '2' detected 51_D **Key_3** DTMF key '3' detected 52_D **Key_4** DTMF key '4' detected 53_D **Key_5** DTMF key '5' detected 54_D **Key_6** DTMF key '6' detected 55_D **Key_7** DTMF key '7' detected 56_D **Key_8** DTMF key '8' detected 57_D **Key_9** DTMF key '9' detected 42_D **Key_*** DTMF key '*' detected 35_D **Key_#** DTMF key '#' detected 65_D **Key_A** DTMF key 'A' detected 66_D **Key_B** DTMF key 'B' detected 67_D **Key_C** DTMF key 'C' detected 68_D **Key_D** DTMF key 'D' detected

3.2.1.11 DXS_EVENT_DATA_PULSE_t

Description

This structure contains data specific to the pulse dialing event.

Prototype

```
struct
{
        uint16_t reserved:8;
        uint16_t digit:8;
} DXS_EVENT_DATA_PULSE_t;
```

Parameters

Data Type	Name	Description
uint16_t	reserved:8	Reserved.
uint16_t	digit:8	Pulse digit number information (0 - 9). 1_D **Key_1** key '1' detected 2_D **Key_2** key '2' detected 3_D **Key_3** key '3' detected 4_D **Key_4** key '4' detected 5_D **Key_5** key '5' detected 6_D **Key_6** key '6' detected 7_D **Key_7** key '7' detected 8_D **Key_8** key '8' detected 9_D **Key_9** key '9' detected 11_D **Key_0** key '0' detected

3.2.1.12 DXS_EVENT_DATA_TONE_DET_t

Description

This structure contains data specific to the tone detection event.

Prototype

```
struct
{
        uint8_t index;
} DXS_EVENT_DATA_TONE_DET_t;
```

Parameters

Data Type	Name	Description
uint8_t	index	Tone table index of the tone that was detected.

3.2.1.13 DXS_Event_t

Description

The structure contains data about the DXS event.

Prototype

```
struct
{
        uint8_t dev;
        uint8_t ch;
        DXS_Event_id_t id;
        DXS_Event_data_t data;
} DXS_Event_t;
```

Parameters

Data Type	Name	Description
uint8_t	dev	Device number.
uint8_t	ch	Channel number.
DXS_Event_id_t	id	Event ID number.
DXS_Event_data_t	data	Container for event specific data, union type.

3.2.1.14 DXS_GpioConfig_t

Description

This structure contains data specific to the GPIO configuration.

Prototype

```
struct
{
        uint8_t gpio0_PU;
        uint8_t gpio1_PU;
        uint8_t gpio2_PU;
        uint8_t gpio3_PU;
        uint8_t gpio0_DIR;
        uint8_t gpio1_DIR;
        uint8_t gpio2_DIR;
        uint8_t gpio3_DIR;
} DXS_GpioConfig_t;
```

Parameters

Data Type	Name	Description
uint8_t	gpio0_PU	GPIO Pull-up Resistor for GPIO line 0: 0_D **Disabled** Resistor disabled. 1_D **Enabled** Resistor enabled.
uint8_t	gpio1_PU	GPIO Pull-up Resistor for GPIO line 1: 0_D **Disabled** Resistor disabled. 1_D **Enabled** Resistor enabled.
uint8_t	gpio2_PU	GPIO Pull-up Resistor for GPIO line 2: 0_D **Disabled** Resistor disabled. 1_D **Enabled** Resistor enabled.
uint8_t	gpio3_PU	GPIO Pull-up Resistor for GPIO line 3: 0_D **Disabled** Resistor disabled. 1_D **Enabled** Resistor enabled.
uint8_t	gpio0_DIR	Direction of GPIO line 0: 0_D **Output** 1_D **Input**
uint8_t	gpio1_DIR	Direction of GPIO line 1: 0_D **Output** 1_D **Input**
uint8_t	gpio2_DIR	Direction of GPIO line 2: 0_D **Output** 1_D **Input**
uint8_t	gpio3_DIR	Direction of GPIO line 3: 0_D **Output** 1_D **Input**

3.2.1.15 DXS_GR909Config_t

Description

This structure contains GR-909 measurement limit values.

Prototype

```
struct
{
        uint32_t settle_time;
        uint32_t HPT_W2G_AC_Lim;
        uint32_t HPT_W2W_AC_Lim;
        uint32_t HPT_W2G_DC_Lim;
        uint32_t HPT_W2W_DC_Lim;
        uint32_t FEMF_W2G_AC_Lim;
        uint32_t FEMF_W2W_AC_Lim;
        uint32_t FEMF_W2G_DC_Lim;
        uint32_t FEMF_W2W_DC_Lim;
        uint32_t RFT_Res_Lim;
        uint32_t ROH_Lin_Lim;
        uint32_t RIT_Low_Lim;
```

```
        uint32_t RIT_High_Lim;
} DXS_GR909Config_t;
```

Parameters

Data Type	Name	Description
uint32_t	settle_time	Settle Time after Discharge, [ms].
uint32_t	HPT_W2G_AC_Lim	HPT Wire to GND AC Limit, [mVrms].
uint32_t	HPT_W2W_AC_Lim	HPT Wire to Wire AC Limit, [mVrms].
uint32_t	HPT_W2G_DC_Lim	HPT Wire to GND DC Limit, [mV].
uint32_t	HPT_W2W_DC_Lim	HPT Wire to Wire DC Limit, [mV].
uint32_t	FEMF_W2G_AC_Lim	FEMF Wire to GND AC Limit, [mVrms].
uint32_t	FEMF_W2W_AC_Lim	FEMF Wire to Wire AC Limit, [mVrms].
uint32_t	FEMF_W2G_DC_Lim	FEMF Wire to GND DC Limit, [mV].
uint32_t	FEMF_W2W_DC_Lim	FEMF Wire to Wire DC Limit, [mV].
uint32_t	RFT_Res_Lim	RFT Resistance Limit, [Ohm].
uint32_t	ROH_Lin_Lim	ROH Linearity Limit, [%].
uint32_t	RIT_Low_Lim	RIT Lower Limit, [Ohm].
uint32_t	RIT_High_Lim	RIT Higher Limit, [Ohm].

3.2.1.16 DXS_GR909Result_t

Description

This structure contains data specific to the GR-909 result.

Prototype

```
struct
{
        uint8_t HPT_Valid;
        uint8_t FEMF_Valid;
        uint8_t RFT_Valid;
        uint8_t ROH_Valid;
        uint8_t RIT_Valid;
        uint8_t HPT_Pass;
        uint8_t FEMF_Pass;
        uint8_t RFT_Pass;
        uint8_t ROH_Pass;
        uint8_t RIT_Pass;
        uint32_t HPT_AC_R2G;
        uint32_t HPT_AC_T2G;
        uint32_t HPT_AC_T2R;
        int32_t HPT_DC_R2G;
        int32_t HPT_DC_T2G;
        int32_t HPT_DC_T2R;
        uint32_t RFT_R2G;
        uint32_t RFT_T2G;
```

```
        uint32_t RFT_T2R;
        uint32_t ROH_Low;
        uint32_t ROH_High;
        uint32_t RIT_Res;
} DXS_GR909Result_t;
```

Parameters

Data Type	Name	Description
uint8_t	HPT_Valid	HPT test results: 0_D **Invalid** The results of the last HPT tests are not valid. 1_D **Valid** The results of the last HPT tests are valid.
uint8_t	FEMF_Valid	FEMF test results: 0_D **Invalid** The results of the last FEMF tests are not valid. 1_D **Valid** The results of the last FEMF tests are valid.
uint8_t	RFT_Valid	RFT test results: 0_D **Invalid** The results of the last RFT tests are not valid. 1_D **Valid** The results of the last RFT tests are valid.
uint8_t	ROH_Valid	ROH test results: 0_D **Invalid** The results of the last ROH tests are not valid. 1_D **Valid** The results of the last ROH tests are valid.
uint8_t	RIT_Valid	RIT test results: 0_D **Invalid** The results of the last RIT tests are not valid. 1_D **Valid** The results of the last RIT tests are valid.
uint8_t	HPT_Pass	HPT test limits: 0_D **No** The HPT test was not within the limits. 1_D **Yes** The HPT test passed within the limits.
uint8_t	FEMF_Pass	FEMF test limits: 0_D **No** The FEMF test was not within the limits. 1_D **Yes** The FEMF test passed within the limits.
uint8_t	RFT_Pass	RFT test limits: 0_D **No** The RFT test was not within the limits. 1_D **Yes** The RFT test passed within the limits.
uint8_t	ROH_Pass	ROH test limits: 0_D **No** The ROH test was not within the limits. 1_D **Yes** The ROH test passed within the limits.
uint8_t	RIT_Pass	RIT test limits: 0_D **No** The RIT test was not within the limits. 1_D **Yes** The RIT test passed within the limits.
uint32_t	HPT_AC_R2G	Test Result HPT or FEMF AC Ring to GND, [mVrms].
uint32_t	HPT_AC_T2G	Test Result HPT or FEMF AC Tip to GND, [mVrms].
uint32_t	HPT_AC_T2R	Test Result HPT or FEMF AC Tip to Ring, [mVrms].
int32_t	HPT_DC_R2G	Test Result HPT or FEMF DC Ring to GND, [mV].
int32_t	HPT_DC_T2G	Test Result HPT or FEMF DC Tip to GND, [mV].
int32_t	HPT_DC_T2R	Test Result HPT or FEMF DC Tip to Ring, [mV].
uint32_t	RFT_R2G	Resistive Fault Ring to Ground Result, [Ohm].

Data Type	Name	Description
uint32_t	RFT_T2G	Resistive Fault Tip to Ground Result, [Ohm].
uint32_t	RFT_T2R	Resistive Fault Tip to Ring Result, [Ohm].
uint32_t	ROH_Low	Test Result ROH Low, [Ohm].
uint32_t	ROH_High	Test Result ROH High, [Ohm].
uint32_t	RIT_Res	Test Result RIT, [Ohm].

3.2.1.17 DXS_HookValTime_t

Description

This structure is used for the validation times of hook, hook flash, and pulse dialing.

This is an example of typical timings:

- 80 ms <= flash time <= 200 ms
- 30 ms <= digit low time <= 80 ms
- 30 ms <= digit high time <= 80 ms
- Interdigit time = 300 ms
- Off-hook time = 40 ms
- On-hook time = 400 ms; Only min. time is validated and pre-initialized for interdigit, off-hook, and on-hook time.

Prototype

```
struct
{
        DXS_HOOK_VALIDATION_TYPE_t nType;
        uint32_t nMinTime;
        uint32_t nMaxTime;
} DXS_HookValTime_t;
```

Parameters

Data Type	Name	Description
DXS_HOOK_VALIDATION_TYPE_t	nType	Type of validation time setting.
uint32_t	nMinTime	Minimum time for validation in ms.
uint32_t	nMaxTime	Maximum time for validation in ms.

3.2.1.18 DXS_OLCalibration_t

Description

This structure contains data specific to the open loop calibration.

Prototype

```
struct
{
        uint8_t GRG;
        uint8_t GTG;
        uint8_t GTR;
        uint16_t CRG;
```

```
        uint16_t CTG;
        uint16_t CTR;
} DXS_OLCalibration_t;
```

Parameters

Data Type	Name	Description
uint8_t	GRG	Open Loop Conductance Ring to Ground, [S].
uint8_t	GTG	Open Loop Conductance Tip to Ground, [S].
uint8_t	GTR	Open Loop Conductance Tip to Ring, [S].
uint16_t	CRG	Open Loop Capacitance Ring to Ground, [nF].
uint16_t	CTG	Open Loop Capacitance Tip to Ground, [nF].
uint16_t	CTR	Open Loop Capacitance Tip to Ring, [nF].

3.2.1.19 DXS_RingCadence_t

Description

This structure is used to configure the ringing cadence.

Prototype

```
struct
{
        uint8_t data[DXS_RING_CADENCE_MAX_LEN_BYTES];
        int32_t valid_bits;
} DXS_RingCadence_t;
```

Parameters

Data Type	Name	Description
uint8_t	data[DXS_RING_CADENCE_MAX_LEN_BYTES]	Ringing cadence data bits.
int32_t	valid_bits	Number of valid bits in the data field.

3.2.1.20 DXS_Version_t

Description

This structure contains data specific to the version.

Prototype

```
struct
{
        DXS_Dev_t device;
        DXS_Package_t package;
        uint8_t fw_maj;
        uint8_t fw_min;
        uint8_t fw_hfix;
        uint32_t fw_time;
```

```
        uint32_t lib_ver;
        uint32_t api_ver;
} DXS_Version_t;
```

Parameters

Data Type	Name	Description
DXS_Dev_t	device	Device code defined by DXS_Dev_t type.
DXS_Package_t	package	Device package defined by DXS_Package_t type.
uint8_t	fw_maj	Firmware version major number.
uint8_t	fw_min	Firmware version minor number.
uint8_t	fw_hfix	Firmware version hotfix number.
uint32_t	fw_time	Firmware build timestamp.
uint32_t	lib_ver	DXS API Device Driver version in A.B.C.D packed in a 32-bit word.
uint32_t	api_ver	DXS API version in E.F.G packed in a 32-bit word.

3.2.2 Union Reference

Table 27 contains the Union reference.

Table 27 Union Overview

Name	Description
DXS_CID_Timers_t	This is a union of structures containing data specific to CID generator timers.
DXS_Event_data_t	This union contains data specific to the event.

3.2.2.1 DXS_CID_Timers_t

Description

This is a union of structures containing data specific to CID generator timers.

Prototype

```
union
{
        DXS_CID_ETSI_Timers_t etsi;
        DXS_CID_TELCORDIA_Timers_t tc;
        DXS_CID_BT_Timers_t bt;
        DXS_CID_NTT_Timers_t ntt;
} DXS_CID_Timers_t;
```

Parameters

Data Type	Name	Description
DXS_CID_ETSI_Timers_t	etsi	Structure for CID ETSI timers configuration.
DXS_CID_TELCORDIA_Timers_t	tc	Structure for CID Telcordia timers configuration.

Data Type	Name	Description
DXS_CID_BT_Timers_t	bt	Structure for CID SIN BT timers configuration.
DXS_CID_NTT_Timers_t	ntt	Structure for CID NTT timers configuration.

3.2.2.2 DXS_Event_data_t

Description

This union contains data specific to the event.

Prototype

```
union
{
        DXS_EVENT_DATA_DTMF_t dtmf;
        DXS_EVENT_DATA_TONE_DET_t tone_det;
        DXS_EVENT_DATA_PULSE_t pulse;
        DXS_EVENT_DATA_CERR_t cerr;
        DXS_EVENT_DATA_CID_ERR_t cid_err;
} DXS_Event_data_t;
```

Parameters

Data Type	Name	Description
DXS_EVENT_DATA_DTMF_t	dtmf	DTMF digit information.
DXS_EVENT_DATA_TONE_DET_t	tone_det	Tone detection information.
DXS_EVENT_DATA_PULSE_t	pulse	Pulse digit information.
DXS_EVENT_DATA_CERR_t	cerr	Command error event details.
DXS_EVENT_DATA_CID_ERR_t	cid_err	Caller ID error event details.

3.2.3 Enumerator Reference

Table 28 contains the Enumerator reference.

Table 28 Enumerator Overview

Name	Description
DXS_ACCESS_MODE_t	DXS access mode.
DXS_ACLM_Measurement_t	AC Level Meter measurement types.
DXS_CID_AS_t	CID Alert signals.
DXS_CID_DATA_FMT_t	CID data transmission format.
DXS_CID_MsgParam_t	Enumeration for CID parameters.
DXS_CID_MsgType_t	Enumeration for CID message type.
DXS_CID_Std_t	Supported CID standards.
DXS_DCDC_Var_t	DXS DC/DC converter type.
DXS_Dev_t	DXS device type.
DXS_EVENT_DATA_CID_ERR_t	This enumeration contains data specific to the CID error event.
DXS_Event_id_t	Definitions for the events.
DXS_HOOK_VALIDATION_TYPE_t	Validation types used for the DXS_HookValTime_t structure.
DXS_LINE_MODE_t	Definitions for line feeding.
DXS_Package_t	DXS package type.
DXS_status_t	Definition of all error codes.

3.2.3.1 DXS_LINE_MODE_t

Description

Definitions for line feeding.

Prototype

```
enum
{
        DXS_LINE_FEED_DISABLED = 0,
        DXS_LINE_FEED_RING_BURST = 1,
        DXS_LINE_FEED_STANDBY = 2,
        DXS_LINE_FEED_ACTIVE = 3,
        DXS_LINE_FEED_GROUND_START = 5,
        DXS_LINE_FEED_CALIBRATE = 6,
        DXS_LINE_FEED_GR909 = 7,
        DXS_LINE_FEED_ACTIVE_MWI = 12,
        DXS_LINE_FEED_ACTIVE_HOWL = 13,
        DXS_LINE_FEED_RINGING_REVPOL = 17,
        DXS_LINE_FEED_ACTIVE_REVPOL = 19,
        DXS_LINE_FEED_GROUND_START_T2G = 21,
        DXS_LINE_FEED_CAP_MEAS = 23,
        DXS_LINE_FEED_ACTIVE_HOWL_REVPOL = 29
} DXS_LINE_MODE_t;
```

Parameters

Name	Value	Description
DXS_LINE_FEED_DISABLED	0	Set operating mode to power down.
DXS_LINE_FEED_RING_BURST	1	Set operating mode to ringing mode with normal polarity.
DXS_LINE_FEED_STANDBY	2	Set operating mode to standby mode.
DXS_LINE_FEED_ACTIVE	3	Set operating mode to active with normal polarity. Normal polarity means that the ring wire is near VBAT and the tip wire is near GND.
DXS_LINE_FEED_GROUND_START	5	Set operating mode to ground start mode.
DXS_LINE_FEED_CALIBRATE	6	Set operating mode to calibration.
DXS_LINE_FEED_GR909	7	Set operating mode to GR909 line testing.
DXS_LINE_FEED_ACTIVE_MWI	12	Set mode to active with normal polarity and message waiting indication. This operating mode is only available when the subscriber is on-hook.
DXS_LINE_FEED_ACTIVE_HOWL	13	Send howler tone with normal polarity settings. Sending of a howler tone requires a special setup (due to high amplitude). This operating mode is only available when the subscriber is off-hook.
DXS_LINE_FEED_RINGING_REVPOL	17	Set operating mode to ringing with reverse polarity.
DXS_LINE_FEED_ACTIVE_REVPOL	19	Set operating mode to active reverse polarity. Reverse polarity means that the tip wire is near VBAT and the ring wire is near GND.
DXS_LINE_FEED_GROUND_START_T2G	21	Set operating mode to ground start tip to ground.
DXS_LINE_FEED_CAP_MEAS	23	Set operating mode to capacitance line testing measurement.
DXS_LINE_FEED_ACTIVE_HOWL_REVPOL	29	Send howler tone with reverse polarity settings. Sending of a howler tone requires a special setup (due to high amplitude). This operating mode is only available in ACT off-hook.

3.2.3.2 DXS_Event_id_t

Description

Definitions for the events.

Prototype

```
enum
{
        DXS_EVENT_NONE = 0,
        DXS_EVENT_FXS_ONHOOK = 1,
        DXS_EVENT_FXS_OFFHOOK = 2,
        DXS_EVENT_FXS_FLASH = 3,
        DXS_EVENT_NLT_END = 4,
        DXS_EVENT_NLT_ABORT = 5,
        DXS_EVENT_PULSE_DIGIT = 6,
        DXS_EVENT_PULSE_START = 7,
        DXS_EVENT_DTMF_DIGIT = 8,
        DXS_EVENT_CALIBRATION_END = 9,
        DXS_EVENT_METERING_END = 10,
        DXS_EVENT_OVERTEMP = 11,
        DXS_EVENT_OVERTEMP_END = 12,
        DXS_EVENT_GROUND_FAULT = 13,
        DXS_EVENT_GROUND_FAULT_END = 14,
        DXS_EVENT_TONE_DET_CPT = 15,
        DXS_EVENT_TONE_DET_CPT_END = 16,
        DXS_EVENT_DEBUG_CERR = 17,
        DXS_EVENT_CID_REQ_DATA = 18,
        DXS_EVENT_CID_BUF_UNDERFLOW = 19,
        DXS_EVENT_CID_END = 20,
        DXS_EVENT_CID_SEQ_END = 21,
        DXS_EVENT_CID_SEQ_ERROR = 22,
        DXS_EVENT_GROUND_KEY = 23,
        DXS_EVENT_GROUND_KEY_END = 24,
        DXS_EVENT_FAULT_FW_WATCHDOG = 25,
        DXS_EVENT_T2G_TIMER_EXPIRED = 26,
        DXS_EVENT_FXS_RAW_ONHOOK = 27,
        DXS_EVENT_FXS_RAW_OFFHOOK = 28,
        DXS_EVENT_MAX = 63
} DXS_Event_id_t;
```

Parameters

Name	Value	Description
DXS_EVENT_NONE	0	
DXS_EVENT_FXS_ONHOOK	1	Hook event: on-hook.
DXS_EVENT_FXS_OFFHOOK	2	Hook event: off-hook.
DXS_EVENT_FXS_FLASH	3	Hook event: flash hook.

Name	Value	Description
DXS_EVENT_NLT_END	4	Line testing finished (results are ready, where applicable).
DXS_EVENT_NLT_ABORT	5	Line testing aborted.
DXS_EVENT_PULSE_DIGIT	6	Pulse digit detected.
DXS_EVENT_PULSE_START	7	Indicates start of pulse dialing. It is possible to use this event to stop the dial tone.
DXS_EVENT_DTMF_DIGIT	8	DTMF tone detected.
DXS_EVENT_CALIBRATION_END	9	End of calibration. The calibration process has finished or was stopped because of an error.
DXS_EVENT_METERING_END	10	End of metering pulse.
DXS_EVENT_OVERTEMP	11	Over temperature detected.
DXS_EVENT_OVERTEMP_END	12	Over temperature finished.
DXS_EVENT_GROUND_FAULT	13	Ground fault detected.
DXS_EVENT_GROUND_FAULT_END	14	Ground fault finished.
DXS_EVENT_TONE_DET_CPT	15	Call progress tone detected.
DXS_EVENT_TONE_DET_CPT_END	16	Call progress tone detection ended.
DXS_EVENT_DEBUG_CERR	17	Debug command error event.
DXS_EVENT_CID_REQ_DATA	18	Caller ID sender request data.
DXS_EVENT_CID_BUF_UNDERFLOW	19	Caller ID buffer underflow.
DXS_EVENT_CID_END	20	Caller ID sending finished.
DXS_EVENT_CID_SEQ_END	21	Caller ID sequence finished.
DXS_EVENT_CID_SEQ_ERROR	22	Caller ID sequence aborted.
DXS_EVENT_GROUND_KEY	23	Ground key detected.
DXS_EVENT_GROUND_KEY_END	24	Ground key finished.
DXS_EVENT_FAULT_FW_WATCHDOG	25	Firmware watchdog (crash). The application must reinitialize the device upon reception of this event.
DXS_EVENT_T2G_TIMER_EXPIRED	26	
DXS_EVENT_FXS_RAW_ONHOOK	27	Raw hook event: on-hook.
DXS_EVENT_FXS_RAW_OFFHOOK	28	Raw hook event: off-hook.
DXS_EVENT_MAX	63	

3.2.3.3 DXS_EVENT_DATA_CID_ERR_t

Description

This enumeration contains data specific to the CID error event.

Prototype

```
enum
{
        DXS_EVENT_CID_FSK_BUF = 1,
        DXS_EVENT_CID_TIMEOUT = 2,
        DXS_EVENT_CID_HOOK_STATE = 3,
        DXS_EVENT_CID_HW_RING_LIMIT = 4
} DXS_EVENT_DATA_CID_ERR_t;
```

Parameters

Name	Value	Description
DXS_EVENT_CID_FSK_BUF	1	FSK buffer underflow.
DXS_EVENT_CID_TIMEOUT	2	CID SM timeout.
DXS_EVENT_CID_HOOK_STATE	3	CID SM incorrect hook state.
DXS_EVENT_CID_HW_RING_LIMIT	4	The DC/DC HW limitation of simultaneously ringing lines is reached.

3.2.3.4 DXS_Dev_t

Description

DXS device type.

Prototype

```
enum
{
        DXS_DEV_UNKNOWN = 0,
        DXS_DEV_PEF32002_A11 = 1,
        DXS_DEV_PEF32001_A11 = 2,
        DXS_DEV_PEF32002_A12 = 3,
        DXS_DEV_PEF32001_A12 = 4,
        DXC_DEV_PEF31002_A12 = 5,
        DXC_DEV_PEF31001_A12 = 6,
        DXS_DEV_PEF32002_V12_A13 = 7,
        DXS_DEV_PEF32001_V12_A11 = 8,
        DXC_DEV_PEF31002_V12_A13 = 9,
        DXC_DEV_PEF31001_V12_A11 = 10
} DXS_Dev_t;
```

Parameters

Name	Value	Description
DXS_DEV_UNKNOWN	0	unknown or unfused device
DXS_DEV_PEF32002_A11	1	DXS102, Version 1.1, Engineering Sample.
DXS_DEV_PEF32001_A11	2	DXS101, Version 1.1, Engineering Sample.
DXS_DEV_PEF32002_A12	3	DXS102, Version 1.1.
DXS_DEV_PEF32001_A12	4	DXS101, Version 1.1.
DXC_DEV_PEF31002_A12	5	DXC102, Version 1.1.
DXC_DEV_PEF31001_A12	6	DXC101, Version 1.1.
DXS_DEV_PEF32002_V12_A13	7	DXS102, Version 1.2, Step A13.
DXS_DEV_PEF32001_V12_A11	8	DXS101, Version 1.2, Step A11.
DXC_DEV_PEF31002_V12_A13	9	DXC102, Version 1.2, Step A13.
DXC_DEV_PEF31001_V12_A11	10	DXC101, Version 1.2, Step A11.

3.2.3.5 DXS_Package_t

Description

DXS package type.

Prototype

```
enum
{
        DXS_PACKAGE_UNKNOWN = 0,
        DXS_PACKAGE_VQFN68 = 1,
        DXS_PACKAGE_VQFN48 = 2,
        DXS_PACKAGE_VQFN44 = 3
} DXS_Package_t;
```

Parameters

Name	Value	Description
DXS_PACKAGE_UNKNOWN	0	unknown or unfused package
DXS_PACKAGE_VQFN68	1	VQFN68 package.
DXS_PACKAGE_VQFN48	2	VQFN48 package.
DXS_PACKAGE_VQFN44	3	VQFN44 package.

3.2.3.6 DXS_ACCESS_MODE_t

Description

DXS access mode.

Prototype

```
enum
```

```
{
        DXS_ACCESS_SPI = 0,
        DXS_ACCESS_CSI = 1,
        DXS_ACCESS_LAST = 2
} DXS_ACCESS_MODE_t;
```

Parameters

Name	Value	Description
DXS_ACCESS_SPI	0	SPI access.
DXS_ACCESS_CSI	1	CSI access.
DXS_ACCESS_LAST	2	Last number.

3.2.3.7 DXS_DCDC_Var_t

Description

DXS DC/DC converter type.

Prototype

```
enum
{
        DXS_DCDC_UNDEF = -1,
        DXS_DCDC_IBB12 = 0,
        DXS_DCDC_CIBB12 = 1,
        DXS_DCDC_IB12 = 2,
        DXS_DCDC_BB48 = 4,
        DXS_DCDC_CBB48 = 3,
        DXS_DCDC_IFB12 = 6,
        DXS_DCDC_CIFB12 = 7,
        DXS_DCDC_IBGD12 = 8,
        DXS_DCDC_IBVD3 = 10,
        DXS_DCDC_IFB12CH8 = 29,
        DXS_DCDC_MAX
} DXS_DCDC_Var_t;
```

Parameters

Name	Value	Description
DXS_DCDC_UNDEF	-1	Undefined.
DXS_DCDC_IBB12	0	Inverting Buck-Boost Converter per channel with PNP switching transistor and typically 12 V input voltage.
DXS_DCDC_CIBB12	1	Combined Inverting Buck-Boost Converter. One converter supplies both channels and is using a PNP switching transistor with typically 12 V input voltage.
DXS_DCDC_IB12	2	Inverting Boost Converter per channel with NMOS switching transistor typically 12 V input voltage.

Name	Value	Description
DXS_DCDC_BB48	4	Buck-or-Boost Converter per channel with selectable buck or boost switching to connect to -48 V battery.
DXS_DCDC_CBB48	3	Combined Buck-or-Boost Converter with selectable buck or boost switching to connect to -48 V battery.
DXS_DCDC_IFB12	6	Inverting Flyback Converter per channel with flyback concept and NMOS switching transistor (typically 12 V input supply).
DXS_DCDC_CIFB12	7	Combined Inverting Flyback Converter per channel with flyback concept and NMOS switching transistor (typically 12 V input supply).
DXS_DCDC_IBGD12	8	Inverting Boost Converter with Gate Driver to control a standard level NMOS switching transistor for typically 12 V input voltage.
DXS_DCDC_IBVD3	10	Inverting Boost Converter with Voltage Doubler per channel with NMOS switching transistor for low input voltage (for example 3.3V).
DXS_DCDC_IFB12CH8	29	Single Inverting FlybackDC/DC Converter for 12 V shared for all 8 channels on four DXS devices with 85 V peak open circuit ringing capability.
DXS_DCDC_MAX		The last member.

3.2.3.8 DXS_HOOK_VALIDATION_TYPE_t

Description

Validation types used for the **DXS_HookValTime_t** structure.

Prototype

```
enum
{
        DXS_HOOK_VT_HOOKOFF_TIME = 0x0,
        DXS_HOOK_VT_HOOKON_TIME = 0x1,
        DXS_HOOK_VT_HOOKFLASH_TIME = 0x2,
        DXS_HOOK_VT_DIGITLOW_TIME = 0x3,
        DXS_HOOK_VT_DIGITHIGH_TIME = 0x4,
        DXS_HOOK_VT_INTERDIGIT_TIME = 0x5,
        DXS_HOOK_VT_FLASH_HOLDOFF_TIME = 0x6
} DXS_HOOK_VALIDATION_TYPE_t;
```

Parameters

Name	Value	Description
DXS_HOOK_VT_HOOKOFF_TIME	0x0	Settings for off-hook validation; an exception is raised in case the time reaches the nMinTime parameter. The parameter nMaxTime is unused.
DXS_HOOK_VT_HOOKON_TIME	0x1	Settings for on-hook validation; an exception is raised in case the time reaches the nMinTime parameter. The parameter nMaxTime is unused.

Name	Value	Description
DXS_HOOK_VT_HOOKFLASH_TIME	0x2	Settings for hook flash validation - also known as register recall. When the time value lies between the times defined in the fields nMinTime and nMaxTime, an exception is raised.
DXS_HOOK_VT_DIGITLOW_TIME	0x3	Settings for pulse digit low, open loop and make validation. The time value must lie between the times defined in the fields nMinTime and nMaxTime to allow recognition of a pulse dialing event.
DXS_HOOK_VT_DIGITHIGH_TIME	0x4	Settings for pulse digit high, close loop and break validation. The time value must lie between the times defined in the fields nMinTime and nMaxTime to allow recognition of a pulse dialing event.
DXS_HOOK_VT_INTERDIGIT_TIME	0x5	Settings for pulse digit pause; the time must reach the nMinTime parameter to allow recognition of a pulse dialing event. The parameter nMaxTime is unused.
DXS_HOOK_VT_FLASH_HOLDOFF_TIME	0x6	Setting for hook flash holdoff period. The nMinTime defines the period after a flash break during which no further hook event must occur so that the flash is reported. The parameter nMaxTime is unused.

3.2.3.9 DXS_ACLM_Measurement_t

Description

AC Level Meter measurement types.

Prototype

```
enum
{
        DXS_ACLM_FR = 1,
        DXS_ACLM_TH = 2,
        DXS_ACLM_GT = 3,
        DXS_ACLM_SNR = 4
} DXS_ACLM_Measurement_t;
```

Parameters

Name	Value	Description
DXS_ACLM_FR	1	Frequency Response measurement.
DXS_ACLM_TH	2	Transhybrid measurement.
DXS_ACLM_GT	3	Gain Tracking measurement.
DXS_ACLM_SNR	4	Signal to Noise Ratio measurement.

3.2.3.10 DXS_CID_Std_t

Description

Supported CID standards.

Prototype

```
enum
{
        DXS_CID_STD_UNDEF = 0,
        DXS_CID_STD_ETSI = 1,
        DXS_CID_STD_TELCORDIA = 2,
        DXS_CID_STD_BT = 3,
        DXS_CID_STD_NTT = 4
} DXS_CID_Std_t;
```

Parameters

Name	Value	Description
DXS_CID_STD_UNDEF	0	Undefined.
DXS_CID_STD_ETSI	1	ETSI ES 200 778 v1.2.2.
DXS_CID_STD_TELCORDIA	2	Telcordia GR-30-CORE.
DXS_CID_STD_BT	3	SIN 227 for BT Network.
DXS_CID_STD_NTT	4	NTT (Japan)

3.2.3.11 DXS_CID_AS_t

Description

CID Alert signals.

Prototype

```
enum
{
        DXS_CID_AS_NONE = 0,
        DXS_CID_AS_DTAS = 1,
        DXS_CID_AS_LR_DTAS = 2,
        DXS_CID_AS_RPAS = 3,
        DXS_CID_AS_CAR = 4,
        DXS_CID_AS_OSI = 5
} DXS_CID_AS_t;
```

Parameters

Name	Value	Description
DXS_CID_AS_NONE	0	No alert.
DXS_CID_AS_DTAS	1	DTAS alert.
DXS_CID_AS_LR_DTAS	2	DTAS alert after Line Reversal.

Name	Value	Description
DXS_CID_AS_RPAS	3	RPAS alert.
DXS_CID_AS_CAR	4	CAR alert.
DXS_CID_AS_OSI	5	OSI alert.

3.2.3.12 DXS_CID_DATA_FMT_t

Description

CID data transmission format.

Prototype

```
enum
{
        DXS_CID_DATA_UNDEF = 0,
        DXS_CID_DATA_FSK = 1,
        DXS_CID_DATA_DTMF = 2
} DXS_CID_DATA_FMT_t;
```

Parameters

Name	Value	Description
DXS_CID_DATA_UNDEF	0	Undefined.
DXS_CID_DATA_FSK	1	FSK.
DXS_CID_DATA_DTMF	2	DTMF.

3.2.3.13 DXS_CID_MsgType_t

Description

Enumeration for CID message type.

Prototype

```
enum
{
        DXS_CID_MSG_TYPE_ND = 0x40,
        DXS_CID_MSG_TYPE_CS = 0x80,
        DXS_CID_MSG_TYPE_MWI = 0x82,
        DXS_CID_MSG_TYPE_AOC = 0x86,
        DXS_CID_MSG_TYPE_SMS = 0x89,
        DXS_CID_MSG_TYPE_RESERVED = 0xF1
} DXS_CID_MsgType_t;
```

Parameters

Name	Value	Description
DXS_CID_MSG_TYPE_ND	0x40	Number Display.
DXS_CID_MSG_TYPE_CS	0x80	Call Set-up.

Name	Value	Description
DXS_CID_MSG_TYPE_MWI	0x82	Message Waiting Indicator.
DXS_CID_MSG_TYPE_AOC	0x86	Advice of Charge.
DXS_CID_MSG_TYPE_SMS	0x89	Short Message Service.
DXS_CID_MSG_TYPE_RESERVED	0xF1	Reserved for Network Operator use.

3.2.3.14 DXS_CID_MsgParam_t

Description

Enumeration for CID parameters.

Prototype

```
enum
{
        DXS_CID_MSG_PARAM_DATE_TIME = 0x01,
        DXS_CID_MSG_PARAM_CALLING_LINE_ID = 0x02,
        DXS_CID_MSG_PARAM_CALLED_LINE_ID = 0x03,
        DXS_CID_MSG_PARAM_RA_CALLING_LINE_ID = 0x04,
        DXS_CID_MSG_PARAM_CALLING_PARTY_NAME = 0x07,
        DXS_CID_MSG_PARAM_RA_CALLING_PARTY_NAME = 0x08,
        DXS_CID_MSG_PARAM_VI = 0x0B,
        DXS_CID_MSG_PARAM_MSG_ID = 0x0D,
        DXS_CID_MSG_PARAM_LM_CLI = 0x0E,
        DXS_CID_MSG_PARAM_COMP_DATE_TIME = 0x0F,
        DXS_CID_MSG_PARAM_COMP_CALLING_LINE_ID = 0x10,
        DXS_CID_MSG_PARAM_CALL_TYPE = 0x11,
        DXS_CID_MSG_PARAM_FIRST_CALLED_LINE_ID = 0x12,
        DXS_CID_MSG_PARAM_NUM_MESSAGES = 0x13,
        DXS_CID_MSG_PARAM_TYPE_FWD_CALL = 0x15,
        DXS_CID_MSG_PARAM_TYPE_CALLING_USER = 0x16,
        DXS_CID_MSG_PARAM_REDIR_NUM = 0x1A,
        DXS_CID_MSG_PARAM_CHARGE = 0x20,
        DXS_CID_MSG_PARAM_ADDON_CHARGE = 0x21,
        DXS_CID_MSG_PARAM_DURA_CALL = 0x23,
        DXS_CID_MSG_PARAM_NET_PROV_ID = 0x30,
        DXS_CID_MSG_PARAM_CARRIER_ID = 0x31,
        DXS_CID_MSG_PARAM_SEL_TERMINAL_FUNC = 0x40,
        DXS_CID_MSG_PARAM_DISP_INFO = 0x50,
        DXS_CID_MSG_PARAM_SERV_INFO = 0x55,
        DXS_CID_MSG_PARAM_EXTENSION = 0xE0,
        DXS_CID_MSG_PARAM_RESERVED = 0xE1
} DXS_CID_MsgParam_t;
```

Parameters

Name	Value	Description
DXS_CID_MSG_PARAM_DATE_TIME	0x01	Date and Time.
DXS_CID_MSG_PARAM_CALLING_LINE_ID	0x02	Calling Line Identity.
DXS_CID_MSG_PARAM_CALLED_LINE_ID	0x03	Called Line Identity.
DXS_CID_MSG_PARAM_RA_CALLING_LINE_ID	0x04	Reason for Absence of Calling Line Identity.
DXS_CID_MSG_PARAM_CALLING_PARTY_NAME	0x07	Calling Party Name.
DXS_CID_MSG_PARAM_RA_CALLING_PARTY_NAME	0x08	Reason for absence of Calling Party Name.
DXS_CID_MSG_PARAM_VI	0x0B	Visual Indicator.
DXS_CID_MSG_PARAM_MSG_ID	0x0D	Message Identification.
DXS_CID_MSG_PARAM_LM_CLI	0x0E	Last Message CLI.
DXS_CID_MSG_PARAM_COMP_DATE_TIME	0x0F	Complementary Date and Time.
DXS_CID_MSG_PARAM_COMP_CALLING_LINE_ID	0x10	Complementary Calling Line Identity.
DXS_CID_MSG_PARAM_CALL_TYPE	0x11	Call type.
DXS_CID_MSG_PARAM_FIRST_CALLED_LINE_ID	0x12	First Called Line Identity.
DXS_CID_MSG_PARAM_NUM_MESSAGES	0x13	Number of Messages.
DXS_CID_MSG_PARAM_TYPE_FWD_CALL	0x15	Type of Forwarded call.
DXS_CID_MSG_PARAM_TYPE_CALLING_USER	0x16	Type of Calling user.
DXS_CID_MSG_PARAM_REDIR_NUM	0x1A	Redirecting Number.
DXS_CID_MSG_PARAM_CHARGE	0x20	Charge.
DXS_CID_MSG_PARAM_ADDON_CHARGE	0x21	Additional Charge.
DXS_CID_MSG_PARAM_DURA_CALL	0x23	Duration of the Call.
DXS_CID_MSG_PARAM_NET_PROV_ID	0x30	Network Provider Identity.
DXS_CID_MSG_PARAM_CARRIER_ID	0x31	Carrier Identity.
DXS_CID_MSG_PARAM_SEL_TERMINAL_FUNC	0x40	Selection Of Terminal Function.
DXS_CID_MSG_PARAM_DISP_INFO	0x50	Display Information.
DXS_CID_MSG_PARAM_SERV_INFO	0x55	Service Information.
DXS_CID_MSG_PARAM_EXTENSION	0xE0	Extension for network operator use.
DXS_CID_MSG_PARAM_RESERVED	0xE1	Reserved for network operator use.

3.2.3.15 DXS_status_t

Description

Definition of all error codes.

Prototype

```
enum
{
        DXS_statusError = -1,
        DXS_statusOk = 0,
        DXS_statusDevNotFound = 0x4001,
        DXS_statusChannelNotFound = 0x4002,
        DXS_statusDevInitFailed = 0x4003,
        DXS_statusFeatNotCompiledIn = 0x4004,
        DXS_statusFeatNotSupportedCaps = 0x4005,
        DXS_statusRegReadError = 0x4006,
        DXS_statusRegWriteError = 0x4007,
        DXS_statusThreadCreatError = 0x4008,
        DXS_statusThreadStopError = 0x4009,
        DXS_statusObxHandlerStartError = 0x400A,
        DXS_statusMsgQueueCreatError = 0x400B,
        DXS_statusInvalidParam = 0x400C,
        DXS_statusCmdWriteError = 0x400D,
        DXS_statusCmdReadError = 0x400E,
        DXS_statusPcmIfActError = 0x400F,
        DXS_statusPcmChActError = 0x4010,
        DXS_statusPcmChMuteError = 0x4011,
        DXS_statusPcmIfNotEnabled = 0x4012,
        DXS_statusPcmChNotDisabled = 0x4013,
        DXS_statusNotInitResource = 0x4014,
        DXS_statusCapVersUpdateError = 0x4015,
        DXS_statusSddOpmodeSetError = 0x4016,
        DXS_statusCmdMbxWriteError = 0x4017,
        DXS_statusCmdMbxWriteTmout = 0x4018,
        DXS_statusCmdLengthInvalid = 0x4019,
        DXS_statusCmdIbxNoSpace = 0x401A,
        DXS_statusMpoolInitError = 0x401B,
        DXS_statusSpiOpenError = 0x401C,
        DXS_statusSpiAccError = 0x401D,
        DXS_statusSpiTxLenError = 0x401E,
        DXS_statusCmdObDataOvld = 0x401F,
        DXS_statusCmdObRdErr = 0x4020,
        DXS_statusSetBootCfgErr = 0x4021,
        DXS_statusCtrlResErr = 0x4022,
        DXS_statusFwDwldTimeout = 0x4023,
        DXS_statusDwldBinErr = 0x4024,
        DXS_statusCmdLengthErr = 0x4025,
        DXS_statusWaitListObjInv = 0x4026,
        DXS_statusWaitListDevAddErr = 0x4027,
        DXS_statusCmdObDataRdTmout = 0x4027,
```

```
DXS_statusBbdDCDCHwCfgMismatch = 0x4029,
DXS_statusBbdMasterBlkInvalid = 0x402A,
DXS_statusBbdUnknownBlk = 0x402B,
DXS_statusBbdUnknownDcDcOpt = 0x402C,
DXS_statusBbdDcDcReconfig = 0x402D,
DXS_statusDevNotInitialized = 0x402E,
DXS_statusPramPatchNotDownloaded = 0x402F,
DXS_statusBbdNotDownloaded = 0x4030,
DXS_statusParamsOutRange = 0x4031,
DXS_statusMwlNotAllowed = 0x4032,
DXS_statusSddEvtWaitError = 0x4033,
DXS_statusSddEvtWaitTmout = 0x4034,
DXS_statusGpioNotConfigured = 0x4035,
DXS_statusGpioWriteAccessIgnored = 0x4036,
DXS_statusLineModeInvalidTransition = 0x4037,
DXS_statusMeteringLineModeNotActive = 0x4038,
DXS_statusMeteringPreviousPulseNotFinished = 0x4039,
DXS_statusWaitingLineToRecover = 0x403A,
DXS_statusCalVersInvalid = 0x403B,
DXS_statusAutomaticCalibrationFailed = 0x403C,
DXS_statusCalInProgress = 0x403D,
DXS_statusTimerCreateError = 0x403E,
DXS_statusDialZeroParam = 0x403F,
DXS_statusDialMaxMinParam = 0x4040,
DXS_statusDialTypeParam = 0x4041,
DXS_statusFdOpenError = 0x4042,
DXS_statusIntConfError = 0x4043,
DXS_statusFskEnableNotAllowed = 0x4044,
DXS_statusFskStandardChangeNotAllowed = 0x4045,
DXS_statusUtdToneIdxMissing = 0x4046,
DXS_statusOLCalibrationInProgress = 0x4047,
DXS_statusOLCalibrationNoResults = 0x4048,
DXS_statusAclmInbCalcError = 0x4049,
DXS_statusAclmOutbCalcError = 0x404A,
DXS_statusAclmResultsNotAvail = 0x404B,
DXS_statusAclmInProgress = 0x404C,
DXS_statusAclmStartErrInvOpmode = 0x404D,
DXS_statusAclmStartErrActOpmodeTmout = 0x404E,
DXS_statusOLCalibrationWaitError = 0x404F,
DXS_statusCidStdNotSupported = 0x4050,
DXS_statusCidMsgNotInit = 0x4051,
DXS_statusCidInvalidLineMode = 0x4052,
DXS_statusCidInvalidAlertSignal = 0x4053,
DXS_statusCidInProgress = 0x4054,
DXS_statusPramPatchCksumError = 0x4055,
DXS_statusPramPatchInvalid = 0x4056,
DXS_statusPramPatchDwldFail = 0x4057,
DXS_statusCidMsgLenWrong = 0x4058,
DXS_statusBbdDCDCMasterNotSet = 0x4059,
DXS_statusDcDcRingCapsExceeded = 0x405A,
DXS_statusRingCadConfInvalid = 0x405B,
```

```
        DXS_statusRingCadConfNotPossible = 0x405C,
        DXS_statusRingCadStartNotConf = 0x405D,
        DXS_statusSleepModeNotAllowed = 0x405E,
        DXS_statusEventEnableNotAllowed = 0x405F,
        DXS_statusEventDisableNotAllowed = 0x4060
} DXS_status_t;
```

Parameters

Name	Value	Description
DXS_statusError	-1	Error.
DXS_statusOk	0	Success, no error occurred.
DXS_statusDevNotFound	0x4001	Device not found.
DXS_statusChannelNotFound	0x4002	Channel not found.
DXS_statusDevInitFailed	0x4003	Device initialization failed.
DXS_statusFeatNotCompiledIn	0x4004	Feature is not compiled in (disabled at compile time).
DXS_statusFeatNotSupportedCaps	0x4005	Feature is not supported by device capabilities.
DXS_statusRegReadError	0x4006	Register read operation failed.
DXS_statusRegWriteError	0x4007	Register write operation failed.
DXS_statusThreadCreatError	0x4008	Thread creation failed.
DXS_statusThreadStopError	0x4009	Thread termination failed.
DXS_statusObxHandlerStartError	0x400A	Outbox data handler start failed.
DXS_statusMsgQueueCreatError	0x400B	Message queue creation failed.
DXS_statusInvalidParam	0x400C	Invalid parameters.
DXS_statusCmdWriteError	0x400D	Firmware command write operation failed.
DXS_statusCmdReadError	0x400E	Firmware command read operation failed.
DXS_statusPcmIfActError	0x400F	PCM interface activation failed.
DXS_statusPcmChActError	0x4010	PCM channel activation/deactivation failed.
DXS_statusPcmChMuteError	0x4011	PCM channel mute/unmute failed.
DXS_statusPcmIfNotEnabled	0x4012	PCM interface is not enabled.
DXS_statusPcmChNotDisabled	0x4013	PCM channels are not disabled.
DXS_statusNotInitResource	0x4014	Not initialized resource.
DXS_statusCapVersUpdateError	0x4015	Update of capabilities and version failed.
DXS_statusSddOpmodeSetError	0x4016	Line feeding mode set failed.
DXS_statusCmdMbxWriteError	0x4017	Writing command to device mailbox failed.
DXS_statusCmdMbxWriteTmout	0x4018	Timeout waiting for free space in command inbox.
DXS_statusCmdLengthInvalid	0x4019	Invalid value in length field of command.
DXS_statusCmdIbxNoSpace	0x401A	Not enough inbox space for writing command.
DXS_statusMpoolInitError	0x401B	Memory buffer pool initialization failed.
DXS_statusSpiOpenError	0x401C	OS SPI open error.
DXS_statusSpiAccError	0x401D	OS SPI access error.
DXS_statusSpiTxLenError	0x401E	OS SPI transmit length error.

Name	Value	Description
DXS_statusCmdObDataOvld	0x401F	More data in command outbox than expected.
DXS_statusCmdObRdErr	0x4020	Reading from the command outbox failed.
DXS_statusSetBootCfgErr	0x4021	Setting of boot configuration register failed.
DXS_statusCtrlResErr	0x4022	Controller reset failed.
DXS_statusFwDwldTimeout	0x4023	Firmware download timeout.
DXS_statusDwldBinErr	0x4024	Download of the firmware binary failed.
DXS_statusCmdLengthErr	0x4025	Command length error.
DXS_statusWaitListObjInv	0x4026	Reference object is not a waiting list.
DXS_statusWaitListDevAddErr	0x4027	It is not possible to add a device to the waiting list.
DXS_statusCmdObDataRdTmout	0x4027	Timeout waiting for command read response.
DXS_statusBbdDCDCHwCfgMismatch	0x4029	DC/DC HW option mismatch between BBD file and device configuration.
DXS_statusBbdMasterBlkInvalid	0x402A	Invalid master block in BBD file.
DXS_statusBbdUnknownBlk	0x402B	Unknown unsupported BBD block in BBD file.
DXS_statusBbdUnknownDcDcOpt	0x402C	Unknown unsupported DC/DC HW option in BBD file.
DXS_statusBbdDcDcReconfig	0x402D	Reconfiguration of BBD DC/DC config block is denied.
DXS_statusDevNotInitialized	0x402E	Device is not initialized.
DXS_statusPramPatchNotDownloaded	0x402F	Device PRAM patch is not downloaded.
DXS_statusBbdNotDownloaded	0x4030	BBD file is not downloaded.
DXS_statusParamsOutRange	0x4031	Parameters are out of range.
DXS_statusMwlNotAllowed	0x4032	Message Waiting Lamp is not allowed for this DC/DC variant.
DXS_statusSddEvtWaitError	0x4033	Error while waiting for SDD event.
DXS_statusSddEvtWaitTmout	0x4034	Timeout waiting for SDD event.
DXS_statusGpioNotConfigured	0x4035	Attempt to access GPIO write/read before port configuration.
DXS_statusGpioWriteAccessIgnored	0x4036	Attempt to perform write operation at GPIO configured as input.
DXS_statusLineModeInvalidTransition	0x4037	Invalid line mode state transition.
DXS_statusMeteringLineModeNotActive	0x4038	It is not possible to enable a Metering pulse when line mode is not active.
DXS_statusMeteringPreviousPulseNotFinished	0x4039	Metering pulse is enabled already.
DXS_statusWaitingLineToRecover	0x403A	Ground fault or over temperature detected on the line and not recovered yet.
DXS_statusCalVersInvalid	0x403B	Incorrect calibration version.
DXS_statusAutomaticCalibrationFailed	0x403C	Automatic calibration failed.
DXS_statusCalInProgress	0x403D	Current line mode is CALIBRATE.
DXS_statusTimerCreateError	0x403E	Timer creation failed.
DXS_statusDialZeroParam	0x403F	Value zero not allowed as hook state validation time.

Name	Value	Description
DXS_statusDialMaxMinParam	0x4040	Max must be larger than min hook state validation time.
DXS_statusDialTypeParam	0x4041	Unknown hook state validation type parameter.
DXS_statusFdOpenError	0x4042	Opening file descriptor failed.
DXS_statusIntConfError	0x4043	Interrupt configuration failed.
DXS_statusFskEnableNotAllowed	0x4044	FSK enable while DTMF is running.
DXS_statusFskStandardChangeNotAllowed	0x4045	FSK change standard while FSK is enabled.
DXS_statusUtdToneIdxMissing	0x4046	Tone index does not exist in the tone table.
DXS_statusOLCalibrationInProgress	0x4047	OL Calibration is already running.
DXS_statusOLCalibrationNoResults	0x4048	C-Measurement or GR-909 results were not received for OL calibration.
DXS_statusAclmInbCalcError	0x4049	AC Level Meter measurement result inband calculation error.
DXS_statusAclmOutbCalcError	0x404A	AC Level Meter measurement result outband calculation error.
DXS_statusAclmResultsNotAvail	0x404B	AC Level Meter measurement results are not available.
DXS_statusAclmInProgress	0x404C	AC Level Meter measurement is already in progress.
DXS_statusAclmStartErrInvOpmode	0x404D	AC Level Meter measurement start error - invalid line mode.
DXS_statusAclmStartErrActOpmodeTmout	0x404E	AC Level Meter measurement start error - timeout switching to active line mode.
DXS_statusOLCalibrationWaitError	0x404F	Line feed mode did not return to disabled during OL calibration.
DXS_statusCidStdNotSupported	0x4050	Caller ID standard is not supported.
DXS_statusCidMsgNotInit	0x4051	Caller ID message is not initialized.
DXS_statusCidInvalidLineMode	0x4052	Caller ID invalid line mode.
DXS_statusCidInvalidAlertSignal	0x4053	Caller ID invalid alert signal type for standard.
DXS_statusCidInProgress	0x4054	Caller ID is already in progress.
DXS_statusPramPatchCksumError	0x4055	Device PRAM patch checksum error.
DXS_statusPramPatchInvalid	0x4056	Device PRAM patch format is invalid or not supported.
DXS_statusPramPatchDwldFail	0x4057	Device PRAM patch download failed.
DXS_statusCidMsgLenWrong	0x4058	Caller ID message length is incorrect.
DXS_statusBbdDCDCMasterNotSet	0x4059	Master channel required for DC/DC configuration download is not set.
DXS_statusDcDcRingCapsExceeded	0x405A	Ringing capabilities are exceeded for a DC/DC.
DXS_statusRingCadConfInvalid	0x405B	It is not possible to set the ringing cadence configuration (invalid content).
DXS_statusRingCadConfNotPossible	0x405C	It is not possible to set the ringing cadence configuration (another ringing cadence is running).
DXS_statusRingCadStartNotConf	0x405D	Ringing cadence start attempt while ringing cadence is not configured.

Name	Value	Description
DXS_statusSleepModeNotAllowed	0x405E	Setting of the Sleep mode is not allowed.
DXS_statusEventEnableNotAllowed	0x405F	Enabling event is not allowed.
DXS_statusEventDisableNotAllowed	0x4060	Disabling event is not allowed.

Literature References

[1] DXS102/DXS101 V1.2/V1.3 Data Sheet Rev. 2.0

[2] DXS101 (PEF32001VSV12/PEF32001VSV13) Data Sheet Rev. 2.0

[3] DXS User's Manual System Description Rev. 2.2

[4] DXS/DXC System Package 4.0 Release Notes Rev. 1.0

Attention: Refer to the latest revision of the documents.

Standards References

[5] ETSI Standard ES 200 778, Access and Terminals (AT); Analogue access to the Public Switched Telephone Network (PSTN); Protocol over the local loop for display and related services; Terminal equipment requirements

[6] ETSI European Telecommunication Standard ETS 300 659, Access and Terminals (AT); Analogue access to the Public Switched Telephone Network (PSTN); Subscriber line protocol over the local loop for display (and related) services

[7] BT SIN 227, Suppliers' Information Note: CALLING LINE IDENTIFICATION SERVICE, SERVICE DESCRIPTION

[8] BT SIN 242, Suppliers' Information Note: CDS CALLING LINE IDENTIFICATION SERVICE, TERMINAL EQUIPMENT REQUIREMENTS

[9] Telcordia Technologies Generic Requirements GR-30-CORE, LSSGR: Voiceband Data Transmission Interface (FSD 05-01-0100)

[10] Nippon Telegraph and Telephone Corporation (NTT) Technical Reference, TELEPHONE SERVICE INTERFACES

Terminology

A	
ALM	Analog Line Module
API	Application Program Interface
B	
BBD	Block Based Download
BSP	Board Support Package
C	
CAR	Signal receiver seizing signal (NTT Caller ID protocol)
CFG	Configuration
CH	Channel
CID	Caller ID
CPE	Customer Premises Equipment
CPU	Central Processing Unit
D	
DC	Direct Current
DTAS	Dual Tone Alerting Signal
DTMF	Dual Tone Multiple Frequency
G	
GPIO	General Purpose Input Output
H	
HL	High Level
I	
IO	Input/Output
IOCTL	Input/Output Control
L	
LL	Low Level
M	
MWL	Message Waiting Lamp
N	
NTT	Nippon Telegraph and Telephone Corporation
P	
PCM	Pulse Code Modulation
R	
RPAS	Ringing Pulse Alerting Signal
RX	Receive
S	
SoC	System on Chip
SPI	Serial Peripheral Interface

T

TAPI	Telephone API
TX	Transmit

V

VoIP	Voice over IP