July 2006

TAPI

Telephone Application Programming Interface

Preliminary

User's Manual

Programmer's Reference
Revision 1.1

Communication Solutions

infineon

Never stop thinking

Edition 2006-07-11

Published by
Infineon Technologies AG
81726 München, Germany

Legal Disclaimer

The information given in this document shall in no event be regarded as a guarantee of conditions or characteristics (“Beschaffenheitsgarantie”). With respect to any examples or hints given herein, any typical values stated herein and/or any information regarding the application of the device, Infineon Technologies hereby disclaims any and all warranties and liabilities of any kind, including without limitation warranties of non-infringement of intellectual property rights of any third party.

Information

For further information on technology, delivery terms and conditions and prices please contact your nearest Infineon Technologies Office (**www.infineon.com**).

Warnings

Due to technical requirements components may contain dangerous substances. For information on the types in question please contact your nearest Infineon Technologies Office.

Infineon Technologies Components may only be used in life-support devices or systems with the express written approval of Infineon Technologies, if a failure of such components can reasonably be expected to cause the failure of that life-support device or system, or to affect the safety or effectiveness of that device or system. Life support devices or systems are intended to be implanted in the human body, or to support and/or maintain and sustain and/or protect human life. If they fail, it is reasonable to assume that the health of the user or other persons may be endangered.

Revision History: 2006-07-11, Revision 1.1

Previous Version: Revision 1.0, 2006-05-23.

Page	Subjects (major changes since last revision)
Page 18	Chapter renamed and restructured, Chapter 2.1.
Page 22	Added information about packet handling, Chapter 2.1.3.
Page 25	Improved description of event reporting interfaces, Chapter 2.5.
Page 45	Added description of LEC interfaces supporting window based LEC implementation, Chapter 3.1.4.
Page 59	Added new timing diagrams for called ID, Chapter 3.6.2
Page 73	Added description of fax/modem support, Chapter 3.7.
Page 77	Improved parameter description of most ioctls, Chapter 4.1.
Page 199	Modified the default values for FSK transmission and reception, Chapter 4.2.4.7.
	Editorial changes throughout the entire document.

Table of Contents

Table of Contents 4
List of Figures 11
List of Tables 12
Preface 14
1 Development Environment Setup 15
1.1 Introduction to the TAPI V3.x 15
1.2 Compilation 16
1.2.1 Linux 16
1.2.1.1 Loading of the TAPI Modules and Registration 16
1.2.1.1.1 Support of proc File System 17
1.2.2 VxWorks 17
2 First Steps 18
2.1 Driver Interfaces 18
2.1.1 File Descriptors 18
2.1.2 Phone and Data Channel Resources 20
2.1.3 Packet Handling 22
2.2 Initialization 22
2.2.1 Basic Device Driver Init 22
2.2.2 TAPI Channel Initialization 22
2.3 Set-up Line Feeding Modes 23
2.4 Set-up Audio Modes 24
2.5 Event Reporting Interfaces 25
2.5.1 Event Message Format 25
2.5.2 Enable/Disable Event Reporting 26
2.5.3 Examples of Event Reporting and Masking 27
2.5.4 Old Interfaces 30
2.6 Configure Hook Validation Timing 30
2.7 Establishing Connections 31
2.7.1 IP Phone Applications 33
2.7.1.1 Voice Call 33
2.7.1.2 In-call Announcement 34
2.7.2 Gateway-like Applications 35
2.7.2.1 Internal Call 35
2.7.2.2 Voice Call with External VoIP Party 36
2.8 Conferencing 38
2.8.1 Initialization 38
2.8.2 Conference with an Internal and an External VoIP Party 38
2.8.3 Conference with Two VoIP Parties 40
3 Feature Description 42
3.1 Channel Configuration 42
3.1.1 Analog Line Channel Volume 42
3.1.2 Audio Channel Volume and Mute 43
3.1.3 PCM 44
3.1.3.1 Example of System using PCM Interface 44
3.1.4 LEC 45
3.1.5 Jitter Buffer 46

3.1.6 RTP . . . 47
3.1.6.1 RTP Payload Type Tables . . . 47
3.1.6.2 RTP Session Parameters . . . 48
3.2 Encoder/Decoder Control . . . 48
3.3 RTCP and Jitter Buffer Statistics . . . 49
3.4 Tone API . . . 50
3.4.1 Tone Table . . . 50
3.4.2 Playing Tones . . . 51
3.4.3 Defining Simple Tones . . . 52
3.4.4 Defining Composed Tones . . . 53
3.4.5 Predefined Tones . . . 54
3.4.6 Generating Tones with High-level Output . . . 55
3.4.7 Generating Special Tones . . . 56
3.4.8 Call Progress Tone Detection . . . 56
3.5 IP Phone Ringing . . . 57
3.6 Power Ringing and Caller ID Support . . . 57
3.6.1 Introduction to Power Ringing and Caller ID Features . . . 57
3.6.1.1 Information on Caller ID . . . 58
3.6.2 CID Configuration . . . 59
3.6.3 Ringing Support . . . 66
3.6.3.1 Configure Ringing Type . . . 66
3.6.3.2 Configure Ring Cadence . . . 66
3.6.3.3 Start / Stop Ringing . . . 66
3.6.4 CID TX . . . 67
3.6.4.1 Prepare CID Message . . . 67
3.6.4.2 Examples of CID TX . . . 68
3.6.5 CID RX - FSK Receiver . . . 72
3.7 Fax/Modem Support . . . 73
3.7.1 Detect Fax/Modem Signals . . . 73
3.7.2 Pass-Through Mode . . . 74
3.7.3 T.38 Data-Pump Mode . . . 75

4 TAPI Interfaces . . . 77
4.1 ioctl Commands of TAPI Interfaces . . . 77
4.1.1 CID Features Service . . . 78
4.1.1.1 IFX_TAPI_CID_CFG_SET . . . 79
4.1.1.2 IFX_TAPI_CID_RX_DATA_GET . . . 80
4.1.1.3 IFX_TAPI_CID_RX_START . . . 81
4.1.1.4 IFX_TAPI_CID_RX_STATUS_GET . . . 82
4.1.1.5 IFX_TAPI_CID_RX_STOP . . . 83
4.1.1.6 IFX_TAPI_CID_TX_INFO_START . . . 83
4.1.1.7 IFX_TAPI_CID_TX_SEQ_START . . . 85
4.1.2 Connection Control Service . . . 86
4.1.2.1 IFX_TAPI_DEC_LEVEL_GET . . . 87
4.1.2.2 IFX_TAPI_DEC_START . . . 88
4.1.2.3 IFX_TAPI_DEC_STOP . . . 89
4.1.2.4 IFX_TAPI_DEC_TYPE_SET . . . 89
4.1.2.5 IFX_TAPI_DEC_VOLUME_SET . . . 90
4.1.2.6 IFX_TAPI_ENC_FRAME_LEN_GET . . . 91
4.1.2.7 IFX_TAPI_ENC_FRAME_LEN_SET . . . 92
4.1.2.8 IFX_TAPI_ENC_LEVEL_SET . . . 92
4.1.2.9 IFX_TAPI_ENC_START . . . 93

4.1.2.10 IFX_TAPI_ENC_STOP . . . 94
4.1.2.11 IFX_TAPI_ENC_TYPE_SET . . . 95
4.1.2.12 IFX_TAPI_ENC_VAD_CFG_SET . . . 95
4.1.2.13 IFX_TAPI_ENC_VOLUME_SET . . . 96
4.1.2.14 IFX_TAPI_JB_CFG_SET . . . 97
4.1.2.15 IFX_TAPI_JB_STATISTICS_GET . . . 98
4.1.2.16 IFX_TAPI_JB_STATISTICS_RESET . . . 98
4.1.2.17 IFX_TAPI_MAP_DATA_ADD . . . 99
4.1.2.18 IFX_TAPI_MAP_DATA_REMOVE . . . 100
4.1.2.19 IFX_TAPI_MAP_PCM_ADD . . . 101
4.1.2.20 IFX_TAPI_MAP_PCM_REMOVE . . . 102
4.1.2.21 IFX_TAPI_MAP_PHONE_ADD . . . 103
4.1.2.22 IFX_TAPI_MAP_PHONE_REMOVE . . . 104
4.1.2.23 IFX_TAPI_PKT_AAL_CFG_SET . . . 104
4.1.2.24 IFX_TAPI_PKT_AAL_PROFILE_SET . . . 105
4.1.2.25 IFX_TAPI_PKT_RTCP_STATISTICS_GET . . . 106
4.1.2.26 IFX_TAPI_PKT_RTCP_STATISTICS_RESET . . . 107
4.1.2.27 IFX_TAPI_PKT_RTP_CFG_SET . . . 108
4.1.2.28 IFX_TAPI_PKT_RTP_PT_CFG_SET . . . 109
4.1.3 Dial Service . . . 111
4.1.3.1 IFX_TAPI_PKT_EV_OOB_SET . . . 111
4.1.3.2 IFX_TAPI_PULSE_ASCII_GET . . . 112
4.1.3.3 IFX_TAPI_PULSE_GET . . . 113
4.1.3.4 IFX_TAPI_PULSE_READY . . . 114
4.1.3.5 IFX_TAPI_TONE_DTMF_ASCII_GET . . . 115
4.1.3.6 IFX_TAPI_TONE_DTMF_GET . . . 115
4.1.3.7 IFX_TAPI_TONE_DTMF_READY_GET . . . 117
4.1.4 Fax T.38 Service . . . 119
4.1.4.1 IFX_TAPI_T38_DEMOD_START . . . 119
4.1.4.2 IFX_TAPI_T38_MOD_START . . . 120
4.1.4.3 IFX_TAPI_T38_STATUS_GET . . . 121
4.1.4.4 IFX_TAPI_T38_STOP . . . 122
4.1.5 Initialization Service . . . 124
4.1.5.1 IFX_TAPI_CH_INIT . . . 124
4.1.5.2 IFX_TAPI_DWLD . . . 125
4.1.6 Metering Service . . . 126
4.1.6.1 IFX_TAPI_METER_CFG_SET . . . 126
4.1.6.2 IFX_TAPI_METER_START . . . 127
4.1.6.3 IFX_TAPI_METER_STOP . . . 128
4.1.7 Miscellaneous Services . . . 129
4.1.7.1 IFX_TAPI_CAP_CHECK . . . 129
4.1.7.2 IFX_TAPI_CAP_LIST . . . 131
4.1.7.3 IFX_TAPI_CAP_NR . . . 132
4.1.7.4 IFX_TAPI_CH_STATUS_GET . . . 133
4.1.7.5 IFX_TAPI_DEBUG_REPORT_SET . . . 135
4.1.7.6 IFX_TAPI_EXCEPTION_GET . . . 136
4.1.7.7 IFX_TAPI_EXCEPTION_MASK . . . 137
4.1.7.8 IFX_TAPI_VERSION_CHECK . . . 138
4.1.7.9 IFX_TAPI_VERSION_GET . . . 139
4.1.8 Event Reporting Services . . . 140
4.1.8.1 IFX_TAPI_EVENT_GET . . . 140

4.1.8.2 IFX_TAPI_EVENT_ENABLE . . . 141
4.1.8.3 IFX_TAPI_EVENT_EXT_DTMF . . . 142
4.1.8.4 IFX_TAPI_EVENT_EXT_DTMF_CFG . . . 142
4.1.8.5 IFX_TAPI_EVENT_DISABLE . . . 143
4.1.9 Operation Control Service . . . 143
4.1.9.1 IFX_TAPI_LEC_PCM_CFG_GET . . . 144
4.1.9.2 IFX_TAPI_LEC_PCM_CFG_SET . . . 145
4.1.9.3 IFX_TAPI_LEC_PHONE_CFG_GET . . . 146
4.1.9.4 IFX_TAPI_LEC_PHONE_CFG_SET . . . 146
4.1.9.5 IFX_TAPI_LINE_FEED_SET . . . 147
4.1.9.6 IFX_TAPI_LINE_HOOK_STATUS_GET . . . 148
4.1.9.7 IFX_TAPI_LINE_HOOK_VT_SET . . . 149
4.1.9.8 IFX_TAPI_LINE_LEVEL_SET . . . 150
4.1.9.9 IFX_TAPI_PCM_VOLUME_SET . . . 151
4.1.9.10 IFX_TAPI_PHONE_VOLUME_SET . . . 152
4.1.9.11 IFX_TAPI_WLEC_PCM_CFG_GET . . . 152
4.1.9.12 IFX_TAPI_WLEC_PCM_CFG_SET . . . 153
4.1.9.13 IFX_TAPI_WLEC_PHONE_CFG_GET . . . 154
4.1.9.14 IFX_TAPI_WLEC_PHONE_CFG_SET . . . 154
4.1.10 PCM Service . . . 156
4.1.10.1 IFX_TAPI_PCM_ACTIVATION_GET . . . 156
4.1.10.2 IFX_TAPI_PCM_ACTIVATION_SET . . . 157
4.1.10.3 IFX_TAPI_PCM_CFG_GET . . . 157
4.1.10.4 IFX_TAPI_PCM_CFG_SET . . . 158
4.1.11 Power Ringing Service . . . 160
4.1.11.1 IFX_TAPI_RING_CADENCE_HR_SET . . . 160
4.1.11.2 IFX_TAPI_RING_CADENCE_SET . . . 161
4.1.11.3 IFX_TAPI_RING_CFG_GET . . . 162
4.1.11.4 IFX_TAPI_RING_CFG_SET . . . 163
4.1.11.5 IFX_TAPI_RING_START . . . 164
4.1.11.6 IFX_TAPI_RING_STOP . . . 164
4.1.12 Signal Detection Service . . . 166
4.1.12.1 IFX_TAPI_SIG_DETECT_DISABLE . . . 166
4.1.12.2 IFX_TAPI_SIG_DETECT_ENABLE . . . 167
4.1.12.3 IFX_TAPI_TONE_CPTD_START . . . 168
4.1.12.4 IFX_TAPI_TONE_CPTD_STOP . . . 169
4.1.13 Test Services . . . 170
4.1.13.1 IFX_TAPI_TEST_HOOKGEN . . . 170
4.1.13.2 IFX_TAPI_TEST_LOOP . . . 170
4.1.14 Tone Control Service . . . 172
4.1.14.1 IFX_TAPI_TONE_LOCAL_PLAY . . . 173
4.1.14.2 IFX_TAPI_TONE_NET_PLAY . . . 174
4.1.14.3 IFX_TAPI_TONE_STATUS_GET . . . 174
4.1.14.4 IFX_TAPI_TONE_STOP . . . 175
4.1.14.5 IFX_TAPI_TONE_TABLE_CFG_SET . . . 176
4.1.15 Audio Channel Control . . . 178
4.1.15.1 IFX_TAPI_AUDIO_MODE_SET . . . 178
4.1.15.2 IFX_TAPI_AUDIO_MUTE_SET . . . 179
4.1.15.3 IFX_TAPI_AUDIO_ICA_SET . . . 179
4.1.15.4 IFX_TAPI_AUDIO_RING_START . . . 180
4.1.15.5 IFX_TAPI_AUDIO_RING_STOP . . . 180

4.1.15.6 IFX_TAPI_AUDIO_ROOM_TYPE_SET . . . 181
4.1.15.7 IFX_TAPI_AUDIO_VOLUME_SET . . . 182
4.2 Type Definition Reference . . . 183
4.2.1 Basic Type Definitions . . . 183
4.2.1.1 IFX_return_t . . . 183
4.2.1.2 IFX_boolean_t . . . 183
4.2.1.3 IFX_uint8_t . . . 184
4.2.1.4 IFX_int8_t . . . 184
4.2.1.5 IFX_uint32_t . . . 184
4.2.1.6 IFX_int32_t . . . 184
4.2.1.7 IFX_uint16_t . . . 185
4.2.1.8 IFX_int16_t . . . 185
4.2.1.9 IFX_char_t . . . 185
4.2.1.10 IFX_void_t . . . 185
4.2.1.11 IFX_float_t . . . 185
4.2.1.12 IFX_operation_t . . . 186
4.2.2 IO-control Reference . . . 186
4.2.3 Constant Reference . . . 189
4.2.4 Structure Reference . . . 190
4.2.4.1 IFX_TAPI_CAP_t . . . 193
4.2.4.2 IFX_TAPI_CH_INIT_t . . . 193
4.2.4.3 IFX_TAPI_CH_STATUS_t . . . 194
4.2.4.4 IFX_TAPI_CID_ABS_REASON_t . . . 197
4.2.4.5 IFX_TAPI_CID_CFG_t . . . 198
4.2.4.6 IFX_TAPI_CID_DTMF_CFG_t . . . 198
4.2.4.7 IFX_TAPI_CID_FSK_CFG_t . . . 199
4.2.4.8 IFX_TAPI_CID_MSG_DATE_t . . . 200
4.2.4.9 IFX_TAPI_CID_MSG_STRING_t . . . 200
4.2.4.10 IFX_TAPI_CID_MSG_t . . . 201
4.2.4.11 IFX_TAPI_CID_MSG_TRANSPARENT_t . . . 201
4.2.4.12 IFX_TAPI_CID_MSG_VALUE_t . . . 202
4.2.4.13 IFX_TAPI_CID_RX_DATA_t . . . 202
4.2.4.14 IFX_TAPI_CID_RX_STATUS_t . . . 203
4.2.4.15 IFX_TAPI_CID_STD_ETSI_DTMF_t . . . 203
4.2.4.16 IFX_TAPI_CID_STD_ETSI_FSK_t . . . 205
4.2.4.17 IFX_TAPI_CID_STD_NTT_t . . . 205
4.2.4.18 IFX_TAPI_CID_STD_SIN_t . . . 206
4.2.4.19 IFX_TAPI_CID_STD_TELCORDIA_t . . . 207
4.2.4.20 IFX_TAPI_CID_TIMING_t . . . 208
4.2.4.21 IFX_TAPI_DWLD_t . . . 209
4.2.4.22 IFX_TAPI_EXCEPTION_BITS_t . . . 209
4.2.4.23 IFX_TAPI_EVENT_t . . . 211
4.2.4.24 IFX_TAPI_EVENT_DATA_DTMF_t . . . 211
4.2.4.25 IFX_TAPI_EVENT_DATA_EXT_KEYPAD_t . . . 212
4.2.4.26 IFX_TAPI_EVENT_DATA_FAX_SIG_t . . . 212
4.2.4.27 IFX_TAPI_EVENT_DATA_PULSE_t . . . 213
4.2.4.28 IFX_TAPI_EVENT_DATA_RFC2833_t . . . 213
4.2.4.29 IFX_TAPI_EVENT_DATA_TONE_t . . . 213
4.2.4.30 IFX_TAPI_EVENT_EXT_DTMF_t . . . 214
4.2.4.31 IFX_TAPI_EVENT_EXT_DTMF_CFG_t . . . 214
4.2.4.32 IFX_TAPI_JB_CFG_t . . . 215

4.2.4.33 IFX_TAPI_JB_STATISTICS_t . . . 215
4.2.4.34 IFX_TAPI_LEC_CFG_t . . . 218
4.2.4.35 IFX_TAPI_LINE_HOOK_VT_t . . . 218
4.2.4.36 IFX_TAPI_LINE_VOLUME_t . . . 219
4.2.4.37 IFX_TAPI_MAP_DATA_t . . . 220
4.2.4.38 IFX_TAPI_MAP_PCM_t . . . 221
4.2.4.39 IFX_TAPI_MAP_PHONE_t . . . 222
4.2.4.40 IFX_TAPI_METER_CFG_t . . . 223
4.2.4.41 IFX_TAPI_PCK_AAL_CFG_t . . . 223
4.2.4.42 IFX_TAPI_PCK_AAL_PROFILE_t . . . 224
4.2.4.43 IFX_TAPI_PCM_CFG_t . . . 224
4.2.4.44 IFX_TAPI_PKT_RTCP_STATISTICS_t . . . 225
4.2.4.45 IFX_TAPI_PKT_RTP_CFG_t . . . 226
4.2.4.46 IFX_TAPI_PKT_RTP_PT_CFG_t . . . 227
4.2.4.47 IFX_TAPI_RING_CADENCE_t . . . 227
4.2.4.48 IFX_TAPI_RING_CFG_t . . . 228
4.2.4.49 IFX_TAPI_SIG_DETECTION_t . . . 229
4.2.4.50 IFX_TAPI_T38_DEMOD_DATA_t . . . 229
4.2.4.51 IFX_TAPI_T38_MOD_DATA_t . . . 231
4.2.4.52 IFX_TAPI_T38_STATUS_t . . . 232
4.2.4.53 IFX_TAPI_TEST_LOOP_t . . . 233
4.2.4.54 IFX_TAPI_TONE_COMPOSED_t . . . 234
4.2.4.55 IFX_TAPI_TONE_CPTD_t . . . 234
4.2.4.56 IFX_TAPI_TONE_SIMPLE_t . . . 235
4.2.4.57 IFX_TAPI_WLEC_CFG_t . . . 237
4.2.4.58 IFX_TAPI_VERSION_t . . . 237
4.2.5 Union Reference . . . 237
4.2.5.1 IFX_TAPI_CID_MSG_ELEMENT_t . . . 238
4.2.5.2 IFX_TAPI_CID_STD_TYPE_t . . . 238
4.2.5.3 IFX_TAPI_EXCEPTION_t . . . 239
4.2.5.4 IFX_TAPI_EVENT_DATA_t . . . 239
4.2.5.5 IFX_TAPI_TONE_t . . . 240
4.2.6 Enumerator Reference . . . 240
4.2.6.1 DEV_ERR . . . 242
4.2.6.2 IFX_TAPI_AUDIO_MODE_t . . . 246
4.2.6.3 IFX_TAPI_AUDIO_ROOM_TYPE_t . . . 246
4.2.6.4 IFX_TAPI_CAP_PORT_t . . . 247
4.2.6.5 IFX_TAPI_CAP_SIG_DETECT_t . . . 247
4.2.6.6 IFX_TAPI_CAP_TYPE_t . . . 248
4.2.6.7 IFX_TAPI_CH_INIT_COUNTRY_t . . . 249
4.2.6.8 IFX_TAPI_CH_INIT_MODE_t . . . 249
4.2.6.9 IFX_TAPI_CID_ABSREASON_t . . . 250
4.2.6.10 IFX_TAPI_CID_ALERT_ETSI_t . . . 250
4.2.6.11 IFX_TAPI_CID_HOOK_MODE_t . . . 251
4.2.6.12 IFX_TAPI_CID_MSG_TYPE_t . . . 251
4.2.6.13 IFX_TAPI_CID_RX_ERROR_t . . . 253
4.2.6.14 IFX_TAPI_CID_RX_STATE_t . . . 253
4.2.6.15 IFX_TAPI_CID_SERVICE_TYPE_t . . . 253
4.2.6.16 IFX_TAPI_CID_STD_t . . . 255
4.2.6.17 IFX_TAPI_CID_VMWI_t . . . 255
4.2.6.18 IFX_TAPI_DATA_MAP_START_STOP_t . . . 256

4.2.6.19 IFX_TAPI_DEBUG_REPORT_SET_t . . . 256
4.2.6.20 IFX_TAPI_DIALING_STATUS_t . . . 257
4.2.6.21 IFX_TAPI_DWLD_TYPE_t . . . 257
4.2.6.22 IFX_TAPI_ENC_TYPE_t . . . 258
4.2.6.23 IFX_TAPI_ENC_VAD_t . . . 259
4.2.6.24 IFX_TAPI_EVENT_ID_t . . . 259
4.2.6.25 IFX_TAPI_EVENT_GET_RET_t . . . 262
4.2.6.26 IFX_TAPI_EVENT_TYPE_t . . . 263
4.2.6.27 IFX_TAPI_JB_LOCAL_ADAPT_t . . . 264
4.2.6.28 IFX_TAPI_JB_PKT_ADAPT_t . . . 265
4.2.6.29 IFX_TAPI_JB_TYPE_t . . . 265
4.2.6.30 IFX_TAPI_LEC_GAIN_t . . . 266
4.2.6.31 IFX_TAPI_LEC_NLP_t . . . 266
4.2.6.32 IFX_TAPI_LEC_TYPE_t . . . 267
4.2.6.33 IFX_TAPI_LINE_FEED_t . . . 267
4.2.6.34 IFX_TAPI_LINE_HOOK_STATUS_t . . . 268
4.2.6.35 IFX_TAPI_LINE_HOOK_VALIDATION_TYPE_t . . . 268
4.2.6.36 IFX_TAPI_LINE_LEVEL_t . . . 269
4.2.6.37 IFX_TAPI_LINE_STATUS_t . . . 270
4.2.6.38 IFX_TAPI_MAP_DATA_TYPE_t . . . 271
4.2.6.39 IFX_TAPI_MAP_DEC_t . . . 271
4.2.6.40 IFX_TAPI_MAP_ENC_t . . . 272
4.2.6.41 IFX_TAPI_MAP_TYPE_t . . . 272
4.2.6.42 IFX_TAPI_METER_MODE_t . . . 273
4.2.6.43 IFX_TAPI_PCM_RES_t . . . 273
4.2.6.44 IFX_TAPI_PKT_AAL_PROFILE_RANGE_t . . . 274
4.2.6.45 IFX_TAPI_PKT_EV_NUM_t . . . 274
4.2.6.46 IFX_TAPI_PKT_EV_OOB_t . . . 276
4.2.6.47 IFX_TAPI_PKT_EV_OOBPLAY_t . . . 276
4.2.6.48 IFX_TAPI_RING_CFG_MODE_t . . . 277
4.2.6.49 IFX_TAPI_RING_CFG_SUBMODE_t . . . 277
4.2.6.50 IFX_TAPI_RUNTIME_ERROR_t . . . 278
4.2.6.51 IFX_TAPI_SIG_t . . . 278
4.2.6.52 IFX_TAPI_T38_ERROR_t . . . 280
4.2.6.53 IFX_TAPI_T38_STATUS_t . . . 281
4.2.6.54 IFX_TAPI_T38_STD_t . . . 281
4.2.6.55 IFX_TAPI_TONE_CPTD_DIRECTION_t . . . 282
4.2.6.56 IFX_TAPI_TONE_FREQ_t . . . 282
4.2.6.57 IFX_TAPI_TONE_GROUP_t . . . 283
4.2.6.58 IFX_TAPI_TONE_MODULATION_t . . . 283
4.2.6.59 IFX_TAPI_TONE_TG_t . . . 284
4.2.6.60 IFX_TAPI_TONE_TYPE_t . . . 284
4.2.6.61 IFX_TAPI_WLEC_NLP_t . . . 285
4.2.6.62 IFX_TAPI_WLEC_TYPE_t . . . 285

References . . . 286

Terminology . . . 287

List of Figures

Figure 1 TAPI V3.x Architecture . . . 15
Figure 2 File Descriptors Example for a 4-channel ATA . . . 19
Figure 3 File Descriptors Example for a Four Channels IP Phone . . . 20
Figure 4 Example of File Descriptors and Resources (1) . . . 21
Figure 5 Example of File Descriptors and Resources (2) . . . 21
Figure 6 Sequence of Line Feeding Modes . . . 23
Figure 7 Sequence for Event Reporting . . . 25
Figure 8 One VoIP Calls . . . 33
Figure 9 In-call Announcement . . . 34
Figure 10 Internal Call . . . 35
Figure 11 Two Independent VoIP Calls . . . 36
Figure 12 External and Internal Conference . . . 39
Figure 13 VoIP Conference . . . 40
Figure 14 Connection of a DuSLIC via PCM Interface . . . 45
Figure 15 Simple and Composed Tones . . . 51
Figure 16 Overview of the Configuration Structure . . . 60
Figure 17 Timing for Telcordia and ETSI CID Type 1 with Transmission After First Ring . . . 61
Figure 18 Timing for some Possible ETSI CID Type 1 with Transmission Prior to Ringing . . . 62
Figure 19 Timing for NTT CID Type 1 with Transmission Prior to Ringing . . . 62
Figure 20 Timing for Telcordia MWI . . . 63
Figure 21 Timing for Telcordia CID Type 2 . . . 63
Figure 22 Timing for ETSI CID Type 2, Successful Transmission . . . 64
Figure 23 Timing for ETSI CID Type 2, Unsuccessful Transmission . . . 64
Figure 24 Timing for NTT CID Type 2 . . . 64

List of Tables

Table 1 Compiler Switches . . . 16
Table 2 Topics of Chapter 2 . . . 18
Table 3 Device Nodes for a System Including one VoIP Subsystem and One VINETIC®-2CPE . . . 19
Table 4 Event Message Format. . . . 26
Table 5 Event IDs Requiring the data Field . . . 26
Table 6 Event Enable/disable Examples . . . 27
Table 7 Event Detection Examples . . . 28
Table 8 Interfaces for Mapping Services . . . 32
Table 9 Topics of Chapter 3 . . . 42
Table 10 Topics of Chapter 3.1 . . . 42
Table 11 Topics of Chapter 3.4 . . . 50
Table 12 Tone Table - Predefined Tones . . . 54
Table 13 Topics of Chapter 3.4 . . . 57
Table 14 Caller ID Types . . . 58
Table 15 Caller ID Generation - On-hook CID (type 1) . . . 58
Table 16 Caller ID Generation - On-hook MWI . . . 58
Table 17 Caller ID Generation - Off-hook CID (type 2) and off-hook MWI . . . 59
Table 18 Caller ID Generation - Abbreviations . . . 59
Table 19 Relevant Timing Applicable to the Different Standards . . . 61
Table 20 Caller ID/MWI Type Selection . . . 68
Table 21 TAPI Interface Overview . . . 77
Table 22 IO-control Overview of CID Features . . . 78
Table 23 Structure Overview of CID Features . . . 78
Table 24 Union Overview of CID Features . . . 78
Table 25 Enumerator Overview of CID Features . . . 79
Table 26 IO-control Overview of Connection Control Service . . . 86
Table 27 Structure Overview of Connection Control Service . . . 87
Table 28 Enumerator Overview of Connection Control Service . . . 87
Table 29 IO-control Overview of Dial Service . . . 111
Table 30 IO-control Overview of Fax T.38 Service . . . 119
Table 31 Structure Overview of Fax T.38 Service . . . 119
Table 32 Enumerator Overview of Fax T.38 Service . . . 119
Table 33 IO-control Overview of Initialization Service . . . 124
Table 34 Structure Overview of Initialization Service . . . 124
Table 35 Enumerator Overview of Initialization Service . . . 124
Table 36 IO-control Overview of Metering Service . . . 126
Table 37 Structure Overview of Metering Service . . . 126
Table 38 IO-control Overview of Miscellaneous Services . . . 129
Table 39 Structure Overview of Miscellaneous Services . . . 129
Table 40 Union Overview of Miscellaneous Services . . . 129
Table 41 Enumerator Overview of Miscellaneous Service . . . 129
Table 42 IO-control Overview of Miscellaneous Services . . . 140
Table 43 Structure Overview of Miscellaneous Services . . . 140
Table 44 Union Overview of Miscellaneous Services . . . 140
Table 45 Enumerator Overview of Miscellaneous Service . . . 140
Table 46 IO-control Overview of Operation Control Service . . . 143
Table 47 Structure Overview of Operation Control Service . . . 144
Table 48 Enumerator Overview of Operation Control Service . . . 144
Table 49 IO-control Overview of PCM Service . . . 156

Table 50 IO-control Overview of Ringing Service 160
Table 51 Structure Overview of Ringing Service 160
Table 52 IO-control Overview of Signal Detection Service 166
Table 53 Structure Overview of Signal Detection Service 166
Table 54 Enumerator Overview of Signal Detection Service 166
Table 55 IO-control Overview of Tone Control Service 172
Table 56 Constant Overview of Tone Control Service 172
Table 57 Structure Overview of Tone Control Service 172
Table 58 Union Overview of Tone Control Service 172
Table 59 Enumerator Overview of Tone Control Service 172
Table 60 IO-control Overview of Tone Control Service 178
Table 61 Enumerator Overview of Tone Control Service 178
Table 62 IO-control Overview of TAPI Interfaces 186
Table 63 Constant Reference for TAPI Interfaces 189
Table 64 Structure Overview of TAPI Interfaces 190
Table 65 Union Overview of TAPI Interfaces 237
Table 66 Enumerator Overview of TAPI Interfaces 240

Preface

This document describes the Infineon Telephone API (TAPI) driver usage.

Organization of this Document

This document is divided into 4 chapters. It is organized as follows:

- **Chapter 1**, **Development Environment Setup**
  Overview of the TAPI device driver. Description of the device driver compilation and its configuration options.
- **Chapter 2**, **First Steps**
  First steps necessary for software setup and TAPI usage for some basic operations.
- **Chapter 3**, **Feature Description**
  Description of the TAPI services.
- **Chapter 4**, **TAPI Interfaces**
  Reference for the TAPI interfaces.

Applicability to Infineon Products

The TAPI is a software layer used to control telephony features in Infineon products of the VINETIC® and DuSLIC families as well as products including the Infineon VoIP Subsystem (for example INCA-IP2 and Danube).

TAPI includes services common across all product lines as well as application specific services.

The description in **Chapter 2** and **Chapter 3** introduces a rough classification of the TAPI services, giving an explicit indication of the applicability to the different scenarios. For details on features supported for a specific product please refer to the product release note.

Code Examples

This document includes several fragments of C pseudo-code used to explain usage of the interfaces. Compilable C code and error checking has not been used in order to reduce the examples' complexity. Being written in pseudo-code format, the examples are not compilable.

1 Development Environment Setup

This chapter describes how to set up the TAPI software development environment.

1.1 Introduction to the TAPI V3.x

TAPI support has been implemented in two layers: TAPI High Level (HL), abstracting the features up to a none device specific level, and TAPI Low Level (LL) implementing the device specific part (for example HW/FW access).

With the introduction of version 3.0, TAPI is able to support the VoIP function on multiple Infineon devices/families, including the latest IP-Phone device, VoIP processor and residential gateway SoC.

TAPI is able of supporting multiple Infineon devices belonging to different families. The most noticeable architectural change in the TAPI V3.x is delivering TAPI HL as a separate driver, the TAPI LL is implemented in a separate binary per supported device.

Both control and data paths use TAPI interfaces. **Figure 1** provides an overview of the TAPI architecture, in the particular configuration two different Infineon devices are controlled by TAPI[1]. As shown in the figure, three device drivers must be loaded. To be noted that some Infineon device drivers include device specific commands (such as device initialization) that, although not part of TAPI[2], are passed through the TAPI OS interface. A classification of TAPI and non-TAPI commands is done by the ioctl dispatcher (see **Figure 1**).

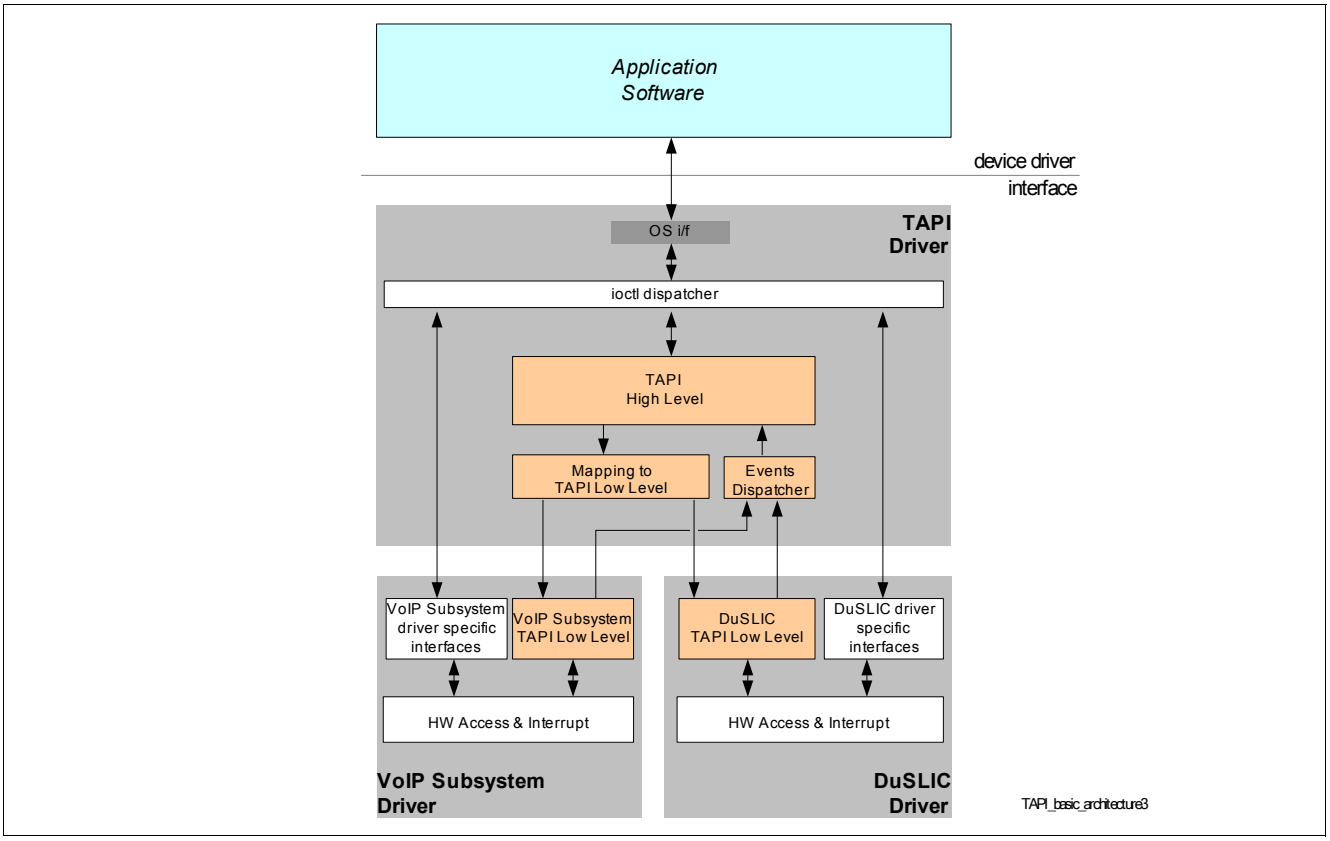

Figure 1 TAPI V3.x Architecture

1) TAPI controls the telephony features of the Infineon device.

2) And for this reason not described in this document, please refer to the device specific documentation.

1.2 Compilation

This chapter describes how to compile the TAPI device driver for Linux (kernal 2.4) and VxWorks. For Linux, the GNU toolchain (autoconf, automake) is used. For VxWorks, the Tornado project files are required.

To retrieve the device driver sources and to obtain the execution rights and directory structure, the following command has to be used. It will extract all sources into a subdirectory.

```
tar xvzf drv_tapi-3.1.x.x.tar.gz
```

1.2.1 Linux

Building the device driver is done in two steps:

- Go to the directory where you extracted the sources and type in `./configure` with the options described in **Table 1** and then
- Execute make or make install

Prerequisite are: the toolchain is in place, the path to the cross-compiler is included in the PATH and the availability of path to the Linux kernel header files.

Table 1 Compiler Switches

Option	Description	Always Required
--enable-debug --disable-debug	Enable/disable debug messages.	No
--enable-kernelincl	Set the Linux kernel include path.	Yes
--enable-lt --disable-lt	Enable/disable TAPI line testing services	No
--enable-voice --disable-voice	Enable/disable TAPI Voice support.[1)]	No
--enable-dtmf --disable-dtmf	Enable/disable TAPI DTMF detection support.[1)]	No
--enable-cid --disable-cid	Enable/disable TAPI Caller ID support.[1)]	No
--enable-fax --disable-fax	Enable/disable TAPI T.38 FAX support.[1)]	No
--enable-extkeypad	Enable TAPI EXT KEYPAD support.	No
--enable-audioch	Enable TAPI AUDIO CHANNEL support.	No
--enable-el_debug	Enable event logger debugging (requires additional eventlogger package).	No
--enable-udp-redirect	Enable QoS - quality of service and UDP redirection.	No
--enable-trace	Enable runtime traces.	No

1) Per default voice, dtmf, cid and fax are enabled. This will change in a next TAPI version.

1.2.1.1 Loading of the TAPI Modules and Registration

TAPI driver and low-level device drivers (including TAPI LL) are defined to be implemented as kernel modules, which can be inserted or removed from the kernel dynamically. The device drivers must be loaded after the High Level TAPI is loaded.

On insmod the version information of the Device driver is displayed on the console.

If CONFIG_DEVFS_FS is supported, device nodes are created by the High Level TAPI on insmod of the low-level driver. The template is /dev/<devName>/<deviceNumber><channelNumber>

Example - Registration

```
/* TAPI Module is built as "drv_tapi" */
# insmod drv_tapi

/* Now load TAPI LL part with default parameters: */
/* Use default major number and device node name */
/* drv_vmmc is the TAPI LL for the VoIP subsystem */
# insmod drv_vmmc

/* As alternative, TAPI LL is loaded using customer parameters */
/* major = device driver major number */
/* devName = device node name to be used */
# insmod drv_vmmc major=244 devName=vmmc
```

1.2.1.1.1 Support of proc File System

If CONFIG_PROC_FS is supported, the proc file system reports the list of successfully registered low-level device drivers and version of the TAPI.

Example - Proc File System

```
/* Retrieves the registered low level drivers, example */
# cat /proc/driver/tapi/registered_drivers

Driver          version       major    devices       devname
=======================================================
VMMC           0.9.2.2        230      1             /dev/vmmc

/* Retrieves the version information of High Level TAPI, example */
# cat /proc/driver/tapi/version
TAPI Driver, Version 3.2.0.1
Compiled on Feb 20 2006, 16:58:09 for Linux kernel 2.4.31-tqm-dpram-ralph
```

1.2.2 VxWorks

Not supported yet.

2 First Steps

This chapter introduces the first steps using the device driver interfaces, from the initialization of the device driver to some simple connection scenarios. A series of commented code examples is given to demonstrate the usage of the driver interfaces.

The topics described in this chapters are listed in **Table 2**.

Table 2 Topics of Chapter 2

Topic	Chapter	Applicability
Driver Interfaces	**Chapter 2.1**	All type of applications.
Initialization of device driver and channel	**Chapter 2.2**	All type of applications.
Set-up line feeding modes	**Chapter 2.3**	For ATA, and Gateway applications.
Set-up audio modes (handset, headset, etc) and acoustic echo canceller optimization.	**Chapter 2.4**	IP Phone application.
Reporting of events to the application software	**Chapter 2.5**	All type of applications.
Configure Hook Validation Timing	**Chapter 2.6**	For ATA and Gateway applications.
Establishment of connections and conferences	**Chapter 2.7**	All type of applications.
Establishment of conferences	**Chapter 2.8**	All type of applications.

2.1 Driver Interfaces

Communication between application software and the driver is achieved through chararacter device driver interfaces calls performed on specific file descriptors.

File descriptors are obtained through the POSIX-compliant open system call whose arguments specify the pathname to the device/channel special device file one wants to access.

The driver implements the following interfaces:

- open/close, to get/release a file descriptor, see also **Chapter 2.1.1** and **Chapter 2.1.2**.
- select, to block on interrupts/events, see also **Chapter 2.5**.
- read/write, to receive/send packets from/to the device, see also **Chapter 2.1.1**.
- ioctl, to configure and control the device. **Chapter 2** and **Chapter 3** describe how to use the ioctl commands. A reference of all supported commands is given in **Chapter 4.1**.

2.1.1 File Descriptors

Two types of file descriptors are defined: device-specific and channel-specific. As the name suggests, the device file descriptors (one per device) are used for device-wide control, whereas the channel file descriptors (one per device channel) are for channel-specific actions, a channel being a subset of the available device hardware and firmware resources.

The driver can handle one or more devices connected to the same host controller, several device nodes are specified to give the programmer the ability to address a specific device and its channels.

The channel file descriptor addresses a given device channel, **Table 3** shows an example of mapping resources-device nodes for a driver controlling two devices. Different channel-file descriptors address different resource sets, reflecting the modular nature of the products.

The device-specific nodes are of the format “/dev/*devname*X0”, where X corresponds to the device number. These device nodes are used for controlling the device as a whole. For example, in a system with two devices, /dev/*devname*10 addresses device number 1 and /dev/*devname*20 addresses device number 2.

The channel-specific nodes are of the format “/dev/*devname*XY”, addressing device number X, channel Y. For example, in a system with two devices, /dev/vin11 addresses device number 1/channel 1 and /dev/*devname*23 addresses device number 2 channel 3.

A channel file descriptor must be opened for read and write access, implementing packet upstream and packet downstream. Exceptions are reported on the device file descriptors via interface **IFX_TAPI_EVENT_GET**.

Table 3 Device Nodes for a System Including one VoIP Subsystem and One VINETIC®-2CPE

Device Node	Device Number	Addressed Channel	Included Resources
/dev/dan10	Danube Device 1	Entire device	VoIP subsystem chip
/dev/dan11	Danube Device 1	Channel 1	ALM0, SIG0, COD0, PCM0 resources
/dev/dan12	Danube Device 1	Channel 2	ALM1, SIG1, COD1, PCM1 resources
/dev/dan13	Danube Device 1	Channel 3	SIG2, COD2, PCM2 resources
/dev/dan14	Danube Device 1	Channel 4	SIG3, COD3, PCM3 resources
/dev/vin10	VINETIC® Device 1	Entire device	VINETIC chip
/dev/vin11	VINETIC® Device 1	Channel 1	ALM0, SIG0, COD0, PCM0 resources
/dev/vin12	VINETIC® Device 1	Channel 2	ALM1, SIG1, COD1, PCM1 resources
/dev/vin13	VINETIC® Device 1	Channel 3	SIG2, COD2, PCM2 resources
/dev/vin14	VINETIC® Device 1	Channel 4	SIG3, COD3, PCM3 resources

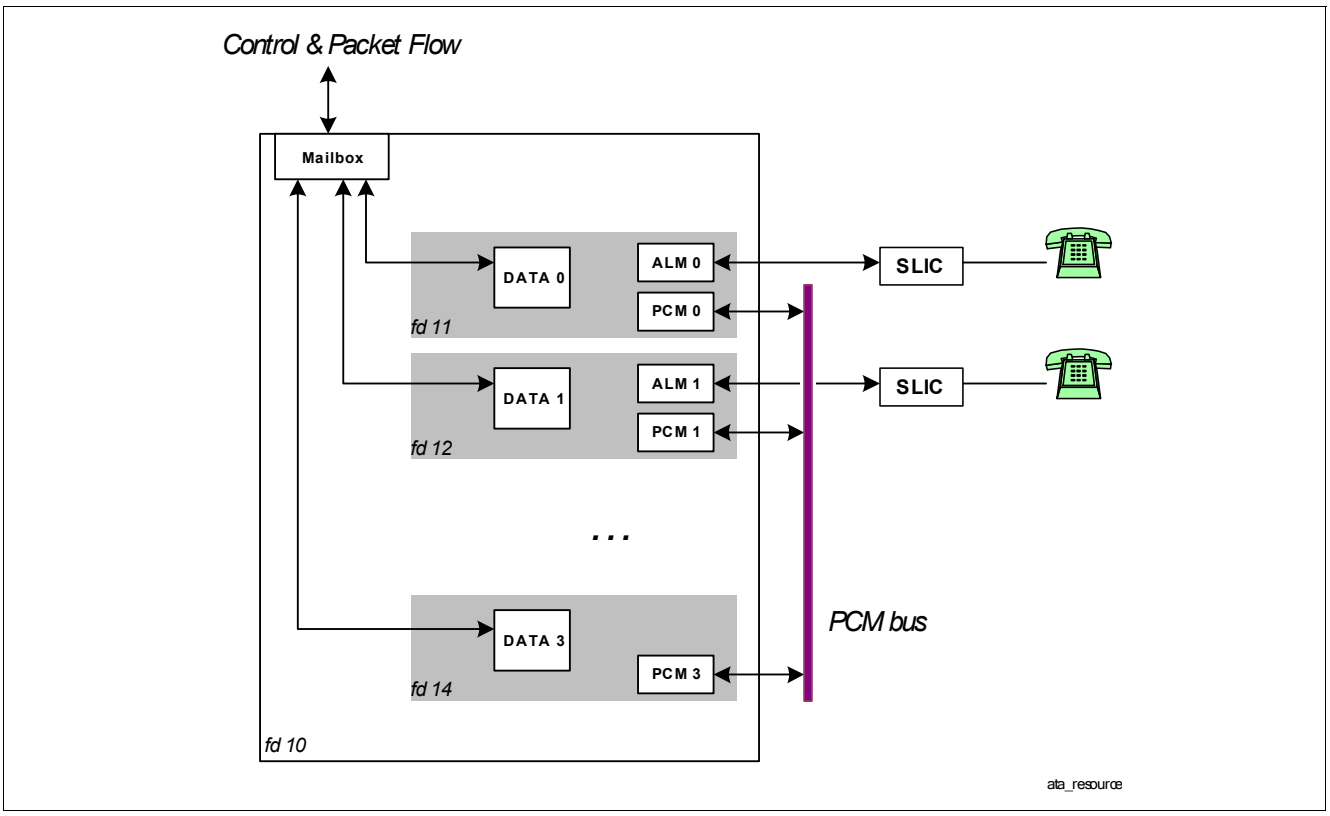

Figure 2 File Descriptors Example for a 4-channel ATA

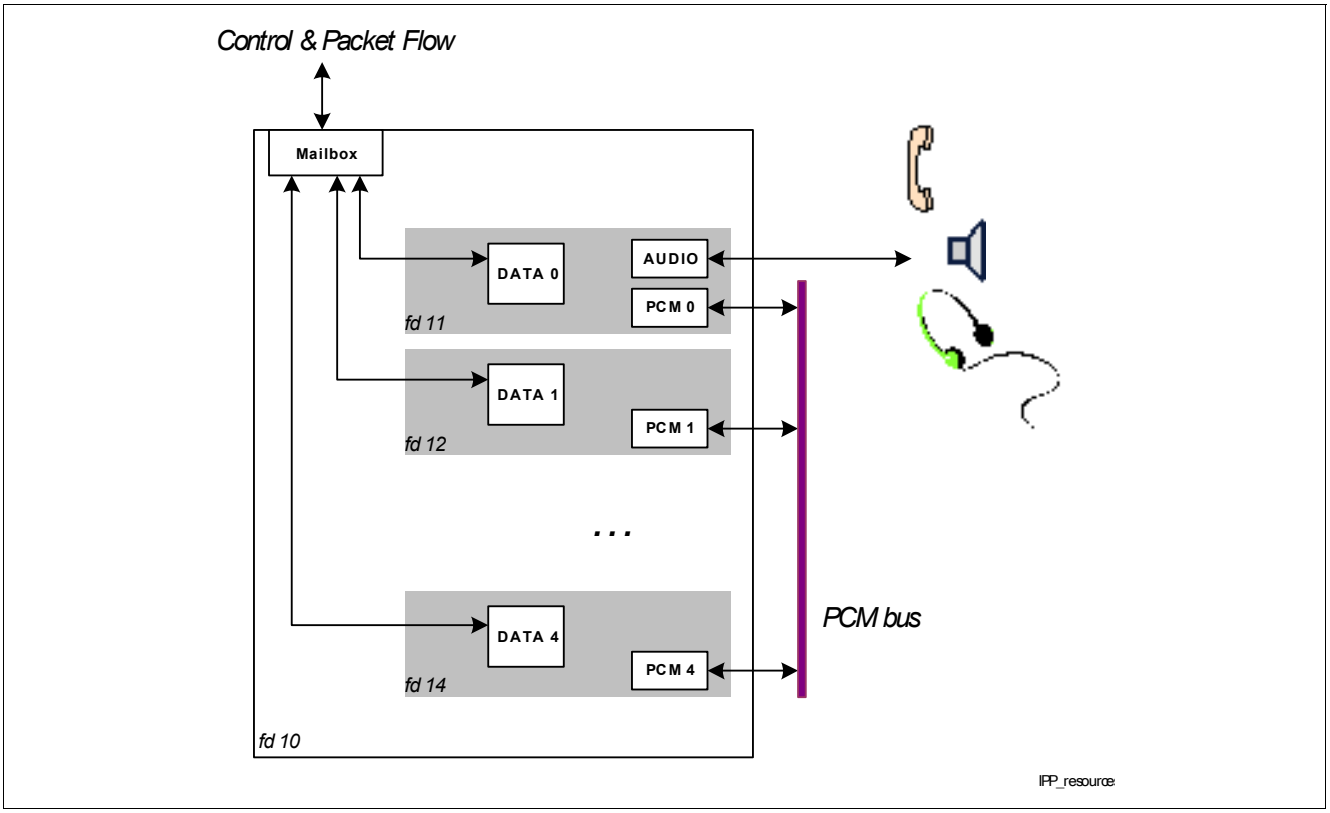

Figure 3 File Descriptors Example for a Four Channels IP Phone

2.1.2 Phone and Data Channel Resources

Within a channel, a distinction is made between “data channel” resources, in charge of complex signal processing (such as speech compression, signal generation/detection) and packetizations, and “phone channel” resources, seen as a kind of I/O port for the digitalized voice.

For ATA, and VoIP Gateway applications PCM and analog line module (alias ALM) resources belong to phone channels while SIG and COD resources belong to data channels. Difference for IP Phone being that an Audio Channel is available instead of the ALMs.

Data channel and Phone concept allows an effective usage of the device resources: the modular architecture permits to decouple data channel resources from phone channel resources within the same channel. This results in the possibility that any data channel can be linked to any other phone channel (see also example in **Figure 5**).

The selection of hardware and firmware modules is done indirectly through the ioctl interface issued for the given file descriptor. For example, command **IFX_TAPI_ENC_START** applies only to data channels, while **IFX_TAPI_LINE_FEED_SET** and **IFX_TAPI_PCM_CFG_SET** apply to phone channels (respectively ALM and PCM resources). **Chapter 4**, in the reference description of each TAPI command, states whether a command must be applied to a data or phone channel.

Examples in **Figure 4** and **Figure 5** show two possible ways (for ATA and VoIP Gateway applications) of linking phone channels to data channels while implementing the same application scenario (two parallel calls involving local phone to VoIP party).

In the example in **Figure 4**, phone and data channels used by both voice calls belong to the same VINETIC® channel. It means that for handling of voice call, an **IFX_TAPI_ENC_START** and **IFX_TAPI_LINE_FEED_SET** will apply to the same file descriptor (fd0). In a similar fashion, voice call B is handled only using file descriptor (fd1).

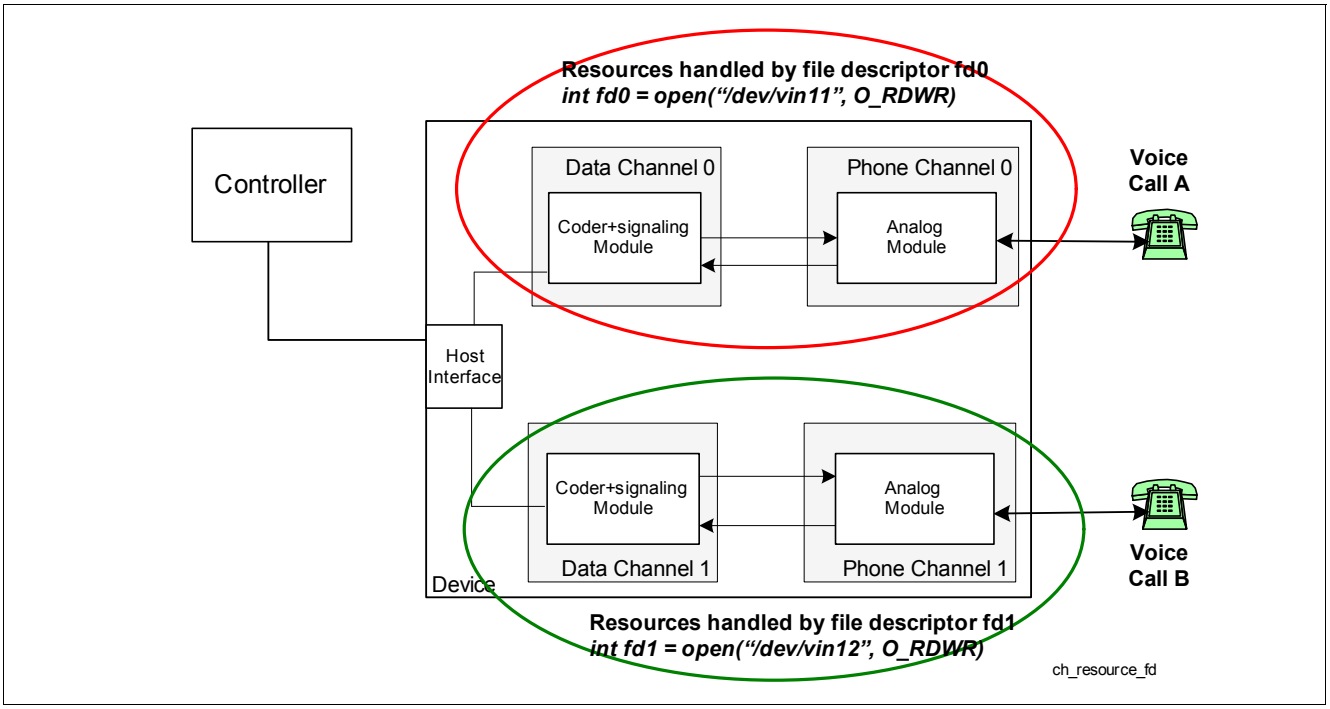

Figure 4 Example of File Descriptors and Resources (1)

In the example shown in **Figure 5**, voice call A involves phone channel 0 and data channel 1. It means that for configuring voice call B, command **IFX_TAPI_ENC_START** must be issued on fd1 (to use data channel 1) and command **IFX_TAPI_LINE_FEED_SET** must be applied on fd0 (to use phone channel 0). The same principle applies to voice call B: phone channel commands must use fd1 and data channel commands must use fd2.

For information about linking of phone and data resources, please refer to **Chapter 2.7**.

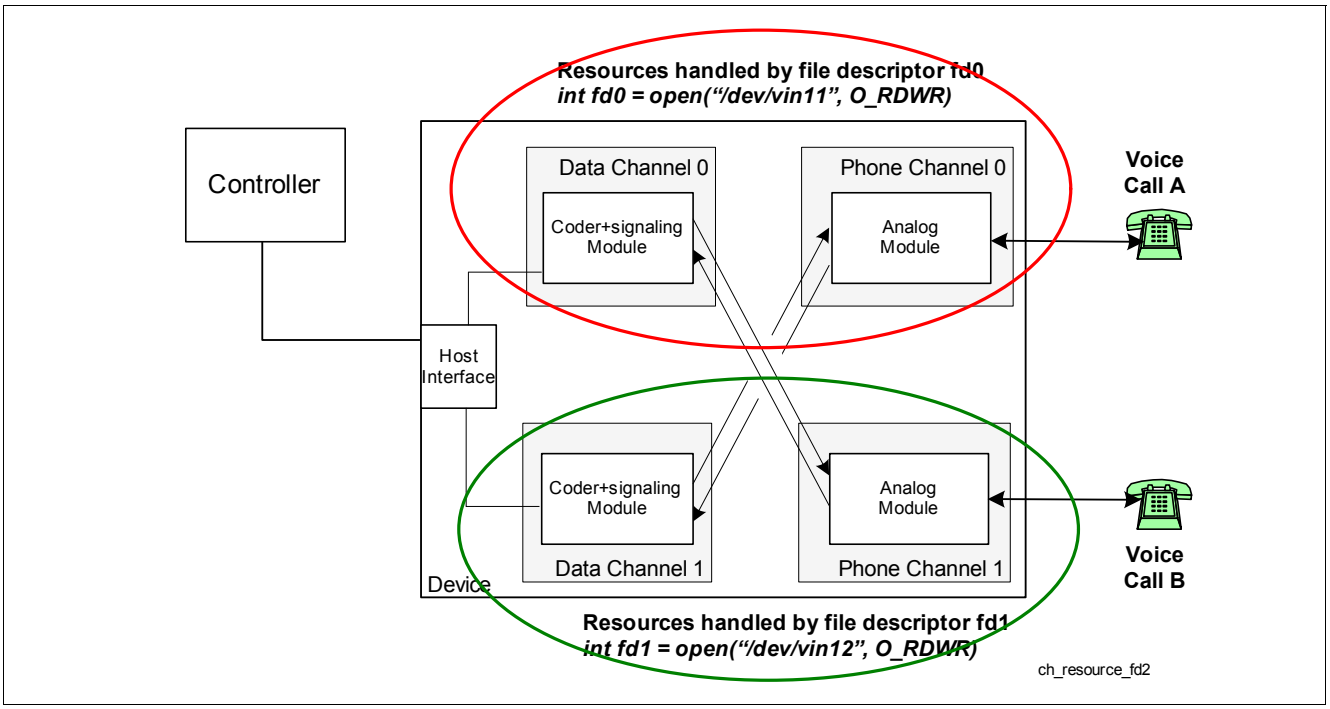

Figure 5 Example of File Descriptors and Resources (2)

2.1.3 Packet Handling

Packet handling (RTP and T.38 data pump) is done via read/write interfaces.

- read (fd, buf, bufsize), returns the number of read bytes or **IFX_ERROR**.
- write (fd, buf, len), returns the number of written bytes or **IFX_ERROR**.

The file descriptor specifies which device channel must be used. It is important to note that, in the RTP case, buf is a pointer to a complete RTP frame (including header).

Below is reported an example of read/write usage, with a simple read-write loop.

Example - Packet Handling

```
while (1)
{
  /* now wait for events and data */
  memcpy((void*)&trfds, (void*)&rfds, sizeof(fd_set));
  select(width + 1, &trfds, IFX_NULL, IFX_NULL, IFX_NULL);

  if (FD_ISSET(fd[0], &trfds))
  {
    /* Read packet from channel 0 and sent it to local channel 1 */
    len = read(fd[0], buf, sizeof(buf));
    write(fd[1], buf, len);
  }
  if (FD_ISSET(fd[1], &trfds))
  {
    /* Read packet from channel 1 and send it to local channel 0 */
    len = read(fd[1], buf, sizeof(buf));
    write(fd[0], buf, len);
  }

} /* while */);
```

2.2 Initialization

The foundation for all subsequent operations is the successful initialization of the device driver and the device itself. First operation is a device driver “open” on the device configuration node (e.g. /dev/vin10) as well as on the channel nodes to initialize for further utilization.

2.2.1 Basic Device Driver Init

At compile time, the driver is configured with details about the access mode, and the number of devices in the system. Configuration of device base address and the interrupt line to attach its interrupt handler to, is done using device driver services. For more details please refer to the product device driver documentation.

2.2.2 TAPI Channel Initialization

The channel initialization is done with command **IFX_TAPI_CH_INIT**, the argument is a pointer to **IFX_TAPI_CH_INIT_t**. It is necessary to configure:

- nMode: valid TAPI channel type is **IFX_TAPI_CH_INIT_MODE_VOICE_CODER**. This configuration has be done for each channel.
- pProc: a pointer to device-specific initialization details (depending on the device type, for example struct VMMC_IO_INIT for INCA-IP2) taking optionally as parameters the binaries for firmware and block based download (BBD) files.

– Firmware download: issued only one time after device reset.
– BBD files for ATA and VoIP Gateway: these contain coefficients that are dependent on SLIC/DAA and line type; it is possible to download a different set of coefficients per channel. This operation is defined only for channels including an analog port.
– BBD files for IP Phone applications: optimized audio coefficients and volume levels

Attention: After the initialization of each channel, a data channel is connected to a analog/audio channel.

2.3 Set-up Line Feeding Modes

It is possible to change the line feeding mode using **IFX_TAPI_LINE_FEED_SET**. The line feeding modes are defined in **IFX_TAPI_LINE_FEED_t**.

Figure 6 depicts a sequence of line feeding modes used in typical systems (example for outgoing calls).

For incoming calls, the flow depends on Caller ID standard (for example using line reversal or not). Furthermore, during ringing, some special line feeding modes are used internally by the driver.

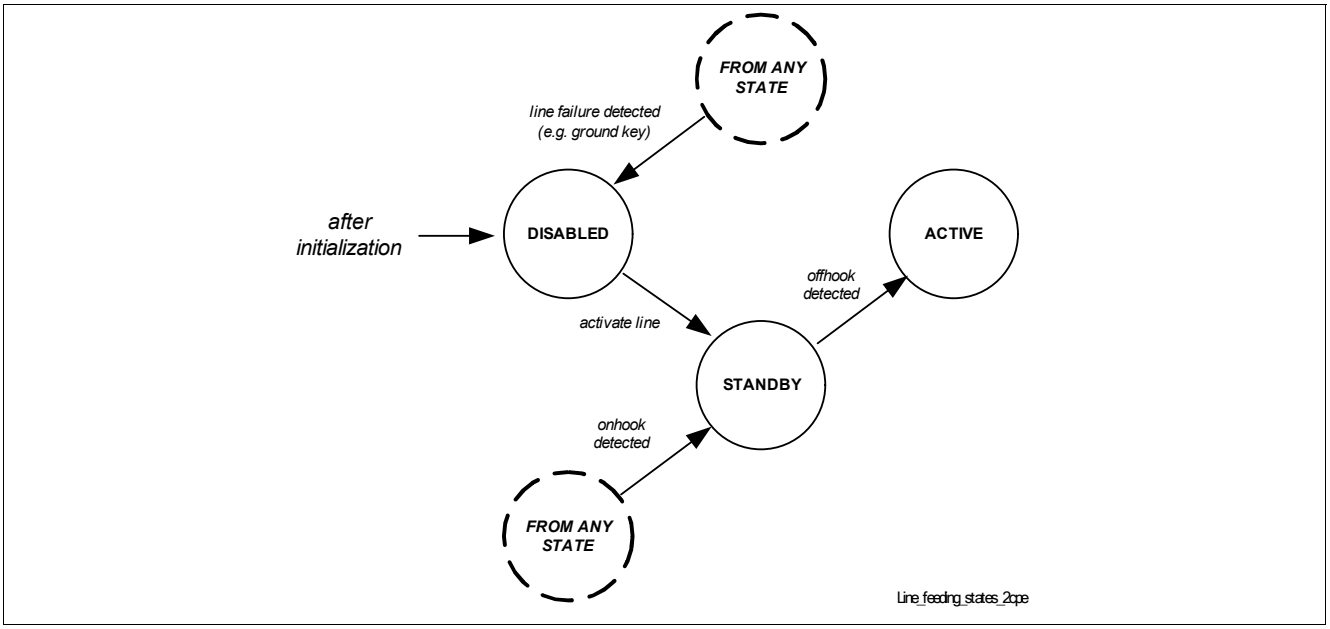

Figure 6 Sequence of Line Feeding Modes

Example - Set Line Feeding Mode

```
/* Device already initialized with IFX_TAPI_CH_INIT */
/* It means that all lines are already set to IFX_TAPI_LINE_FEED_DISABLED */
IFX_int32_t lineMode;

/* Enable line addressed by fd */
lineMode = IFX_TAPI_LINE_FEED_STANDBY;
ioctl(fd, IFX_TAPI_LINE_FEED_SET, lineMode);
/* Offhook detected for the line, now switch immediately */
/* to ACTIVE mode */
/* Required to have the possibility to detect DTMF and */
/* playing a busy tone */
lineMode = IFX_TAPI_LINE_FEED_ACTIVE;
ioctl(fd, IFX_TAPI_LINE_FEED_SET, lineMode);

/* Onhook detected, call is terminated, now switch */
```

```
/* to STANDBY mode */
/* Line feeding is enough for hook detection */
/* and line testing */
lineMode = IFX_TAPI_LINE_FEED_STANDBY;
ioctl(fd, IFX_TAPI_LINE_FEED_SET, lineMode);
```

2.4 Set-up Audio Modes

Command **IFX_TAPI_AUDIO_MODE_SET** must be used to select the audio module working mode, for example handset (**IFX_TAPI_AUDIO_MODE_DISABLED**). The audio modes are defined in **IFX_TAPI_AUDIO_MODE_t**.

Interface **IFX_TAPI_AUDIO_ROOM_TYPE_SET** can be used in case of hands-free mode (**IFX_TAPI_AUDIO_MODE_HANDSFREE**) in order to optimize the acoustic echo cancellation (AEC) algorithm; giving an indication of the room type (with respect to acoustic echo characteristics). Enum **IFX_TAPI_AUDIO_ROOM_TYPE_t** defines the room type.

Below are reported usage examples for the documented interfaces.

Example - Set-up Audio Mode

```
IFX_int32_t audioMode;

/* Choose handset mode */
audioMode = IFX_TAPI_AUDIO_MODE_DISABLED;
ioctl(fd, IFX_TAPI_AUDIO_MODE_SET, audioMode);

/* ... */

/* Key pressed: user wants to switch to headset mode */
audioMode = IFX_TAPI_AUDIO_MODE_HEADSET;
ioctl(fd, IFX_TAPI_AUDIO_MODE_SET, audioMode);
/* ... */

/* Key pressed: user wants to use the open listening feature */
/* while in headset mode */
audioMode = IFX_TAPI_AUDIO_MODE_HEADSET_OPENL;
ioctl(fd, IFX_TAPI_AUDIO_MODE_SET, audioMode);
```

Example - Select Room Type

```
IFX_int32_t audioMode;
IFX_TAPI_AUDIO_ROOM_TYPE_t roomType;

/* Audio channel is in handsfree mode */
audioMode = IFX_TAPI_AUDIO_MODE_HANDSFREE;
ioctl(fd, IFX_TAPI_AUDIO_MODE_SET, audioMode);

/* Optimize the AEC for an echoic room */
roomType = IFX_TAPI_AUDIO_ROOM_TYPE_ECHOIC;
ioctl(fd, IFX_TAPI_AUDIO_ROOM_TYPE_SET, (IFX_int32_t) roomType);
```

2.5 Event Reporting Interfaces

TAPI provides interfaces for masking and reporting of detected “events” such as detection of DTMF tone, on-/off-hook in the analog line, and fax/modem tone detected.

Events not masked are reported by waking up a device file descriptor. The application can sleep on the file descriptor with the system call “select” provided by many operating systems (Linux, VxWorks,...).

The application software can read (get) the event message, coded in a **IFX_TAPI_EVENT_t** struct, by using **IFX_TAPI_EVENT_GET**. **Figure 7** depicts a typical flow.

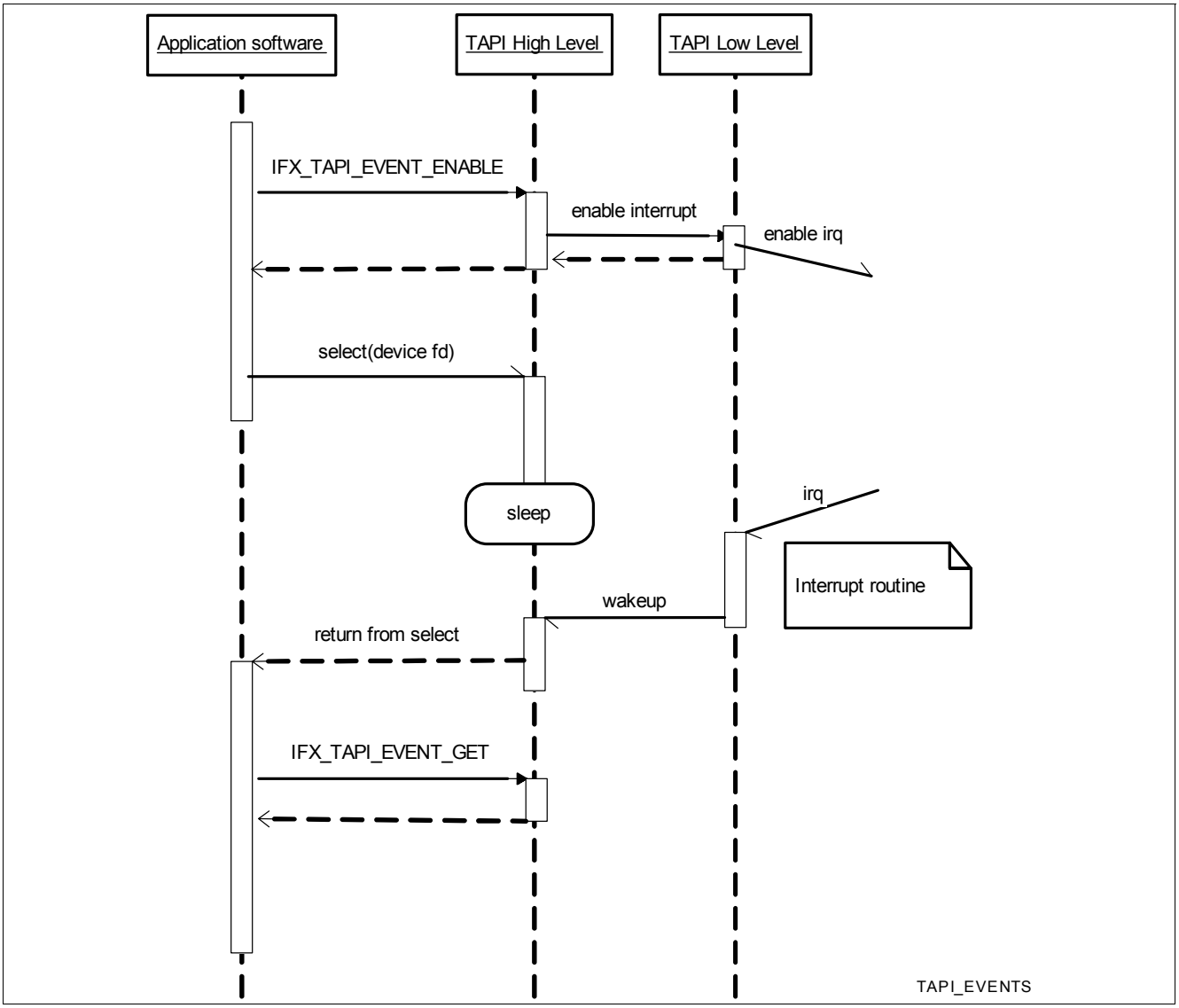

Figure 7 Sequence for Event Reporting

2.5.1 Event Message Format

Table 4 describes the composition of the event message (struct **IFX_TAPI_EVENT_t**).

Table 4 Event Message Format

Field	Data Type	Description	Note
id	IFX_TAPI_EVENT_ID_t	Field used to identify a certain event.	Each entry in IFX_TAPI_EVENT_ID_t is derived from one entry[1] in IFX_TAPI_EVENT_TYPE_t.
ch	IFX_uint16_t	Channel where the event occurred. The first channel in a device is ch=0.	For example, in a device with four channels: ch=0 for the first channel ch=3 for the fourth channel
tail	IFX_uint16_t	This field contains the information whether a new event message is ready (IFX_TRUE) or not (IFX_FALSE).	This field is filled only by IFX_TAPI_EVENT_GET.
data	IFX_TAPI_EVENT_DATA_t	Additional information necessary to characterize a certain event. This field is defined only for a few event types.	Table 5 reports the event IDs requiring the data field.

1) IEntries in IFX_TAPI_EVENT_TYPE_t can be used as bitmask for one or more event IDs.

Table 5 Event IDs Requiring the data Field

Event ID	Used Field	Data Type
IFX_TAPI_EVENT_PULSE_DIGIT	pulse	IFX_TAPI_EVENT_DATA_PULSE_t
IFX_TAPI_EVENT_DTMF_DIGIT	dtmf	IFX_TAPI_EVENT_DATA_DTMF_t
IFX_TAPI_EVENT_TONE_GEN_END	tone	IFX_TAPI_EVENT_DATA_TONE_t
IFX_TAPI_EVENT_TONE_DET_CPT	tone	IFX_TAPI_EVENT_DATA_TONE_t
All event IDs derived from event type IFX_TAPI_EVENT_TYPE_FAXMODEM_SIGNAL	fax_sig	IFX_TAPI_EVENT_DATA_FAX_SIG_t
IFX_TAPI_EVENT_FAULT_GENERAL	value	IFX_uint16_t *Note: The error messages are enumerated in DEV_ERR.*

2.5.2 Enable/Disable Event Reporting

For enabling/disabling detection of event reporting to the application is achieved using the interfaces IFX_TAPI_EVENT_ENABLE and IFX_TAPI_EVENT_DISABLE on a device file descriptor (representing the device where to enable/disable the event) and passing a pointer to a IFX_TAPI_EVENT_t struct with appropriate configuration.

Table 6 contains some examples of how to fill IFX_TAPI_EVENT_t struct to enable/disable events.

Table 6 Event Enable/disable Examples

Event	id	ch	data
DTMF digit from FXS or FXO port, channel number 1	IFX_TAPI_EVENT_DTMF_DIGIT	1	local=1 network=0 digit=unused
DTMF digit from VoIP port (RTP inband), channel number 1	IFX_TAPI_EVENT_DTMF_DIGIT	1	local=0 network=1 digit=unused
RFC2833 packet received, channel number 1	IFX_TAPI_EVENT_RFC2833_EVENT	1	unused
Pulse digit from FXS port, channel number 1	IFX_TAPI_EVENT_PULSE_DIGIT	1	digit=unused
Off hook on FXS port, channel 3	IFX_TAPI_EVENT_FXS_OFFHOOK	3	unused
On hook on FXS port, channel 3	IFX_TAPI_EVENT_FXS_ONHOOK	3	unused
Flash hook on FXS port, channel 3	IFX_TAPI_EVENT_FXS_FLASH	3	unused
Ringing on FXS port, channel 3	IFX_TAPI_EVENT_FXS_RING	3	unused
End of CID information TX on data channel 3	IFX_TAPI_EVENT_CID_TX_INFO_END	3	unused
Tone generation ended on local port, channel 3	IFX_TAPI_EVENT_TONE_GEN_END	3	local=1 network=0 index=unused
Call progress tone detection from local port, channel 3	IFX_TAPI_EVENT_TONE_DET_CPT	3	local=1 network=0 digit=unused
Call progress tone index detection from VoIP port, channel 3	IFX_TAPI_EVENT_TONE_DET_CPT	3	local=0 network=1 digit=unused
Fax signal DIS detected from local port, channel 2	IFX_TAPI_EVENT_FAXMODEM_DIS	2	local=1 network=0

2.5.3 Examples of Event Reporting and Masking

After enabling the detection of events, the application software should block on a select system call for a device file descriptor . As soon as an event is detected the event message will be internally queued (in a channel specific queue) and select will return.

Interface **IFX_TAPI_EVENT_GET** should be used to get the event message, using a device file descriptor (representing the device where the event is expected) and specifying the event source in the ch field:

- ch = **IFX_TAPI_EVENT_ALL_CHANNELS** to get the event message from any channel
- ch = <channel number>, to get the event message from a specific channel

Table 6 contains some examples of how events are returned by **IFX_TAPI_EVENT_GET** in struct **IFX_TAPI_EVENT_t**.

To be noted that tail field (in **IFX_TAPI_EVENT_t**) indicates whether at least one event message is ready to be read see **Table 4**.

Table 7 Event Detection Examples

Event	id	ch	data
DTMF digit "3" from FXS or FXO port, channel number 1	IFX_TAPI_EVENT_DTMF_DIGIT	1	local=1 network=0 digit=3
DTMF digit "3" from VoIP port (RTP inband), channel number 1	IFX_TAPI_EVENT_DTMF_DIGIT	1	local=0 network=1 digit=3
RFC2833 packet received with DTMF digit "3", channel number 1	IFX_TAPI_EVENT_RFC2833_EVENT	1	event=3
Pulse digit "3" from FXS port, channel number 1	IFX_TAPI_EVENT_PULSE_DIGIT	1	digit=3
Off hook on FXS port, channel 3	IFX_TAPI_EVENT_FXS_OFFHOOK	3	unused
On hook on FXS port, channel 3	IFX_TAPI_EVENT_FXS_ONHOOK	3	unused
Flash hook on FXS port, channel 3	IFX_TAPI_EVENT_FXS_FLASH	3	unused
Ringing on FXS port, channel 3	IFX_TAPI_EVENT_FXS_RING	3	unused
End of CID sequence on FXS port, channel 3	IFX_TAPI_EVENT_CID_TX_INFO_END	3	unused
Tone generation ended to local port, channel 3	IFX_TAPI_EVENT_TONE_GEN_END	3	local=1 network=0 index=55
Call progress tone index 44 detected from local port, channel 3	IFX_TAPI_EVENT_TONE_DET_CPT	3	local=1 network=0 digit=44
Call progress tone index 44 detected from VoIP port, channel 3	IFX_TAPI_EVENT_TONE_DET_CPT	3	local=0 network=1 digit=44
Fax signal DIS detected from local port, channel 3	IFX_TAPI_EVENT_FAXMODEM_DIS	3	local=1 network=0

Example - Event Reporting

```
IFX_TAPI_EVENT_t tapiEvent;
IFX_int32_t fd_dev, fd[2], i;
fd_set rfds;
IFX_int32_t width;
IFX_return_t ret;

FD_ZERO(&rfds);
/* Open device file descriptor */
fd_dev = open("/dev/vin10", O_RDWR);
FD_SET (fd_dev, &rfds);
width = fd_dev;
/* Open channel file descriptor for channel 0 */
fd[0] = open("/dev/vin11", O_RDWR, 0x644);
FD_SET(fd[0], &rfds);
if (width < fd[0])
{
 width = fd[0];
}
```

```
fd[1] = open("/dev/vin12", O_RDWR, 0x644);
/* Open channel file descriptor for channel 1 */
FD_SET(fd[1], &rfds);
if (width < fd[1])
{
 width = fd[1];
}

/* Now wait for events and data */
ret = select(width + 1, &rfds, IFX_NULL, IFX_NULL, IFX_NULL);
/* Select woke up from exception on fd_dev or data on fd[x] */
if (FD_ISSET(fd_dev, &rfds))
{
 /* an exception occurred, get status of two channels */
 for (i=0;i<2;i++)
 {
  memset (&tapiEvent, 0, sizeof(tapiEvent));
  tapiEvent.ch = i;
  ret = ioctl(fd_dev, IFX_TAPI_EVENT_GET, (IFX_int32_t) &tapiEvent);
  if ((ret == IFX_SUCCESS) && (tapiEvent.id != IFX_TAPI_EVENT_NONE))
  {
   printf("Event %d occurred in channel %d \n", tapiEvent.id, i);
  }
 }
}

for (i = 0; i < 2; i++)
{
 if (FD_ISSET(fd[i], &rfds))
 {
  printf("Handle data\n");
 }
}
```

Example - Detect Digit

```
/* This example shows the call of the status information of one channel */
IFX_TAPI_EVENT_t tapiEvent;
IFX_int32_t ret;

/* Get the status only for channel 0 ("/dev/vin11") */
while (1)
{
 /* Only query this channel */
 tapiEvent.ch = 0;
 ret = ioctl(fd, IFX_TAPI_EVENT_GET, (IFX_int32_t) &tapiEvent);
 /* Check whether a DTMF digit has been detected */
 if (tapiEvent.id == IFX_TAPI_EVENT_DTMF_DIGIT)
 {
  printf("Digit pressed\n");
 }
}
```

Example - Masking Events

```
/* Mask reporting of exceptions */
IFX_TAPI_EVENT_t tapiEvent;

/* Mask DTMF event on local side in channel 1 */
tapiEvent.id = IFX_TAPI_EVENT_DTMF_DIGIT;
tapiEvent.data.dtmf.local = IFX_TRUE;
tapiEvent.data.ch = 1;

/* Now mask the selected exceptions */
ioctl(fd, IFX_TAPI_EVENT_DISABLE, (IFX_int32_t) &tapiEvent);
```

2.5.4 Old Interfaces

The following interfaces become obsolete with the introduction of IFX_TAPI_EVENT_* commands :

- Enabling/disabling reporting of exceptions **IFX_TAPI_EXCEPTION_MASK**, **IFX_TAPI_EXCEPTION_GET**
- Channel exception status **IFX_TAPI_CH_STATUS_GET**
- Pulse digit detection **IFX_TAPI_PULSE_GET**, **IFX_TAPI_PULSE_ASCII_GET**, **IFX_TAPI_PULSE_READY**
- DTMF digit detection **IFX_TAPI_TONE_DTMF_GET**, **IFX_TAPI_TONE_DTMF_ASCII_GET**, **IFX_TAPI_TONE_DTMF_READY_GET**
- Fax/modem signal detection **IFX_TAPI_SIG_DETECT_DISABLE**, **IFX_TAPI_SIG_DETECT_DISABLE**.

Attention: The obsolete interfaces will be skipped from a future TAPI release. Please consider porting the application software to the new IFX_TAPI_EVENT_* interfaces. Mixing usage of old interfaces and IFX_TAPI_EVENT_* interfaces may lead to unpredictable system behavoiur!

2.6 Configure Hook Validation Timing

Using ioctl **IFX_TAPI_LINE_HOOK_VT_SET** it is possible to configure detection timing for the following parameters. To be noted that for some parameters only minimum time can be configured.

- on-hook: minimum time
- off-hook: minimum time
- Flash hook (alias register recall): validation interval
- Pulse dialing: digit low time validation interval
- Pulse dialing: digit high time validation interval
- Pulse dialing: minimum time between digits

The ioctl can configure one timing at time using structure **IFX_TAPI_LINE_HOOK_VT_t** and specifying

- Field nType: Kind of timing is to be configured, selecting one value out of enum **IFX_TAPI_LINE_HOOK_VALIDATION_TYPE_t**
- Field nMinTime: minimum time (if required)
- Field nMaxTime: maximum time (if required)

Example - Configure Hook Validation Timing - Hook Events

```
IFX_TAPI_LINE_HOOK_VT_t param;
IFX_int32_t fd;

/* Open channel file descriptor for channel 0 */
fd = open("/dev/vin11", O_RDWR, 0x644);
memset (&param, 0, sizeof (IFX_TAPI_LINE_HOOK_VT_t));

/* Set onhook timing: minimum 400 ms */
```

```
param.nType = IFX_TAPI_LINE_HOOK_VT_ONHOOK_TIME;
param.nMinTime = 400;
/* Note that nMaxTime is not required for onhook detection! */
ioctl(fd, IFX_TAPI_LINE_HOOK_VT_SET, (IFX_int32_t) &param);

/* Set offhook timing: minimum 40 ms */
param.nType = IFX_TAPI_LINE_HOOK_VT_OFFHOOK_TIME;
param.nMinTime = 40;
/* Note that nMaxTime is not required for offhook detection! */
ioctl(fd, IFX_TAPI_LINE_HOOK_VT_SET, (IFX_int32_t) &param);

/* Set flash hook timing: minimum 40 ms, maximum 200 ms */
param.nType = IFX_TAPI_LINE_HOOK_VT_FLASHHOOK_TIME;
param.nMinTime = 40;
param.nMinTime = 200;
ioctl(fd, IFX_TAPI_LINE_HOOK_VT_SET, (IFX_int32_t) &param);

/* Close open fd */
close(fd);
```

Example - Configure Hook Validation Timing - Pulse Dialing

```
IFX_TAPI_LINE_HOOK_VT_t param;
IFX_int32_t fd;

/* Open channel file descriptor for channel 0 */
fd = open("/dev/vin11", O_RDWR, 0x644);

memset (&param, 0, sizeof (IFX_TAPI_LINE_HOOK_VT_t));
/* Set pulse dialing */
param.nType = IFX_TAPI_LINE_HOOK_VT_DIGITLOW_TIME;
param.nMinTime = 40;
param.nMaxTime = 60;
ioctl(fd, IFX_TAPI_LINE_HOOK_VT_SET, (IFX_int32_t) &param);
param.nType = IFX_TAPI_LINE_HOOK_VT_DIGITHIGH_TIME;
param.nMinTime = 40;
param.nMaxTime = 60;
ioctl(fd, IFX_TAPI_LINE_HOOK_VT_SET, (IFX_int32_t) &param);

/* Close open fd */
close(fd);
```

2.7 Establishing Connections

Several connection types are supported; for ATA and Gateway applications, any phone connected to an ALM/PCM port can be connected to another ALM/PCM port or to a VoIP party. In case of IP Phone applications, the audio channel can be connected to any data channel or to the PCM port. Three-party conferencing is also possible considering any combination of ports.

This chapter introduces the API required for linking the device ports, and describes how to establish different type of calls and three-party conferences.

- **Chapter 2.7.1**, IP Phone applications.
- **Chapter 2.7.2**, Gateway-like applications.

Attention: For typical applications, each analog or audio channel used must be linked to one dedicated data channel.

Table 8 Interfaces for Mapping Services

Interfaces	Addressed Resource Set	Note
IFX_TAPI_MAP_DATA_ADD IFX_TAPI_MAP_DATA_REMOVE	To be used for adding/removing a link between a data resource (it means a set of coder and signaling firmware resources) to any other type of resource (phone, analog or PCM).	These interfaces are relevant for establishing a VoIP call with a phone attached to the analog/audio or to the PCM port. .
IFX_TAPI_MAP_PCM_ADD IFX_TAPI_MAP_PCM_REMOVE	To be used for adding/removing a link between a PCM port to other PCM port or an analog/audio port.	These interfaces are relevant for scenarios where a voice connection is going through the PCM interface.
IFX_TAPI_MAP_PHONE_ADD IFX_TAPI_MAP_PHONE_REMOVE	To be used for adding/removing a link between analog module ports (ALM) to other analog ports or PCM ports	Applicable only to ATA and Gateway applications.

2.7.1 IP Phone Applications

This chapter describes how to prepare the device before establishing a voice connections.

2.7.1.1 Voice Call

Figure 8 depicts the usage of the resources for one VoIP call in IP Phone applications.

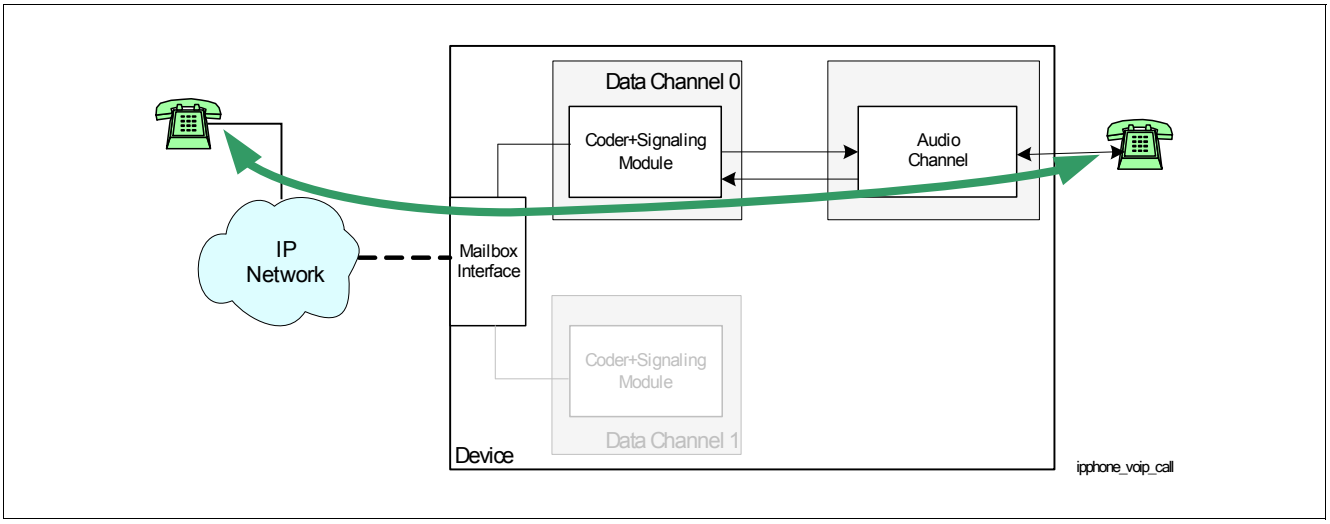

Figure 8 One VoIP Calls

Example - IP Phone VoIP Call

```
IFX_TAPI_CH_INIT_t InitCh1;

memset(&InitCh1, 0, sizeof(IFX_TAPI_CH_INIT_t));

/* Initialize the channel for VoIP applications */
InitCh1.nMode = IFX_TAPI_INIT_MODE_VOICE_CODER;

ioctl(fd0, IFX_TAPI_CH_INIT, &InitCh1);
/* Now the channel is initialized as in Figure */
/* It is possible to start all call services
```

2.7.1.2 In-call Announcement

Figure 9 depicts the usage of the resources for two VoIP calls in parallel for ATA and VoIP Gateway applications. The in-call announcement have to be directed to the auxiliary input of the audio channel (IFX_TAPI_MAP_DATA_TYPE_AUDIO_AUX).

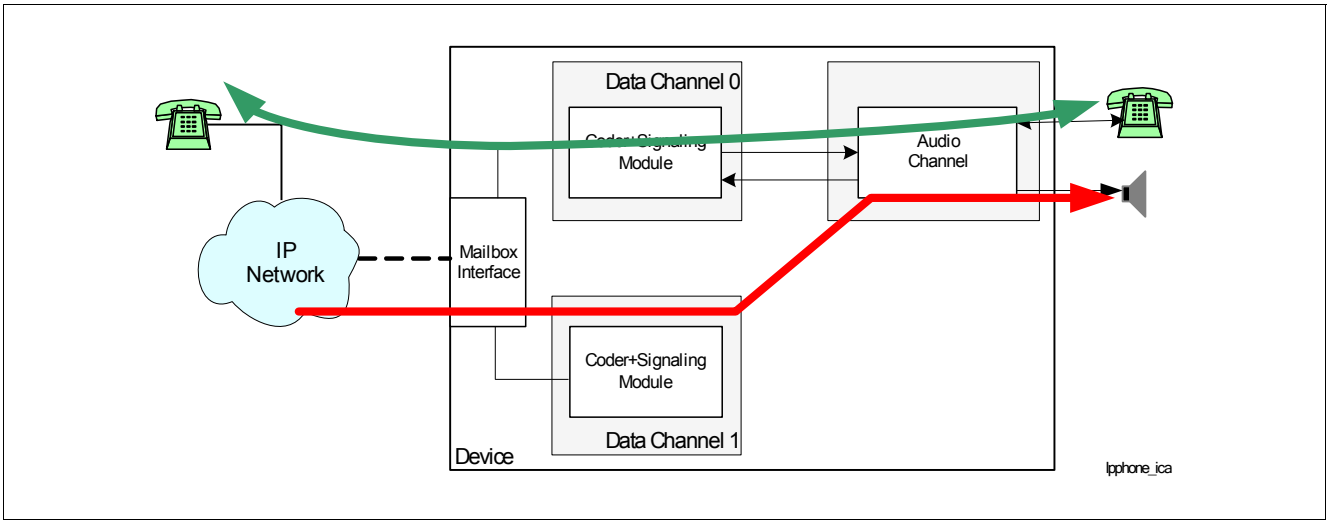

Figure 9 In-call Announcement

Example - In-call Announcement

```
   IFX_TAPI_CH_INIT_t InitCh1;
   IFX_int32_t fd1;
   IFX_TAPI_MAP_DATA_t datamap;

   memset(&InitCh1, 0, sizeof(IFX_TAPI_CH_INIT_t));

   /* Initialize the channel for VoIP applications */
   InitCh1.nMode = IFX_TAPI_INIT_MODE_VOICE_CODER;
   ioctl(fd0, IFX_TAPI_CH_INIT, &InitCh1);

/* Assume a VoIP call is already ongoing using data channel 0

/* In-call announcement using data channel 1 */
/* Connect auxiliary input of the audio channel (nDstCh=0) to data channel 1(fd1) */
   datamap.nDstCh = 0;
   datamap.nDstCh = IFX_TAPI_MAP_DATA_TYPE_AUDIO_AUX;
   ioctl(fd1, IFX_TAPI_MAP_DATA_ADD, &datamap);

   /* Now the data channel 1is connected to the audio channel */
   /* The announcement will be heard on the speaker */
   /* Note: The announcement must be passed through data channel 1 !! */
```

2.7.2 Gateway-like Applications

For ATA and Gateway applications

2.7.2.1 Internal Call

Internal calls can be established on ATA and VoIP Gateway applications simply connecting the two phone channels using interface **IFX_TAPI_MAP_PHONE_ADD**.

Figure 10 depicts a typical example. It is very important to note the role of the data channel: it is required for signal detection/generation (such as DTMF, Fax/Modem tones, call progress tones, caller ID, ...) while the coder part is idle (jitter buffer and RTP are inactive).

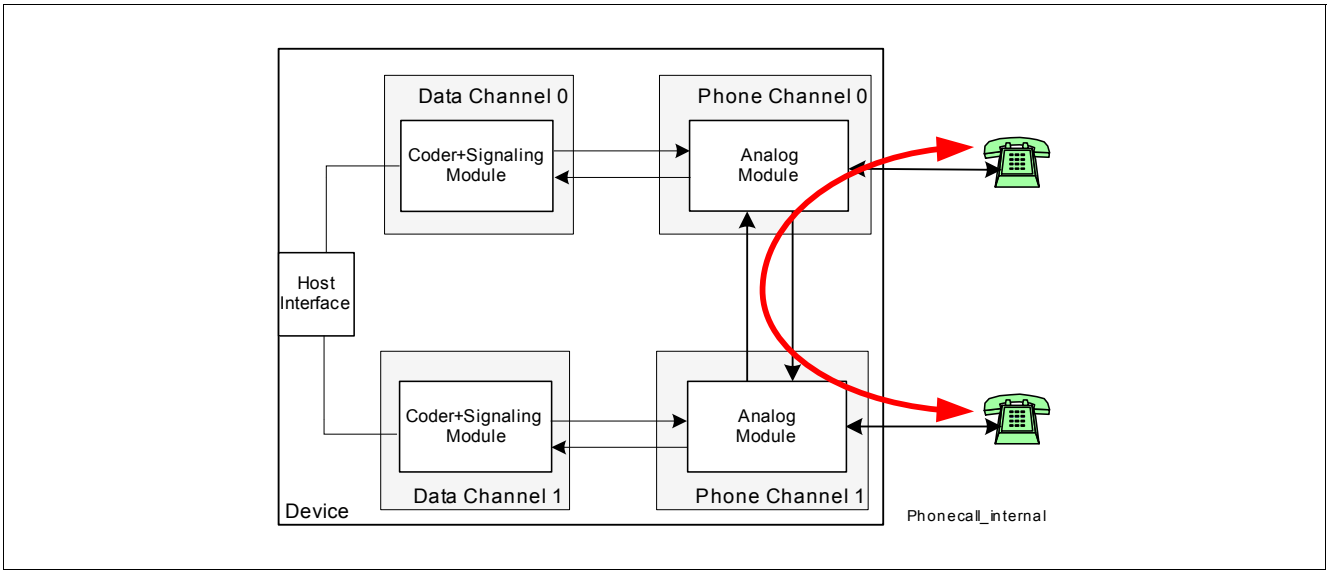

Figure 10 Internal Call

Example - Internal Call

```
IFX_TAPI_CH_INIT_t InitCh;
IFX_TAPI_MAP_PHONE_t param;
IFX_int32_t fd0, fd1;

memset(&InitCh, 0, sizeof(IFX_TAPI_CH_INIT_t));
memset(&param, 0, sizeof(IFX_TAPI_MAP_PHONE_t));

/* Open channels the two channels */
fd0 = open("/dev/vin11", O_RDWR, 0x644);
fd1 = open("/dev/vin12", O_RDWR, 0x644);

/* Initialize the channels */
/* Each channel contains one analog and one data resource */
InitCh.nMode = IFX_TAPI_INIT_MODE_VOICE_CODER;
ioctl(fd0, IFX_TAPI_CH_INIT, &InitCh);
ioctl(fd1, IFX_TAPI_CH_INIT, &InitCh);

/* Now connect the two channels, as in Figure */
/* Connect phone channel 1 (nPhoneCh=1) to phone channel 0 (fd0) */
param.nPhoneCh = 1;
```

```
param.nChType = IFX_TAPI_MAP_TYPE_PHONE;
ioctl(fd0, IFX_TAPI_MAP_PHONE_ADD, &param);

/* Or the same result can be achieved with the following code */
/* Connect phone channel 0 (nPhoneCh=0) to phone channel 1 (fd1) */
param.nPhoneCh = 0;
param.nChType = IFX_TAPI_MAP_TYPE_PHONE;
ioctl(fd1, IFX_TAPI_MAP_PHONE_ADD, &param);
```

2.7.2.2 Voice Call with External VoIP Party

Figure 11 depicts the usage of the resources for two VoIP calls in parallel for ATA and VoIP Gateway applications.

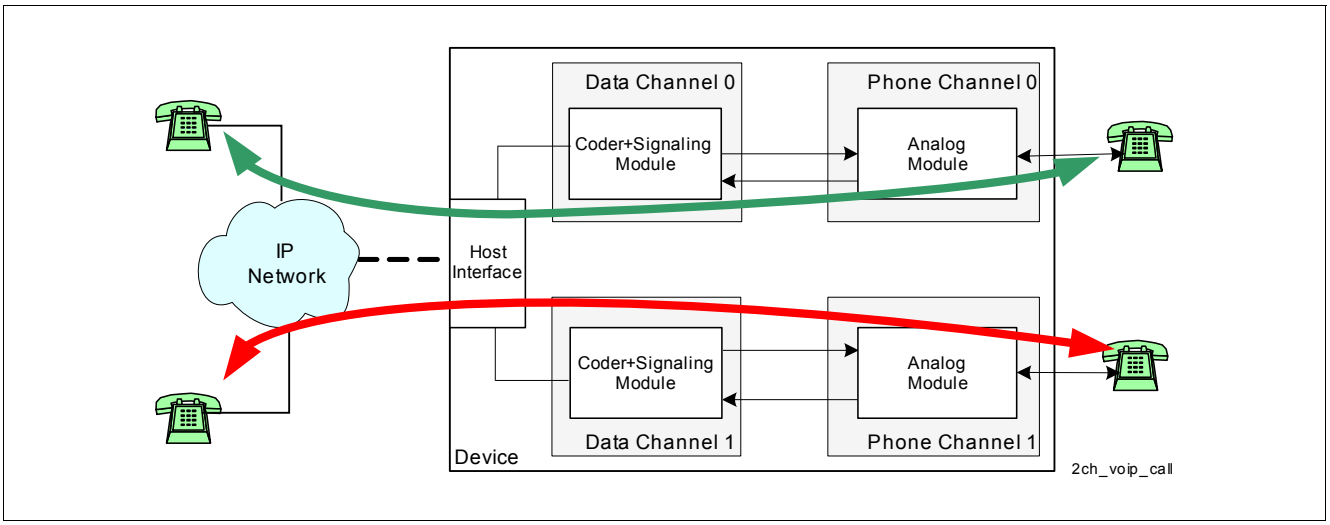

Figure 11 Two Independent VoIP Calls

Example - Two Independent VoIP Call - Directly After Channel Initialization

```
IFX_TAPI_CH_INIT_t InitCh;

memset(&InitCh, 0, sizeof(IFX_TAPI_CH_INIT_t));

/* Initialize the channels for VoIP applications */
/* Each channel contains one analog and one data resource */
InitCh.nMode = IFX_TAPI_INIT_MODE_VOICE_CODER;
ioctl(fd0, IFX_TAPI_CH_INIT, &InitCh);
ioctl(fd1, IFX_TAPI_CH_INIT, &InitCh);
/* Now the two channels are initialized as in Figure */
/* It is possible to start all call services */
```

Example - Two Independent VoIP Call - Unmap Defaul Resources

```
/* IFX_TAPI_CH_INIT Assigns by default data channels to each analog module */
/* This example shows how to ummap the data channels assigned by default */

IFX_TAPI_CH_INIT_t InitCh;
IFX_TAPI_MAP_DATA_t datamap;
```

```
memset(&InitCh, 0, sizeof(IFX_TAPI_CH_INIT_t));

/* Initialize the channels */
InitCh1.nMode = IFX_TAPI_INIT_MODE_VOICE_CODER;
InitCh2.nMode = IFX_TAPI_INIT_MODE_VOICE_CODER;
ioctl(fd0, IFX_TAPI_CH_INIT, &InitCh);
ioctl(fd1, IFX_TAPI_CH_INIT, &InitCh);

/* Unmap the data channels that are per default mapped */

memset(&datamap, 0, sizeof(IFX_TAPI_MAP_DATA_t));
/* Remove connection between phone channel 0 (nDstCh=0) and */
/* data channel 0 (fd0) */
datamap.nDstCh = 0;
datamap.nChType = IFX_TAPI_MAP_DATA_TYPE_PHONE;
ioctl(fd0, IFX_TAPI_MAP_DATA_REMOVE, &datamap);
/* Remove connection between phone channel 1 (nDstCh=1) and */
/* data channel 1 (fd1) */
datamap.nDstCh = 1;
datamap.nChType = IFX_TAPI_MAP_DATA_TYPE_PHONE;
ioctl(fd1, IFX_TAPI_MAP_DATA_REMOVE, &datamap);
/* Now phone and data resources are unmapped */

/* Connect phone channel 0 (nDstCh=0) to data channel 0 (fd0) */
datamap.nDstCh = 0;
datamap.nChType = IFX_TAPI_MAP_DATA_TYPE_PHONE;
ioctl(fd0, IFX_TAPI_MAP_DATA_ADD, &datamap);
/* Connect phone channel 1 (nDstCh=1) to data channel 1 (fd1) */
datamap.nDstCh = 1;
datamap.nChType = IFX_TAPI_MAP_DATA_TYPE_PHONE;
ioctl(fd1, IFX_TAPI_MAP_DATA_ADD, &datamap);
```

2.8 Conferencing

This chapter lists the conferencing services with some scenarios. Conferencing is implicitely supported by the mapping services **IFX_TAPI_MAP_***.

The TAPI default setup maps data channel 0 to phone channel 0, data 1 to phone 1, and so on. This enables external VoIP connection without reconfiguration and avoids resource management if no conferencing is used.

In case of conferencing, the TAPI distinguishes between external and (device) internal calls.

External calls are established using a data channel. Internal calls can also be established connecting phone channels internally in the chip. The following subchapter show how to establish conferences, please note that **Chapter 2.8.1** and **Chapter 2.8.3** are the only relevant for IP Phone applications.

In the following examples, *fd0* refers to data and phone channel 0, *fd1* to data and phone channel 1, and so on.

2.8.1 Initialization

If using TAPI conferencing with resource management, the data channels should be disconnected after channel initialization and assigned only as soon as required (for example off-hook detected in a channel). This ensures that the maximum number of data channel resources are available in a system.

To simplify the example, return values are not handled.

```
/* The example is valid for ATA/VoIP GW applications */;
/* For IP Phone applications IFX_TAPI_MAP_DATA_TYPE_AUDIO has to be used */;
/* instead of IFX_TAPI_MAP_DATA_TYPE_PHONE */;

IFX_TAPI_MAP_DATA_t datamap;
IFX_TAPI_CH_INIT_t Init;
IFX_TAPI_CAP_t cap;

/* Initialize the device and all channels. It sets up the configuration */
memset(&Init, 0, sizeof(IFX_TAPI_CH_INIT_t));
ioctl(fd0, IFX_TAPI_CH_INIT, &Init);
ioctl(fd1, IFX_TAPI_CH_INIT, &Init);

/* Unmap the data channels that are per default mapped */
memset(&datamap, 0, sizeof(IFX_TAPI_MAP_DATA_t));
datamap.nDstCh = 0;
datamap.nChType = IFX_TAPI_MAP_DATA_TYPE_PHONE;
ioctl(fd0, IFX_TAPI_MAP_DATA_REMOVE, &datamap);
datamap.nDstCh = 1;
ioctl(fd1, IFX_TAPI_MAP_DATA_REMOVE, &datamap);

/* Check coder channel amount */
cap.captype = IFX_TAPI_CAP_TYPE_CODEC;
ioctl(fd0, IFX_TAPI_CAP_CHECK, (int)&cap);
printf("%d available data channels \n", cap.cap);
/* Resource manager must check all managed capabilities */
```

2.8.2 Conference with an Internal and an External VoIP Party

This scenario is not applicable to IP Phone applications.

Continuing the setup from **Chapter 2.7.2.1**, an external party can be added (see **Figure 12**). The mapped data channel can be either of type coder or PCM.

Note: The resource handling must consider the limitation of the chip for coder and for PCM.

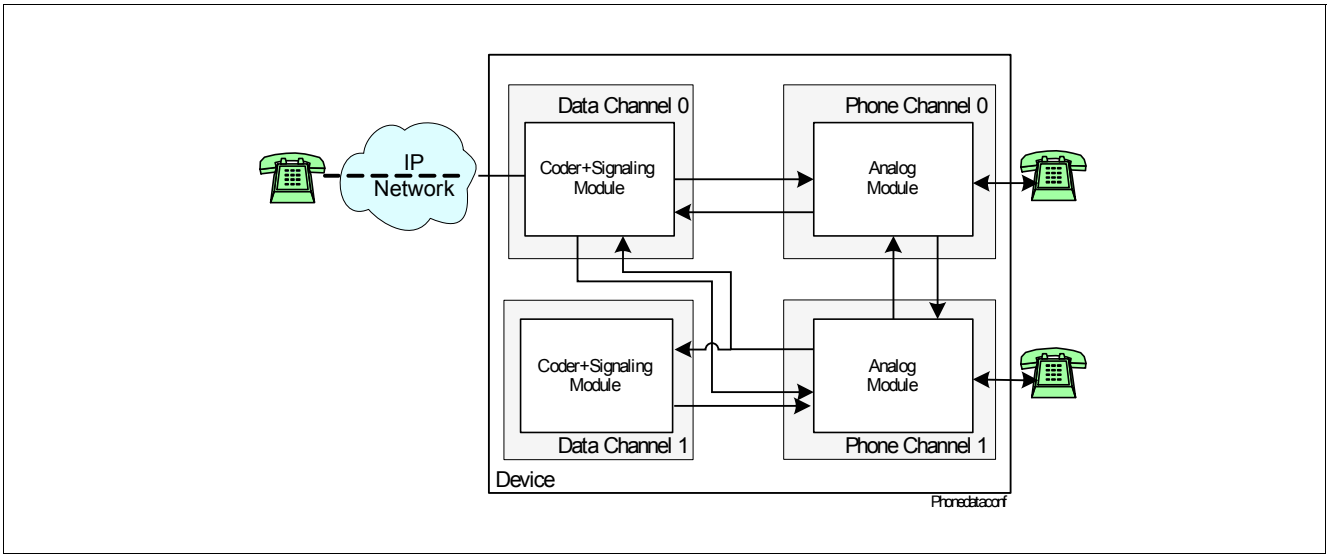

Figure 12 External and Internal Conference

In this example the data channel 0 is mapped to the first phone (channel 0):

```
IFX_TAPI_MAP_PHONE_t phonemap;
IFX_TAPI_MAP_DATA_t datamap;

memset(&phonemap, 0, sizeof(IFX_TAPI_MAP_PHONE_t));
memset(&datamap, 0, sizeof(IFX_TAPI_MAP_DATA_t));

/* Add phone 0 to phone 1 */
phonemap.nPhoneCh = 0;
phonemap.nChType = IFX_TAPI_MAP_TYPE_PHONE;
ioctl(fd1, IFX_TAPI_MAP_PHONE_ADD, &phonemap);
/* Add phone 0 to data 0 */
datamap.nDstCh = 0;
datamap.nChType = IFX_TAPI_MAP_DATA_TYPE_PHONE;
ioctl(fd0, IFX_TAPI_MAP_DATA_ADD, &datamap);
/* Add phone 1 to data 0 */
datamap.nDstCh = 1;
datamap.nChType = IFX_TAPI_MAP_DATA_TYPE_PHONE;
ioctl(fd0, IFX_TAPI_MAP_DATA_ADD, &datamap);
```

Removing the Connection

To get back the setup as after the initialization described in **Chapter 2.8.1**, the connections are removed with the following commands:

```
IFX_TAPI_MAP_PHONE_t phonemap;
IFX_TAPI_MAP_DATA_t datamap;

memset(&phonemap, 0, sizeof(IFX_TAPI_MAP_PHONE_t));
memset(&datamap, 0, sizeof(IFX_TAPI_MAP_DATA_t));

/* Remove phone 0 from data 0 */
datamap.nDstCh = 0;
datamap.nChType = IFX_TAPI_MAP_DATA_TYPE_PHONE;
ioctl(fd0, IFX_TAPI_MAP_DATA_REMOVE, &datamap);
```

```
/* The data channel is still connected to phone 1 -> remove the connection */
datamap.nDstCh = 1;
datamap.nChType = IFX_TAPI_MAP_DATA_TYPE_PHONE;
ioctl(fd0, IFX_TAPI_MAP_DATA_REMOVE, &datamap);
/* Remove phone 0 from phone 1 */
phonemap.nPhoneCh = 1;
phonemap.nChType = IFX_TAPI_MAP_TYPE_PHONE;
ioctl(fd0, IFX_TAPI_MAP_PHONE_REMOVE, &phonemap);
```

Note: Without removing the data channel from phone 1, the connection still exists. This can be a requirement, when phone 0 hangs up and phone 1 should be still in the call.

2.8.3 Conference with Two VoIP Parties

Starting from the setup from **Chapter 2.8.1**, two external parties can be added to the local phone channel 0 (see **Figure 13**). In this case, data channels 0 and 1 are mapped to the phone channel 0. The connection between the data channels is done automatically by TAPI.

```
/* The example is valid for ATA/VoIP GW applications */;
/* For IP Phone applications IFX_TAPI_MAP_DATA_TYPE_AUDIO has to be used */;
/* instead of IFX_TAPI_MAP_DATA_TYPE_PHONE */;

IFX_TAPI_MAP_DATA_t datamap;
IFX_int32_t fd0, fd1;

memset(&datamap, 0, sizeof(IFX_TAPI_MAP_DATA_t));

/* Add the phone channel 0 to the data channel 0 (fd0) */
datamap.nDstCh = 0;
datamap.nChType = IFX_TAPI_MAP_DATA_TYPE_PHONE;
ioctl(fd0, IFX_TAPI_MAP_DATA_ADD, &datamap);
/* Add the phone channel 0 to the data channel 1 (fd1) */
ioctl(fd1, IFX_TAPI_MAP_DATA_ADD, &datamap);

/* Enable the data transfer for both data channels */
ioctl(fd0, IFX_TAPI_ENC_START, 0);
ioctl(fd1, IFX_TAPI_ENC_START, 0);
```

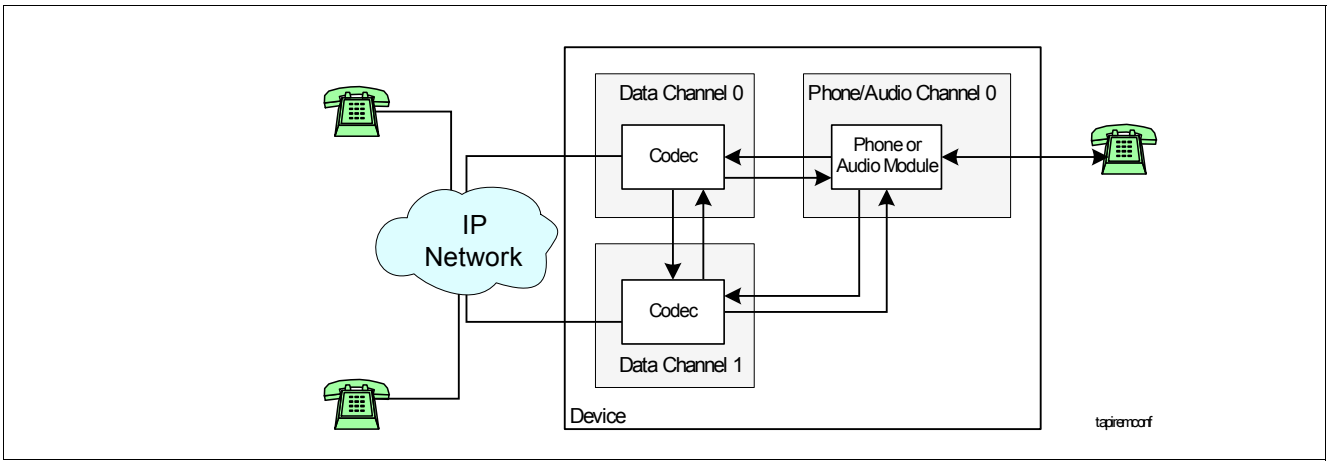

Figure 13 VoIP Conference

Removing the Connection

To get back the setup as after the initialization described in **Chapter 2.8.1**, the connections are removed with the following commands:

```
/* The example is valid for ATA/VoIP GW applications */;
/* For IP Phone applications IFX_TAPI_MAP_DATA_TYPE_AUDIO has to be used */;
/* instead of IFX_TAPI_MAP_DATA_TYPE_PHONE */;

IFX_TAPI_MAP_DATA_t datamap;

memset(&datamap, 0, sizeof(IFX_TAPI_MAP_DATA_t));
ioctl(fd0, IFX_TAPI_ENC_STOP, 0);
ioctl(fd1, IFX_TAPI_ENC_STOP, 0);
/* Unmap data channels */;
datamap.nDstCh = 0;
datamap.nChType = IFX_TAPI_MAP_DATA_TYPE_PHONE;
ioctl(fd0, IFX_TAPI_MAP_DATA_REMOVE, &datamap);
ioctl(fd1, IFX_TAPI_MAP_DATA_REMOVE, &datamap);
```

3 Feature Description

The topics described in this chapters are reported in **Table 9**.

Table 9 Topics of Chapter 3

Topic	Chapter	Applicability
Channel configuration	**Chapter 3.1**	All application types.
Configuration and control of encoder/decoder	**Chapter 3.2**	All application types.
Connection statistics retrieval	**Chapter 3.3**	All application types.
Tone management (play, detect call progress tones, etc)	**Chapter 3.4**	All application types.
IP Phone ringing support	**Chapter 3.5**	For IP Phone applications.
Caller ID and power ringing support	**Chapter 3.6**	For ATA and Gateway applications.
Fax/modem support	**Chapter 3.7**	For ATA and Gateway applications.

3.1 Channel Configuration

This chapter describes configuration options for a TAPI channel, in particular how to set up and/or use.

Table 10 Topics of Chapter 3.1

Topic	Chapter	Applicability
Analog Line Channel Volume (RX and TX gains)	**Chapter 3.1.1**	For ATA and Gateway applications.
Audio Channel Volume and Mute	**Chapter 3.1.2**	For IP Phone applications.
PCM interface	**Chapter 3.1.3**	All application types.
Line Echo Canceller (LEC)	**Chapter 3.1.4**	For ATA and Gateway applications.
Jitter Buffer parameters	**Chapter 3.1.5**	All application types.
RTP parameters	**Chapter 3.1.6**	All application types.

3.1.1 Analog Line Channel Volume

It is possible to tune input (RX) and output (TX) gains independently for the analog phone port[1)] using the interface **IFX_TAPI_PHONE_VOLUME_SET** for the channel file descriptor.

Programming RX and TX gains directly influences input and output level; however the absolute signal levels are given by several components.

The measured output level is normally given by:

- Source level: level of the signal source, for example when playing a busy tone the level is specified when configuring the tone table. Output gain of the FW blocks processing the signal: these are configurable through FW coefficient download.
- TX gain: configured using **IFX_TAPI_PHONE_VOLUME_SET**.
- Downloaded country-specific coefficients and line impedance (depending on HW design and line characteristics).

For playing tones with relatively high absolute levels (above about 3 dBm), please also refer to **Chapter 3.4.7**.

The absolute input level that will be coded in the RTP flow is normally given by:

- Source level at tip&ring.
- Analog coefficients downloaded to the device and line impedance (depending on HW design and line characteristics).

1) Note that in a typical VINETIC® system, not all channels contain an analog phone port (ALM).

- RX gain: configured using **IFX_TAPI_PHONE_VOLUME_SET**.
- Input gain of the FW blocks processing the signal. Possible to configure through FW coefficient download.

Attention: The relative gain in the TX/RX Paths should be always set using the BBD coefficients, while keeping the programmable TX-RX gain to the default values of 0 dB. The latter might be used for any variation of the downloaded BBD coefficients, or for any application requiring an "on-the-fly" adaptation of the signal levels. Moreover, any arbitrary change of RX and/or TX gains may influence the overall system performance. Therefore the allowed range of the gain variation must be carefully evaluated.

Example - Volume

```
/* Configure RX and TX gains */
IFX_TAPI_LINE_VOLUME_t volume;

memset(&volume, 0, sizeof(IFX_TAPI_LINE_VOLUME_t));

/* Set the input gain to, for example, -2 dB */
volume.nGainRx = -2;
/* Set the output gain to, for example, 0 dB */
volume.nGainTx = 0;

ioctl(fd, IFX_TAPI_PHONE_VOLUME_SET, (IFX_int32_t) &volume);
```

3.1.2 Audio Channel Volume and Mute

It is possible to set the volume level step for the audio channel using interface **IFX_TAPI_AUDIO_VOLUME_SET** for the channel file descriptor. The volume levels steps are downloaded.

The audio channel can be muted at any time using interface **IFX_TAPI_AUDIO_MUTE_SET**.

Attention: The volume level steps to be supported must be downloaded in a BBD file.

Example - Audio Channel Volume and Mute

```
/* Configure Audio Channel Volume and Mute */
IFX_int32_t volume;
IFX_operation_t mute;

/* Configure volume step 3 */
volume = 3;
ioctl(fd, IFX_TAPI_AUDIO_VOLUME_SET, (IFX_int32_t) volume);

/* Do something ... */

/* Now mute the audio channel */
mute = IFX_ENABLE;
ioctl(fd, IFX_TAPI_AUDIO_MUTE_SET, (IFX_int32_t) mute);

/* Do something ... */

/* Now unmute the audio channel, restoring the voice path */
mute = IFX_DISABLE;
ioctl(fd, IFX_TAPI_AUDIO_MUTE_SET, (IFX_int32_t) mute);
```

3.1.3 PCM

PCM interface communication can be used to connect the product (for example the VINETIC® or DANUBE®) to an external device (such as DuSLIC or DECT chip set).

Each product can communicate with external devices over the PCM interface. With **IFX_TAPI_PCM_CFG_SET**, the application must assign RX/TX time slots, used PCM highway and coding (8-bit A-law, 8-bit µ–law or 16-bit linear).

Once the communication is configured, command **IFX_TAPI_PCM_ACTIVATION_SET** activates the communication from the product side: the device will start transmitting/receiving on the programmed time slots and highway.

Attention: Activation of LEC in the PCM interface might be required for ATA and Gateway applications; see also Chapter 3.1.4.

Example - Program PCM Interface

```
IFX_TAPI_PCM_CFG_t pcmConf;

memset(&pcmConf, 0, sizeof (IFX_TAPI_PCM_CFG_t));

pcmConf.nHighway = 0;
pcmConf.nResolution = IFX_TAPI_PCM_RES_ALAW_8BIT;
pcmConf.nRate = 2048;

pcmConf.nTimeslotRX = 0;
pcmConf.nTimeslotTX = 1;

ioctl(fd, IFX_TAPI_PCM_CFG_SET, (IFX_int32_t) &pcmConf);

/* PCM channel has been configured, however the communication is not active yet */
ioctl(fd0, IFX_TAPI_PCM_ACTIVATION_SET, (IFX_int32_t) 1);

/* Now PCM communication is started */
```

3.1.3.1 Example of System using PCM Interface

The example in **Figure 14** shows some possible flows for a scenaro including the VoIP subsystem and a DuSLIC-S chip set (only codec + SLIC, DSP functionality not provided).

- Flow A: Classical VoIP call, not involving PCM.
- Flow B: Call switched between VoIP subsystem and DuSLIC, data resources are used only for signal detection, tone generation and Caller ID, not for RTP and JB.
- Flow C: DuSLIC VoIP call using VoIP subsystem resources.

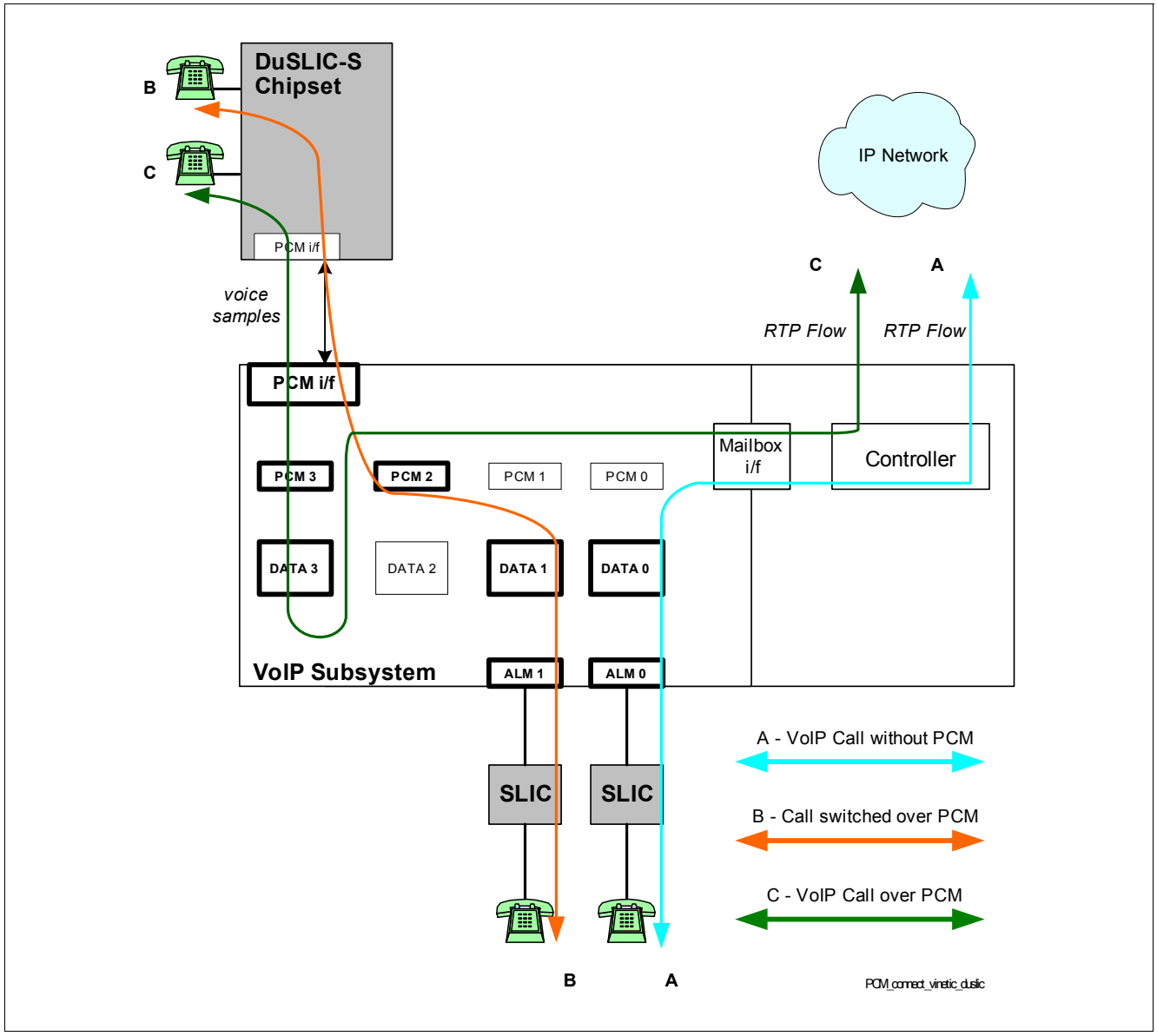

Figure 14 Connection of a DuSLIC via PCM Interface

3.1.4 LEC

The LEC services are used to choose the LEC type[1)] (near-end or near-end plus far-end) and to select whether or not to activate NLP (Non-Linear Processing).

The LEC (Line Echo Canceller) can be activated in the analog phone interface (**IFX_TAPI_WLEC_PHONE_CFG_SET**) or in the PCM interface (**IFX_TAPI_WLEC_PCM_CFG_SET**); decision where to activate the LEC depends on the phone channel resource used (ALM or PCM). Parameter is a pointer to a **IFX_TAPI_WLEC_CFG_t** struct:

- Field nType must be used to control the LEC
  - **IFX_TAPI_WLEC_TYPE_NE** to activate near-end echo cancellation.
  - **IFX_TAPI_WLEC_TYPE_NFE** to activate both near-end and far-end cancellation.
  - **IFX_TAPI_WLEC_TYPE_OFF** to disable the LEC functionality.

1) Please refer to the system release notes for the list of supported LEC types.

- Field bNlp must be used to control the non-linear processing (if the LEC is active)
  - **IFX_TAPI_WLEC_NLP_ON** to activate the NLP
  - **IFX_TAPI_WLEC_NLP_OFF** to deactivate the NLP

In systems where the analog telephone is directly connected to analog interface (for example flow A in **Figure 14**), the LEC has to be activated in the analog interface. For systems using PCM to connect additional analog lines (for example flow C in **Figure 14**), the LEC must[1)] be activated in the PCM interface.

Attention: WLEC is the name of a new type of LEC FW algorithn, included in some Infineon products to add far-end line echo cancellation. Although referring to the new WLEC, the interfaces documented in this chapter can be used with any near-end LEC implementation already supported by older TAPI versions. Customer using interfaces IFX_TAPI_LEC_* should migrate to the new IFX_TAPI_WLEC interfaces.

Example - LEC Configuration

```
/* Activate LEC with NLP for analog port */

IFX_TAPI_WLEC_CFG_t lecConf;
memset(&lecConf, 0, sizeof (IFX_TAPI_WLEC_CFG_t));

/* Enable LEC: near-end cancellation */;
lecConf.nType = IFX_TAPI_WLEC_TYPE_NE;
/* Activate NLP */;
lecConf.bNlp = IFX_TAPI_WLEC_NLP_ON;
ioctl(fd, IFX_TAPI_WLEC_PHONE_CFG_SET, (IFX_int32_t) &lecConf);

/* Now deactivate LEC */
lecConf.nType = IFX_TAPI_WLEC_TYPE_OFF;
ioctl(fd, IFX_TAPI_WLEC_PHONE_CFG_SET, (IFX_int32_t) &lecConf);
```

3.1.5 Jitter Buffer

Configuration of the Jitter Buffer (JB) is done using service **IFX_TAPI_JB_CFG_SET**, which makes it possible to set the following JB properties:

- Type: fixed or adaptive JB
- Local adaptation type
- Optimization for voice traffic or data (fax/modem) traffic
- Scaling
- Initial, minimum and maximum size

Example - Jitter Buffer

```
IFX_TAPI_JB_CFG_t jbCfgVoice;

memset (&jbCfgVoice, 0, sizeof(IFX_TAPI_JB_CFG_t));

/* Adaptive JB */
jbCfgVoice.nJbType = IFX_TAPI_JB_TYPE_ADAPTIVE;
/* Optimization for voice */
jbCfgVoice.nPckAdpt = IFX_TAPI_JB_PKT_ADAPT_VOICE;
/* nScaling multiplied by the packet length determines the play-out delay */
```

1) Only if the device providing the analog interface does not use the LEC.

```
jbCfgVoice.nScaling = 0x10;
/* Initial JB size, in 125 µs steps: 90 ms = 0x2D0 * 125 µs */
jbCfgVoice.nInitialSize = 0x2D0;
/* Minimum JB size, in 125 µs steps: 10 ms = 0x50 * 125 µs */
jbCfgVoice.nMinSize = 0x50;
/* Minimum JB size, in 125 µs steps: 180 ms = 0x5A0 * 125 µs */
jbCfgVoice.nMaxSize = 0x5A0;

ioctl(fd, IFX_TAPI_JB_CFG_SET, (IFX_int32_t) &jbCfgVoice);
```

3.1.6 RTP

Before starting any RTP session, the application software has to set up several RTP connection parameters for each channel (specified by the channel fd). Two interfaces are provided for it:

- **IFX_TAPI_PKT_RTP_PT_CFG_SET** to configure the payload type tables, see **Chapter 3.1.6.1**.
- **IFX_TAPI_PKT_RTP_CFG_SET** to configure RTP session parameters, see **Chapter 3.1.6.2**.

3.1.6.1 RTP Payload Type Tables

Interface **IFX_TAPI_PKT_RTP_PT_CFG_SET** must be used to configure, on a per-channel basis, the association vocoder-payload type in so-called payload tables. It is possible to configure payload types independently for upstream and downstream direction. The list of encoding types is defined in **IFX_TAPI_ENC_TYPE_t**.

In downstream, only RTP frames with “known” payload types will be decoded. RTP frames with payload type not programmed using **IFX_TAPI_PKT_RTP_PT_CFG_SET** will be discarded. This means that the tables must be carefully programmed with all vocoders to be supported.

In upstream direction, the generated RTP frames will use the payload type associated with the operating vocoder.

Attention: For some vocoders the same encoding type is used for different bitrates (for example G.723), for some other algorithms (for example G.726) a dynamic[1] payload type value is defined for different bitrates. For detailed information on how to choose the correct payload type please refer to RFC 3551 (especially Section 6).

Example - RTP Payload Types Tables

```
/* Configure RTP Payload Type tables */
IFX_TAPI_PKT_RTP_PT_CFG_t rtpPTConf;

memset(&rtpPTConf, 0, sizeof(IFX_TAPI_PKT_RTP_PT_CFG_t));

/* First prepare the table values */
/* Payload Types, assuming the application supports G.711 A/µLaw, */
/* G.729AB and iLBC */
/* Assign different payload type for iLBC in upstream and downstream */
/* Values for G.711, G.729 are taken from RFC 3551, values for iLBC are */
/* not actual! */

/* Payload types table - Upstream */
rtpPTConf.nPTup[IFX_TAPI_ENC_TYPE_MLAW] = 0;
rtpPTConf.nPTup[IFX_TAPI_ENC_TYPE_ALAW] = 8;
rtpPTConf.nPTup[IFX_TAPI_ENC_TYPE_G729_AB] = 18;
rtpPTConf.nPTup[IFX_TAPI_ENC_TYPE_ILBC_152] = 99;
```

1) RFC 3551 terminology.

```
/* Payload types table - Downstream*/
rtpPTConf.nPTdown[IFX_TAPI_ENC_TYPE_MLAW] = 0;
rtpPTConf.nPTdown[IFX_TAPI_ENC_TYPE_ALAW] = 8;
rtpPTConf.nPTdown[IFX_TAPI_ENC_TYPE_G729_AB] = 18;
rtpPTConf.nPTdown[IFX_TAPI_ENC_TYPE_ILBC_152] = 97;

/* Program the channel (addressed by fd) with the specified payload types */
ioctl(fd, IFX_TAPI_PKT_RTP_PT_CFG_SET, (IFX_int32_t) &rtpPTConf);
```

3.1.6.2 RTP Session Parameters

Parameters relevant for the RTP session, according to RFC 3550, are configured using **IFX_TAPI_PKT_RTP_CFG_SET**: initial value for the sequence number, SSRC for voice and SID frames, and payload type for RFC 2833 packets.

Handling of DTMF and Fax/Modem signal events are also configured with the same interface; in particular, the user can select whether the events will be sent in-band and/or out-of-band using RFC 2833.

Example - RTP Session Parameters

```
/* This is only an example to demonstrate how the API can be used */
/* The used configuration parameters must be adapted to the real product */
IFX_TAPI_PKT_RTP_CFG_t rtpConf;

memset(&rtpConf, 0, sizeof(IFX_TAPI_PKT_RTP_CFG_t));

rtpConf.nSeqNr = 0;
rtpConf.nSsrc = 0;

/* Enable only out-of-band (RFC 2833 packet) transmission */
rtpConf.nEvents = IFX_TAPI_PKT_EV_OOB_ONLY;
/* Configure payload type for RFC 2833 packets*/
rtpConf.nEventPT = 18;
/* Play out the signal upon RFC 2833 packets reception */
rtpConf.nEventsPlay = IFX_TAPI_PKT_EV_OOBPLAY_PLAY;
ioctl(fd, IFX_TAPI_PKT_RTP_CFG_SET, (IFX_int32_t) &rtpConf);
```

3.2 Encoder/Decoder Control

Configuration of the encoder is done using interfaces:

- **IFX_TAPI_ENC_TYPE_SET**, to select the encoder (alias vocoder) type
- **IFX_TAPI_ENC_FRAME_LEN_SET**, to set the packetization time
- **IFX_TAPI_ENC_VAD_CFG_SET**, to enable/disable Voice Activity Detection (silence compression and comfort noise generation)
- **IFX_TAPI_ENC_VOLUME_SET**, to configure the encoding level

After configuration of the encoder parameters, start/stop encoding can be controlled via **IFX_TAPI_ENC_START** and **IFX_TAPI_ENC_STOP**. This will also automatically start/stop generation of RTP frames. Before starting encoding, packetization session parameters have to be set (for RTP see also **Chapter 3.1.6**); otherwise, random RTP parameters will apply!

In a similar fashion, decoding of the RTP frames can be also controlled with **IFX_TAPI_DEC_START** and **IFX_TAPI_DEC_STOP**.

Attention: Issuing IFX_TAPI_DEC_STOP and/or IFX_TAPI_ENC_STOP, lead to reset of the connection statistics maintained in the device!

Example - Encoder/Decoder Control

```
IFX_int32_t encFrameLen, encType, encVAD;

/* Encoding time 10 ms */
encFrameLen = 10;
ioctl(fd, IFX_TAPI_ENC_FRAME_LEN_SET, (IFX_int32_t) encFrameLen);

/* Set the encoder */
encType = IFX_TAPI_ENC_TYPE_MLAW;
ioctl(fd, IFX_TAPI_ENC_TYPE_SET, (IFX_int32_t) encType);

/* Set the VAD on */
encVAD = IFX_TAPI_ENC_VAD_ON;
ioctl(fd, IFX_TAPI_ENC_VAD_CFG_SET, (IFX_int32_t) encVAD);

/* Start encoding: it starts also RTP packet flow */
/* Parameter is not needed */
ioctl(fd, IFX_TAPI_ENC_START, (IFX_int32_t) 0);

/* Do something ... */

/* Stop encoding: it stops also RTP packet flow */
/* Parameter is not needed */
ioctl(fd, IFX_TAPI_ENC_STOP, (IFX_int32_t) 0);
```

3.3 RTCP and Jitter Buffer Statistics

RTCP and jitter buffer statistics, for a certain channel, can be read at any time using respectively **IFX_TAPI_PKT_RTCP_STATISTICS_GET** and **IFX_TAPI_JB_STATISTICS_GET**.

Attention: Disabling encoder and/or decoder will reset all statistics!

Example - Read Statistics

```
/* Configure RTP Payload Type tables */
/* This is only an example to demonstrate how the API can be used */
/* Read all connection statistics available in firmware */
IFX_TAPI_PKT_RTCP_STATISTICS_t rtcpStat;
IFX_TAPI_JB_STATISTICS_t jbStat;

memset(&rtcpStat, 0, sizeof(rtcpStat));
memset(&jbStat, 0, sizeof(jbStat));

ioctl(fd, IFX_TAPI_PKT_RTCP_STATISTICS_GET, &rtcpStat);
ioctl(fd, IFX_TAPI_JB_STATISTICS_GET, &jbStat);
```

3.4 Tone API

The tone management API allows generation and detections of tones specified and stored in an internal table called “Tone Table.” Besides typical DTMF and call progress tones[1], it is possible to define tones with more complex cadences and combinations of frequencies.

Table 11 Topics of Chapter 3.4

Topic	Chapter	Applicability
Usage of the Tone Table	**Chapter 3.4.1**	All application types.
Play tones	**Chapter 3.4.2**	All application types.
Definition of simple tones	**Chapter 3.4.3**	All application types.
Definition of composed tones	**Chapter 3.4.4**	All application types.
Predefined tones	**Chapter 3.4.5**	All application types.
Play tones with high output level	**Chapter 3.4.6**	For ATA and Gateway applications.
Play special tones	**Chapter 3.4.7**	All application types.
Call progress tone detection	**Chapter 3.4.8**	All application types.

3.4.1 Tone Table

Before tones can be generated (and also recognized), tones have to be defined using a tone descriptor (**IFX_TAPI_TONE_SIMPLE_t** and/or **IFX_TAPI_TONE_COMPOSED_t**) and stored in an internal Tone Table, using **IFX_TAPI_TONE_TABLE_CFG_SET**. The Tone Table can store up to 256 different tones. The first 32 entries of the table (0-31) are predefined for DTMF and some other predefined tones[2]. The rest of the table can be used to store simple and composed tones.

The following figure shows examples of a simple and a composed tone.

1) For example busy tone, ring back, disconnect tone, etc.

2) Example of dial tone, busy tone and ringing tone and other tones often used in telephony.

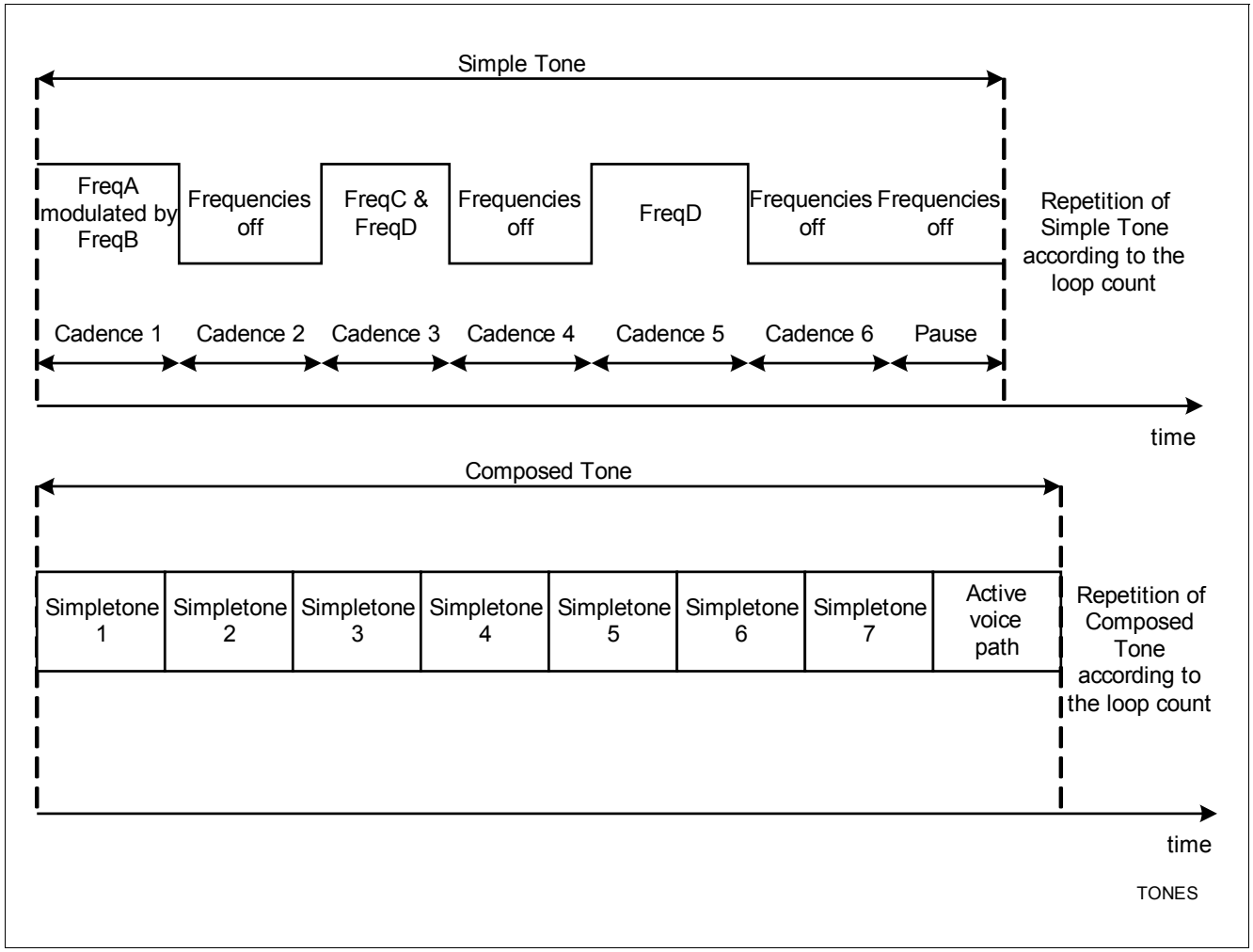

Figure 15 Simple and Composed Tones

As depicted in **Figure 15**, a simple tone can consist of four to six different cadences using four frequencies (f1, f2, f3, f4), and it is possible to define the level for each of the four frequencies. For each cadence, up to four frequencies can be active at the same time. Optionally it is possible to enable modulation; in this case, frequency f1 will be modulated using frequency f2; frequencies f3 and f4 will be summed up (see also **IFX_TAPI_TONE_SIMPLE_t**).

The composed tones consist of up to seven simple tones that are played in a sequence. If the loop count of a composed tone is greater than zero, it is also possible to activate the voice path between the loops.

3.4.2 Playing Tones

Playing tones is possible only for tones already present in the tone table using **IFX_TAPI_TONE_LOCAL_PLAY** and passing the tone index as a parameter.

For generation of tones towards the network, **IFX_TAPI_TONE_NET_PLAY** must be used.

In order to stop the tone generation, tone index 0 (zero) must be passed or the interface **IFX_TAPI_TONE_STOP** can be used.

Examples

```
/* start playing one simple tone, tone defined in tone table index 71 */
ioctl(fd, IFX_TAPI_TONE_LOCAL_PLAY, (IFX_int32_t) 71);
```

```
/* wait a little, but wait enough to hear whole tone */
sleep(5);

/* stop playing tone */
ioctl(fd, IFX_TAPI_TONE_LOCAL_PLAY, (IFX_int32_t) 0);

/* pause */
sleep(5);

/* now start playing a complex tone, tone defined in tone table index 100 */
ioctl(fd, IFX_TAPI_TONE_LOCAL_PLAY, (IFX_int32_t) 100);
/* wait a little, but wait enough to hear whole tone */
sleep(15);

/* stop playing tone */
ioctl(fd, IFX_TAPI_TONE_LOCAL_PLAY, (IFX_int32_t) 0);
```

3.4.3 Defining Simple Tones

The definition of a simple tone is done by filling up a **IFX_TAPI_TONE_t** union using the `.simple` fields: This means that the **IFX_TAPI_TONE_SIMPLE_t** struct is used. After filling the relevant fields, the structure must be inserted in the tone table using **IFX_TAPI_TONE_TABLE_CFG_SET**.

Example - Tone Configuration

```
IFX_TAPI_TONE_t tone;

memset(&tone, 0, sizeof(IFX_TAPI_TONE_t));
/* define a simple tone for tone table index 71 */
tone.simple.format = IFX_TAPI_TONE_TYPE_SIMPLE;
/* tone table index where to insert the tone */
tone.simple.index = 71;
/* using two frequencies */
/* 0 <= till < 4000 Hz */
tone.simple.freqA = 480;
tone.simple.freqB = 620;
/* tone level for freqA */
/* -300 < till < 0 */
tone.simple.levelA = -15;
/* tone level for freqB */
tone.simple.levelB = -20;
/* program first cadences (on time) */
tone.simple.cadence[0] = 2000;
/* program second cadences (off time) */
tone.simple.cadence[1] = 2000;
/* in the first cadence, both frequencies must be played */
tone.simple.frequencies[0] = IFX_TAPI_TONE_FREQA | IFX_TAPI_TONE_FREQB;
/* in the second cadence, all frequencies are off */
tone.simple.frequencies[1] = IFX_TAPI_TONE_FREQNONE;
/* the tone will be played two times (2 loops) */
tone.simple.loop = 2;
```

```
/* at the end of each loop there is a pause */
tone.simple.pause = 200;

/* update the tone table with the simple tone */
ioctl(fd, IFX_TAPI_TONE_TABLE_CFG_SET, (IFX_int32_t) &tone);
/* now the simple tone is added to the tone table at index 71 */

/* define a simple tone for tone table index 72 */
memset(&tone,0, sizeof(IFX_TAPI_TONE_t));
tone.simple.format = IFX_TAPI_TONE_TYPE_SIMPLE;
/* tone table index where to insert the tone */
tone.simple.index = 72;
tone.simple.freqA = 480;
tone.simple.levelA = -15;
tone.simple.cadence[0] = 2000;
tone.simple.cadence[1] = 500;
tone.simple.frequencies[0] = IFX_TAPI_TONE_FREQA;
tone.simple.frequencies[1] = IFX_TAPI_TONE_FREQNONE;
/* the tone will be played one time (1 loop) */
tone.simple.loop = 1;
tone.simple.pause = 0;

/* add simple tone to the tone table */
ioctl(fd, IFX_TAPI_TONE_TABLE_CFG_SET, (IFX_int32_t) &tone);
/* now the simple tone is added to the tone table at index 72 */
```

3.4.4 Defining Composed Tones

The definition of a composed tone is done by filling up a **IFX_TAPI_TONE_t** union using the `.composed` fields: This means that the **IFX_TAPI_TONE_COMPOSED_t** struct is used. After filling the relevant fields the structure must be inserted in the tone table using **IFX_TAPI_TONE_TABLE_CFG_SET**.

Example - Composed Tone Configuration

```
IFX_TAPI_TONE_t tone;

/* define a complex tone for tone table index 100 */
memset(&tone, 0, sizeof(IFX_TAPI_TONE_t));

tone.composed.format = IFX_TAPI_TONE_TYPE_COMPOSED;
/* tone table index where to insert the tone */
tone.composed.index = 100;
/* the complex tone uses two simple tones already present in the */
/* tone table */
tone.composed.count = 2;
/* now list the simple tones that compose the complex tone */
/* assumption that two simple tones are defined in tone table */
/* indexes 71 and 72 */
/* First simple tone 71 will be played (so many times as his loop is) */
/* then second simple tone 72 will be played (so many times as his loop is) */
tone.composed.tones[0] = 71;
tone.composed.tones[1] = 72;
```

```
/* add complex tone to the tone table */
ioctl(fd, IFX_TAPI_TONE_TABLE_CFG_SET, (IFX_int32_t) &tone);
/* now the complex tone is added to the tone table at index 100
```

3.4.5 Predefined Tones

The same interfaces can be used to play and stop predefined tones. The only difference between handling of predefined and user-defined tones is that for the reserved tone table indexes, interface **IFX_TAPI_TONE_TABLE_CFG_SET** can be used only to define on/off time (using the first two elements of the cadence array) and level. Other parameters defined in **IFX_TAPI_TONE_SIMPLE_t** are ignored; frequencies are predefined and can not be modified.

Table 12 reports the predefined tones entries in the tone table.

Table 12 Tone Table - Predefined Tones

Index	Frequency 1	Level1 [dB]	Frequency 2	Level 2 [dB]	Note
1	697	-11	1209	-9	DTMF digit 1
2	697	-11	1336	-9	DTMF digit 2
3	697	-11	1477	-9	DTMF digit 3
4	770	-11	1209	-9	DTMF digit 4
5	770	-11	1336	-9	DTMF digit 5
6	770	-11	1477	-9	DTMF digit 6
7	852	-11	1209	-9	DTMF digit 7
8	852	-11	1336	-9	DTMF digit 8
9	852	-11	1477	-9	DTMF digit 9
10	941	-11	1209	-9	DTMF digit * (star)
11	941	-11	1336	-9	DTMF digit 0 (zero)
12	941	-11	1477	-9	DTMF digit # (number)
13	800	-9	-	0	-
14	1000	-9	-	0	-
15	1250	-9	-	0	-
16	950	-9	-	0	-
17	1100	-9	-	0	CNG Tone
18	1400	-9	-	0	-
19	1500	-9	-	0	-
20	1600	-9	-	0	-
21	1800	-9	-	0	-
22	2100	-9	-	0	CED Tone
23	2300	-9	-	0	-
24	2450	-9	-	0	-
25	350	-11	440	-9	Dial tone example
26	440	-11	480	-9	Ringing tone example
27	480	-11	620	-9	Busy tone example
28	697	-11	1633	-9	DTMF digit A
29	770	-11	1633	-9	DTMF digit B

Table 12 Tone Table - Predefined Tones (cont'd)

Index	Frequency 1	Level1 [dB]	Frequency 2	Level 2 [dB]	Note
30	852	-11	1633	-9	DTMF digit C
31	941	-11	1633	-9	DTMF digit D

3.4.6 Generating Tones with High-level Output

This chapter is relevant only to ATA and VoIP Gateway applications.

Some howler tones require relative high levels (BT howler tone requires up to +15 dBm) in order to enable the output levels above the maximum level normally permitted (about 3.14 dBm). The service **IFX_TAPI_LINE_LEVEL_SET** for the ALM fd should be used to enable maximum output level path. If this is not done, the maximum output level will be limited to about +3.14 dBm. Interface **IFX_TAPI_PHONE_VOLUME_SET** can be used to fine-tune the signal output level (see also **"Analog Line Channel Volume" on Page 42**).

Assuming that the signal stored in the RTP sequence is encoded to reach up to level $7FFF_H$ (maximum digital value) and with output gain of 0 dB and path set to "high level," the output signal will reach the maximum level permitted by the device.

Attention: Immediately after playing an howler tone, if previously enabled, the high-level path must be now disabled using the service IFX_TAPI_LINE_LEVEL_SET. Furthermore, if a dedicated output gain (using IFX_TAPI_PHONE_VOLUME_SET) has been set because of the howler tone, it is important to reconfigure the output gain to the original value.

Example - Howler Tones with High-level Output

```
   /* Example of playing Howler tone with output level exceeding +3.14 dBm */
   IFX_TAPI_LINE_VOLUME_t gains_normal, gains_howler;

   memset(&gains_normal, 0, sizeof(IFX_TAPI_LINE_VOLUME_t));
   memset(&gains_howler, 0, sizeof(IFX_TAPI_LINE_VOLUME_t));

   /* set gains for normal operation mode */
   gains_normal.nGainRx = -3;
   gains_normal.nGainTx = 0;

   /* set TX gain for howler tone, if required */
   gains_howler.nGainRx = 24; /* more than 3 dB */
   /* initialize system with default gains */
   ioctl(fd, IFX_TAPI_PHONE_VOLUME_SET, (IFX_int32_t) &gains_normal);

 /* do something else... */

   /* now play the howler tone (the tone has been defined earlier) */
   ioctl(fd, IFX_TAPI_TONE_LOCAL_PLAY, (IFX_int32_t) TONE_INDEX_HOWLER);

   /* wait a little to play tone */
   sleep(5);

   /* after some time the device can stop playing the tone */
   ioctl(fd, IFX_TAPI_TONE_STOP, (IFX_int32_t) TONE_INDEX_HOWLER);

   /* application decides that the howler tone must be played */
   /* (if required) change TX gain */
```

```
ioctl(fd, IFX_TAPI_PHONE_VOLUME_SET, (IFX_int32_t) &gains_howler);
/* enable device to generate high level output */
ioctl(fd, IFX_TAPI_LINE_LEVEL_SET, (IFX_int32_t) IFX_TAPI_LINE_LEVEL_ENABLE );
/* now play the howler tone (the tone has been defined earlier) */
ioctl(fd, IFX_TAPI_TONE_LOCAL_PLAY, (IFX_int32_t) TONE_INDEX_HOWLER);

/* wait a little to play tone */
sleep(5);

/* after some time the device can stop playing the tone */
ioctl(fd, IFX_TAPI_TONE_STOP, (IFX_int32_t) TONE_INDEX_HOWLER);
/* return to normal output level */
ioctl(fd, IFX_TAPI_LINE_LEVEL_SET,
(IFX_int32_t) IFX_TAPI_LINE_LEVEL_DISABLE);
/* (if gain changed at the beginning of the sequence) change TX gain */
ioctl(fd, IFX_TAPI_PHONE_VOLUME_SET, (IFX_int32_t) &gains_normal);
```

3.4.7 Generating Special Tones

The Infineon products can automatically generate a variety of tones; nevertheless some special tones (such as the howler tone defined by BT) can not be directly generated by using internal product resources. In order to generate this kind of special tone, support from application software is required. The application software has to provide the TAPI with RTP packets containing the encoding[1)] of the special tone.

3.4.8 Call Progress Tone Detection

Some Infineon devices can be programmed to detect Call Progress Tones (CPTs, for example a busy tone) or in general, all tones that can be defined as simple tones in the tone table. In order to detect a tone, it must first be added to the tone table.

Detection starts on a certain data channel after programming the CPT detector with command **IFX_TAPI_TONE_CPTD_START** for the channel file description, and giving as parameter a pointer to a structure **IFX_TAPI_TONE_CPTD_t** that must be programmed with the tone table index in order to be detected, and the direction (receive or transmit).

The detection of the tone is signaled through the event reporting interface, with event id (**IFX_TAPI_EVENT_TONE_DET_CPT**, reporting the direction (local or network).

Disabling detection of the tone, for the same file descriptor, is done using the interface **IFX_TAPI_TONE_CPTD_STOP**. This command does not require additional parameters.

Examples

```
IFX_TAPI_TONE_CPTD_t cpt;

memset (&cpt, 0, sizeof(IFX_TAPI_TONE_CPTD_t));
/* Detect busy tone, present in the tone table at index BUSY_TONE */
cpt.tone = BUSY_TONE;
/* Tone to be detected in transmit direction (typical for FXO) */
cpt.signal = IFX_TAPI_TONE_CPTD_DIRECTION_TX;
/* Start CPT detector */
ioctl(fd, IFX_TAPI_TONE_CPTD_START, (IFX_int32_t) &cpt);
```

1) The coding type must be one of the vocoders supported by the product.

```
/* Applications waits for CPTD event */

/* Wait for IFX_TAPI_EVENT_TONE_DET_CPT event */

/* After some time, application wants to stop CPT detection */
ioctl(fd, IFX_TAPI_TONE_CPTD_STOP, (IFX_int32_t) 0);
```

3.5 IP Phone Ringing

Command **IFX_TAPI_AUDIO_RING_START** must be used to start ringing, giving as parameter the tone table index where the ringing cadence is defined.

Command **IFX_TAPI_AUDIO_RING_STOP** is used to stop ringing.

Example - IP Phone Ringing

```
IFX_int32_t ringCadenceIndex;

/* Position of the ring cadence in the tone table */
ringCadenceIndex = RING_TONE;

ioctl(fd, IFX_TAPI_AUDIO_RING_START, ringCadenceIndex);

/* .... do something.... */

/* Stop ringing */
ioctl(fd, IFX_TAPI_AUDIO_RING_START, 0);
```

3.6 Power Ringing and Caller ID Support

The following sections in this chapter describe how to program Caller ID (CID) and power ringing usage.

Table 13 Topics of Chapter 3.4

Topic	Chapter	Applicability
Introduction to CID and Ringing support	Chapter 3.6.1	For ATA and Gateway applications.
CID Configuration	Chapter 3.6.2	For ATA and Gateway applications.
Ringing	Chapter 3.6.3	For ATA and Gateway applications.
CID Transmission	Chapter 3.6.4	For ATA and Gateway applications.
CID Reception	Chapter 3.6.5	For ATA and Gateway applications.

3.6.1 Introduction to Power Ringing and Caller ID Features

TAPI interfaces are provided for CID transmission/reception and ringing, the following services are provided

- Caller ID transmission associated with ringing
- Caller ID transmission not associated with ringing
- Ringing without Caller ID transmission
- Caller ID reception

An overview of implemented CID types is shown in **Table 14**; the various types are supported[1)] according to Telcordia, ETSI, BT and NTT standards.

1) For a reference list of supported CID feature and standards, please refer to the driver release notes.

Table 14 Caller ID Types

Type	Associated with Ringing	Hook Status
Caller ID type 1	Yes	On-hook
Caller ID type 2	No	Off-hook
Message Waiting Indication/Visual Message Waiting Indication	No	On-hook

A fundamental step before starting any type of CID service is the configuration of the CID “engine” (**Chapter 3.6.2**). It allows selection among CID standards, and optionally tunes the automatic generation of the CID sequences (for example program timing, ack tone).

After configuration of the CID engine, the application starts CID transmission (**Chapter 3.6.4**) or CID reception (**Chapter 3.6.5**) for a given channel. Before starting a CID type 1, a ringing cadence must also be programmed.

A service is available for issuing a ring without CID transmission (ringing support in described **Chapter 3.6.3**)

3.6.1.1 Information on Caller ID

Generation of Caller ID can be divided into four phases:

- Phase 1 - Send Alert
- Phase 2 - Wait for ACK (not always required, depends on type and standard)
- Phase 3 - Send CID information
- Phase 4 - Ringing (only for CID type 1)

The operations required in the four phases are summarized in **Table 15** (on-hook transmission associated with ringing), **Table 16** (on-hook transmission not associated with ringing), **Table 17** (off-hook transmission) and **Table 18** (abbreviations used).

Table 15 Caller ID Generation - On-hook CID (type 1)

Standard	Phase 1 - Alert	Phase 2 - ACK	Phase 3 - CID Tx	Phase 4 - Ringing
Telcordia on-hook	Ring (RB)	N/A	FSK	Periodical ringing
ETSI “during ringing”	Ring (RB)	N/A	FSK/DTMF	Periodical ringing
ETSI “prior to ringing” with DTAS	DTAS	N/A	FSK/DTMF	Periodical ringing
ETSI “prior to ringing” with LR	LR - DTAS	N/A	FSK/DTMF	LR - Periodical ringing
ETSI “prior to ringing” with RP	Ring (RP)	N/A	FSK/DTMF	Periodical ringing
SIN 227 (BT) on-hook	LR - DTAS	N/A	FSK	LR - Periodical ringing
NTT (Japan) on-hook	LR - CAR	off-hook	FSK - on-hook	LR - Periodical ringing

Table 16 Caller ID Generation - On-hook MWI

Standard	Phase 1 - Alert	Phase 2 - ACK	Phase 3 - CID Tx	Phase 4 - Ringing
Telcordia on-hook	OSI	N/A	FSK	N/A
ETSI “during ringing”	N/A	N/A	N/A	N/A
ETSI “prior to ringing” with DTAS	DTAS	N/A	FSK/DTMF	N/A
ETSI “prior to ringing” with LR	LR - DTAS	N/A	FSK/DTMF	N/A
ETSI “prior to ringing” with RP	Ring (RP)	N/A	FSK/DTMF	N/A

Table 16 Caller ID Generation - On-hook MWI (cont'd)

Standard	Phase 1 - Alert	Phase 2 - ACK	Phase 3 - CID Tx	Phase 4 - Ringing
SIN 227 (BT) on-hook	LR - DTAS	N/A	FSK	N/A
NTT (Japan) on-hook	N/A	N/A	N/A	N/A

Table 17 Caller ID Generation - Off-hook CID (type 2) and off-hook MWI

Standard	Phase 1 - Alert	Phase 2 - ACK	Phase 3 - CID Tx	Phase 4 - Ringing
Telcordia off-hook	CAS	'D'	FSK	N/A
ETSI off-hook	DTAS	'D'. Optional 'A', 'B', 'C'.	FSK/DTMF	N/A
SIN 227 (BT) off-hook	DTAS	'D'	FSK	N/A
NTT (Japan) off-hook	AS	N/A	FSK	N/A

Table 18 Caller ID Generation - Abbreviations

Abbreviation	Definition	Note
RB	Ring Burst	
RP	Ring Pulse	It is a short Ring Burst (defined by ETSI)
CAR	Signal Receiver Seizing Signal	It is a short Ring Burst (defined by NTT)
FSK	Frequency Shift Keying	CID data are transmitted using FSK modem modulation, using V.23 or Bell202 standard. Japanese CID uses a modified V.23 data coding.
DTMF	Dual Tone Multi-Frequency	CID data are transmitted as sequence of DTMF tones.
ACK	Acknowledge	Signal produced by CPE to indicate "ready" for processing CID protocol. It is typically a DTMF digit or a short off-hook.
DTAS	Dual Tone Alert Signal	Tone used to alert the CPE that a CID protocol transmission is going to start. ETSI / SIN terminology.
CAS	CPE Alert Signal	Tone used to alert the CPE that a CID protocol transmission is going to start. Telcordia terminology.
AS	Alert Signal	Tone used to alert the CPE that a CID protocol transmission is going to start. NTT (Japan) terminology.
LR	Line Reversal (alias Polarity Reversal)	Normal Polarity is re-established with the beginning of the periodic ringing
OSI	Open Switching Interval	A period of line high impedance

3.6.2 CID Configuration

Selection of CID standard to be used is done with service **IFX_TAPI_CID_CFG_SET** and struct **IFX_TAPI_CID_CFG_t**, and minimum configuration required is to specify which CID standard shall be used for CID operations; this is selectable through field **nStandard**.

Furthermore, the user has the possibility to modify several parameters CID standard specific through **IFX_TAPI_CID_STD_TYPE_t**, which is a union of structs defining parameters (such as timing, FSK or DTMF parameters, etc.) typical for each supported CID standard. **Figure 16** gives an overview of the **IFX_TAPI_CID_CFG_t** structure.

The application software does not have to configure all fields of struct replacing **IFX_TAPI_CID_STD_TYPE_t**; for fields not configured, default values will be used. Default values for all configurable fields are given in the struct

documentation. As an example, if a pointer to a **IFX_TAPI_CID_TIMING_t** struct is not given, default values for the CID timing apply.

Ring cadence must be programmed (using **IFX_TAPI_RING_CADENCE_HR_SET**) before starting a CID sequence associated with ringing; see also **Chapter 3.6.3**.

Please refer to the coding examples on **Page 65**.

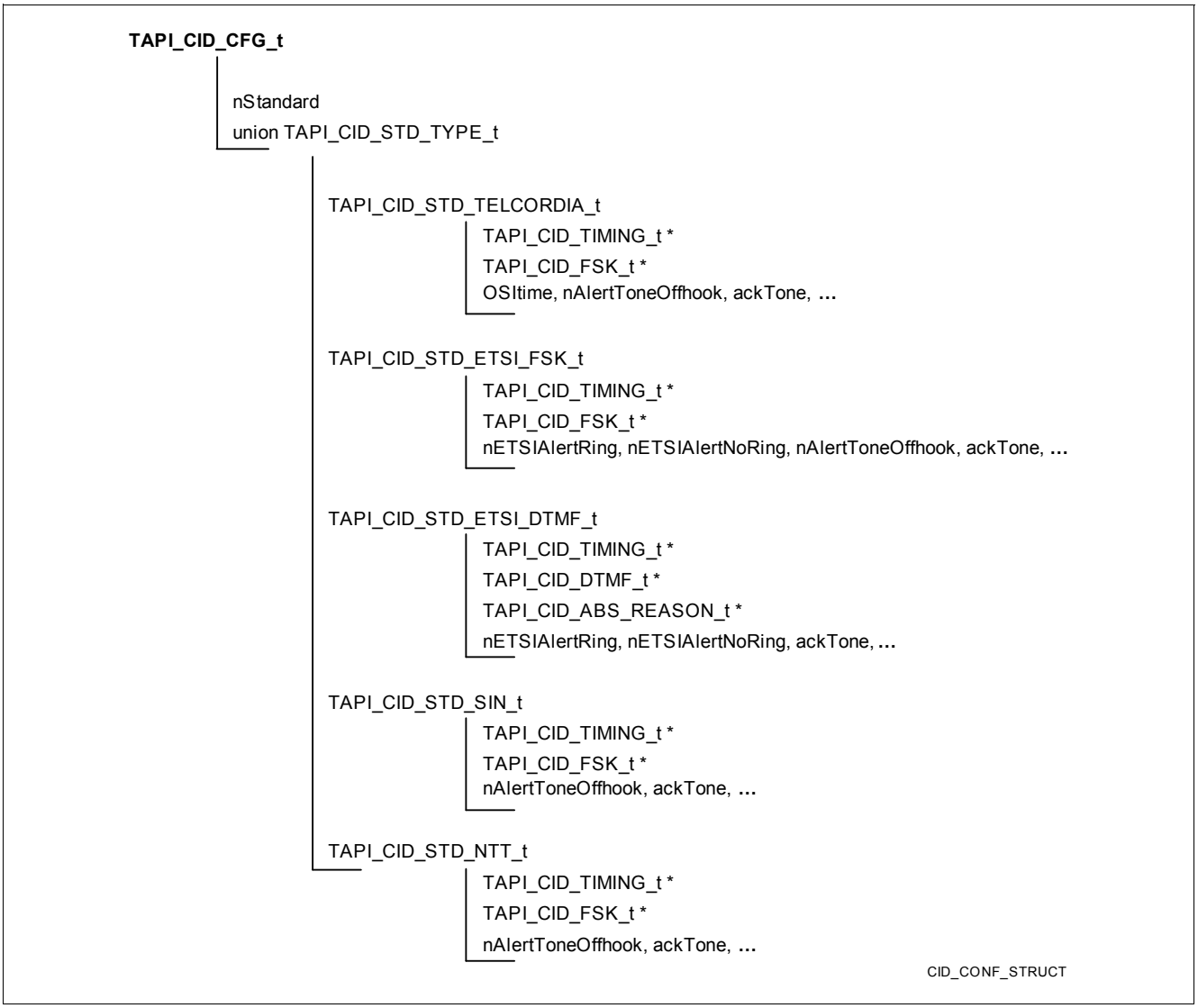

Figure 16 Overview of the Configuration Structure

CID Timing Configuration

It is possible to adjust timing of CID sequence transmission (**IFX_TAPI_CID_TX_SEQ_START**) modifying parameters included in struct **IFX_TAPI_CID_TIMING_t**.

Table 19 reports which parameters apply to which CID transmission standard, to be noted that some fields are located in the standard specific struct, not in **IFX_TAPI_CID_TIMING_t**.

Figure 17 up to **Figure 23** represent the parameters in different CID transmission scenarios. For a description of each timing parameter, please refer to the struct description.

The ring cadence used by CID type 1 is programmed using different interfaces, please refer to **Chapter 3.6.3.2**.

Table 19 Relevant Timing Applicable to the Different Standards

Timing Parameter	Telcordia /Bellcore	ETSI	SIN (BT)	NTT	Parameter Location
beforeData	N/A	N/A	N/A	Off-hook	IFX_TAPI_CID_TIMING_t
dataOut2restoreTimeOnhook	N/A	On-hook[1)]	On-hook	On-hook	IFX_TAPI_CID_TIMING_t
dataOut2restoreTimeOffhook	Off-hook	Off-hook	Off-hook	Off-hook	IFX_TAPI_CID_TIMING_t
ack2dataOutTime	Off-hook	Off-hook	Off-hook	On-hook	IFX_TAPI_CID_TIMING_t
cas2ackTime	Off-hook	Off-hook	Off-hook	N/A	IFX_TAPI_CID_TIMING_t
afterAckTimeout	Off-hook	Off-hook	Off-hook	N/A	IFX_TAPI_CID_TIMING_t
afterFirstRing	On-hook associated with ringing	On-hook associated with ringing[2)]	N/A	N/A	IFX_TAPI_CID_TIMING_t
afterRingPulse	N/A	On-hook[3)]	N/A	N/A	IFX_TAPI_CID_TIMING_t
afterDTASOnhook	N/A	On-hook[4)]	On-hook	N/A	IFX_TAPI_CID_TIMING_t
afterLineReversal	N/A	On-hook[5)]	On-hook	On-hook	IFX_TAPI_CID_TIMING_t
afterOSI	On-hook	N/A	N/A	N/A	IFX_TAPI_CID_TIMING_t
ringPulseTime	N/A	N/A	N/A	On-hook	IFX_TAPI_CID_STD_NTT_t
ringPulseTimeLoop	N/A	N/A	N/A	On-hook	IFX_TAPI_CID_STD_NTT_t
ringPulseOffTime	N/A	N/A	N/A	On-hook	IFX_TAPI_CID_STD_NTT_t
dataOut2incomingSuccessfulTimeout	N/A	N/A	N/A	On-hook	IFX_TAPI_CID_STD_NTT_t

1) Only for data transmission before ringing, not defined for data transmission during ringing
2) Only for data transmission between first and second ring
3) Only for data transmission after a ring pulse (RP)
4) Only for data transmission after DTAS alert or line reversal and DTAS alert
5) Only for data transmission after line reversal and DTAS alert

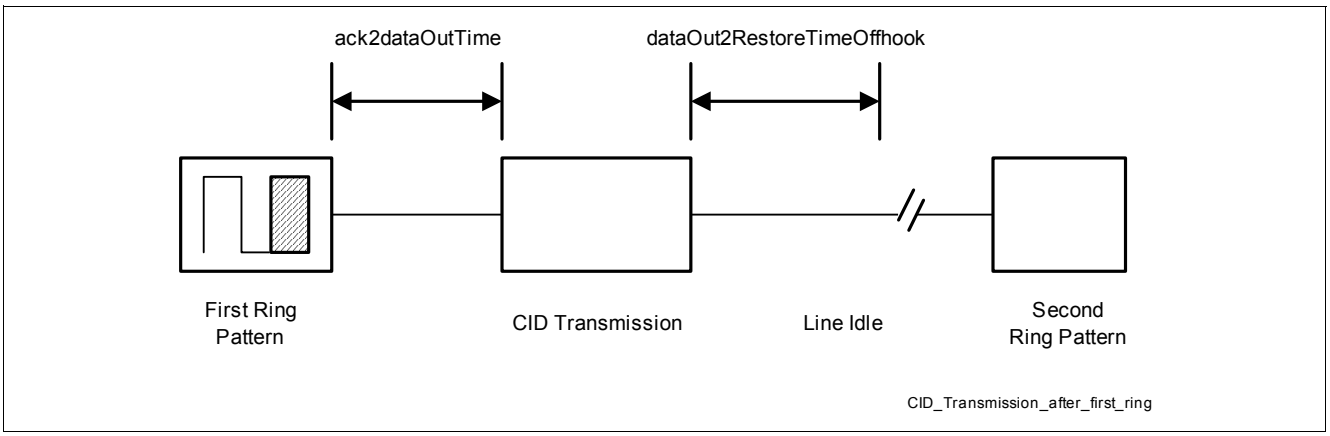

Figure 17 Timing for Telcordia and ETSI CID Type 1 with Transmission After First Ring

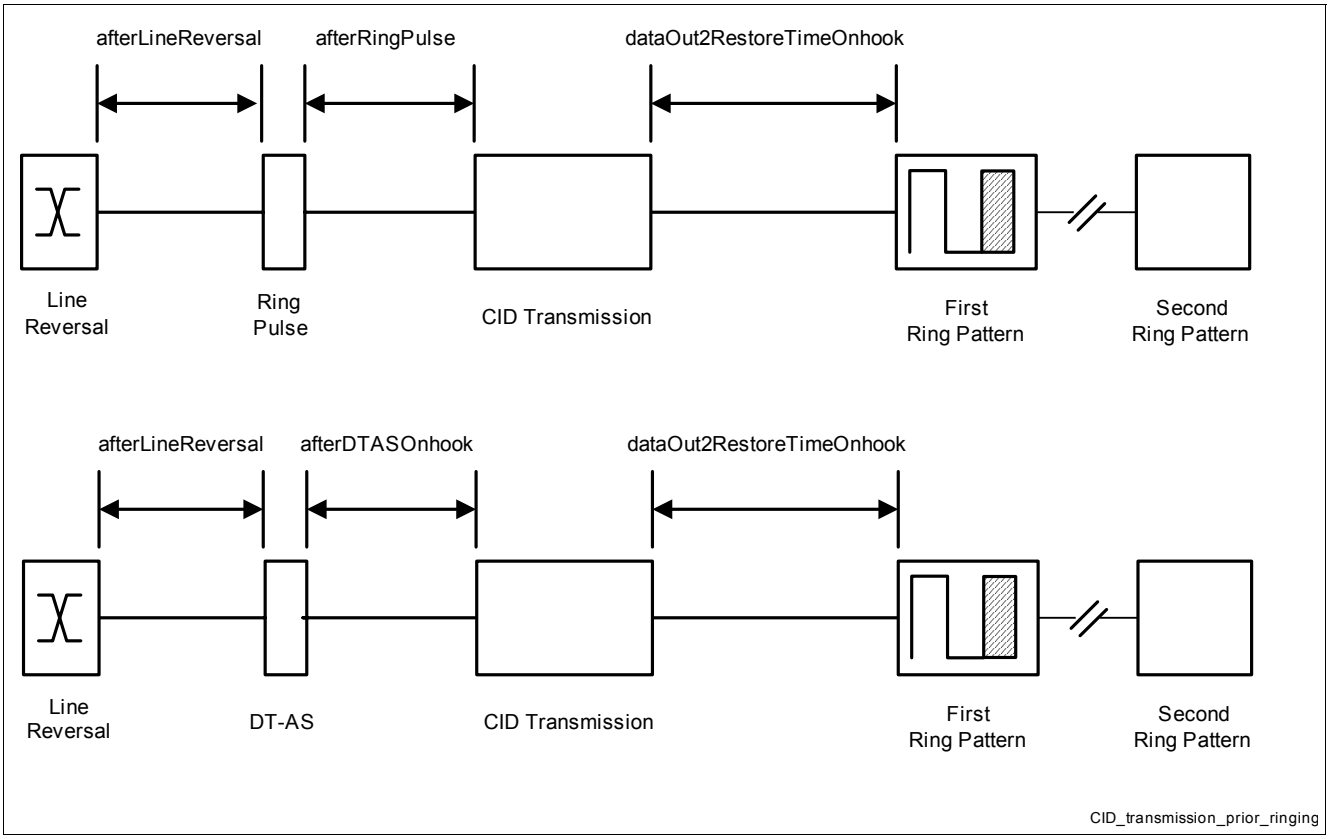

Figure 18 Timing for some Possible ETSI CID Type 1 with Transmission Prior to Ringing

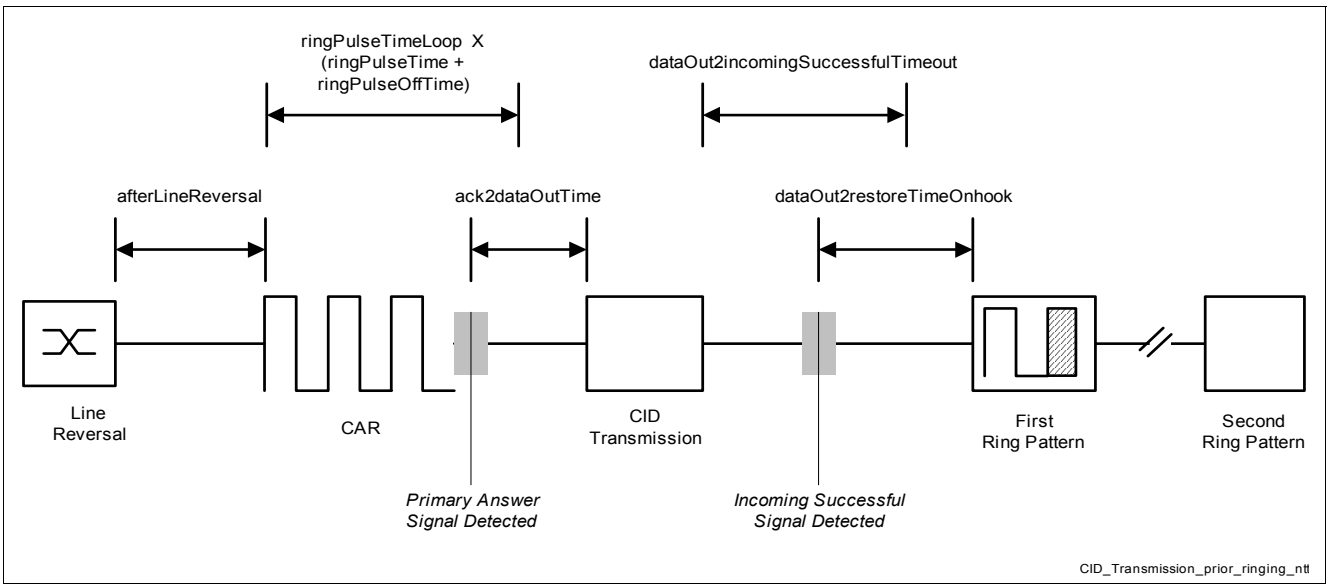

Figure 19 Timing for NTT CID Type 1 with Transmission Prior to Ringing

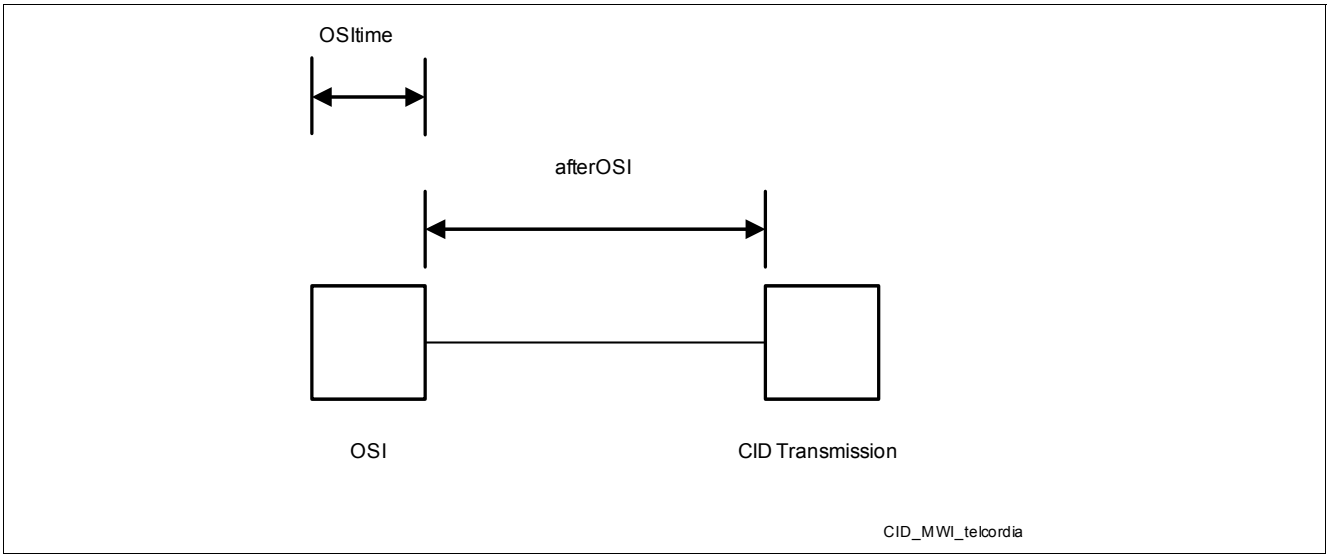

Figure 20 Timing for Telcordia MWI

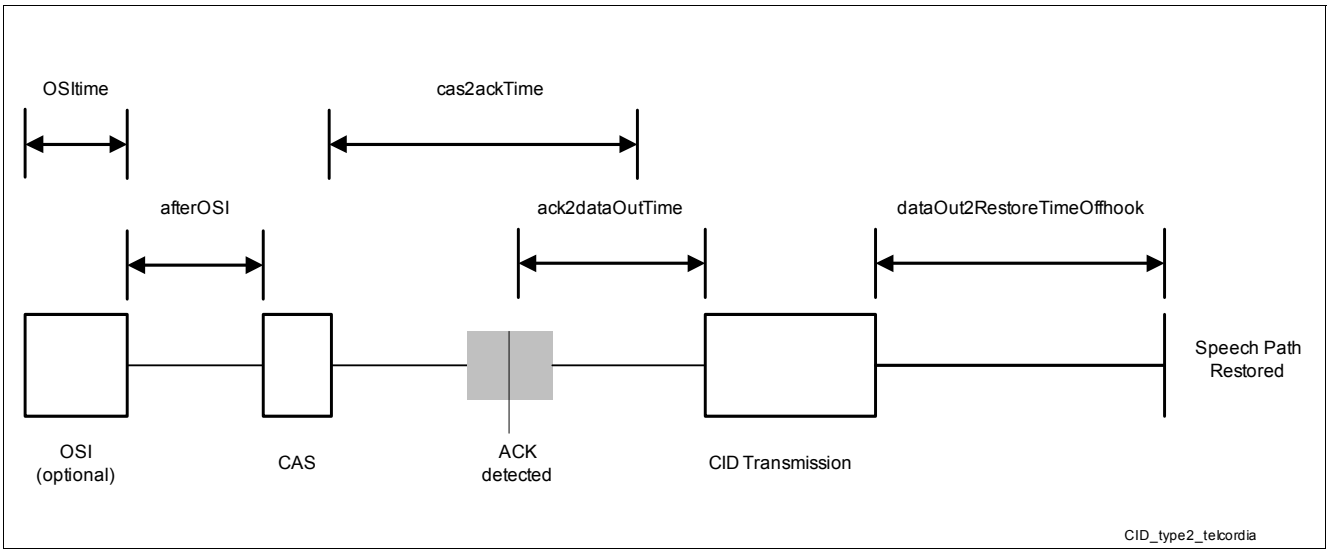

Figure 21 Timing for Telcordia CID Type 2

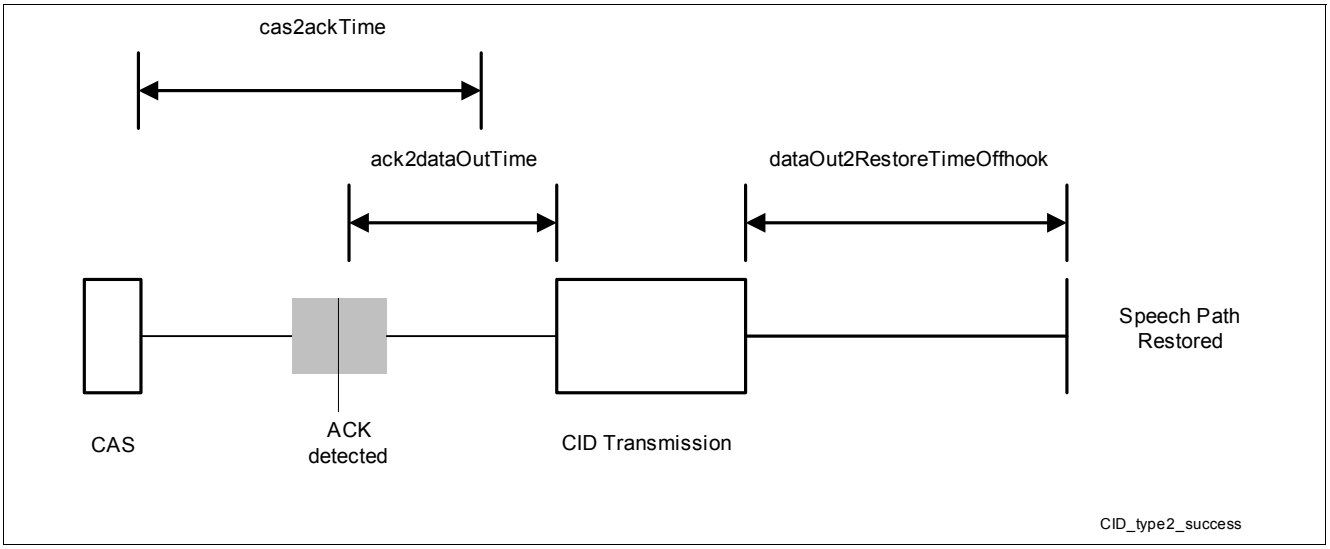

Figure 22 Timing for ETSI CID Type 2, Successful Transmission

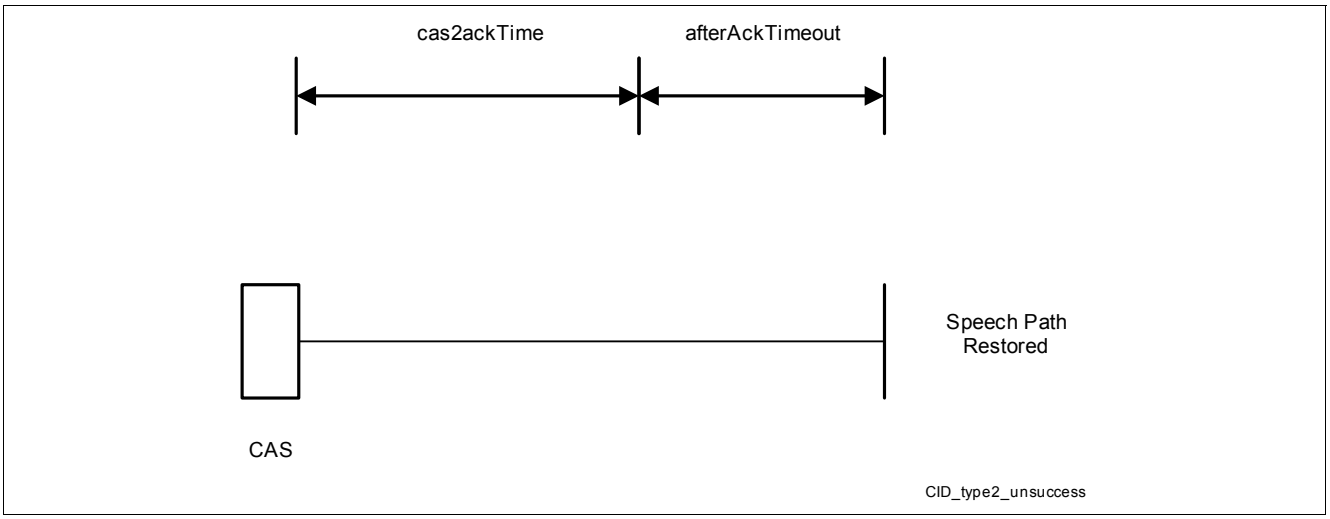

Figure 23 Timing for ETSI CID Type 2, Unsuccessful Transmission

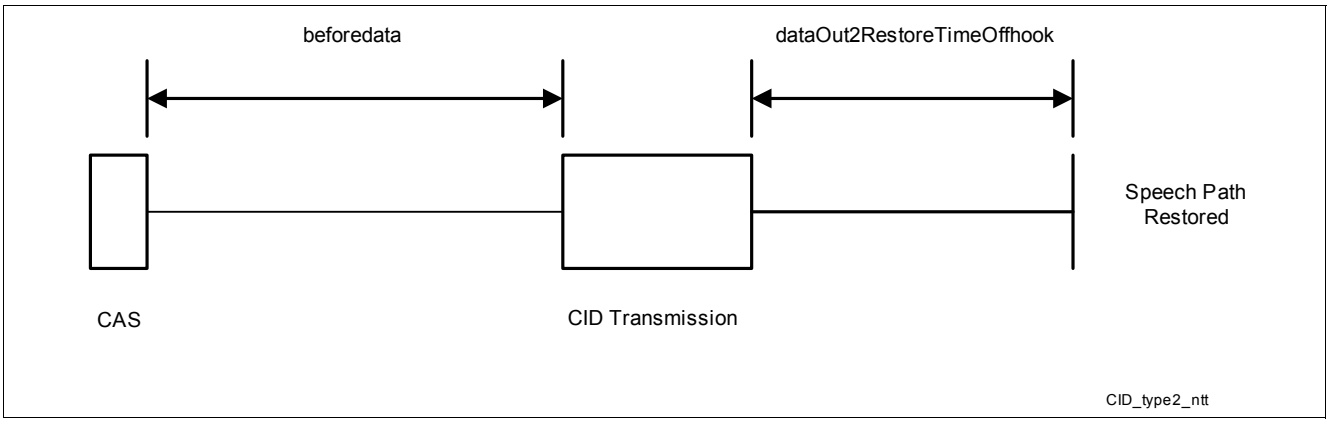

Figure 24 Timing for NTT CID Type 2

Example - CID Configuration using Default Values

```
/* Configure Caller ID support, selecting only used standard */
/* The driver uses default parameters */
/* For example use caller ID settings for UK SIN227 standard */
IFX_TAPI_CID_CFG_t cidConf;

memset(&cidConf, 0, sizeof(cidConf));

/* Standard selection is done in the following line */
cidConf.nStandard = IFX_TAPI_CID_STD_SIN;

ioctl(fd, IFX_TAPI_CID_CFG_SET, &cidConf);
/* From now on the SIN227 default configuration is used */
```

Example - CID Configuration Modifying some Default Values

```
/* Configuration of Caller ID, modifying several parameters */
/* Use Telcordia/Bellcore CID standard for USA */
/* Variable fskConf contains FSK transmission parameters */
IFX_TAPI_CID_FSK_CFG_t fskConf;
/* Variable timingConf contains timing parameters */
IFX_TAPI_CID_TIMING_t timingConf;
/* Variable sinConf contains SIN227 specific parameters */
/* and is used to link fskConf and timingConf */
IFX_TAPI_CID_STD_TELCORDIA_t telcordiaConf;
/* cidConf to select CID standard and contains telcordiaConf*/
IFX_TAPI_CID_CFG_t cidConf;

memset(&fskConf, 0, sizeof(fskConf));
memset(&timingConf, 0, sizeof(timingConf));
memset(&telcordiaConf, 0, sizeof(telcordiaConf));
memset(&cidConf, 0, sizeof(cidConf));
/* Choose Telcordia standard */
cidConf.nStandard = IFX_TAPI_CID_STD_TELCORDIA;

/* Modify some FSK parameters */
/* for example FSK TX level to -10 dB and minimum FSK RX level to -20 dB */
fskConf.levelTX = -10;
fskConf.levelRX = -20;

/* To see all FSK parameters please refer to IFX_TAPI_CID_FSK_CFG_t */
/* For all remaining parameters default values will be used */
/* Modify timing */
/* for example modify delay 1st ring to FSK and timeout for CID type 2 ack */
timingConf.afterFirstRing = 400;
timingConf.cas2ackTime = 200;

/* To see all timing parameters please refer to IFX_TAPI_CID_TIMING_t */
/* For all remaining parameters default values will be used */
/* Modify some other CID parameters, relevant for Telcordia standard */
/* for example reconfigure offhook DTAS and OSI length to 150 ms */
/* New DTAS tone already added to the tone table at position DTAS_OFFHOOK_INDEX */
```

```
telcordiaConf.nAlertToneOffhook = DTAS_OFFHOOK_INDEX;
telcordiaConf.OSItime = 150;

/* For all Telcordia parameters please refer to IFX_TAPI_CID_STD_TELCORDIA */
/* For all remaining parameters default values will be used */
/* Now it is important to link all structures together */
telcordiaConf.pCIDTiming = &timingConf;
telcordiaConf.pFSKConf = &fskConf;

cidConf.cfg = (IFX_TAPI_CID_STD_TYPE_t*) &telcordiaConf;
ioctl(fd, IFX_TAPI_CID_CFG_SET, &cidConf);
```

3.6.3 Ringing Support

Ringing is to be handled in two steps: first program the ring cadence pattern and ringing type, then start periodical ringing (using the preprogrammed cadence pattern).

3.6.3.1 Configure Ringing Type

Service **IFX_TAPI_RING_CFG_SET** shall be used to choose ringing mode for example internal or external (listed ringing mode listed in **IFX_TAPI_RING_CFG_MODE_t**) and the ring trip type (listed ringing mode listed in **IFX_TAPI_RING_CFG_SUBMODE_t**).

Example - Ringing Type Configuration

```
 /* Configure Ringing Type */
IFX_TAPI_RING_CFG_t ringingType;
 memset (&ringingType, 0,  sizeof (IFX_TAPI_RING_CFG_t));
 ringingType.mode = IFX_TAPI_RING_CFG_MODE_INT_BALANCED;
 ringingType.submode = IFX_TAPI_RING_CFG_SUBMODE_DC_RNG_TRIP_FAST;
 ret = ioctl(fd, IFX_TAPI_RING_CFG_SET, (IFX_int32_t) &ringingType)
```

3.6.3.2 Configure Ring Cadence

The ringing cadence pattern can be programmed with relatively fine granularity (50 ms) with interface **IFX_TAPI_RING_CADENCE_HR_SET**. The cadence pattern is coded in a bit sequence: each 1 corresponds to a 50 ms ring burst and each 0 corresponds to a 50 ms ring pause. For programming convenience, the bit sequence is represented in an array of char. Each entry in the array (8 bits) corresponds to 400 ms, which means that an array entry 0xFF corresponds to a 400 ms ring burst and 0x00 corresponds to 400 ms ring pause. A ring cadence has to start with a ring burst, which means that at least the first bit of the coded cadence must be a 1.

The array is part of the **IFX_TAPI_RING_CADENCE_t** structure, also containing a field specifying the number of valid bits in the array.

Some examples of cadence patterns:

- 1-second ring burst and 1-second ring pause is represented as FF FF F0 00 00, with 40 valid bits (2000 ms / 50 ms = 40).
- 3-second ring burst and 1-second ring pause is represented as FF FF FF FF FF FF FF F0 00 00, with 80 valid bits (4000 ms / 50 ms = 80).
- 0.5-second ring burst and 0.4-second ring pause is represented as FF C0 00, with 18 valid bits (900 ms / 50 ms = 18).

3.6.3.3 Start / Stop Ringing

Two possibilities are given for starting ringing:

- In applications requiring a CID transmission associated with ringing, service **IFX_TAPI_CID_TX_SEQ_START** starts a sequence synchronizing ringing with CID transmission.
- For applications requiring ringing without CID, service **IFX_TAPI_RING_START**[1] starts ringing.

Ringing can be stopped using **IFX_TAPI_RING_STOP**. Ringing stops automatically upon off-hook detection.

Example - Ringing Support

```
/* Cadence pattern of 3 sec. ring and 1 sec. pause */
IFX_char_t data[10] = {0xFF, 0xFF, 0xFF, 0xFF, 0xFF,
                       0xFF, 0xFF, 0xF0, 0x00, 0x00};
IFX_TAPI_RING_CADENCE_t ringCadence;

memset(&ringCadence, 0, sizeof(IFX_TAPI_RING_CADENCE_t));
/* Program the cadence */
memcpy(&ringCadence.data, data, sizeof(data));
ringCadence.nr = sizeof(data) * 8;
ioctl(fd, IFX_TAPI_RING_CADENCE_HR_SET, &ringCadence);
/* Do ringing (NO CID will be sent with this Interface) */
ioctl(fd, IFX_TAPI_RING_START, 0);

/* .... do something.... */
sleep(2);

/* Stop ringing */
ioctl(fd, IFX_TAPI_RING_STOP, 0);
```

3.6.4 CID TX

Two possible ways of transmitting CID are supported:

- **IFX_TAPI_CID_TX_SEQ_START**, to issue execution of the CID transmission sequence according to the preconfigured CID standard.
- **IFX_TAPI_CID_TX_INFO_START**, to encode and transmit only a data link message. In this case, the application software implements CID signaling sequence: issue and synchronize alert signal/ACK, polarity reversal, ringing, etc.

Both interfaces receive as a parameter a pointer to **IFX_TAPI_CID_MSG_t**, containing the CID information to be transmitted and also some information on the CID transmission type (type 1/2 and MWI).

Chapter 3.6.4.1 describes how to prepare the CID message, and **Chapter 3.6.4.2** gives some examples of CID transmission.

3.6.4.1 Prepare CID Message

CID transmission type and message are set up in a **IFX_TAPI_CID_MSG_t** variable that contains the following fields:

- `txMode`, for example, on-hook or off-hook.
- `messageType`, for example, call set-up (for caller ID) or MWI.
- `message`, pointer to an array of CID message elements, each element representing a part of the CID message to be transmitted, such as calling/called number or name, date and time, ...
- `nMsgElements`, CID message element array size.

1) CID can not be issued with this interface; this was supported with driver up to release 1.0.x.

Fields `txMode` and `msgType` determine CID type; see also **Table 20** for configuration examples. Note that Visual Message Waiting Indication (VMWI) is an MWI on-hook with service type **IFX_TAPI_CID_ST_VISINDIC**.

Each entry in the array can be one of the message element types supported by union **IFX_TAPI_CID_MSG_ELEMENT_t**. The idea is that information to be transmitted can be represented as a string (for example, calling number and name), date/time or value.

In some scenarios, application software already has available a message coded in the correct data link format (for FSK already including message framing, parameter coding and CRC). This is supported as “transparent transmission” (see example following). In this case, the message element array shall consist of only one element, containing the entire CID message.

Examples of CID message preparation for CID type 1, type 2, VMWI and transparent transmission are given in **Chapter 3.6.4.2**.

Table 20 Caller ID/MWI Type Selection

CID/MWI Type	txMode	msgType
CID type 1	**IFX_TAPI_CID_HM_ONHOOK**	**IFX_TAPI_CID_MT_CSUP**
CID type 2	**IFX_TAPI_CID_HM_OFFHOOK**	**IFX_TAPI_CID_MT_CSUP**
MWI on-hook	**IFX_TAPI_CID_HM_ONHOOK**	**IFX_TAPI_CID_MT_MWI**
MWI off-hook	**IFX_TAPI_CID_HM_OFFHOOK**	**IFX_TAPI_CID_MT_MWI**

Transmission modes

- On-hook, **IFX_TAPI_CID_HM_ONHOOK**
- Off-hook, **IFX_TAPI_CID_HM_OFFHOOK**

Message types

- Call setup, **IFX_TAPI_CID_MT_CSUP**
- Message Waiting Indication, **IFX_TAPI_CID_MT_MWI**

Service types

- Date and Time, **IFX_TAPI_CID_ST_DATE**
- Calling Line Identity, **IFX_TAPI_CID_ST_CLI**
- Reason for absence of Calling line Identity, **IFX_TAPI_CID_ST_ABSCLI** (Unavailable and Private)
- Calling party name, **IFX_TAPI_CID_ST_NAME**
- Reason for absence of calling party name, **IFX_TAPI_CID_ST_ABSNAME** (Unavailable and Private)
- (only for MWI) Visual Indicator, **IFX_TAPI_CID_ST_VISINDIC** (Indicator off and Indicator on)
- Information to be sent with transparent transmission **IFX_TAPI_CID_ST_TRANSPARENT**

3.6.4.2 Examples of CID TX

This chapter gives some examples of CID TX

- **Example 1 - CID Configuration and Caller ID type 1**
- **Example 2 - Caller ID type 2**
- **Example 3 - Caller ID type 2 and calling number not available**
- **Example 4 - Visual Message Waiting Indication**
- **Example 5 - Transparent Transmission**

Example 1 - CID Configuration and Caller ID type 1

```
const IFX_char_t* PHONE_NUMBER = "123456789\0";
const IFX_char_t* PHONE_NAME = "test\0";
```

```
const IFX_int32_t MAX_MSG_LENGTH = 3;

/* Application decides to issue a CID Type 1 */
IFX_TAPI_CID_CFG_t cidType1;
IFX_TAPI_CID_MSG_ELEMENT_t message[MAX_MSG_LENGTH];
IFX_TAPI_CID_CFG_t cidConfiguration;

/* Fill cidConfiguration with information about standard */
/* For example, use Telcordia/Bellcore standard */
memset(&cidConfiguration, 0, sizeof (IFX_TAPI_CID_CFG_t));
cidConfiguration.nStandard = IFX_TAPI_CID_STD_TELCORDIA;

/* Default parameters apply to the CID configuration for */
/* Telcordia/Bellcore */
ioctl(fd, IFX_TAPI_CID_CFG_SET, &cidConfiguration);
/* Now TAPI is configured with a CID standard */
/* Normally execution of IFX_TAPI_CID_CFG_SET is required only */
/* one time after boot */

memset(&cidType1, 0, sizeof(cidType1));
memset(&message, 0, sizeof(message));

/* Caller ID type 1: On hook transmission & Call Set-Up service */
cidType1.txMode = IFX_TAPI_CID_HM_ONHOOK;
cidType1.messageType = IFX_TAPI_CID_MT_CSUP;
/* Three message elements to be send(calling number, date/time and name) */
cidType1.nMsgElements = MAX_MSG_LENGTH;
/* Prepare message to be displayed: calling number */
message[0].string.elementType = IFX_TAPI_CID_ST_CLI;
/* Calling number size */
message[0].string.len = strlen(PHONE_NUMBER);
/* Calling number formatted as string */
strncpy(message[0].string.element, &PHONE_NUMBER[0],
        sizeof(message[0].string.element));
/* Prepare message to be displayed: date & time */
message[1].date.elementType = IFX_TAPI_CID_ST_DATE;
/* Date and time 11.01.2006 16:10*/
message[1].date.day = 11;
message[1].date.month = 1;
message[1].date.hour = 16;
message[1].date.mn = 10;
/* setup name element */
message[2].string.elementType = IFX_TAPI_CID_ST_NAME;
message[2].string.len = strlen(PHONE_NAME);
strncpy(message[2].string.element, PHONE_NAME,
        sizeof(message[2].string.element));

/* Message is now prepared */
cidType1.message = message;
/* Start caller id sequence */
ioctl(fd, IFX_TAPI_CID_TX_SEQ_START, &cidType1);
```

Example 2 - Caller ID type 2

```
const IFX_char_t* PHONE_NUMBER = "123456789";
const IFX_int32_t MAX_MSG_LENGTH = 3;

/* Application decides to issue a CID Type 2 */
/* Sending only calling number */
IFX_TAPI_CID_MSG_t cidType2;
IFX_TAPI_CID_MSG_ELEMENT_t message[MAX_MSG_LENGTH];

memset(&cidType2, 0, sizeof(cidType2));
memset(&message, 0, sizeof(message));
/* Caller ID type 2: Off hook transmission & Call Set-Up service */
cidType2.txMode = IFX_TAPI_CID_HM_ONHOOK;
cidType2.messageType = IFX_TAPI_CID_MT_CSUP;

/* One message element to be transmitted (calling number) */
cidType2.nMsgElements = 1;
/* Prepare message to be displayed: calling number */
message[0].string.elementType = IFX_TAPI_CID_ST_CLI;
/* Calling number size */
message[0].string.len = strlen(PHONE_NUMBER);
/* Calling number formatted as string */
strncpy(message[0].string.element, PHONE_NUMBER, message[0].string.len);
/* Message is now prepared */
cidType2.message = message;
/* Start caller id sequence */
ioctl(fd, IFX_TAPI_CID_TX_SEQ_START, &cidType2);
```

Example 3 - Caller ID type 2 and calling number not available

```
/* Application decides to issue a CID Type 2 */
/* However calling number is not available */
IFX_TAPI_CID_MSG_t cidType2;
IFX_TAPI_CID_MSG_ELEMENT_t message[MAX_MSG_LENGTH];

memset ( &cidType2, 0, sizeof(cidType2) );
memset ( &message, 0, sizeof(message) );

/* Caller ID type 2: Off-hook transmission & Call Set-Up service */
cidType2.txMode = IFX_TAPI_CID_HM_OFFHOOK;
cidType2.messageType = IFX_TAPI_CID_MT_CSUP;
/* One message element to be transmitted (absence reason of calling number) */
cidType2.nMsgElements = 1;
/* Prepare message to be displayed: absence reason of calling number */
message[0].value.elementType = IFX_TAPI_CID_ST_ABSCLI;
/* Absence reason: number unavailable/unknown */
message[0].value.element = IFX_TAPI_CID_ABSREASON_UNAV;
/* Message is now prepared */
cidType2.pMessage = message;
/* Start caller id sequence */
ioctl ( fd, IFX_TAPI_CID_TX_SEQ_START, &cidType2 );
```

Example 4 - Visual Message Waiting Indication

```
const IFX_char_t* PHONE_NUMBER = "123456789";
const IFX_int32_t MAX_MSG_LENGTH = 3;

/* Application decides to issue a VMWI, to light the CPE LED */
/* Implemented only for on-hook */
IFX_TAPI_CID_MSG_t cidTypeVMWI;
IFX_TAPI_CID_MSG_ELEMENT_t message[MAX_MSG_LENGTH];

memset(&cidTypeVMWI, 0, sizeof(cidTypeVMWI));
memset(&message, 0, sizeof(message));

/* VMWI: On hook transmission & Message Waiting Indication service */
cidTypeVMWI.txMode = IFX_TAPI_CID_HM_ONHOOK;
cidTypeVMWI.msgType = IFX_TAPI_CID_MT_MWI;
/* One message element to be transmitted (enable VMWI) */
cidTypeVMWI.nMsgElements = 1;
/* Prepare message to be displayed: Visual Indication */
message[0].value.elementType = IFX_TAPI_CID_ST_VISINDIC;
/* VMWI Enable (to light the LED) */
message[0].value.element = IFX_TAPI_CID_VMWI_EN;

/* Message is now prepared */
cidTypeVMWI.message = message;

/* If the complete on-hook sequence is required (e.g. Telcordia standard) */
ioctl(fd, IFX_TAPI_CID_TX_SEQ_START, &cidTypeVMWI);
/* Otherwise, if only the FSK transmission is required */
ioctl(fd, IFX_TAPI_CID_TX_INFO_START, &cidTypeVMWI);
```

Example 5 - Transparent Transmission

```
/* Application decides to send a CID type 1, already coded in the right */
/* FSK format */
IFX_TAPI_CID_MSG_t cid_info;
IFX_TAPI_CID_MSG_ELEMENT_t message;

/* FSK string to present the number "08923403330" */
IFX_uint8_t data[16] = {0x80, 0x0D, 0x02, 0x0B, 0x30, 0x38, 0x39, 0x32,
                        0x33, 0x34, 0x30, 0x33, 0x33, 0x33, 0x30, 0x33};

memset(&message, 0, sizeof(message));
memset(&cid_info, 0, sizeof(cid_info));

/* setup message */
message.transparent.elementType = IFX_TAPI_CID_ST_TRANSPARENT;
message.transparent.len = sizeof(data);
message.transparent.data = data;
/* setup cid info */
cid_info.txMode = IFX_TAPI_CID_HM_ONHOOK;
cid_info.msgType = IFX_TAPI_CID_MT_CSUP;
cid_info.nMsgElements = 1;
```

```
cid_info.msg = &message;
/* Send sequence */
ioctl(fd, IFX_TAPI_CID_TX_SEQ_START, &cid_info);
```

3.6.5 CID RX - FSK Receiver

Using the CID RX interfaces it is possible to control a FSK detection capability provided by some Infineon devices:

- **IFX_TAPI_CID_RX_START**, to start the FSK detector. It is necessary to pass as parameter the hook mode, defined in enum **IFX_TAPI_CID_HOOK_MODE_t**.
  - **IFX_TAPI_CID_HM_ONHOOK** for on-hook reception (required, for example, for CID type 1)
  - **IFX_TAPI_CID_HM_OFFHOOK** for off-hook reception (required, for example, for CID type 2)
- **IFX_TAPI_CID_RX_STOP**, to stop the detector. It is recommended to stop the FSK detector as soon as the CID message has been detected.
- **IFX_TAPI_CID_RX_DATA_GET**, to read the detected message. To be noted that this interface provides the data link layer information of the message, with exception of mark and seizure bits.
- **IFX_TAPI_CID_RX_STATUS_GET**, to report receiver status (active/inactive), reception progress (ongoing, data ready) and errors during CID reception. Alternatively, the application software can wait for a CID RX event: **IFX_TAPI_EVENT_CID_RX_FSK_END**

Caller ID reception should start as soon as the application software gets an indication that a CID information is about to be transmitted on the line.

For example, in case of CID type 1 (Telcordia standard), the CID receiver must be enabled as soon as an incoming ringing signal is detected on the FXO port. In case of CID type 2, the start of a CID transmission is typically indicated by a **IFX_TAPI_EVENT_CID_RX_CAS** event.

Attention: For caller ID reception, it is necessary to distinguish between DTMF and FSK transmission: the interfaces documented in this chapter can be used only for FSK caller ID. In case of DTMF caller ID, the single digits must be collected using the event reporting interface.

Example - CID RX - FSK Receiver

```
IFX_TAPI_CID_HOOK_MODE_t cidHookMode;
IFX_TAPI_CID_RX_STATUS_t cidRxStatus;
IFX_TAPI_CID_RX_DATA_t cidRxData;
IFX_return_t ret;

memset(&cidRxStatus, 0, sizeof (IFX_TAPI_CID_RX_STATUS_t));
memset(&cidRxData, 0, sizeof (IFX_TAPI_CID_RX_DATA_t));

/* Start CID RX Engine, on-hook detection*/
cidHookMode = IFX_TAPI_CID_HM_ONHOOK;
ret = ioctl(fd, IFX_TAPI_CID_RX_START, (IFX_int32_t) cidHookMode);

/* Check if CID information is available for reading */
ret = ioctl(fd, IFX_TAPI_CID_RX_STATUS_GET, (IFX_int32_t) &cidRxStatus);

/* No error during CID RX operations: cidRxStatus.nError == 0 */
/* CID information received: cidRxStatus.nStatus == 0x3 */
if ((ret == IFX_SUCCESS) && (cidRxStatus.nError == 0)
   && (cidRxStatus.nStatus == 0x3))
{
   /* CID information available, now read it */
   ret = ioctl(fd, IFX_TAPI_CID_RX_DATA_GET, (IFX_int32_t) &cidRxData);
```

```
    /* CID RX Engine can be stopped */
    ret = ioctl(fd, IFX_TAPI_CID_RX_STOP, 0);
} /* if */
```

3.7 Fax/Modem Support

This chapter describes TAPI support for fax/modem transmission

- Detection of Fax/Modem signals; see **Chapter 3.7.1** for details.
- Pass-through mode (also called transparent mode) is supported for both fax and modem; see **Chapter 3.7.2** for details.
- Fax relay mode: control T.38 data pump[1]; see **Chapter 3.7.3** for details.

3.7.1 Detect Fax/Modem Signals

It is possible to enable/disable detection of fax/modem signals using the event reporting interfaces **IFX_TAPI_EVENT_ENABLE** and **IFX_TAPI_EVENT_DISABLE**.

The follow **IFX_TAPI_EVENT_t** fields must be programmed

- id contains one of the fax/modem signals defined in **IFX_TAPI_EVENT_ID_t**.
- ch contains the channel where the signal detection must be enabled/disabled
- data must be used to enable (**IFX_ENABLE**) or disable (**IFX_ENABLE**) detection on a certain direction.
  - data.network for signals to be detected from network, it means RTP inband.
  - data.local for signals to be detected from an analog or PCM port.

Upon signal detection an interrupt is generated and the application software gets the information from **IFX_TAPI_EVENT_GET**.

It is strongly recommended to use the event reporting interfaces, nevertheless backwards compatibility[2] is given for the old interfaces

- **IFX_TAPI_SIG_DETECT_ENABLE**, **IFX_TAPI_SIG_DETECT_DISABLE**
- **IFX_TAPI_CH_STATUS_GET**

Example - Detection of Fax/Modem Signals

```
/* Enable reporting of Fax/Modem events on data channel 1 */
/* Detect DIS from local port and CED from local and network port */

IFX_TAPI_EVENT_t tapiEventDIS;
IFX_TAPI_EVENT_t tapiEventCED;
IFX_TAPI_EVENT_t tapiEventHold;
memset(&tapiEventDIS, 0, sizeof (IFX_TAPI_EVENT_t));
memset(&tapiEventCED, 0, sizeof (IFX_TAPI_EVENT_t));
memset(&tapiEventHold, 0, sizeof (IFX_TAPI_EVENT_t));

/* Detection of DIS */
tapiEventDIS.id = IFX_TAPI_EVENT_FAXMODEM_DIS;
tapiEventDIS.data.fax_sig.local = IFX_ENABLE;
tapiEventDIS.data.fax_sig.network = IFX_DISABLE;
tapiEventDIS.ch = 1;

/* Detection of CED from local and network port */
```

1) The T.38 data pump runs in VINETIC® firmware and complements the T.38 protocol implementation running on top of TAPI.
2) The old interfaces will be discontinued in a next TAPI release.

```
tapiEventCED.id = IFX_TAPI_EVENT_FAXMODEM_CED;
tapiEventCED.data.fax_sig.local = IFX_ENABLE;
tapiEventCED.data.fax_sig.network = IFX_ENABLE;
tapiEventCED.ch = 1;

/* Detection of holding characteristics, direction does not matter */
/* however the direction must be programmed in order to select the detector */
tapiEventHold.id = IFX_TAPI_EVENT_FAXMODEM_HOLD;
tapiEventCED.data.fax_sig.local = IFX_DISABLE;
tapiEventHold.data.fax_sig.network = IFX_ENABLE;
tapiEventHold.ch = 1;

/* Now enable detection of DIS */
ioctl(fd, IFX_TAPI_EVENT_ENABLE, (IFX_int32_t) &tapiEventDIS);

/* After a while the DIS event has been reported */
/* Now disable detection of DIS to lower (for example) power consumption */
ioctl(fd, IFX_TAPI_EVENT_DISABLE, (IFX_int32_t) &tapiEventDIS);

/* Now application software wants to enable CED detection */
ioctl(fd, IFX_TAPI_EVENT_ENABLE, (IFX_int32_t) &tapiEventCED);

/* After a while the CED event has been reported */
/* Now disable detection of CED to lower (for example) power consumption*/
ioctl(fd, IFX_TAPI_EVENT_DISABLE, (IFX_int32_t) &tapiEventCED);

/* Now application software wants to enable holding characteristic */
ioctl(fd, IFX_TAPI_EVENT_ENABLE, (IFX_int32_t) &tapiEventHold);

/* After a while the holding event has been reported */
/* Now disable detection of holding to lower (for example) power consumption*/
ioctl(fd, IFX_TAPI_EVENT_DISABLE, (IFX_int32_t) &tapiEventHold);
```

3.7.2 Pass-Through Mode

The pass-through mode is required to facilitate fax/modem communication using a VoIP link actually set up for voice traffic.

Switching to pass-through mode should be triggered by the detection of a fax/modem signal, either from the local subscriber or from the network (in-band or out-of-band).

TAPI support for pass-through mode is given through interfaces

- **IFX_TAPI_JB_CFG_SET** to configure JB as fixed and in data mode.
- **IFX_TAPI_LEC_PHONE_CFG_SET** (or **IFX_TAPI_LEC_PCM_CFG_SET**) to turn LEC and NLP off.
- **IFX_TAPI_ENC_TYPE_SET** to configure vocoder to G.711 A-law or µ-law; this parameter must be first negotiated with the remote VoIP party.

In some application scenarios, a voice call follows the fax/modem call without first going on-hook: as soon as the end of a fax/modem call is detected, the application software should restore LEC and JB configuration to the values preceding the switch to pass-through mode.

Example - Pass-Through Mode

```
IFX_TAPI_JB_CFG_t jbCfgVoice, jbCfgData;
```

```
IFX_TAPI_WLEC_CFG_t lecConf;
IFX_TAPI_ENC_TYPE_t encTypeVoice, encTypeData;

memset(&jbCfgVoice, 0, sizeof (IFX_TAPI_JB_CFG_t));
memset(&jbCfgData, 0, sizeof (IFX_TAPI_JB_CFG_t));
memset(&encTypeVoice, 0, sizeof (IFX_TAPI_ENC_TYPE_t));
memset(&encTypeData, 0, sizeof (IFX_TAPI_ENC_TYPE_t));

/* .... */
/* jbCfgVoice populated and used to configure JB */
/* lecConf populated and used to configure LEC */
/* .... */
/* During operations fax/modem signals have been detected */
/* It is necessary to switch to fax/modem pass-through mode */
/* VoIP signaling negotiated G.711 ALaw vocoder */
encTypeData = IFX_TAPI_ENC_TYPE_MLAW;
ioctl(fd, IFX_TAPI_ENC_TYPE_SET, (IFX_int32_t) encTypeData);

/* Reconfigure JB for fax/modem communications */
jbCfgData.nJbType = IFX_TAPI_JB_TYPE_FIXED;
jbCfgData.nPckAdpt = IFX_TAPI_JB_PKT_ADAPT_DATA;
/* The JB size are strictly application dependent */

/* Initial JB size 90 ms = 0x2D0 * 125 µs */
jbCfgVoice.nInitialSize = 0x02D0;
/* Minimum JB size 10 ms = 0x50 * 125 µs */
jbCfgVoice.nMinSize = 0x50;
/* Maximum JB size 180 ms = 0x5A0 * 125 µs */
jbCfgVoice.nMaxSize = 0x5A0;

ioctl(fd, IFX_TAPI_JB_CFG_SET, (IFX_int32_t) &jbCfgData);

/* Disable LEC */
lecConf.nType = IFX_TAPI_WLEC_TYPE_OFF;
ioctl(fd, IFX_TAPI_WLEC_PHONE_CFG_SET, (IFX_int32_t) &lecConf);
```

3.7.3 T.38 Data-Pump Mode

TAPI includes interfaces for controlling the T.38 data pump[1)] available in Infineon products.

After device initialization, a channel is normally ready for supporting packetized voice. Command **IFX_TAPI_T38_MOD_START** and **IFX_TAPI_T38_DEMOD_START** start the T.38 data pump modulator and demodulator operations. The specified data channel will switch from voice packetization to T.38 data-pump mode, configured using the given parameters (primary and alternative standard, training sequence, levels, etc.).

TAPI handles T.38 data-pump frames with the same interfaces defined for packetized voice: the read interface to get modulated data-pump frames from the device, and write for sending fax frames to the demodulator.

1)Not to be confused with T.38 frames generated by application software: T.38 data-pump frames are used by the T.38 protocol implementation in software to generate the well-known T.38 TCP/UDP frames. Infineon T.38 stack implementation is described in [1] and [2].

A call to **IFX_TAPI_T38_STOP** switches the device back to voice-processing mode, and the read and write interface will serve voice data (RTP or AAL packets). All T.38 interfaces apply to data channels except otherwise stated.

Interface **IFX_TAPI_T38_STATUS_GET** is used to read the functional and/or error status of the T.38 data pump.

Attention: Before starting usage of T.38 services, it must be ensured that encoding, decoding and LEC are disabled (see Chapter 3.2 and Chapter 3.1.4)). Deactivation of fax/modem signal detectors (see Chapter 3.7.1) is also recommended; optionally a tone- holding detector might be enabled to detect the end of fax transmission.

4 TAPI Interfaces

This section describes all TAPI interfaces.

- **Chapter 4.1** provides a reference for the ioctl services, grouped by functionality.
- **Chapter 4.2** provides the reference for all types defined by TAPI, including
  - **Chapter 4.2.1**, basic type definition
  - **Chapter 4.2.2**, summary of all ioctl
  - **Chapter 4.2.3**, constant reference
  - **Chapter 4.2.4**, struct reference
  - **Chapter 4.2.5**, union reference
  - **Chapter 4.2.6**, enum reference

4.1 ioctl Commands of TAPI Interfaces

This chapter describes the entire interfaces to the TAPI. The ioctl commands are explained by mentioning the return values for each function. The organization is as follows:

Table 21 TAPI Interface Overview

Name	Description
CID Features Service	CID Features
Connection Control Service	Connection Control Service
Dial Service	Dial Service
Fax T.38 Service	Fax T.38 Service
Initialization Service	Initialization Service
Metering Service	Metering Service
Miscellaneous Services	Miscellaneous Services
Event Reporting Services	Event Reporting Services
Operation Control Service	Operation Control Service
PCM Service	PCM Service
Power Ringing Service	Power Ringing Service
Signal Detection Service	Signal Detection Service
Test Services	Test Service
Tone Control Service	Tone Control Service
Audio Channel Control	Audio Channel Service

4.1.1 CID Features Service

Caller ID support.

Table 22 IO-control Overview of CID Features

Name	Description
IFX_TAPI_CID_CFG_SET	Configures the CID transmitter.
IFX_TAPI_CID_RX_DATA_GET	Reads CID Data collected since Caller ID receiver was started.
IFX_TAPI_CID_RX_START	Setup the CID Receiver to start receiving CID Data.
IFX_TAPI_CID_RX_STATUS_GET	Retrieves the current status information of the CID Receiver.
IFX_TAPI_CID_RX_STOP	Stop the CID Receiver.
IFX_TAPI_CID_TX_INFO_START	This interfaces transmits only CID message.
IFX_TAPI_CID_TX_SEQ_START	This interfaces handles the entire CID sequence generation.

Table 23 Structure Overview of CID Features

Name	Description
IFX_TAPI_CID_ABS_REASON_t	ABSCLI/ABSNAME settings.
IFX_TAPI_CID_CFG_t	Structure containing CID configuration possibilities.
IFX_TAPI_CID_DTMF_CFG_t	Structure containing the configuration information for DTMF CID.
IFX_TAPI_CID_FSK_CFG_t	Structure containing the configuration information for FSK transmitter and receiver.
IFX_TAPI_CID_MSG_DATE_t	
IFX_TAPI_CID_MSG_STRING_t	Structure for element types (IFX_TAPI_CID_SERVICE_TYPE_t) with dynamic length (line numbers or names).
IFX_TAPI_CID_MSG_t	Structure containing the CID message type and content.
IFX_TAPI_CID_MSG_VALUE_t	Structure for element types (IFX_TAPI_CID_SERVICE_TYPE_t) with one value (length 1).
IFX_TAPI_CID_STD_ETSI_DTMF_t	Structure containing CID configuration for ETSI standard using DTMF transmission.
IFX_TAPI_CID_STD_ETSI_FSK_t	Structure containing CID configuration for ETSI standard using FSK transmission.
IFX_TAPI_CID_STD_NTT_t	Structure containing CID configuration for NTT standard.
IFX_TAPI_CID_STD_SIN_t	Structure containing CID configuration for BT SIN227 standard.
IFX_TAPI_CID_STD_TELCORDIA_t	Structure containing CID configuration for Telcordia standard.
IFX_TAPI_CID_TIMING_t	Structure containing the timing for CID transmission.

Table 24 Union Overview of CID Features

Name	Description
IFX_TAPI_CID_MSG_ELEMENT_t	CID Message Element
IFX_TAPI_CID_STD_TYPE_t	CID Standard Type

Table 25 Enumerator Overview of CID Features

Name	Description
IFX_TAPI_CID_ABSREASON_t	List of ABSCLI/ABSNAME settings.
IFX_TAPI_CID_ALERT_ETSI_t	List of ETSI Alerts.
IFX_TAPI_CID_HOOK_MODE_t	Caller ID transmission modes.
IFX_TAPI_CID_MSG_TYPE_t	Caller ID message types (defined in ETSI EN 300 659-3).
IFX_TAPI_CID_RX_ERROR_t	CID Receiver Errors.
IFX_TAPI_CID_RX_STATE_t	CID Receiver Status.
IFX_TAPI_CID_SERVICE_TYPE_t	Caller ID Services (defined in ETSI EN 300 659-3).
IFX_TAPI_CID_STD_t	List of CID standards.
IFX_TAPI_CID_VMWI_t	List of VMWI settings.

4.1.1.1 IFX_TAPI_CID_CFG_SET

Description

Configures the CID transmitter.

Prototype

```
IFX_int32_t ioctl (
        IFX_int32_t fd,
        IFX_TAPI_CID_CFG_SET,
        IFX_int32_t param );
```

Parameters

Data Type	Name	Description
IFX_int32_t	fd	File descriptor. It is applicable to data channel file descriptors.
IFX_int32_t	IFX_TAPI_CID_CFG_SET	I/O control identifier for this operation
IFX_int32_t	param	The parameter points to a IFX_TAPI_CID_CFG_t structure

Return Values

Data Type	Description
IFX_int32_t	The return value can be either of the following: • IFX_SUCCESS 0 • IFX_ERROR -1

Remarks

The delay must be programmed in that way, that the CID data still fits between the ring burst in case of appearance mode 1 and 2, otherwise the ioctl IFX_TAPI_RING_START may return an error. CID is stopped when the ringing is stopped or the phone goes off-hook.

Example

```
IFX_TAPI_CID_CFG_t param;
IFX_int32_t fd;

memset(&param, 0, sizeof(IFX_TAPI_CID_CFG_t));
/* Set CID standard to Telcordia/Bellcore default values */
param.nStandard = IFX_TAPI_CID_STD_TELCORDIA;
ioctl(fd, IFX_TAPI_CID_CFG_SET, &param);
```

4.1.1.2 IFX_TAPI_CID_RX_DATA_GET

Description

Reads CID data collected since caller Id receiver was started.

Prototype

```
IFX_int32_t ioctl (
        IFX_int32_t fd,
        IFX_TAPI_CID_RX_DATA_GET,
        IFX_int32_t param );
```

Parameters

Data Type	Name	Description
IFX_int32_t	fd	File descriptor. It is applicable to data channel file descriptors.
IFX_int32_t	IFX_TAPI_CID_RX_DATA_GET	I/O control identifier for this operation
IFX_int32_t	param	The parameter points to a IFX_TAPI_CID_RX_DATA_t structure.

Return Values

Data Type	Description
IFX_int32_t	The return value can be either of the following: • IFX_SUCCESS 0 • IFX_ERROR -1

Remarks

This request can be sent after the CID Receiver status signaled that data is ready for reading or that data collection is ongoing. In the last case, the number of data read correspond to the number of data received since CID receiver was started. Once the data is read after the status signaled that data is ready, new data can be read only if CID Receiver detects new data.

Example

```
IFX_TAPI_CID_RX_STATUS_t cidStatus;
IFX_TAPI_CID_RX_DATA_t cidData;
IFX_int32_t fd;
```

```
IFX_return_t ret;


/* Open channel file descriptor for channel 0 */
fd = open("/dev/vin11", O_RDWR, 0x644);

memset(&cidStatus, 0, sizeof(IFX_TAPI_CID_RX_STATUS_t));
memset(&cidData, 0, sizeof(IFX_TAPI_CID_RX_DATA_t));
/* Read actual status */
ret = ioctl(fd, IFX_TAPI_CID_RX_STATUS_GET, ( IFX_int32_t )&cidStatus);
/* Check if cid data are available for reading */
if ((IFX_SUCCESS == ret)
    && (IFX_TAPI_CID_RX_ERROR_NONE == cidStatus.nError)
    && (IFX_TAPI_CID_RX_STATE_DATA_READY == cidStatus.nStatus))
{
   ret = ioctl(fd, IFX_TAPI_CID_RX_DATA_GET, (IFX_int32_t) &cidData);
}

/* Close all open fds */
close(fd);
```

4.1.1.3 IFX_TAPI_CID_RX_START

Description

Setup the CID Receiver to start receiving CID Data.

Prototype

```
IFX_int32_t ioctl (
       IFX_int32_t fd,
       IFX_TAPI_CID_RX_START,
       IFX_int32_t param );
```

Parameters

Data Type	Name	Description
IFX_int32_t	fd	File descriptor. It is applicable to data channel file descriptors.
IFX_int32_t	IFX_TAPI_CID_RX_START	I/O control identifier for this operation
IFX_int32_t	param	Hook mode for the reception to be choosen from IFX_TAPI_CID_HOOK_MODE_t.

Return Values

Data Type	Description
IFX_int32_t	The return value can be either of the following: • IFX_SUCCESS 0 • IFX_ERROR -1

Remarks

This command must be sent so that the driver can start collecting CID Data.

Example

```
ioctl(fd, IFX_TAPI_CID_RX_START, (IFX_int32_t) IFX_TAPI_CID_HM_ONHOOK);
```

4.1.1.4 IFX_TAPI_CID_RX_STATUS_GET

Description

Retrieves the current status information of the CID Receiver.

Prototype

```
IFX_int32_t ioctl (
        IFX_int32_t fd,
        IFX_TAPI_CID_RX_STATUS_GET,
        IFX_int32_t param );
```

Parameters

Data Type	Name	Description
IFX_int32_t	fd	File descriptor. It is applicable to data channel file descriptors.
IFX_int32_t	IFX_TAPI_CID_RX_STATUS_GET	I/O control identifier for this operation
IFX_int32_t	param	The parameter points to a IFX_TAPI_CID_RX_STATUS_t structure.

Return Values

Data Type	Description
IFX_int32_t	The return value can be either of the following: • IFX_SUCCESS 0 • IFX_ERROR -1

Remarks

Once the CID receiver is activated it signals the status with an exception. The exception is raised if data has been received completely or an error occurred. Afterwards this interface is used to determine if data has been received. In that case it can be read with the interface IFX_TAPI_CID_RX_DATA_GET. This interface can also be used without exception to determine the status of the receiver while reception.

Example

```
IFX_TAPI_CID_RX_STATUS_t cidStatus;
IFX_int32_t fd;
IFX_return_t ret;

memset(&cidStatus, 0, sizeof (IFX_TAPI_CID_RX_STATUS_t));
```

```
ret = ioctl(fd, IFX_TAPI_CID_RX_STATUS_GET, (IFX_int32_t) &cidStatus);
/* Check if cid information are available for reading */
if ((IFX_SUCCESS == ret)
    && (IFX_TAPI_CID_RX_ERROR_NONE == cidStatus.nError)
    && (IFX_TAPI_CID_RX_STATE_DATA_READY == cidStatus.nStatus))
{
   printf ( "Data are ready for reading\n\r" );
}
```

4.1.1.5 IFX_TAPI_CID_RX_STOP

Description

Stop the CID Receiver.

Prototype

```
IFX_int32_t ioctl (
       IFX_int32_t fd,
       IFX_TAPI_CID_RX_STOP,
       IFX_int32_t param );
```

Parameters

Data Type	Name	Description
IFX_int32_t	fd	File descriptor. It is applicable to data channel file descriptors.
IFX_int32_t	IFX_TAPI_CID_RX_STOP	I/O control identifier for this operation
IFX_int32_t	param	This interface expects no parameter. It should be set to 0.

Return Values

Data Type	Description
IFX_int32_t	The return value can be either of the following: • IFX_SUCCESS 0 • IFX_ERROR -1

Example

```
ioctl(fd, IFX_TAPI_CID_RX_STOP, 0);
```

4.1.1.6 IFX_TAPI_CID_TX_INFO_START

Description

This interfaces transmits CID message.

Prototype

```
IFX_int32_t ioctl (
```

```
IFX_int32_t fd,
IFX_TAPI_CID_TX_INFO_START,
IFX_int32_t param );
```

Parameters

Data Type	Name	Description
IFX_int32_t	fd	File descriptor. It is applicable to data channel file descriptors.
IFX_int32_t	IFX_TAPI_CID_TX_INFO_START	I/O control identifier for this operation
IFX_int32_t	param	Pointer to a IFX_TAPI_CID_MSG_t, containing the CID / MWI information to be transmitted.

Return Values

Data Type	Description
IFX_int32_t	The return value can be either of the following: • IFX_SUCCESS 0 • IFX_ERROR -1

Remarks

Before issuing this service, CID engine must be configured with IFX_TAPI_CID_CFG_SET at least one time after boot. This is required to configure country specific settings.

Example

```
IFX_TAPI_CID_MSG_t Cid_Info;
IFX_TAPI_CID_MSG_ELEMENT_t Msg_El[2];
IFX_int32_t fd;
IFX_return_t ret;
IFX_char_t* number = "12345";


/* Open channel file descriptor for channel 0 */
fd = open("/dev/vin11", O_RDWR, 0x644);

/* Reset and fill the caller id structure */
memset(&Cid_Info, 0, sizeof(Cid_Info));
memset(&Msg_El, 0, sizeof(Msg_El));

Cid_Info.txMode = IFX_TAPI_CID_HM_ONHOOK;
/* Message Waiting */
Cid_Info.messageType = IFX_TAPI_CID_MT_MWI;
Cid_Info.nMsgElements = 2;
Cid_Info.message = Msg_El;
/* Mandatory for Message Waiting: set Visual Indicator on */
Msg_El[0].value.elementType = IFX_TAPI_CID_ST_VISINDIC;
Msg_El[0].value.element = IFX_TAPI_CID_VMWI_EN;
/* Add optional CLI (number) element */
```

```
Msg_El[1].string.elementType = IFX_TAPI_CID_ST_CLI;
Msg_El[1].string.len = ret;
strncpy(Msg_El[1].string.element, number, sizeof(Msg_El[1].string.element));
/* Transmit the caller id */
ioctl(fd, IFX_TAPI_CID_TX_INFO_START, &Cid_Info);

/* close all open fds */
close(fd);
```

4.1.1.7 IFX_TAPI_CID_TX_SEQ_START

This service starts a pre-programmed CID sequence driven by TAPI. This is a non-blocking service, the driver will signal the end of the CID sequence.

Before issuing this service, CID engine must be configured with **IFX_TAPI_CID_CFG_SET** at least one time after boot. This is required to configure country specific settings.

Prototype

```
IFX_void_t ioctl (
        IFX_int32_t fd,
        IFX_TAPI_CID_TX_SEQ_START,
        IFX_int32_t pCIDInfo );
```

Parameters

Data Type	Name	Description
IFX_int32_t	fd	File Descriptor. It is applicable to data channel file descriptors.
IFX_int32_t	IFX_TAPI_CID_TX_SEQ_START	I/O control identifier for this operation
IFX_int32_t	pCIDInfo	Pointer to IFX_TAPI_CID_MSG_t, defining the CID/MWI type and containing the information to be transmitted.

Return Values

Data Type	Description
IFX_void_t	No return value

Remarks

For FSK transmission, decision of seizure and mark length is based on configured standard and CID transmission type.

4.1.2 Connection Control Service

Contains all services used for RTP or AAL connection. It also contains services for conferencing.

Table 26 IO-control Overview of Connection Control Service

Name	Description
IFX_TAPI_DEC_LEVEL_GET	This service returns the level of the most recently played signal.
IFX_TAPI_DEC_START	Start the playout of data.
IFX_TAPI_DEC_STOP	Stop the playout of data.
IFX_TAPI_DEC_TYPE_SET	Select a codec for the data channel playout.
IFX_TAPI_DEC_VOLUME_SET	Sets the playout volume.
IFX_TAPI_ENC_FRAME_LEN_GET	This service gets the frame length for the audio packets.
IFX_TAPI_ENC_FRAME_LEN_SET	This service sets the frame length for the audio packets.
IFX_TAPI_ENC_LEVEL_SET	This service sets the level of the most recently recorded signal.
IFX_TAPI_ENC_START	Start recording on the data channel.
IFX_TAPI_ENC_STOP	Stop recording (generating packets) on this channel.
IFX_TAPI_ENC_TYPE_SET	Select a codec for the data channel.
IFX_TAPI_ENC_VAD_CFG_SET	Configures the voice activity detection and silence handling Voice Activity Detection (VAD) is a feature that allows the codec to determine when to send voice data or silence data.
IFX_TAPI_ENC_VOLUME_SET	Sets the recording volume.
IFX_TAPI_JB_CFG_SET	Configures the Jitter Buffer.
IFX_TAPI_JB_STATISTICS_GET	Reads out Jitter Buffer statistics.
IFX_TAPI_JB_STATISTICS_RESET	Resets the jitter buffer statistics.
IFX_TAPI_MAP_DATA_ADD	This interface adds the data channel to an analog phone device.
IFX_TAPI_MAP_DATA_REMOVE	This interface removes a data channel from an analog phone device.
IFX_TAPI_MAP_PCM_ADD	This interface adds the PCM channel to an analog phone device.
IFX_TAPI_MAP_PCM_REMOVE	This interface removes the PCM channel to an analog phone device.
IFX_TAPI_MAP_PHONE_ADD	This interface adds the phone channel to another analog phone channel.
IFX_TAPI_MAP_PHONE_REMOVE	This interface removes a phone channel from an analog phone device.
IFX_TAPI_PKT_AAL_CFG_SET	This interface configures AAL fields for a new connection.
IFX_TAPI_PKT_AAL_PROFILE_SET	AAL profile configuration.
IFX_TAPI_PKT_RTCP_STATISTICS_GET	Retrieves RTCP statistics.
IFX_TAPI_PKT_RTCP_STATISTICS_RESET	Resets the RTCP statistics.
IFX_TAPI_PKT_RTP_CFG_SET	This interface configures RTP and RTCP fields for a new connection.
IFX_TAPI_PKT_RTP_PT_CFG_SET	Change the payload type table.

Table 27 Structure Overview of Connection Control Service

Name	Description
IFX_TAPI_JB_CFG_t	Structure for jitter buffer configuration used by IFX_TAPI_JB_CFG_SET.
IFX_TAPI_JB_STATISTICS_t	Structure for Jitter Buffer statistics used by ioctl IFX_TAPI_JB_STATISTICS_GET.
IFX_TAPI_MAP_DATA_t	Phone channel mapping structure used for IFX_TAPI_MAP_DATA_ADD and IFX_TAPI_MAP_DATA_REMOVE.
IFX_TAPI_MAP_PHONE_t	Phone channel mapping structure used for IFX_TAPI_MAP_PHONE_ADD and IFX_TAPI_MAP_PHONE_REMOVE.
IFX_TAPI_PCK_AAL_CFG_t	Structure used for ioctl IFX_TAPI_PKT_AAL_CFG_SET
IFX_TAPI_PCK_AAL_PROFILE_t	AAL profile setup structure used for IFX_TAPI_PKT_AAL_PROFILE_SET.
IFX_TAPI_PKT_RTCP_STATISTICS_t	Structure for RTCP Statistics.
IFX_TAPI_PKT_RTP_CFG_t	Structure for RTP Configuration.
IFX_TAPI_PKT_RTP_PT_CFG_t	Structure for RTP payload configuration.

Table 28 Enumerator Overview of Connection Control Service

Name	Description
IFX_TAPI_DATA_MAP_START_STOP_t	Start/Stop information for data channel mapping.
IFX_TAPI_ENC_TYPE_t	Possible codecs to select for IFX_TAPI_ENC_TYPE_SET and IFX_TAPI_PCK_AAL_PROFILE_t.
IFX_TAPI_ENC_VAD_t	Enumeration used for ioctl IFX_TAPI_ENC_VAD_CFG_SET.
IFX_TAPI_MAP_DATA_TYPE_t	Data channel destination types (conferencing).
IFX_TAPI_PKT_AAL_PROFILE_RANGE_t	Used for IFX_TAPI_PCK_AAL_PROFILE_t in case one coder range.
IFX_TAPI_PKT_EV_OOB_t	Enumeration for IFX_TAPI_PKT_RTP_CFG_t event setting.

4.1.2.1 IFX_TAPI_DEC_LEVEL_GET

Description

This service returns the level of the most recently decoded signal.

Attention: Interface not implemented yet.

Prototype

```
IFX_int32_t ioctl (
        IFX_int32_t fd,
        IFX_TAPI_DEC_LEVEL_GET,
        IFX_int32_t param );
```

Parameters

Data Type	Name	Description
IFX_int32_t	fd	File descriptor. It is applicable to data channel file descriptors.
IFX_int32_t	IFX_TAPI_DEC_LEVEL_GET	I/O control identifier for this operation
IFX_int32_t	param	Pointer to an integer for the level

Return Values

Data Type	Description
IFX_int32_t	The return value can be either of the following: • IFX_SUCCESS 0 • IFX_ERROR -1

Example

```
IFX_int32_t fd;
IFX_int32_t nLevel;

/* Get the record level */
ioctl(fd, IFX_TAPI_DEC_LEVEL_GET, &nLevel);
/* Output level in dB can be calculated via formula: */
/* outlvl_in_dB = +3.17 + 20 log10 (nRecLevel/32767) */
```

4.1.2.2 IFX_TAPI_DEC_START

Description

Starts the decoding of data.

Prototype

```
IFX_int32_t ioctl (
        IFX_int32_t fd,
        IFX_TAPI_DEC_START,
        IFX_int32_t param );
```

Parameters

Data Type	Name	Description
IFX_int32_t	fd	File descriptor. It is applicable to data channel file descriptors.
IFX_int32_t	IFX_TAPI_DEC_START	I/O control identifier for this operation
IFX_int32_t	param	This interface expects no parameter. It should be set to 0.

Return Values

Data Type	Description
IFX_int32_t	The return value can be either of the following: • IFX_SUCCESS 0 • IFX_ERROR -1

Example

```
ioctl(fd, IFX_TAPI_DEC_START, 0);
```

4.1.2.3 IFX_TAPI_DEC_STOP

Description

Stops the decoding of data.

Attention: Stop of the decoding will lead to a reset of the connection statistics.

Prototype

```
IFX_int32_t ioctl (
        IFX_int32_t fd,
        IFX_TAPI_DEC_STOP,
        IFX_int32_t param );
```

Parameters

Data Type	Name	Description
IFX_int32_t	fd	File descriptor. It is applicable to data channel file descriptors.
IFX_int32_t	IFX_TAPI_DEC_STOP	I/O control identifier for this operation
IFX_int32_t	param	This interface expects no parameter. It should be set to 0.

Return Values

Data Type	Description
IFX_int32_t	The return value can be either of the following: • IFX_SUCCESS 0 • IFX_ERROR -1

Example

```
ioctl(fd, IFX_TAPI_DEC_STOP, 0);
```

4.1.2.4 IFX_TAPI_DEC_TYPE_SET

Description

Select a vocoder type for the data channel playout.

Prototype

```
IFX_int32_t ioctl (
        IFX_int32_t fd,
        IFX_TAPI_DEC_TYPE_SET,
        IFX_int32_t param );
```

Parameters

Data Type	Name	Description
IFX_int32_t	fd	File descriptor. It is applicable to data channel file descriptors.
IFX_int32_t	IFX_TAPI_DEC_TYPE_SET	I/O control identifier for this operation
IFX_int32_t	param	The parameter specifies the codec, default G711, to be chosen from IFX_TAPI_ENC_TYPE_t .

Return Values

Data Type	Description
IFX_int32_t	The return value can be either of the following: • IFX_SUCCESS 0 • IFX_ERROR -1

Remarks

This interface is currently not supported, because the codec is determined by the payload type of the received packets

4.1.2.5 IFX_TAPI_DEC_VOLUME_SET

Description

Sets the decoding playout volume.

Attention: Interface not implemented yet.

Prototype

```
IFX_int32_t ioctl (
        IFX_int32_t fd,
        IFX_TAPI_DEC_VOLUME_SET,
        IFX_int32_t param );
```

Parameters

Data Type	Name	Description
IFX_int32_t	fd	File descriptor. It is applicable to data channel file descriptors.
IFX_int32_t	IFX_TAPI_DEC_VOLUME_SET	I/O control identifier for this operation
IFX_int32_t	param	Playout volume, default 0x100

Return Values

Data Type	Description
IFX_int32_t	The return value can be either of the following: • IFX_SUCCESS 0 • IFX_ERROR -1

Example

```
ioctl(fd, IFX_TAPI_DEC_VOLUME_SET, 0x100 * 2);
```

4.1.2.6 IFX_TAPI_ENC_FRAME_LEN_GET

Description

This service reads the configured encoding length.

Prototype

```
IFX_int32_t ioctl (
        IFX_int32_t fd,
        IFX_TAPI_ENC_FRAME_LEN_GET,
        IFX_int32_t* param );
```

Parameters

Data Type	Name	Description
IFX_int32_t	fd	File descriptor. It is applicable to data channel file descriptors.
IFX_int32_t	IFX_TAPI_ENC_FRAME_LEN_GET	I/O control identifier for this operation
IFX_int32_t*	param	Pointer to the length of frames in milliseconds

Return Values

Data Type	Description
IFX_int32_t	The return value can be either of the following: • IFX_SUCCESS 0 • IFX_ERROR -1

Example

```
IFX_int32_t fd;
IFX_int32_t nFrameLength;

ioctl(fd, IFX_TAPI_ENC_FRAME_LEN_GET, &nFrameLength);
```

4.1.2.7 IFX_TAPI_ENC_FRAME_LEN_SET

Description

This service configures the encoding length.

Prototype

```
IFX_int32_t ioctl (
        IFX_int32_t fd,
        IFX_TAPI_ENC_FRAME_LEN_SET,
        IFX_int32_t param );
```

Parameters

Data Type	Name	Description
IFX_int32_t	fd	File descriptor. It is applicable to data channel file descriptors.
IFX_int32_t	IFX_TAPI_ENC_FRAME_LEN_SET	I/O control identifier for this operation
IFX_int32_t	param	Length of frames in milliseconds

Return Values

Data Type	Description
IFX_int32_t	The return value can be either of the following: • IFX_SUCCESS 0 • IFX_ERROR -1

Example

```
FX_int32_t fd;
IFX_int32_t nFrameLength;

nFrameLength = 10;
ioctl(fd, IFX_TAPI_ENC_FRAME_LEN_SET, nFrameLength);
```

4.1.2.8 IFX_TAPI_ENC_LEVEL_SET

Description

This service sets the encoding level.

Attention: Interface not implemented yet.

Prototype

```
IFX_int32_t ioctl (
        IFX_int32_t fd,
        IFX_TAPI_ENC_LEVEL_SET,
        IFX_int32_t param );
```

Parameters

Data Type	Name	Description
IFX_int32_t	fd	File descriptor. It is applicable to data channel file descriptors.
IFX_int32_t	IFX_TAPI_ENC_LEVEL_SET	I/O control identifier for this operation
IFX_int32_t	param	Pointer to an integer for the level

Return Values

Data Type	Description
IFX_int32_t	The return value can be either of the following: • IFX_SUCCESS 0 • IFX_ERROR -1

Example

```
IFX_int32_t fd;
IFX_int32_t nRecLevel;

/* Get the record level */
ioctl(fd, IFX_TAPI_ENC_LEVEL_SET, &nRecLevel);
/* Output level in dB can be calculated via formula: */
/* outlvl_in_dB = +3.17 + 20 log10 (nRecLevel/32767) */
```

4.1.2.9 IFX_TAPI_ENC_START

Description

Start encoding and packetization.

Prototype

```
IFX_int32_t ioctl (
        IFX_int32_t fd,
        IFX_TAPI_ENC_START,
        IFX_int32_t param );
```

Parameters

Data Type	Name	Description
IFX_int32_t	fd	File descriptor. It is applicable to data channel file descriptors.
IFX_int32_t	IFX_TAPI_ENC_START	I/O control identifier for this operation
IFX_int32_t	param	This interface expects no parameter. It should be set to 0.

Return Values

Data Type	Description
IFX_int32_t	The return value can be either of the following: • IFX_SUCCESS 0 • IFX_ERROR -1

Remarks

The voice is serviced with the drivers read and write interface. The data format is dependent on the selected setup (coder, chip setup, and so on). For example, a RTP setup will receive RTP packets. Read and write are always non-blocking. If the codec has not been set before with IFX_TAPI_ENC_TYPE_SET the encoder is started with G711.

Example

```
ioctl(fd, IFX_TAPI_ENC_START, 0);
```

4.1.2.10 IFX_TAPI_ENC_STOP

Description

Stop encoding and packetization on this channel.

Prototype

```
IFX_int32_t ioctl (
        IFX_int32_t fd,
        IFX_TAPI_ENC_STOP,
        IFX_int32_t param );
```

Parameters

Data Type	Name	Description
IFX_int32_t	fd	File descriptor. It is applicable to data channel file descriptors.
IFX_int32_t	IFX_TAPI_ENC_STOP	I/O control identifier for this operation
IFX_int32_t	param	This interface expects no parameter. It should be set to 0.

Return Values

Data Type	Description
IFX_int32_t	The return value can be either of the following: • IFX_SUCCESS 0 • IFX_ERROR -1

Example

```
ioctl(fd, IFX_TAPI_ENC_STOP, 0);
```

4.1.2.11 IFX_TAPI_ENC_TYPE_SET

Description

Select a vocoder for the encoding.

Prototype

```
IFX_int32_t ioctl (
        IFX_int32_t fd,
        IFX_TAPI_ENC_TYPE_SET,
        IFX_int32_t param );
```

Parameters

Data Type	Name	Description
IFX_int32_t	fd	File descriptor. It is applicable to data channel file descriptors.
IFX_int32_t	IFX_TAPI_ENC_TYPE_SET	I/O control identifier for this operation
IFX_int32_t	param	The parameter specifies the codec as defined in IFX_TAPI_ENC_TYPE_t. Default codec is G711, µ-law

Return Values

Data Type	Description
IFX_int32_t	The return value can be either of the following: • IFX_SUCCESS 0 • IFX_ERROR -1

Example

```
/* Set the codec */
ioctl(fd, IFX_TAPI_ENC_TYPE_SET, IFX_TAPI_ENC_TYPE_G726_16);
```

4.1.2.12 IFX_TAPI_ENC_VAD_CFG_SET

Description

Configures the voice activity detection and silence handling. Voice Activity Detection (VAD) is a feature that allows the codec to determine when to send voice data or silence data.

Prototype

```
IFX_int32_t ioctl (
        IFX_int32_t fd,
        IFX_TAPI_ENC_VAD_CFG_SET,
        IFX_int32_t param );
```

Parameters

Data Type	Name	Description
IFX_int32_t	fd	File descriptor. It is applicable to data channel file descriptors.
IFX_int32_t	IFX_TAPI_ENC_VAD_CFG_SET	I/O control identifier for this operation
IFX_int32_t	param	Select the VAD mode out of IFX_TAPI_ENC_VAD_t.

Return Values

Data Type	Description
IFX_int32_t	The return value can be either of the following: • IFX_SUCCESS 0 • IFX_ERROR -1

Remarks

Codecs G.723.1 and G.729A/B have a built in comfort noise generation (CNG). Explicitly switching on the CNG is not necessary. Voice activity detection may not be available for G.728 and G.729E.

Example

```
/* Set the VAD on */
ioctl(fd, IFX_TAPI_ENC_VAD_CFG_SET, IFX_TAPI_ENC_VAD_ON);
```

4.1.2.13 IFX_TAPI_ENC_VOLUME_SET

Description

Sets the encoding volume.

Attention: Interface not implemented yet.

Prototype

```
IFX_int32_t ioctl (
        IFX_int32_t fd,
        IFX_TAPI_ENC_VOLUME_SET,
        IFX_int32_t param );
```

Parameters

Data Type	Name	Description
IFX_int32_t	fd	File descriptor. It is applicable to data channel file descriptors.
IFX_int32_t	IFX_TAPI_ENC_VOLUME_SET	I/O control identifier for this operation
IFX_int32_t	param	Recording volume, default 0x100

Return Values

Data Type	Description
IFX_int32_t	The return value can be either of the following: • IFX_SUCCESS 0 • IFX_ERROR -1

Example

```
ioctl(fd, IFX_TAPI_ENC_VOLUME_SET, 0x100 * 2);
```

4.1.2.14 IFX_TAPI_JB_CFG_SET

Description

Configures the Jitter Buffer.

Prototype

```
IFX_int32_t ioctl (
        IFX_int32_t fd,
        IFX_TAPI_JB_CFG_SET,
        IFX_int32_t param );
```

Parameters

Data Type	Name	Description
IFX_int32_t	fd	File descriptor. It is applicable to data channel file descriptors.
IFX_int32_t	IFX_TAPI_JB_CFG_SET	I/O control identifier for this operation
IFX_int32_t	param	The parameter points to a IFX_TAPI_JB_CFG_t struct

Return Values

Data Type	Description
IFX_int32_t	The return value can be either of the following: • IFX_SUCCESS 0 • IFX_ERROR -1

Remarks

Select fixed or adaptive Jitter Buffer and set parameters. Modifies the Jitter Buffer settings during run-time.

Example

```
IFX_TAPI_JB_CFG_t param;
IFX_int32_t fd;

memset(&param, 0, sizeof(IFX_TAPI_JB_CFG_t));
/* Set to adaptive jitter buffer type */
```

```
param.nJbType = IFX_TAPI_JB_TYPE_ADAPTIVE;
ret = ioctl(fd, IFX_TAPI_JB_CFG_SET, param);
```

4.1.2.15 IFX_TAPI_JB_STATISTICS_GET

Description

Reads out Jitter Buffer statistics.

Prototype

```
IFX_int32_t ioctl (
        IFX_int32_t fd,
        IFX_TAPI_JB_STATISTICS_GET,
        IFX_int32_t param );
```

Parameters

Data Type	Name	Description
IFX_int32_t	fd	File descriptor. It is applicable to data channel file descriptors.
IFX_int32_t	IFX_TAPI_JB_STATISTICS_GET	I/O control identifier for this operation
IFX_int32_t	param	The parameter points to a IFX_TAPI_JB_STATISTICS_t structure.

Return Values

Data Type	Description
IFX_int32_t	The return value can be either of the following: • IFX_SUCCESS 0 • IFX_ERROR -1

Example

```
IFX_TAPI_JB_STATISTICS_t param;
IFX_int32_t fd;

memset(&param, 0, sizeof(IFX_TAPI_JB_STATISTICS_t));
ioctl(fd, IFX_TAPI_JB_STATISTICS_GET, param);
```

4.1.2.16 IFX_TAPI_JB_STATISTICS_RESET

Description

Resets the jitter buffer statistics.

Prototype

```
IFX_int32_t ioctl (
        IFX_int32_t fd,
```

```
        IFX_TAPI_JB_STATISTICS_RESET,
        IFX_int32_t param );
```

Parameters

Data Type	Name	Description
IFX_int32_t	fd	File descriptor. It is applicable to data channel file descriptors.
IFX_int32_t	IFX_TAPI_JB_STATISTICS_RESET	I/O control identifier for this operation
IFX_int32_t	param	This interface expects no parameter. It should be set to 0.

Return Values

Data Type	Description
IFX_int32_t	The return value can be either of the following: • IFX_SUCCESS 0 • IFX_ERROR -1

Example

```
ioctl(fd, IFX_TAPI_JB_STATISTICS_RESET, 0);
```

4.1.2.17 IFX_TAPI_MAP_DATA_ADD

Description

This interface connects the data channel to a phone channel.

Prototype

```
IFX_int32_t ioctl (
        IFX_int32_t fd,
        IFX_TAPI_MAP_DATA_ADD,
        IFX_int32_t param );
```

Parameters

Data Type	Name	Description
IFX_int32_t	fd	File descriptor. It is applicable to data channel file descriptors.
IFX_int32_t	IFX_TAPI_MAP_DATA_ADD	I/O control identifier for this operation
IFX_int32_t	param	The parameter points to a IFX_TAPI_MAP_DATA_t struct

Return Values

Data Type	Description
IFX_int32_t	The return value can be either of the following: • IFX_SUCCESS 0 • IFX_ERROR -1

Remarks

Applies to a data file descriptor.The host has to take care about the resource management. It has to count the data channel (codec) resource usage and maintains a list, which data channel is mapped to which phone channel. The available resources can be queried by IFX_TAPI_CAP_CHECK.

Example

```
IFX_TAPI_MAP_DATA_t datamap;
IFX_int32_t fd;

memset(&datamap, 0, sizeof (IFX_TAPI_MAP_DATA_t));
/* Add phone 0 to data 0 */
ioctl(fd, IFX_TAPI_MAP_DATA_ADD, &datamap);
datamap.nDstCh = 1;
/* Add phone 1 to data 0*/
ioctl(fd, IFX_TAPI_MAP_DATA_ADD, &datamap);
```

4.1.2.18 IFX_TAPI_MAP_DATA_REMOVE

Description

This interface removes a data channel from a phone channel.

Prototype

```
IFX_int32_t ioctl (
        IFX_int32_t fd,
        IFX_TAPI_MAP_DATA_REMOVE,
        IFX_int32_t param );
```

Parameters

Data Type	Name	Description
IFX_int32_t	fd	File descriptor. It is applicable to data channel file descriptors.
IFX_int32_t	IFX_TAPI_MAP_DATA_REMOVE	I/O control identifier for this operation
IFX_int32_t	param	The parameter points to a IFX_TAPI_MAP_DATA_t struct.

Return Values

Data Type	Description
IFX_int32_t	The return value can be either of the following: • IFX_SUCCESS 0 • IFX_ERROR -1

Remarks

Applies to a data file descriptor.The host has to take care about the resource management. It has to count the data channel (codec) resource usage and maintains a list, which data channel is mapped to which phone channel. The available resources can be queried by IFX_TAPI_CAP_CHECK.

Example

```
IFX_TAPI_MAP_DATA_t datamap;
IFX_int32_t fd;

memset(&datamap, 0, sizeof(IFX_TAPI_MAP_DATA_t));
/* Do something .... */
/* Remove connection between phone channel 1 (nDstCh=1)
and data channel 1 (fd) */
datamap.nDstCh = 1;
ioctl(fd, IFX_TAPI_MAP_DATA_REMOVE, &datamap);
/* Now phone and data resources are unmapped */
```

4.1.2.19 IFX_TAPI_MAP_PCM_ADD

Description

This interface connects the PCM channel to a phone channel.

Prototype

```
IFX_int32_t ioctl (
        IFX_int32_t fd,
        IFX_TAPI_MAP_PCM_ADD,
        IFX_int32_t param );
```

Parameters

Data Type	Name	Description
IFX_int32_t	fd	File descriptor. It is applicable to phone channel file descriptors containing a PCM interface.
IFX_int32_t	IFX_TAPI_MAP_PCM_ADD	I/O control identifier for this operation
IFX_int32_t	param	The parameter points to a IFX_TAPI_MAP_PCM_t struct.

Return Values

Data Type	Description
IFX_int32_t	The return value can be either of the following: • IFX_SUCCESS 0 • IFX_ERROR -1

Example

```
IFX_TAPI_MAP_PCM_t pcmMap;
IFX_int32_t fd;

memset(&pcmMap, 0, sizeof(IFX_TAPI_MAP_PCM_t));
/* Connect PCM2 (addressed by fd) to phone channel 1 (analog) */
pcmMap.nDstCh = 1;
ioctl(fd, IFX_TAPI_MAP_PCM_ADD, &pcmMap);
```

4.1.2.20 IFX_TAPI_MAP_PCM_REMOVE

Description

This interface removes the PCM channel from the phone channel.

Prototype

```
IFX_int32_t ioctl (
        IFX_int32_t fd,
        IFX_TAPI_MAP_PCM_REMOVE,
        IFX_int32_t param );
```

Parameters

Data Type	Name	Description
IFX_int32_t	fd	File descriptor. It is applicable to phone channel file descriptors containing a PCM interface.
IFX_int32_t	IFX_TAPI_MAP_PCM_REMOVE	I/O control identifier for this operation
IFX_int32_t	param	The parameter points to a IFX_TAPI_MAP_PCM_t struct.

Return Values

Data Type	Description
IFX_int32_t	The return value can be either of the following: • IFX_SUCCESS 0 • IFX_ERROR -1

Example

```
IFX_TAPI_MAP_PCM_t param;
```

```
IFX_int32_t fd;

memset(&param, 0, sizeof (IFX_TAPI_MAP_PCM_t));
/* PCM channel 1 is removed from the phone channel 2 */
param.nDstCh = 2;
/* fd1 represents PCM channel 1 */
ioctl(fd, IFX_TAPI_MAP_PCM_REMOVE, &param);
```

4.1.2.21 IFX_TAPI_MAP_PHONE_ADD

Description

This interface connects a phone channel to another phone channel.

Prototype

```
IFX_int32_t ioctl (
        IFX_int32_t fd,
        IFX_TAPI_MAP_PHONE_ADD,
        IFX_int32_t param );
```

Parameters

Data Type	Name	Description
IFX_int32_t	fd	File descriptor. It is applicable to phone channel file descriptors containing an analog interface.
IFX_int32_t	IFX_TAPI_MAP_PHONE_ADD	I/O control identifier for this operation
IFX_int32_t	param	The parameter points to a IFX_TAPI_MAP_PHONE_t struct.

Return Values

Data Type	Description
IFX_int32_t	The return value can be either of the following: • IFX_SUCCESS 0 • IFX_ERROR -1

Remarks

Applies to a phone channel file descriptor. The host has to take care about the resource management.

Example

```
IFX_TAPI_MAP_PHONE_t param;
IFX_int32_t fd;

memset(&param, 0, sizeof(IFX_TAPI_MAP_PHONE_t));
param.nPhoneCh = 1;
ioctl(fd, IFX_TAPI_MAP_PHONE_ADD, &param);
```

4.1.2.22 IFX_TAPI_MAP_PHONE_REMOVE

Description

This interface removes a phone channel from a phone channel.

Prototype

```
IFX_int32_t ioctl (
        IFX_int32_t fd,
        IFX_TAPI_MAP_PHONE_REMOVE,
        IFX_int32_t param );
```

Parameters

Data Type	Name	Description
IFX_int32_t	fd	File descriptor. It is applicable to phone channel file descriptors containing an analog interface.
IFX_int32_t	IFX_TAPI_MAP_PHONE_REMOVE	I/O control identifier for this operation
IFX_int32_t	param	The parameter points to a IFX_TAPI_MAP_PHONE_t structure.

Return Values

Data Type	Description
IFX_int32_t	The return value can be either of the following: • IFX_SUCCESS 0 • IFX_ERROR -1

Remarks

Applies to a phone channel file descriptor.

Example

```
IFX_TAPI_MAP_PHONE_t param;
IFX_int32_t fd;

memset(&param, 0, sizeof(IFX_TAPI_MAP_PHONE_t));
param.nPhoneCh = 1;
ioctl(fd, IFX_TAPI_MAP_PHONE_REMOVE, &param);
```

4.1.2.23 IFX_TAPI_PKT_AAL_CFG_SET

Description

This interface configures AAL fields for a new connection.

Prototype

```
IFX_int32_t ioctl (
```

```
        IFX_int32_t fd,
        IFX_TAPI_PKT_AAL_CFG_SET,
        IFX_int32_t param );
```

Parameters

Data Type	Name	Description
IFX_int32_t	fd	File descriptor. It is applicable to data channel file descriptors containing an analog interface.
IFX_int32_t	IFX_TAPI_PKT_AAL_CFG_SET	I/O control identifier for this operation
IFX_int32_t	param	The parameter points to a IFX_TAPI_PCK_AAL_CFG_t structure.

Return Values

Data Type	Description
IFX_int32_t	The return value can be either of the following: • IFX_SUCCESS 0 • IFX_ERROR -1

Example

```
IFX_TAPI_PKT_AAL_CFG_SET param;
memset (&param, 0,  sizeof (IFX_TAPI_PKT_AAL_CFG_SET));
param.nTimestamp = 0x1234;
ret = ioctl(fd, IFX_TAPI_PKT_AAL_CFG_SET, &param)
```

4.1.2.24 IFX_TAPI_PKT_AAL_PROFILE_SET

Description

AAL profile configuration.

Prototype

```
IFX_int32_t ioctl (
        IFX_int32_t fd,
        IFX_TAPI_PKT_AAL_PROFILE_SET,
        IFX_int32_t param );
```

Parameters

Data Type	Name	Description
IFX_int32_t	fd	File descriptor. It is applicable to data channel file descriptors containing an analog interface.

Data Type	Name	Description
IFX_int32_t	IFX_TAPI_PKT_AAL_PROFILE_SET	I/O control identifier for this operation
IFX_int32_t	param	The parameter points to a IFX_TAPI_PCK_AAL_PROFILE_t structure.

Return Values

Data Type	Description
IFX_int32_t	The return value can be either of the following: • IFX_SUCCESS 0 • IFX_ERROR -1

Remarks

Sets each row in a profile. A maximum number of 10 rows can be set up. This interface is called on initialization to program the selected AAL profile. The following example programs the ATMF profiles 3, 4 and 5

Example

```
IFX_TAPI_PCK_AAL_PROFILE_t Profile;
 switch  (nProfile)
{
       case    3:
       case    4:
       /* ATMF profile 3 and 4 */
      Profile.rows = 1;
      Profile.codec[10][0] = IFX_TAPI_ENC_TYPE_MLAW;
      Profile.len[10][0] = 43;
      Profile.nUUI[10][0] = IFX_TAPI_MAP_DATA_TYPE_DEFAULT;
       break ;
       case    5:
       /* ATMF profile 5 */
      Profile.rows = 2;
      Profile.codec[10][0] = IFX_TAPI_ENC_TYPE_MLAW;
      Profile.len[10][0] = 43;
      Profile.nUUI[10][0] = IFX_TAPI_PKT_AAL_PROFILE_RANGE_0_7;
      Profile.codec[10][1] = IFX_TAPI_ENC_TYPE_G726_32;
      Profile.len[10][1] = 43;
      Profile.nUUI[10][1] = IFX_TAPI_PKT_AAL_PROFILE_RANGE_8_15;
       break ;
}
ret = ioctl (fd, IFX_TAPI_PKT_AAL_PROFILE_SET, (INT)&Profile);
```

4.1.2.25 IFX_TAPI_PKT_RTCP_STATISTICS_GET

Description

Retrieves RTCP statistics.

Prototype

```
IFX_int32_t ioctl (
        IFX_int32_t fd,
        IFX_TAPI_PKT_RTCP_STATISTICS_GET,
        IFX_int32_t param );
```

Parameters

Data Type	Name	Description
IFX_int32_t	fd	File descriptor. It is applicable to data channel file descriptors.
IFX_int32_t	IFX_TAPI_PKT_RTCP_STATISTICS_GET	I/O control identifier for this operation
IFX_int32_t	param	Pointer to a IFX_TAPI_PKT_RTCP_STATISTICS_t structure according RFC 3550/3551 for a sender report.

Return Values

Data Type	Description
IFX_int32_t	The return value can be either of the following: • IFX_SUCCESS 0 • IFX_ERROR -1

Remarks

Not all statistics may be supported by the Infineon device.

Example

```
IFX_TAPI_PKT_RTCP_STATISTICS_t param;
IFX_int32_t fd;

ioctl(fd, IFX_TAPI_PKT_RTCP_STATISTICS_GET, (IFX_int32_t) &param);
```

4.1.2.26 IFX_TAPI_PKT_RTCP_STATISTICS_RESET

Description

Resets the RTCP statistics.

Prototype

```
IFX_int32_t ioctl (
        IFX_int32_t fd,
        IFX_TAPI_PKT_RTCP_STATISTICS_RESET,
        IFX_int32_t param );
```

Parameters

Data Type	Name	Description
IFX_int32_t	fd	File descriptor. It is applicable to data channel file descriptors.
IFX_int32_t	IFX_TAPI_PKT_RTCP_STATISTICS_RESET	I/O control identifier for this operation
IFX_int32_t	param	This interface expects no parameter. It should be set to 0.

Return Values

Data Type	Description
IFX_int32_t	The return value can be either of the following: • IFX_SUCCESS 0 • IFX_ERROR -1

Example

```
ioctl(fd, IFX_TAPI_PKT_RTCP_STATISTICS_RESET, 0);
```

4.1.2.27 IFX_TAPI_PKT_RTP_CFG_SET

Description

This interface configures RTP and RTCP fields for a new connection.

Prototype

```
IFX_int32_t ioctl (
        IFX_int32_t fd,
        IFX_TAPI_PKT_RTP_CFG_SET,
        IFX_int32_t param );
```

Parameters

Data Type	Name	Description
IFX_int32_t	fd	File descriptor. It is applicable to data channel file descriptors.
IFX_int32_t	IFX_TAPI_PKT_RTP_CFG_SET	I/O control identifier for this operation
IFX_int32_t	param	The parameter points to a IFX_TAPI_PKT_RTP_CFG_t structure.

Return Values

Data Type	Description
IFX_int32_t	The return value can be either of the following: • IFX_SUCCESS 0 • IFX_ERROR -1

Example

```
IFX_TAPI_PKT_RTP_CFG_t param;
IFX_int32_t fd;

memset(&param, 0, sizeof(IFX_TAPI_PKT_RTP_CFG_t));
param.nSeqNr = 0x1234;
ioctl(fd, IFX_TAPI_PKT_RTP_CFG_SET, &param);
```

4.1.2.28 IFX_TAPI_PKT_RTP_PT_CFG_SET

Description

Configure the payload type table.

Prototype

```
IFX_int32_t ioctl (
        IFX_int32_t fd,
        IFX_TAPI_PKT_RTP_PT_CFG_SET,
        IFX_int32_t *param );
```

Parameters

Data Type	Name	Description
IFX_int32_t	fd	File descriptor. It is applicable to data channel file descriptors.
IFX_int32_t	IFX_TAPI_PKT_RTP_PT_CFG_SET	I/O control identifier for this operation
IFX_int32_t	*param	The parameter points to a IFX_TAPI_PKT_RTP_PT_CFG_t structure.

Return Values

Data Type	Description
IFX_int32_t	The return value can be either of the following: • IFX_SUCCESS 0 • IFX_ERROR -1

Remarks

The requested payload type should have been negotiated with the peer prior to the connection set-up. Event payload types are negotiated during signaling phase and the driver and the device can be programmed

accordingly at the time of starting the media session. Event payload types shall not be modified when the session is in progress.

Example

```
IFX_TAPI_PKT_RTP_PT_CFG_t param;
IFX_int32_t fd;

memset(&param, 0, sizeof(IFX_TAPI_PKT_RTP_PT_CFG_t));
param.nPTup[6D] = 0x5;
param.nPTdown[12D] = 0x4;
ioctl(fd, IFX_TAPI_PKT_RTP_PT_CFG_SET, &param);
```

4.1.3 Dial Service

Contains services for dialing.

Table 29 IO-control Overview of Dial Service

Name	Description
IFX_TAPI_PKT_EV_OOB_SET	This service controls the DTMF sending mode.
IFX_TAPI_PULSE_ASCII_GET	This service reads the ASCII character representing the pulse digit out of the receive buffer.
IFX_TAPI_PULSE_GET	This service reads the dial pulse digit out of the receive buffer.
IFX_TAPI_PULSE_READY	This service checks if a pulse digit was received.
IFX_TAPI_TONE_DTMF_ASCII_GET	This service reads the ASCII character representing the DTMF digit out of the receive buffer.
IFX_TAPI_TONE_DTMF_GET	This service reads the DTMF digit out of the internal receive buffer.
IFX_TAPI_TONE_DTMF_READY_GET	This service checks if a DTMF digit was received.

4.1.3.1 IFX_TAPI_PKT_EV_OOB_SET

Description

This service controls the DTMF sending mode.

Prototype

```
IFX_int32_t ioctl (
        IFX_int32_t fd,
        IFX_TAPI_PKT_EV_OOB_SET,
        IFX_int32_t param );
```

Parameters

Data Type	Name	Description
IFX_int32_t	fd	File descriptor. It is applicable to data channel file descriptors.
IFX_int32_t	IFX_TAPI_PKT_EV_OOB_SET	I/O control identifier for this operation
IFX_int32_t	param	The parameter is of type IFX_TAPI_PKT_EV_OOB_t

Return Values

Data Type	Description
IFX_int32_t	The return value can be either of the following: • IFX_SUCCESS 0 • IFX_ERROR -1

Remarks

The DTMF out of band controls how events are reported. If switched on (**IFX_TAPI_PKT_EV_OOB_NO** or **IFX_TAPI_PKT_EV_OOB_DEFAULT**) events are reported according to RFC 2833. If switched off (**IFX_TAPI_PKT_EV_OOB_ONLY**) the events are coded as voice.

Example

```
IFX_TAPI_LINE_VOLUME_t param;
IFX_int32_t fd;

/* Set the DTMF sending mode to out of band */
ioctl(fd, IFX_TAPI_PKT_EV_OOB_SET, IFX_TAPI_PKT_EV_OOB_NO);
```

4.1.3.2 IFX_TAPI_PULSE_ASCII_GET

Description

This service reads the ASCII character representing the pulse digit out of the receive buffer.

Prototype

```
IFX_int32_t ioctl (
        IFX_int32_t fd,
        IFX_TAPI_PULSE_ASCII_GET,
        IFX_int32_t param );
```

Parameters

Data Type	Name	Description
IFX_int32_t	fd	File descriptor. It is applicable to phone channel file descriptors containing an analog interface.
IFX_int32_t	IFX_TAPI_PULSE_ASCII_GET	I/O control identifier for this operation
IFX_int32_t	param	This service reads the dial pulse digit out of the receive buffer, in ASCII format.

Return Values

Data Type	Description
IFX_int32_t	The return value can be either of the following: • **IFX_SUCCESS** 0 • **IFX_ERROR** -1

Example

```
IFX_TAPI_LINE_VOLUME_t param;
IFX_int32_t fd;
IFX_int32_t digit;

/* Get pulse digit */
ret = ioctl(fd, IFX_TAPI_PULSE_ASCII_GET, &digit);
```

4.1.3.3 IFX_TAPI_PULSE_GET

Description

This service reads the dial pulse digit out of the receive buffer.

Prototype

```
IFX_int32_t ioctl (
        IFX_int32_t fd,
        IFX_TAPI_PULSE_GET,
        IFX_int32_t* param );
```

Parameters

Data Type	Name	Description
IFX_int32_t	fd	File descriptor. It is applicable to phone channel file descriptors containing an analog interface.
IFX_int32_t	IFX_TAPI_PULSE_GET	I/O control identifier for this operation
IFX_int32_t*	param	The parameter points to the detected digit. It returns the following values: • 0: no digit detected • 1: Digit 1 • ... • 9: Digit 9 • B: 0

Return Values

Data Type	Description
IFX_int32_t	The return value can be either of the following: • **IFX_SUCCESS** 0 • **IFX_ERROR** -1

Remarks

The command **IFX_TAPI_CH_STATUS_GET** can be used upon exception. If the event reporting mask is not set (see **IFX_TAPI_EXCEPTION_MASK**), **IFX_TAPI_CH_STATUS_GET** reads out the digit. It is contained in the status information (field digit and dialing). It is then not available with this command. This approach minimizes the number of ioctls used to query all status information. No additional request to get the digit is necessary.

Example

```
IFX_TAPI_LINE_VOLUME_t param;
IFX_int32_t fd;
IFX_int32_t digit;

/* Get pulse digit */
ret = ioctl(fd, IFX_TAPI_PULSE_GET, &digit);
```

4.1.3.4 IFX_TAPI_PULSE_READY

Description

This service checks if a pulse digit was received.

Prototype

```
IFX_int32_t ioctl (
        IFX_int32_t fd,
        IFX_TAPI_PULSE_READY,
        IFX_int32_t param );
```

Parameters

Data Type	Name	Description
IFX_int32_t	fd	File descriptor. It is applicable to phone channel file descriptors containing an analog interface.
IFX_int32_t	IFX_TAPI_PULSE_READY	I/O control identifier for this operation
IFX_int32_t	param	The parameter points to the status. It returns the following values: • 0: No pulse digit is in the receive queue • 1: pulse digit is in the receive queue

Return Values

Data Type	Description
IFX_int32_t	The return value can be either of the following: • IFX_SUCCESS 0 • IFX_ERROR -1

Remarks

The command IFX_TAPI_CH_STATUS_GET can be used upon exception. If the event reporting mask is not set (see IFX_TAPI_EXCEPTION_MASK), IFX_TAPI_CH_STATUS_GET reads out the digit. It is contained in the status information (field digit and dialing). It is then not available with this command. This approach minimizes the number of ioctls used to query all status information. No additional request to get the digit is necessary.

Example

```
IFX_TAPI_LINE_VOLUME_t param;
IFX_int32_t fd;
IFX_int32_t ready;

/* Check the pulse dialing status */
ioctl(fd, IFX_TAPI_PULSE_READY, &ready);
if (1 == ready)
{
   /* Digit arrived */
}
else
{
```

```
    /* No digit in the buffer */
}
```

4.1.3.5 IFX_TAPI_TONE_DTMF_ASCII_GET

Description

This service reads the ASCII character representing the DTMF digit out of the receive buffer.

Prototype

```
IFX_int32_t ioctl (
        IFX_int32_t fd,
        IFX_TAPI_TONE_DTMF_ASCII_GET,
        IFX_int32_t param );
```

Parameters

Data Type	Name	Description
IFX_int32_t	fd	File descriptor. It is applicable to data channel file descriptors.
IFX_int32_t	IFX_TAPI_TONE_DTMF_ASCII_GET	I/O control identifier for this operation
IFX_int32_t	param	The parameter points to the detected digit in ASCII representation.

Return Values

Data Type	Description
IFX_int32_t	The return value can be either of the following: • IFX_SUCCESS 0 • IFX_ERROR -1

Example

```
IFX_TAPI_LINE_VOLUME_t param;
IFX_int32_t fd;
IFX_int32_t digit;

/* Get DTMF digit */
ret = ioctl(fd, IFX_TAPI_TONE_DTMF_ASCII_GET, &digit);
```

4.1.3.6 IFX_TAPI_TONE_DTMF_GET

Description

This service reads the DTMF digit out of the internal receive buffer.

Prototype

```
IFX_int32_t ioctl (
        IFX_int32_t fd,
```

```
IFX_TAPI_TONE_DTMF_GET,
IFX_int32_t param );
```

Parameters

Data Type	Name	Description
IFX_int32_t	fd	File descriptor. It is applicable to data channel file descriptors.
IFX_int32_t	IFX_TAPI_TONE_DTMF_GET	I/O control identifier for this operation
IFX_int32_t	param	The parameter points to the detected digit. It returns the following values: • 0: no digit detected • 1: Digit 1 • ... • 9: Digit 9 • A: * (star) • B: 0 • C: # (hash) • 1C: A • 1D: B • 1E: C • 1F: D

Return Values

Data Type	Description
IFX_int32_t	The return value can be either of the following: • **IFX_SUCCESS** 0 • **IFX_ERROR** -1

Remarks

The command **IFX_TAPI_CH_STATUS_GET** can be used upon exception. If the event reporting mask is not set (see **IFX_TAPI_EXCEPTION_MASK**), **IFX_TAPI_CH_STATUS_GET** reads out the digit. It is contained in the status information (field digit and dialing). It is then not available with this command. This approach minimizes the number of ioctls used to query all status information. No additional request to get the digit is necessary.

Example

```
IFX_TAPI_LINE_VOLUME_t param;
IFX_int32_t fd;
IFX_int32_t digit;

/* Get DTMF digit */
ret = ioctl(fd, IFX_TAPI_TONE_DTMF_GET, &digit);
```

4.1.3.7 IFX_TAPI_TONE_DTMF_READY_GET

Description

This service checks if a DTMF digit was received.

Prototype

```
IFX_int32_t ioctl (
        IFX_int32_t fd,
        IFX_TAPI_TONE_DTMF_READY_GET,
        IFX_int32_t param );
```

Parameters

Data Type	Name	Description
IFX_int32_t	fd	File descriptor. It is applicable to data channel file descriptors.
IFX_int32_t	IFX_TAPI_TONE_DTMF_READY_GET	I/O control identifier for this operation
IFX_int32_t	param	Pointer to an integer for the status information. It returns the following values: • 0: No DTMF digit is in the receive queue • 1: DTMF digit is in the receive queue

Return Values

Data Type	Description
IFX_int32_t	The return value can be either of the following: • IFX_SUCCESS 0 • IFX_ERROR -1

Remarks

The command IFX_TAPI_CH_STATUS_GET can be used upon exception. If the event reporting mask is not set (see IFX_TAPI_EXCEPTION_MASK), IFX_TAPI_CH_STATUS_GET reads out the digit. It is contained in the status information (field digit and dialing). It is then not available with this command. This approach minimizes the number of ioctls used to query all status information. No additional request to get the digit is necessary.

Example

```
IFX_int32_t fd;
IFX_int32_t ready;

/* Check the DTMF status */
ioctl(fd, IFX_TAPI_TONE_DTMF_READY_GET, &ready);
if (1 == ready)
{
   /* digit arrived */
}
else
{
```

```
      /* no digit in the buffer */
   }
```

4.1.4 Fax T.38 Service

The FAX services switch the corresponding data channel from voice to T.38 data pump.

All fax services apply to data channels except otherwise stated.

Table 30 IO-control Overview of Fax T.38 Service

Name	Description
IFX_TAPI_T38_DEMOD_START	This service configures and enables the demodulator for a T.38 fax session.
IFX_TAPI_T38_MOD_START	This service configures and enables the modulator for a T.38 fax session.
IFX_TAPI_T38_STATUS_GET	This service provides the T.38 fax status on query.
IFX_TAPI_T38_STOP	This service disables the T.38 fax data pump and activates the voice path again.

Table 31 Structure Overview of Fax T.38 Service

Name	Description
IFX_TAPI_T38_DEMOD_DATA_t	Structure to setup the demodulator for T.38 fax and used for IFX_TAPI_T38_DEMOD_START.
IFX_TAPI_T38_MOD_DATA_t	Structure to setup the modulator for T.38 fax and used for IFX_TAPI_T38_MOD_START.
IFX_TAPI_T38_STATUS_t	Structure to read the T.38 fax status and used for IFX_TAPI_T38_STATUS_GET.

Table 32 Enumerator Overview of Fax T.38 Service

Name	Description
IFX_TAPI_T38_ERROR_t	T38 Fax errors.
IFX_TAPI_T38_STATUS_t	T38 Fax Datapump states.
IFX_TAPI_T38_STD_t	T38 Fax standards.

4.1.4.1 IFX_TAPI_T38_DEMOD_START

Description

This service configures and enables the demodulator for a T.38 fax session.

Prototype

```
IFX_int32_t ioctl (
        IFX_int32_t fd,
        IFX_TAPI_T38_DEMOD_START,
        IFX_int32_t param );
```

Parameters

Data Type	Name	Description
IFX_int32_t	fd	File descriptor. It is applicable to data channel file descriptors.

Data Type	Name	Description
IFX_int32_t	IFX_TAPI_T38_DEMOD_START	I/O control identifier for this operation
IFX_int32_t	param	The parameter points to a IFX_TAPI_T38_DEMOD_DATA_t structure.

Return Values

Data Type	Description
IFX_int32_t	The return value can be either of the following: • IFX_SUCCESS 0 • IFX_ERROR -1

Remarks

This service deactivates the voice path and configures the driver for a fax session.

Example

```
IFX_TAPI_T38_DEMOD_DATA_t param;
IFX_int32_t fd;

memset(&param, 0, sizeof(IFX_TAPI_T38_DEMOD_DATA_t));
/* Set V.21 standard as standard used for fax */
param.nStandard1 = 0x01;
/* Set V.17/14400 as alternative standard */
param.nStandard2 = 0x09;
ioctl(fd, IFX_TAPI_T38_DEMOD_START, &param);
```

4.1.4.2 IFX_TAPI_T38_MOD_START

Description

This service configures and enables the modulator for a T.38 fax session.

Prototype

```
IFX_int32_t ioctl (
        IFX_int32_t fd,
        IFX_TAPI_T38_MOD_START,
        IFX_int32_t param );
```

Parameters

Data Type	Name	Description
IFX_int32_t	fd	File descriptor. It is applicable to data channel file descriptors.

Data Type	Name	Description
IFX_int32_t	IFX_TAPI_T38_MOD_START	I/O control identifier for this operation
IFX_int32_t	param	The parameter points to a IFX_TAPI_T38_MOD_DATA_t structure.

Return Values

Data Type	Description
IFX_int32_t	The return value can be either of the following: • IFX_SUCCESS 0 • IFX_ERROR -1

Remarks

This service deactivates the voice path and configures the channel for a fax session.

Example

```
IFX_TAPI_T38_MOD_DATA_t param;
IFX_int32_t fd;

memset(&param, 0, sizeof(IFX_TAPI_T38_MOD_DATA_t));
/* Set V.21 standard */
param.nStandard = 0x01;
ioctl(fd, IFX_TAPI_T38_MOD_START, &param);
```

4.1.4.3 IFX_TAPI_T38_STATUS_GET

Description

This service provides the T.38 fax status on query.

Prototype

```
IFX_int32_t ioctl (
        IFX_int32_t fd,
        IFX_TAPI_T38_STATUS_GET,
        IFX_int32_t param );
```

Parameters

Data Type	Name	Description
IFX_int32_t	fd	File descriptor. It is applicable to data channel file descriptors.
IFX_int32_t	IFX_TAPI_T38_STATUS_GET	I/O control identifier for this operation
IFX_int32_t	param	The parameter points to a IFX_TAPI_T38_STATUS_t structure.

Return Values

Data Type	Description
IFX_int32_t	The return value can be either of the following: • IFX_SUCCESS 0 • IFX_ERROR -1

Remarks

This interface must be used when a fax exception has occurred to know the status. An error or a change of status will be indicated with TAPI exceptions, refer to IFX_TAPI_EXCEPTION_t.

Example

```
IFX_TAPI_T38_STATUS_t param;
IFX_int32_t fd;

memset(&param, 0, sizeof(IFX_TAPI_T38_STATUS_t));
/* Request status */
ioctl(fd, IFX_TAPI_T38_STATUS_GET, &param);
```

4.1.4.4 IFX_TAPI_T38_STOP

Description

This service disables the T.38 fax data pump and activates the voice path again.

Prototype

```
IFX_int32_t ioctl (
        IFX_int32_t fd,
        IFX_TAPI_T38_STOP,
        IFX_int32_t param );
```

Parameters

Data Type	Name	Description
IFX_int32_t	fd	File descriptor. It is applicable to data channel file descriptors.
IFX_int32_t	IFX_TAPI_T38_STOP	I/O control identifier for this operation
IFX_int32_t	param	This interface expects no parameter. It should be set to 0.

Return Values

Data Type	Description
IFX_int32_t	The return value can be either of the following: • IFX_SUCCESS 0 • IFX_ERROR -1

Example

```
IFX_TAPI_LINE_VOLUME_t param;
IFX_int32_t fd;

ioctl(fd, IFX_TAPI_T38_STOP, 0);
```

4.1.5 Initialization Service

This service sets the default initialization of device and hardware.

Table 33 IO-control Overview of Initialization Service

Name	Description
IFX_TAPI_CH_INIT	This service sets the default initialization of device and hardware.
IFX_TAPI_DWLD	Download service.

Table 34 Structure Overview of Initialization Service

Name	Description
IFX_TAPI_CH_INIT_t	TAPI initialization structure used for IFX_TAPI_CH_INIT.
IFX_TAPI_DWLD_t	Structure used for download.

Table 35 Enumerator Overview of Initialization Service

Name	Description
IFX_TAPI_CH_INIT_COUNTRY_t	Country selection for IFX_TAPI_CH_INIT_t.
IFX_TAPI_CH_INIT_MODE_t	TAPI initialization modes, selection for target system.
IFX_TAPI_DWLD_TYPE_t	Definition of download type.

4.1.5.1 IFX_TAPI_CH_INIT

Description

This service sets the default initialization of the device and of the specific channel.

It applies to channel file descriptors.

Prototype

```
IFX_void_t ioctl (
        IFX_int32_t fd,
        IFX_TAPI_CH_INIT,
        IFX_int32_t param );
```

Parameters

Data Type	Name	Description
IFX_int32_t	fd	File descriptor
IFX_int32_t	IFX_TAPI_CH_INIT	I/O control identifier for this operation
IFX_int32_t	param	The parameter points to a IFX_TAPI_CH_INIT_t structure.

Return Values

Data Type	Description
IFX_void_t	No return value

Example

```
IFX_TAPI_CH_INIT_t Init;

Init.nMode = 0;
Init.nCountry = 2;
Init.pProc = &DevInitStruct;
ioctl(fd, IFX_TAPI_CH_INIT, &Init);
```

4.1.5.2 IFX_TAPI_DWLD

Description

This service issues a download.

The type of dowload must be configure in **IFX_TAPI_DWLD_t**. The different download types are defined in **IFX_TAPI_DWLD_TYPE_t**.

It applies to channel file descriptors.

Attention: Interface not implemented yet.

Prototype

```
IFX_void_t ioctl (
        IFX_int32_t fd,
        IFX_TAPI_DWLD,
        IFX_int32_t param );
```

Parameters

Data Type	Name	Description
IFX_int32_t	fd	Channel file descriptor.
IFX_int32_t	IFX_TAPI_DWLD	I/O control identifier for this operation.
IFX_int32_t	param	The parameter points to a **IFX_TAPI_DWLD_t** structure.

Return Values

Data Type	Description
IFX_void_t	No return value.

4.1.6 Metering Service

Contains services for metering.

All metering services apply to phone channels except otherwise stated.

Table 36 IO-control Overview of Metering Service

Name	Description
IFX_TAPI_METER_CFG_SET	This service sets the characteristic for the metering service.
IFX_TAPI_METER_START	This service starts the metering.
IFX_TAPI_METER_STOP	This service stops the metering.

Table 37 Structure Overview of Metering Service

Name	Description
IFX_TAPI_METER_CFG_t	Structure for metering config.

4.1.6.1 IFX_TAPI_METER_CFG_SET

Description

This service sets the characteristic for the metering service.

Prototype

```
IFX_int32_t ioctl (
        IFX_int32_t fd,
        IFX_TAPI_METER_CFG_SET,
        IFX_int32_t param );
```

Parameters

Data Type	Name	Description
IFX_int32_t	fd	File descriptor. It applies to phone channel file descriptors.
IFX_int32_t	IFX_TAPI_METER_CFG_SET	I/O control identifier for this operation
IFX_int32_t	param	The parameter points to a IFX_TAPI_METER_CFG_t structure.

Return Values

Data Type	Description
IFX_int32_t	The return value can be either of the following: • IFX_SUCCESS 0 • IFX_ERROR -1

Remarks

If burst_cnt is not zero the burst number is defined by burst_cnt and stopping is not necessary. If burst_cnt is set to zero the bursts are sent out until a IFX_TAPI_METER_STOP ioctl is called.

Example

```
IFX_TAPI_METER_CFG_t Metering;
memset (&Metering, 0,  sizeof (IFX_TAPI_METER_CFG_t));
 /* Metering mode is already set to 0 */
 /* 100ms burst length */
Metering.burst_len = 100;
 /* 2 min. burst distance */
Metering.burst_dist = 120;
 /* send out 2 bursts */
Metering.burst_cnt = 2;
 /* set metering characteristic */
ioctl(fd, IFX_TAPI_METER_CFG_SET, &Metering);
```

4.1.6.2 IFX_TAPI_METER_START

Description

This service starts the metering.

Prototype

```
IFX_int32_t ioctl (
        IFX_int32_t fd,
        IFX_TAPI_METER_START,
        IFX_int32_t param );
```

Parameters

Data Type	Name	Description
IFX_int32_t	fd	File descriptor. It applies to phone channel file descriptors.
IFX_int32_t	IFX_TAPI_METER_START	I/O control identifier for this operation
IFX_int32_t	param	This interface expects no parameter. It should be set to 0.

Return Values

Data Type	Description
IFX_int32_t	The return value can be either of the following: • IFX_SUCCESS 0 • IFX_ERROR -1

Remarks

Before this service can be used, the metering characteristic must be set (service IFX_TAPI_METER_CFG_SET) and the line mode must be set to normal (service IFX_TAPI_LINE_FEED_SET).

Example

```
ioctl(fd, IFX_TAPI_METER_START, 0);
```

4.1.6.3 IFX_TAPI_METER_STOP

Description

This service stops the metering.

Prototype

```
IFX_int32_t ioctl (
        IFX_int32_t fd,
        IFX_TAPI_METER_STOP,
        IFX_int32_t param );
```

Parameters

Data Type	Name	Description
IFX_int32_t	fd	File descriptor. It applies to phone channel file descriptors.
IFX_int32_t	IFX_TAPI_METER_STOP	I/O control identifier for this operation
IFX_int32_t	param	This interface expects no parameter. It should be set to 0.

Return Values

Data Type	Description
IFX_int32_t	The return value can be either of the following: • IFX_SUCCESS 0 • IFX_ERROR -1

Remarks

If the metering has not been started before this service returns an error

Example

```
ioctl(fd, IFX_TAPI_METER_STOP, 0);
```

4.1.7 Miscellaneous Services

Contains services for status and version information. This is applicable to phone channels except otherwise stated.

Table 38 IO-control Overview of Miscellaneous Services

Name	Description
IFX_TAPI_CAP_CHECK	This service checks if a specific capability is supported.
IFX_TAPI_CAP_LIST	This service returns the capability lists.
IFX_TAPI_CAP_NR	This service returns the number of capabilities.
IFX_TAPI_CH_STATUS_GET	Query status information about the phone line.
IFX_TAPI_DEBUG_REPORT_SET	Set the report levels if the driver is compiled with ENABLE_TRACE.
IFX_TAPI_EXCEPTION_GET	This service returns the current exception status.
IFX_TAPI_EXCEPTION_MASK	This service masks the reporting of the exceptions.
IFX_TAPI_VERSION_CHECK	Check the supported TAPI interface version.
IFX_TAPI_VERSION_GET	Retrieves the TAPI version string.

Table 39 Structure Overview of Miscellaneous Services

Name	Description
IFX_TAPI_CAP_t	Capability structure.
IFX_TAPI_CH_STATUS_t	Structure used for IFX_TAPI_CH_STATUS_GET to query the phone status of one or more channels.
IFX_TAPI_VERSION_t	Structure used for the TAPI version support check.

Table 40 Union Overview of Miscellaneous Services

Name	Description
IFX_TAPI_EXCEPTION_t	Contains TAPI exception information.

Table 41 Enumerator Overview of Miscellaneous Service

Name	Description
IFX_TAPI_CAP_PORT_t	Lists the ports for the capability list.
IFX_TAPI_CAP_SIG_DETECT_t	Lists the signal detectors for the capability list.
IFX_TAPI_CAP_TYPE_t	Enumeration used for phone capabilities types.
IFX_TAPI_DIALING_STATUS_t	Enumeration for dial status events.
IFX_TAPI_LINE_HOOK_STATUS_t	Enumeration for hook status events.
IFX_TAPI_LINE_STATUS_t	Enumeration for phone line status information.

4.1.7.1 IFX_TAPI_CAP_CHECK

Description

This service checks if a specific capability is supported.

Prototype

```
IFX_int32_t ioctl (
```

```
IFX_int32_t fd,
IFX_TAPI_CAP_CHECK,
IFX_int32_t param );
```

Parameters

Data Type	Name	Description
IFX_int32_t	fd	File descriptor. It is applicable to device file descriptors.
IFX_int32_t	IFX_TAPI_CAP_CHECK	I/O control identifier for this operation
IFX_int32_t	param	The parameter points to a IFX_TAPI_CAP_t structure

Return Values

Data Type	Description
IFX_int32_t	The return value can be either of the following: • IFX_SUCCESS 0 (capability not supported) • IFX_ERROR -1 (error) • 1 (capability supported)

Example

```
IFX_TAPI_CAP_t CapList;
IFX_int32_t fd;
IFX_return_t ret;


/* Open channel file descriptor for channel 0 */
fd = open("/dev/vin11", O_RDWR, 0x644);

memset(&CapList, 0, sizeof(IFX_TAPI_CAP_t));
/* Check if G726, 16 kBit/s is supported */
CapList.captype = IFX_TAPI_CAP_TYPE_CODEC;
CapList.cap = IFX_TAPI_ENC_TYPE_G726_16;
ret = ioctl(fd, IFX_TAPI_CAP_CHECK, &CapList);
if (ret > 0)
{
   printf( "G726_16 supported\n" );
}
/* Check how many data channels are supported */
CapList.captype = IFX_TAPI_CAP_TYPE_CODECS;
ret = ioctl(fd, IFX_TAPI_CAP_CHECK, &CapList);
if (ret > 0)
{
   printf( "%d data channels supported\n" , CapList.cap);
}
/* Check if POTS port is available */
CapList.captype = IFX_TAPI_CAP_TYPE_PORT;
CapList.cap = IFX_TAPI_CAP_PORT_POTS;
```

```
ret = ioctl(fd, IFX_TAPI_CAP_CHECK, &CapList);
if (ret > 0)
{
   printf( "POTS port supported\n");
}

/* close all open fds */
close(fd);
```

4.1.7.2 IFX_TAPI_CAP_LIST

Description

This service returns the capability lists.

Prototype

```
IFX_int32_t ioctl (
       IFX_int32_t fd,
       IFX_TAPI_CAP_LIST,
       IFX_int32_t param );
```

Parameters

Data Type	Name	Description
IFX_int32_t	fd	File descriptor. It is applicable to device file descriptors.
IFX_int32_t	IFX_TAPI_CAP_LIST	I/O control identifier for this operation
IFX_int32_t	param	The parameter points to a IFX_TAPI_CAP_t structure

Return Values

Data Type	Description
IFX_int32_t	The return value can be either of the following: • IFX_SUCCESS 0 • IFX_ERROR -1

Example

```
IFX_TAPI_CAP_t *pCapList;
IFX_int32_t fd;
IFX_return_t ret;
IFX_int32_t nCapNr;
IFX_int32_t i;


/* Open channel file descriptor for channel 0 */
fd = open("/dev/vin11", O_RDWR, 0x644);

/* Get the cap list size */
```

```
ioctl(fd, IFX_TAPI_CAP_NR, &nCapNr);
pCapList = (IFX_TAPI_CAP_t*) malloc (nCapNr * sizeof(IFX_TAPI_CAP_t));
/* Get the cap list */
ioctl(fd, IFX_TAPI_CAP_LIST, pCapList);
for (i = 0; i < nCapNr; i++)
{
   switch (pCapList[i].captype)
   {
      case IFX_TAPI_CAP_TYPE_CODEC:
                printf( "Codec: %s\n\r" , pCapList[i].desc);
            break;
      case IFX_TAPI_CAP_TYPE_PCM:
                printf( "PCM: %d\n\r" , pCapList[i].cap);
            break;
    case IFX_TAPI_CAP_TYPE_CODECS:
                printf( "CODER: %d\n\r" , pCapList[i].cap);
            break;
    case IFX_TAPI_CAP_TYPE_PHONES:
                printf( "PHONES: %d\n\r" , pCapList[i].cap);
            break;
    default:
            break;
   }
}

/* Free the allocated memory */
free(pCapList);
pCapList = IFX_NULL;

/* Close all open fds */
close(fd);
```

4.1.7.3 IFX_TAPI_CAP_NR

Description

This service returns the number of capabilities.

Prototype

```
IFX_int32_t ioctl (
       IFX_int32_t fd,
       IFX_TAPI_CAP_NR,
       IFX_int32_t param );
```

Parameters

Data Type	Name	Description
IFX_int32_t	fd	File descriptor. It is applicable to device file descriptors.

Data Type	Name	Description
IFX_int32_t	IFX_TAPI_CAP_NR	I/O control identifier for this operation
IFX_int32_t	param	Pointer to the number of capabilities which are returned

Return Values

Data Type	Description
IFX_int32_t	The return value can be either of the following: • IFX_SUCCESS 0 • IFX_ERROR -1

Example

```
IFX_TAPI_LINE_VOLUME_t param;
IFX_int32_t fd;
IFX_int32_t nCapNr;

/* Get the cap list size */
ioctl(fd, IFX_TAPI_CAP_NR, &nCapNr);
```

4.1.7.4 IFX_TAPI_CH_STATUS_GET

Description

Query status information about the phone line.

Attention: It is strongly recommended to use the new event reporting interfaces, see Chapter 4.1.8. This interface is going to be discontinued in a next TAPI release.

Prototype

```
IFX_int32_t ioctl (
        IFX_int32_t fd,
        IFX_TAPI_CH_STATUS_GET,
        IFX_int32_t param );
```

Parameters

Data Type	Name	Description
IFX_int32_t	fd	File descriptor. It is applicable to device file descriptors.
IFX_int32_t	IFX_TAPI_CH_STATUS_GET	I/O control identifier for this operation
IFX_int32_t	param	The parameter points to a IFX_TAPI_CH_STATUS_t array.

Return Values

Data Type	Description
IFX_int32_t	The return value can be either of the following: • IFX_SUCCESS 0 • IFX_ERROR -1

Remarks

If the exception bits reporting of digits is enabled (see IFX_TAPI_EXCEPTION_MASK), the digit fifo is read out once and the information is passed to the status. It may occur that more than one digit is located in the fifo. Therefore the application must check with IFX_TAPI_TONE_DTMF_READY_GET or IFX_TAPI_PULSE_READY if further data is contained in the fifos. This interface replaces the exception bits.

Example

```
enum { TAPI_STATUS_DTMF = 0x01 };
enum { TAPI_STATUS_PULSE = 0x02 };

IFX_TAPI_CH_STATUS_t status[1];
IFX_int32_t fd;

/* This example shows the call of the status information of one channel */
status[0].channels = 0;
/* Get the status */
ioctl(fd, IFX_TAPI_CH_STATUS_GET, status);

if (status[0].dialing & TAPI_STATUS_DTMF
    || status[0].dialing & TAPI_STATUS_PULSE)
{
   printf( "Dial key %d detected\n", status[0].digit);
}

/* ---------------------------------------------------------------- */

/* This example shows a select waiting on one fd for exceptions and on */
/* two others for data. In one call the status information of all two */
/* channels are queried from the driver */
IFX_TAPI_CH_STATUS_t stat[4];
IFX_int32_t fd_dev, fd[2], i;
fd_set rfds;
IFX_int32_t width;


FD_ZERO (&rfds);
fd_dev = open("/dev/vin10", O_RDWR);
FD_SET (fd_dev, &rfds);
width = fd_dev;
fd[0] = open("/dev/vin11", O_RDWR);
FD_SET (fd[0], &rfds);
if (width < fd[0])
{
```

```
      width = fd[0];
   }
   fd[1] = open("/dev/vin12", O_RDWR);
   FD_SET (fd[1], &rfds);
   if (width < fd[1])
   {
      width = fd[1];
   }
   ret = select(width + 1, &rfds, NULL, NULL, NULL);
   /* Select woke up from exception on fd_dev or data on fd[x] */
   if (FD_ISSET(fd_dev, &rfds))
   {
      /* Exception occurred */
      status[0].channels = 2;
      /* Get the status */
      ioctl(fd_dev, IFX_TAPI_CH_STATUS_GET, stat);
      for (i = 0; i < 2; i++)
      {
         if (stat[i].hook || stat[i].dialing ||
             stat[i].line || stat[i].digit || stat[i].signal)
         {
            printf("Handle exception\n");
         }
      }
   }

   for (i = 0; i < 2; i++)
   {
      if (FD_ISSET(fd[i], &rfds))
      {
         printf("Handle data\n");
   } /* for */

   /* close all open fds */
   close(fd_dev);
   close(fd[0]);
   close(fd[1]);
```

4.1.7.5 IFX_TAPI_DEBUG_REPORT_SET

Description

Set the report levels if the driver is compiled with `ENABLE_TRACE`.

Prototype

```
IFX_void_t ioctl (
       IFX_int32_t fd,
       IFX_TAPI_DEBUG_REPORT_SET,
       IFX_int32_t param );
```

Parameters

Data Type	Name	Description
IFX_int32_t	fd	File descriptor. It is applicable to device file descriptors.
IFX_int32_t	IFX_TAPI_DEBUG_REPORT_SET	I/O control identifier for this operation
IFX_int32_t	param	Valid arguments is one value of IFX_TAPI_DEBUG_REPORT_SET_t

Return Values

Data Type	Description
IFX_void_t	No return value

4.1.7.6 IFX_TAPI_EXCEPTION_GET

Description

This service returns the current exception status.

Attention: It is strongly recommended to use the new event reporting interfaces, see Chapter 4.1.8. This interface is going to be discontinued in a next TAPI release.

Prototype

```
IFX_void_t ioctl (
        IFX_int32_t fd,
        IFX_TAPI_EXCEPTION_GET,
        IFX_int32_t param );
```

Parameters

Data Type	Name	Description
IFX_int32_t	fd	File descriptor. It is applicable to device file descriptors.
IFX_int32_t	IFX_TAPI_EXCEPTION_GET	I/O control identifier for this operation
IFX_int32_t	param	The parameter points to a IFX_TAPI_EXCEPTION_t structure.

Return Values

Data Type	Description
IFX_void_t	No return value

Remarks

The user application must poll cidrx_supdate, once the CID Receiver was started with IFX_TAPI_CID_RX_START. If it is set the application calls IFX_TAPI_CID_RX_STATUS_GET in order to know if CID data is available for reading or if an error occurred during CID Reception.

Example

```
IFX_TAPI_EXCEPTION_t nException;
IFX_int32_t fd;

/* Get the exception */
nException.Status = ioctl(fd, IFX_TAPI_EXCEPTION_GET, 0);
if (nException.Bits.dtmf_ready)
{
   printf ("DTMF detected\n");
}
```

4.1.7.7 IFX_TAPI_EXCEPTION_MASK

Description

This service masks the reporting of the exceptions.

Attention: It is strongly recommended to use the new event reporting interfaces, see Chapter 4.1.8. This interface is going to be discontinued in a next TAPI release.

Prototype

```
IFX_int32_t ioctl (
       IFX_int32_t fd,
       IFX_TAPI_EXCEPTION_MASK,
       IFX_int32_t param );
```

Parameters

Data Type	Name	Description
IFX_int32_t	fd	File descriptor. It is applicable to channel file descriptors.
IFX_int32_t	IFX_TAPI_EXCEPTION_MASK	I/O control identifier for this operation
IFX_int32_t	param	The parameter defines the exception event bits mask, defined in IFX_TAPI_EXCEPTION_BITS_t.

Return Values

Data Type	Description
IFX_int32_t	The return value can be either of the following: • IFX_SUCCESS 0 • IFX_ERROR -1

Remarks

If digit reporting is enabled, digits are also read out in IFX_TAPI_CH_STATUS_GET

Example

```
IFX_int32_t fd;
```

```
IFX_TAPI_EXCEPTION_t nException;

/* Mask reporting of exceptions */
ioctl(fd, IFX_TAPI_EXCEPTION_MASK, nException);
```

4.1.7.8 IFX_TAPI_VERSION_CHECK

Description

Check the supported TAPI interface version.

Prototype

```
IFX_void_t ioctl (
       IFX_int32_t fd,
       IFX_TAPI_VERSION_CHECK,
       IFX_int32_t param );
```

Parameters

Data Type	Name	Description
IFX_int32_t	fd	File descriptor. It is applicable to device file descriptors.
IFX_int32_t	IFX_TAPI_VERSION_CHECK	I/O control identifier for this operation
IFX_int32_t	param	The parameter points to a IFX_TAPI_VERSION_t structure.

Return Values

Data Type	Description
IFX_void_t	No return value

Remarks

Since an application is always build against one specific TAPI interface version it should check if it is supported. If not the application should abort. This interface checks if the current TAPI version supports a particular version. For example the TAPI versions 2.1 will support TAPI 2.0. But version 3.0 might not support 2.0.

Example

```
IFX_TAPI_VERSION_t version;
IFX_int32_t fd;

memset(&version, 0, sizeof(IFX_TAPI_VERSION_t));
version.major = 2;
version.minor = 1;
if (0 == ioctl(fd, IFX_TAPI_VERSION_CHECK, (IFX_int32_t) &version))
{
   printf( "Version 2.1 supported\n");
}
```

4.1.7.9 IFX_TAPI_VERSION_GET

Description

Retrieves the TAPI version string.

Prototype

```
IFX_int32_t ioctl (
        IFX_int32_t fd,
        IFX_TAPI_VERSION_GET,
        IFX_int32_t param );
```

Parameters

Data Type	Name	Description
IFX_int32_t	fd	File descriptor. It is applicable to device file descriptors.
IFX_int32_t	IFX_TAPI_VERSION_GET	I/O control identifier for this operation
IFX_int32_t	param	Pointer to version character string

Return Values

Data Type	Description
IFX_int32_t	The return value can be either of the following: • IFX_SUCCESS 0 • IFX_ERROR -1

Example

```
IFX_int32_t fd;
IFX_char_t Version[80];

ioctl(fd, IFX_TAPI_VERSION_GET, (IFX_char_t*) &Version[0]);
printf("Version:%s\n" , Version);
```

4.1.8 Event Reporting Services

Contains services for event reporting. This is applicable to device file descriptors except otherwise stated.

Table 42 IO-control Overview of Miscellaneous Services

Name	Description
IFX_TAPI_EVENT_GET	Get event.
IFX_TAPI_EVENT_ENABLE	Enable event detection.
IFX_TAPI_EVENT_EXT_DTMF	Report DTMF event from application software.
IFX_TAPI_EVENT_EXT_DTMF_CFG	Configure reporting of DTMF event from application software.
IFX_TAPI_EVENT_DISABLE	Disable event detection.

Table 43 Structure Overview of Miscellaneous Services

Name	Description
IFX_TAPI_EVENT_t	Event structure.
IFX_TAPI_EVENT_DATA_DTMF_t	Information about a DTMF event.
IFX_TAPI_EVENT_DATA_EXT_KEYPAD_t	This structure is used to report a DTMF event to the TAPI from an external software module.
IFX_TAPI_EVENT_DATA_FAX_SIG_t	Information about a fax/modem signal event.
IFX_TAPI_EVENT_DATA_PULSE_t	Information about a pulse event.
IFX_TAPI_EVENT_DATA_RFC2833_t	Information about an RFC2833 event.
IFX_TAPI_EVENT_DATA_TONE_t	Information about a tone generation/detection event.
IFX_TAPI_EVENT_EXT_DTMF_t	External DTMF event structure.
IFX_TAPI_EVENT_EXT_DTMF_CFG_t	External DTMF event configuration structure.

Table 44 Union Overview of Miscellaneous Services

Name	Description
IFX_TAPI_EVENT_DATA_t	Union for the possible events to be reported.

Table 45 Enumerator Overview of Miscellaneous Service

Name	Description
IFX_TAPI_EVENT_ID_t	Event ID.
IFX_TAPI_EVENT_TYPE_t	Event type.

4.1.8.1 IFX_TAPI_EVENT_GET

Description

Read event from TAPI.

Prototype

```
IFX_int32_t ioctl (
        IFX_int32_t fd,
        IFX_TAPI_EVENT_GET,
        IFX_int32_t param );
```

Parameters

Data Type	Name	Description
IFX_int32_t	fd	Device file descriptor.
IFX_int32_t	IFX_TAPI_EVENT_GET	I/O control identifier for this operation.
IFX_int32_t	param	Pointer to a IFX_TAPI_EVENT_t struct.

Return Values

Data Type	Description
IFX_int32_t	The return value can be either of the following: • IFX_SUCCESS 0 • IFX_ERROR -1

4.1.8.2 IFX_TAPI_EVENT_ENABLE

Description

Enable detection of an event.

Prototype

```
IFX_int32_t ioctl (
        IFX_int32_t fd,
        IFX_TAPI_EVENT_ENABLE,
        IFX_int32_t param );
```

Parameters

Data Type	Name	Description
IFX_int32_t	fd	Device file descriptor.
IFX_int32_t	IFX_TAPI_EVENT_ENABLE	I/O control identifier for this operation.
IFX_int32_t	param	Pointer to a IFX_TAPI_EVENT_t struct.

Return Values

Data Type	Description
IFX_int32_t	The return value can be either of the following: • IFX_SUCCESS 0 • IFX_ERROR -1

4.1.8.3 IFX_TAPI_EVENT_EXT_DTMF

Description

This service is used to signal a DTMF event from the application software.

Prototype

```
IFX_int32_t ioctl (
        IFX_int32_t fd,
        IFX_TAPI_EVENT_EXT_DTMF,
        IFX_int32_t param );
```

Parameters

Data Type	Name	Description
IFX_int32_t	fd	Device file descriptor.
IFX_int32_t	IFX_TAPI_EVENT_EXT_DTMF	I/O control identifier for this operation.
IFX_int32_t	param	Pointer to a IFX_TAPI_EVENT_EXT_DTMF_t struct.

Return Values

Data Type	Description
IFX_int32_t	The return value can be either of the following: • IFX_SUCCESS 0 • IFX_ERROR -1

4.1.8.4 IFX_TAPI_EVENT_EXT_DTMF_CFG

Description

Configure the support of external DTMF event signaled to TAPI.

Prototype

```
IFX_int32_t ioctl (
        IFX_int32_t fd,
        IFX_TAPI_EVENT_EXT_DTMF_CFG,
        IFX_int32_t param );
```

Parameters

Data Type	Name	Description
IFX_int32_t	fd	Device file descriptor.
IFX_int32_t	IFX_TAPI_EVENT_EXT_DTMF_CFG	I/O control identifier for this operation.
IFX_int32_t	param	Pointer to a IFX_TAPI_EVENT_EXT_DTMF_CFG_t struct.

Return Values

Data Type	Description
IFX_int32_t	The return value can be either of the following: • IFX_SUCCESS 0 • IFX_ERROR -1

4.1.8.5 IFX_TAPI_EVENT_DISABLE

Description

Disable detection of an event.

Prototype

```
IFX_int32_t ioctl (
        IFX_int32_t fd,
        IFX_TAPI_EVENT_DISABLE,
        IFX_int32_t param );
```

Parameters

Data Type	Name	Description
IFX_int32_t	fd	Device file descriptor.
IFX_int32_t	IFX_TAPI_EVENT_DISABLE	I/O control identifier for this operation.
IFX_int32_t	param	Pointer to a IFX_TAPI_EVENT_t struct.

Return Values

Data Type	Description
IFX_int32_t	The return value can be either of the following: • IFX_SUCCESS 0 • IFX_ERROR -1

4.1.9 Operation Control Service

Table 46 IO-control Overview of Operation Control Service

Name	Description
IFX_TAPI_LEC_PCM_CFG_GET	Get the LEC configuration for PCM.
IFX_TAPI_LEC_PCM_CFG_SET	Set the line echo canceller (LEC) configuration for PCM.
IFX_TAPI_LEC_PHONE_CFG_GET	Get the LEC configuration.
IFX_TAPI_LEC_PHONE_CFG_SET	Set the line echo canceller (LEC) configuration.
IFX_TAPI_LINE_FEED_SET	This service sets the line feeding mode.
IFX_TAPI_LINE_HOOK_STATUS_GET	This service reads the hook status from the driver.
IFX_TAPI_LINE_HOOK_VT_SET	Specifies the time for hook, pulse digit and hook flash validation.
IFX_TAPI_LINE_LEVEL_SET	This service enables or disables a high level path of a phone channel.

Table 46 IO-control Overview of Operation Control Service (cont'd)

Name	Description
IFX_TAPI_PCM_VOLUME_SET	Sets the PCM interface volume settings.
IFX_TAPI_PHONE_VOLUME_SET	Sets the speaker phone and microphone volume settings.
IFX_TAPI_WLEC_PCM_CFG_GET	Get the LEC configuration for PCM.
IFX_TAPI_WLEC_PCM_CFG_SET	Set the line echo canceller (LEC) configuration for PCM.
IFX_TAPI_WLEC_PHONE_CFG_GET	Set the line echo canceller (LEC) configuration.
IFX_TAPI_WLEC_PHONE_CFG_SET	Get the LEC configuration.

Table 47 Structure Overview of Operation Control Service

Name	Description
IFX_TAPI_LEC_CFG_t	Line echo canceller (LEC) configuration.
IFX_TAPI_LINE_HOOK_VT_t	Structure used for validation times of hook, hook flash and pulse dialing.
IFX_TAPI_LINE_VOLUME_t	Phone speaker phone and microphone volume settings used for ioctl IFX_TAPI_PHONE_VOLUME_SET.

Table 48 Enumerator Overview of Operation Control Service

Name	Description
IFX_TAPI_LEC_GAIN_t	LEC Gain Levels.
IFX_TAPI_LEC_NLP_t	LEC NLP (Non Linear Processor) Settings.
IFX_TAPI_LEC_TYPE_t	LEC type selection.

4.1.9.1 IFX_TAPI_LEC_PCM_CFG_GET

Description

Get the LEC configuration for PCM.

Attention: It is strongly recommended to use the new IFX_TAPI_WLEC_* interfaces. This interface is going to be discontinued in a next TAPI release.

Prototype

```
IFX_int32_t ioctl (
        IFX_int32_t fd,
        IFX_TAPI_LEC_PCM_CFG_GET,
        IFX_int32_t param );
```

Parameters

Data Type	Name	Description
IFX_int32_t	fd	File descriptor. It applies to phone channel file descriptors.
IFX_int32_t	IFX_TAPI_LEC_PCM_CFG_GET	I/O control identifier for this operation
IFX_int32_t	param	The parameter points to a IFX_TAPI_LEC_CFG_t structure.

Return Values

Data Type	Description
IFX_int32_t	The return value can be either of the following: • IFX_SUCCESS 0 • IFX_ERROR -1

4.1.9.2 IFX_TAPI_LEC_PCM_CFG_SET

Description

Set the line echo canceller (LEC) configuration for PCM.

Attention: It is strongly recommended to use the new IFX_TAPI_WLEC_* interfaces. This interface is going to be discontinued in a next TAPI release.

Prototype

```
IFX_int32_t ioctl (
        IFX_int32_t fd,
        IFX_TAPI_LEC_PCM_CFG_SET,
        IFX_int32_t param );
```

Parameters

Data Type	Name	Description
IFX_int32_t	fd	File descriptor. It applies to phone channel file descriptors.
IFX_int32_t	IFX_TAPI_LEC_PCM_CFG_SET	I/O control identifier for this operation
IFX_int32_t	param	The parameter points to a IFX_TAPI_LEC_CFG_t structure.

Return Values

Data Type	Description
IFX_int32_t	The return value can be either of the following: • IFX_SUCCESS 0 • IFX_ERROR -1

Example

```
IFX_TAPI_LEC_CFG_t param;
IFX_int32_t fd;

memset(&param, 0, sizeof (IFX_TAPI_LEC_CFG_t));
param.nGainOut = IFX_TAPI_LEC_GAIN_MEDIUM;
param.nGainIn = IFX_TAPI_LEC_GAIN_MEDIUM;
param.nLen = 16; /* 16ms */
param.bNlp = IFX_TAPI_LEC_NLP_DEFAULT;
ioctl(fd, IFX_TAPI_LEC_PCM_CFG_SET, &param);
```

4.1.9.3 IFX_TAPI_LEC_PHONE_CFG_GET

Description

Get the LEC configuration.

Attention: It is strongly recommended to use the new IFX_TAPI_WLEC_* interfaces. This interface is going to be discontinued in a next TAPI release.

Prototype

```
IFX_int32_t ioctl (
        IFX_int32_t fd,
        IFX_TAPI_LEC_PHONE_CFG_GET,
        IFX_int32_t param );
```

Parameters

Data Type	Name	Description
IFX_int32_t	fd	File descriptor. It applies to phone channel file descriptors.
IFX_int32_t	IFX_TAPI_LEC_PHONE_CFG_GET	I/O control identifier for this operation
IFX_int32_t	param	The parameter points to a IFX_TAPI_LEC_CFG_t structure.

Return Values

Data Type	Description
IFX_int32_t	The return value can be either of the following: • IFX_SUCCESS 0 • IFX_ERROR -1

4.1.9.4 IFX_TAPI_LEC_PHONE_CFG_SET

Description

Set the line echo canceller (LEC) configuration.

Attention: It is strongly recommended to use the new IFX_TAPI_WLEC_* interfaces. This interface is going to be discontinued in a next TAPI release.

Prototype

```
IFX_int32_t ioctl (
        IFX_int32_t fd,
        IFX_TAPI_LEC_PHONE_CFG_SET_t,
        IFX_int32_t param );
```

Parameters

Data Type	Name	Description
IFX_int32_t	fd	File descriptor. It applies to phone channel file descriptors.
IFX_int32_t	IFX_TAPI_LEC_PHONE_CFG_SET_t	I/O control identifier for this operation
IFX_int32_t	param	The parameter points to a IFX_TAPI_LEC_CFG_t structure.

Return Values

Data Type	Description
IFX_int32_t	The return value can be either of the following: • IFX_SUCCESS 0 • IFX_ERROR -1

Example

```
IFX_TAPI_LEC_CFG_t param;
IFX_int32_t fd;

memset (&param, 0, sizeof (IFX_TAPI_LEC_CFG_t));
param.nGainOut = IFX_TAPI_LEC_GAIN_MEDIUM;
param.nGainIn = IFX_TAPI_LEC_GAIN_MEDIUM;
param.nLen = 16; /* 16ms */
param.bNlp = IFX_TAPI_LEC_NLP_DEFAULT;
ioctl(fd, IFX_TAPI_LEC_PHONE_CFG_SET, &param);
```

4.1.9.5 IFX_TAPI_LINE_FEED_SET

Description

This service sets the line feeding mode.

Prototype

```
IFX_int32_t ioctl (
        IFX_int32_t fd,
        param );
```

Parameters

Data Type	Name	Description
IFX_int32_t	fd	File descriptor. It is applicable to phone channel file descriptors containing an analog interface.
IFX_int32_t	param	The parameter represents a mode value of IFX_TAPI_LINE_FEED_t.

Return Values

Data Type	Description
IFX_int32_t	The return value can be either of the following: • IFX_SUCCESS 0 • IFX_ERROR -1

Remarks

For all battery switching modes the hardware must be able to support it for example by programming coefficients.

Example

```
/* Set line mode to power down */
ioctl(fd, IFX_TAPI_LINE_FEED_SET, IFX_TAPI_LINE_FEED_DISABLED);
```

4.1.9.6 IFX_TAPI_LINE_HOOK_STATUS_GET

Description

This service reads the hook status from the driver.

Prototype

```
IFX_int32_t ioctl (
        IFX_int32_t fd,
        IFX_TAPI_LINE_HOOK_STATUS_GET,
        IFX_int32_t param );
```

Parameters

Data Type	Name	Description
IFX_int32_t	fd	File descriptor. It is applicable to phone channel file descriptors containing an analog interface.
IFX_int32_t	IFX_TAPI_LINE_HOOK_STATUS_GET	I/O control identifier for this operation
IFX_int32_t	param	The parameter is a pointer to the status. 0: no hook detected 1: hook detected

Return Values

Data Type	Description
IFX_int32_t	The return value can be either of the following: • IFX_SUCCESS 0 • IFX_ERROR -1

Example

```
IFX_TAPI_LEC_CFG_t param;
IFX_int32_t fd;
IFX_int32_t nHook;
```

```
ioctl(fd, IFX_TAPI_LINE_HOOK_STATUS_GET, &nHook);
switch (nHook)
{
   case 0:
         /* On hook */
      break;
   case 1:
         /* Off hook */
      break;
   default :
         /* Unknown state */
      break;
}
```

4.1.9.7 IFX_TAPI_LINE_HOOK_VT_SET

Description

Specifies the time for hook, pulse digit and hook flash validation.

Prototype

```
IFX_int32_t ioctl (
       IFX_int32_t fd,
       IFX_TAPI_LINE_HOOK_VT_SET,
       IFX_int32_t param );
```

Parameters

Data Type	Name	Description
IFX_int32_t	fd	File descriptor. It is applicable to phone channel file descriptors containing an analog interface.
IFX_int32_t	IFX_TAPI_LINE_HOOK_VT_SET	I/O control identifier for this operation
IFX_int32_t	param	The parameter points to a IFX_TAPI_LINE_HOOK_VT_t structure.

Return Values

Data Type	Description
IFX_int32_t	The return value can be either of the following: • IFX_SUCCESS 0 • IFX_ERROR -1

Remarks

The following conditions must be met:

- DIGIT_LOW_TIME min. and max < HOOKFLASH_TIME min. and max
- HOOKFLASH_TIME min. and max < HOOKON_TIME min. and max

Example

```
IFX_TAPI_LINE_HOOK_VT_t param;
IFX_int32_t fd;

memset (&param, 0, sizeof (IFX_TAPI_LINE_HOOK_VT_t));
/* Set pulse dialing */
param.nType = IFX_TAPI_LINE_HOOK_VT_DIGITLOW_TIME;
param.nMinTime = 40;
param.nMaxTime = 60;
ioctl(fd, IFX_TAPI_LINE_HOOK_VT_SET, (IFX_int32_t) &param);
param.nType = IFX_TAPI_LINE_HOOK_VT_DIGITHIGH_TIME;
param.nMinTime = 40;
param.nMaxTime = 60;
ioctl(fd, IFX_TAPI_LINE_HOOK_VT_SET, (IFX_int32_t) &param);
```

4.1.9.8 IFX_TAPI_LINE_LEVEL_SET

Description

This service enables or disables a high level path of a phone channel.

Prototype

```
IFX_int32_t ioctl (
        IFX_int32_t fd,
        IFX_TAPI_LINE_LEVEL_SET,
        IFX_int32_t param );
```

Parameters

Data Type	Name	Description
IFX_int32_t	fd	File descriptor. It is applicable to phone channel file descriptors containing an analog interface.
IFX_int32_t	IFX_TAPI_LINE_LEVEL_SET	I/O control identifier for this operation
IFX_int32_t	param	The parameter represents a boolean value of IFX_TAPI_LINE_LEVEL_t.

Return Values

Data Type	Description
IFX_int32_t	The return value can be either of the following: • IFX_SUCCESS 0 • IFX_ERROR -1

Remarks

This service is intended for phone channels only and must be used in combination with IFX_TAPI_PHONE_VOLUME_SET or IFX_TAPI_PCM_VOLUME_SET to set the max. level or to restore level.

Example

```
IFX_TAPI_LINE_HOOK_VT_t param;
IFX_int32_t fd;

ioctl(fd, IFX_TAPI_LINE_LEVEL_SET, IFX_TAPI_LINE_LEVEL_ENABLE );
/* Play out some high level tones or samples */
ioctl(fd, IFX_TAPI_LINE_LEVEL_SET, IFX_TAPI_LINE_LEVEL_DISABLE);
```

4.1.9.9 IFX_TAPI_PCM_VOLUME_SET

Description

Sets the PCM interface volume settings.

Prototype

```
IFX_int32_t ioctl (
        IFX_int32_t fd,
        IFX_TAPI_PCM_VOLUME_SET,
        IFX_int32_t param );
```

Parameters

Data Type	Name	Description
IFX_int32_t	fd	File descriptor. It is applicable to phone channel file descriptors containing a PCM interface.
IFX_int32_t	IFX_TAPI_PCM_VOLUME_SET	I/O control identifier for this operation
IFX_int32_t	param	The parameter points to a IFX_TAPI_LINE_VOLUME_t structure.

Return Values

Data Type	Description
IFX_int32_t	The return value can be either of the following: • IFX_SUCCESS 0 • IFX_ERROR -1

Example

```
IFX_TAPI_LINE_VOLUME_t param;
IFX_int32_t fd;

memset(&param, 0, sizeof(IFX_TAPI_LINE_VOLUME_t));
param.nGainTx = 24;
param.nGainRx = 0;
ioctl(fd, IFX_TAPI_PCM_VOLUME_SET, &param);
```

4.1.9.10 IFX_TAPI_PHONE_VOLUME_SET

Description

Sets the speaker phone and microphone volume settings.

Prototype

```
IFX_int32_t ioctl (
        IFX_int32_t fd,
        IFX_TAPI_PHONE_VOLUME_SET,
        IFX_int32_t param );
```

Parameters

Data Type	Name	Description
IFX_int32_t	fd	File descriptor. It is applicable to phone channel file descriptors containing an analog interface.
IFX_int32_t	IFX_TAPI_PHONE_VOLUME_SET	I/O control identifier for this operation
IFX_int32_t	param	The parameter points to a IFX_TAPI_LINE_VOLUME_t structure.

Return Values

Data Type	Description
IFX_int32_t	The return value can be either of the following: • IFX_SUCCESS 0 • IFX_ERROR -1

Example

```
IFX_TAPI_LINE_VOLUME_t param;
IFX_int32_t fd;

memset(&param, 0, sizeof(IFX_TAPI_LINE_VOLUME_t));
param.nGainTx = 24;
param.nGainRx = 0;
ioctl(fd, IFX_TAPI_PHONE_VOLUME_SET, &param);
```

4.1.9.11 IFX_TAPI_WLEC_PCM_CFG_GET

Description

Get the LEC configuration for PCM.

Prototype

```
IFX_int32_t ioctl (
        IFX_int32_t fd,
        IFX_TAPI_WLEC_PCM_CFG_GET,
        IFX_int32_t param );
```

Parameters

Data Type	Name	Description
IFX_int32_t	fd	File descriptor. It applies to phone channel file descriptors.
IFX_int32_t	IFX_TAPI_WLEC_PCM_CFG_GET	I/O control identifier for this operation
IFX_int32_t	param	The parameter points to a IFX_TAPI_WLEC_CFG_t structure.

Return Values

Data Type	Description
IFX_int32_t	The return value can be either of the following: • IFX_SUCCESS 0 • IFX_ERROR -1

4.1.9.12 IFX_TAPI_WLEC_PCM_CFG_SET

Description

Set the line echo canceller (LEC) configuration for PCM.

Prototype

```
IFX_int32_t ioctl (
        IFX_int32_t fd,
        IFX_TAPI_WLEC_PCM_CFG_SET,
        IFX_int32_t param );
```

Parameters

Data Type	Name	Description
IFX_int32_t	fd	File descriptor. It applies to phone channel file descriptors.
IFX_int32_t	IFX_TAPI_WLEC_PCM_CFG_SET	I/O control identifier for this operation
IFX_int32_t	param	The parameter points to a IFX_TAPI_WLEC_CFG_t structure.

Return Values

Data Type	Description
IFX_int32_t	The return value can be either of the following: • IFX_SUCCESS 0 • IFX_ERROR -1

4.1.9.13 IFX_TAPI_WLEC_PHONE_CFG_GET

Description

Get the LEC configuration.

Prototype

```
IFX_int32_t ioctl (
        IFX_int32_t fd,
        IFX_TAPI_WLEC_PHONE_CFG_GET,
        IFX_int32_t param );
```

Parameters

Data Type	Name	Description
IFX_int32_t	fd	File descriptor. It applies to phone channel file descriptors.
IFX_int32_t	IFX_TAPI_WLEC_PHONE_CFG_GET	I/O control identifier for this operation
IFX_int32_t	param	The parameter points to a IFX_TAPI_WLEC_CFG_t structure.

Return Values

Data Type	Description
IFX_int32_t	The return value can be either of the following: • IFX_SUCCESS 0 • IFX_ERROR -1

4.1.9.14 IFX_TAPI_WLEC_PHONE_CFG_SET

Description

Set the line echo canceller (LEC) configuration.

Prototype

```
IFX_int32_t ioctl (
        IFX_int32_t fd,
        IFX_TAPI_WLEC_PHONE_CFG_SET_t,
        IFX_int32_t param );
```

Parameters

Data Type	Name	Description
IFX_int32_t	fd	File descriptor. It applies to phone channel file descriptors.

Data Type	Name	Description
IFX_int32_t	IFX_TAPI_WLEC_PHONE_CFG_SET_t	I/O control identifier for this operation
IFX_int32_t	param	The parameter points to a IFX_TAPI_WLEC_CFG_t structure.

Return Values

Data Type	Description
IFX_int32_t	The return value can be either of the following: • IFX_SUCCESS 0 • IFX_ERROR -1

4.1.10 PCM Service

Contains services for PCM configuration. Apply to phone channels except otherwise stated.

Table 49 IO-control Overview of PCM Service

Name	Description
IFX_TAPI_PCM_ACTIVATION_GET	This service gets the activation status of the PCM time slots configured for this channel.
IFX_TAPI_PCM_ACTIVATION_SET	This service activate/deactivates the PCM time slots configured for this channel.
IFX_TAPI_PCM_CFG_GET	This service gets the configuration of the PCM interface.
IFX_TAPI_PCM_CFG_SET	This service sets the configuration of the PCM interface.

4.1.10.1 IFX_TAPI_PCM_ACTIVATION_GET

Description

This service gets the activation status of the PCM time slots configured for this channel.

Prototype

```
IFX_int32_t ioctl (
        IFX_int32_t fd,
        IFX_TAPI_PCM_ACTIVATION_GET,
        IFX_int32_t param );
```

Parameters

Data Type	Name	Description
IFX_int32_t	fd	File descriptor. It is applicable to phone channel file descriptors containing a PCM interface.
IFX_int32_t	IFX_TAPI_PCM_ACTIVATION_GET	I/O control identifier for this operation
IFX_int32_t	param	The parameter points to an integer which returns the status 0: The time slot is deactive 1: The time slot is active

Return Values

Data Type	Description
IFX_int32_t	The return value can be either of the following: • IFX_SUCCESS 0 • IFX_ERROR -1

Example

```
IFX_TAPI_LINE_VOLUME_t param;
IFX_int32_t fd;
IFX_return_t ret;
IFX_int32_t bAct;
```

```
ret = ioctl(fd, IFX_TAPI_PCM_ACTIVATION_GET, &bAct);
if ((IFX_SUCCESS == ret) && (1 == bAct))
{
   printf("Activated\n");
}
```

4.1.10.2 IFX_TAPI_PCM_ACTIVATION_SET

Description

This service activate / deactivates the PCM time slots configured for this channel.

Prototype

```
IFX_int32_t ioctl (
        IFX_int32_t fd,
        IFX_TAPI_PCM_ACTIVATION_SET,
        IFX_int32_t param );
```

Parameters

Data Type	Name	Description
IFX_int32_t	fd	File descriptor. It is applicable to phone channel file descriptors containing a PCM interface.
IFX_int32_t	IFX_TAPI_PCM_ACTIVATION_SET	I/O control identifier for this operation
IFX_int32_t	param	The parameter defines the activation status 0 deactivate the time slot 1 activate the time slot

Return Values

Data Type	Description
IFX_int32_t	The return value can be either of the following: • IFX_SUCCESS 0 • IFX_ERROR -1

Example

```
/* Activate the PCM timeslot */
ioctl(fd, IFX_TAPI_PCM_ACTIVATION_SET, 1);
```

4.1.10.3 IFX_TAPI_PCM_CFG_GET

Description

This service gets the configuration of the PCM interface.

Prototype

```
IFX_int32_t ioctl (
        IFX_int32_t fd,
```

```
        IFX_TAPI_PCM_CFG_GET,
        IFX_int32_t param );
```

Parameters

Data Type	Name	Description
IFX_int32_t	fd	File descriptor. It is applicable to phone channel file descriptors containing a PCM interface.
IFX_int32_t	IFX_TAPI_PCM_CFG_GET	I/O control identifier for this operation
IFX_int32_t	param	The parameter points to a IFX_TAPI_PCM_CFG_t structure.

Return Values

Data Type	Description
IFX_int32_t	The return value can be either of the following: • IFX_SUCCESS 0 • IFX_ERROR -1

Example

```
IFX_TAPI_PCM_CFG_t Pcm;
IFX_int32_t fd;

memset(&Pcm, 0, sizeof(IFX_TAPI_PCM_CFG_t));
ioctl(fd, IFX_TAPI_PCM_CFG_GET, &Pcm);
```

4.1.10.4 IFX_TAPI_PCM_CFG_SET

Description

This service sets the configuration of the PCM interface.

Prototype

```
IFX_int32_t ioctl (
        IFX_int32_t fd,
        IFX_TAPI_PCM_CFG_SET,
        IFX_int32_t param );
```

Parameters

Data Type	Name	Description
IFX_int32_t	fd	File descriptor. It is applicable to phone channel file descriptors containing a PCM interface.
IFX_int32_t	IFX_TAPI_PCM_CFG_SET	I/O control identifier for this operation
IFX_int32_t	param	The parameter points to a IFX_TAPI_PCM_CFG_t structure.

Return Values

Data Type	Description
IFX_int32_t	The return value can be either of the following: • IFX_SUCCESS 0 • IFX_ERROR -1

Remarks

The parameter rate must be set to the PCM rate, which is applied to the device, otherwise error is returned.

Example

```
IFX_TAPI_PCM_CFG_t Pcm;
IFX_int32_t fd;

memset(&Pcm, 0, sizeof(IFX_TAPI_PCM_CFG_t));
Pcm.nTimeslotRX = 5;
Pcm.nTimeslotTX = 5;
Pcm.nHighway = 0;
/* 16 bit resolution */
Pcm.nResolution = IFX_TAPI_PCM_RES_LINEAR_16BIT;
/* 2048 kHz sample rate */
Pcm.nRate = 2048;
/* Configure PCM interface */
ioctl(fd, IFX_TAPI_PCM_CFG_SET, &Pcm);
```

4.1.11 Power Ringing Service

Table 50 IO-control Overview of Ringing Service

Name	Description
IFX_TAPI_RING_CADENCE_HR_SET	This service sets the high resolution ring cadence for the non-blocking ringing services.
IFX_TAPI_RING_CADENCE_SET	This service sets the ring cadence for the non-blocking ringing services.
IFX_TAPI_RING_CFG_GET	This service gets the ring configuration for the non-blocking ringing services.
IFX_TAPI_RING_CFG_SET	This service sets the ring configuration for the non-blocking ringing services.
IFX_TAPI_RING_START	This service starts the non-blocking ringing on the phone line and sends out CID.
IFX_TAPI_RING_STOP	This service stops non-blocking ringing on the phone line which was started before with service IFX_TAPI_RING_START.

Table 51 Structure Overview of Ringing Service

Name	Description
IFX_TAPI_RING_CADENCE_t	Structure for ring cadence used in IFX_TAPI_RING_CADENCE_HR_SET.
IFX_TAPI_RING_CFG_t	Ringing configuration structure used for ioctl IFX_TAPI_RING_CFG_SET.

4.1.11.1 IFX_TAPI_RING_CADENCE_HR_SET

Description

This service sets the high resolution ring cadence for the non-blocking ringing services.

Prototype

```
IFX_int32_t ioctl (
        IFX_int32_t fd,
        IFX_TAPI_RING_CADENCE_HR_SET,
        IFX_int32_t param );
```

Parameters

Data Type	Name	Description
IFX_int32_t	fd	File descriptor. It is applicable to phone channel file descriptors containing an analog interface.
IFX_int32_t	IFX_TAPI_RING_CADENCE_HR_SET	I/O control identifier for this operation
IFX_int32_t	param	The parameter points to a IFX_TAPI_RING_CADENCE_t structure.

Return Values

Data Type	Description
IFX_int32_t	The return value can be either of the following: • IFX_SUCCESS 0 • IFX_ERROR -1 The number of ring bursts can be counted via exception PSTN line is ringing of service IFX_TAPI_EXCEPTION_GET.

Example

```
/* Pattern of 3 sec. ring and 1 sec. pause */
char data[10] = { 0xFF, 0xFF, 0xFF, 0xFF, 0xFF,
                  0xFF, 0xFF, 0xF0, 0x00, 0x00};
/* Initial 1 sec. ring and 600 ms non ringing signal before ringing */
char initial[4] = { 0xFF,0xFF,0xF0,0x0 };
IFX_TAPI_RING_CADENCE_t Cadence;
IFX_int32_t fd;

memset(&Cadence, 0, sizeof(IFX_TAPI_RING_CADENCE_t));
memcpy(&Cadence.data[0], data, sizeof(data));
Cadence.nr = sizeof (data) * 8;
/* Set size in bits */
memcpy(&Cadence.initial[0], initial, sizeof(initial));
Cadence.initialNr = 4 * 8;
/* Set the cadence sequence */
ioctl(fd, IFX_TAPI_RING_CADENCE_HR_SET, &Cadence);
```

4.1.11.2 IFX_TAPI_RING_CADENCE_SET

Description

This service sets the ring cadence for the non-blocking ringing services.

Prototype

```
IFX_int32_t ioctl (
       IFX_int32_t fd,
       IFX_TAPI_RING_CADENCE_SET,
       IFX_int32_t param );
```

Parameters

Data Type	Name	Description
IFX_int32_t	fd	File descriptor. It is applicable to phone channel file descriptors containing an analog interface.

Data Type	Name	Description
IFX_int32_t	IFX_TAPI_RING_CADENCE_SET	I/O control identifier for this operation
IFX_int32_t	param	The parameter defines the cadence. This value contains the encoded cadence sequence. One bit represents ring cadence voltage for 0.5 sec.

Return Values

Data Type	Description
IFX_int32_t	The return value can be either of the following: • IFX_SUCCESS 0 • IFX_ERROR -1

Remarks

The number of ring bursts can be counted via exception PSTN line is ringing of service IFX_TAPI_EXCEPTION_GET.

Example

```
/* pattern of 3 sec. ring and 1 sec. pause : 1111 1100 1111 1100 ... */
IFX_uint32_t nCadence = 0xFCFCFCFC;
IFX_int32_t fd;

/* set the cadence sequence */
ioctl(fd, IFX_TAPI_RING_CADENCE_SET, nCadence);
```

4.1.11.3 IFX_TAPI_RING_CFG_GET

Description

This service gets the ring configuration for the non-blocking ringing services.

Prototype

```
IFX_int32_t ioctl (
        IFX_int32_t fd,
        IFX_TAPI_RING_CFG_GET,
        IFX_int32_t param );
```

Parameters

Data Type	Name	Description
IFX_int32_t	fd	File descriptor. It is applicable to phone channel file descriptors containing an analog interface.
IFX_int32_t	IFX_TAPI_RING_CFG_GET	I/O control identifier for this operation
IFX_int32_t	param	The parameter points to a IFX_TAPI_RING_CFG_t structure

Return Values

Data Type	Description
IFX_int32_t	The return value can be either of the following: • IFX_SUCCESS 0 • IFX_ERROR -1

4.1.11.4 IFX_TAPI_RING_CFG_SET

Description

This service sets the ring configuration for the non-blocking ringing services.

Prototype

```
IFX_int32_t ioctl (
        IFX_int32_t fd,
        IFX_TAPI_RING_CFG_SET,
        IFX_int32_t param );
```

Parameters

Data Type	Name	Description
IFX_int32_t	fd	File descriptor. It is applicable to phone channel file descriptors containing an analog interface.
IFX_int32_t	IFX_TAPI_RING_CFG_SET	I/O control identifier for this operation
IFX_int32_t	param	The parameter points to a IFX_TAPI_RING_CFG_t structure

Return Values

Data Type	Description
IFX_int32_t	The return value can be either of the following: • IFX_SUCCESS 0 • IFX_ERROR -1

Remarks

The configuration has to be set before ringing starts using the interface IFX_TAPI_RING_START.

Example

```
IFX_TAPI_RING_CFG_t param;
memset (&param, 0,  sizeof (IFX_TAPI_RING_CFG_t));
param.mode = 0;
param.submode = 1;
ret = ioctl(fd, IFX_TAPI_RING_CFG_SET, &param)
```

4.1.11.5 IFX_TAPI_RING_START

Description

This service starts the non-blocking ringing on the phone line using the preconfigured ring cadence.

Prototype

```
IFX_int32_t ioctl (
        IFX_int32_t fd,
        IFX_TAPI_CID_TX_SEQ_START,
        IFX_int32_t param );
```

Parameters

Data Type	Name	Description
IFX_int32_t	fd	File descriptor. It is applicable to phone channel file descriptors containing an analog interface.
IFX_int32_t	IFX_TAPI_CID_TX_SEQ_START	I/O control identifier for this operation
IFX_int32_t	param	Parameter is ignored.

Return Values

Data Type	Description
IFX_int32_t	The return value can be either of the following: • IFX_SUCCESS 0 • IFX_ERROR -1

Remarks

This interface does not implement CID.

4.1.11.6 IFX_TAPI_RING_STOP

Description

This service stops non-blocking ringing on the phone line which was started before with service IFX_TAPI_RING_START.

Prototype

```
IFX_int32_t ioctl (
        IFX_int32_t fd,
        IFX_TAPI_RING_STOP,
        IFX_int32_t param );
```

Parameters

Data Type	Name	Description
IFX_int32_t	fd	File descriptor. It is applicable to phone channel file descriptors containing an analog interface.
IFX_int32_t	IFX_TAPI_RING_STOP	I/O control identifier for this operation
IFX_int32_t	param	This interface expects no parameter. It should be set to 0.

Return Values

Data Type	Description
IFX_int32_t	The return value can be either of the following: • **IFX_SUCCESS** 0 • **IFX_ERROR** -1

Example

```
ioctl(fd, IFX_TAPI_RING_STOP, 0);
```

4.1.12 Signal Detection Service

Table 52 IO-control Overview of Signal Detection Service

Name	Description
IFX_TAPI_SIG_DETECT_DISABLE	Disables the signal detection for Fax or modem signals.
IFX_TAPI_SIG_DETECT_ENABLE	Enables the signal detection for Fax or modem signals.
IFX_TAPI_TONE_CPTD_START	Start the call progress tone detection based on a previously defined simple tone.
IFX_TAPI_TONE_CPTD_STOP	Stops the call progress tone detection.

Table 53 Structure Overview of Signal Detection Service

Name	Description
IFX_TAPI_SIG_DETECTION_t	Structure used for enable and disable signal detection.

Table 54 Enumerator Overview of Signal Detection Service

Name	Description
IFX_TAPI_SIG_t	List the tone detection options.
IFX_TAPI_TONE_CPTD_DIRECTION_t	Specifies the CPT signal for CPT detection.

4.1.12.1 IFX_TAPI_SIG_DETECT_DISABLE

Description

Disables the signal detection for Fax or modem signals.

Attention: It is strongly recommended to use the new event reporting interfaces, see Chapter 4.1.8. This interface is going to be discontinued in a next TAPI release.

Prototype

```
IFX_int32_t ioctl (
        IFX_int32_t fd,
        IFX_TAPI_SIG_DETECT_DISABLE,
        IFX_int32_t param );
```

Parameters

Data Type	Name	Description
IFX_int32_t	fd	File descriptor. It is applicable to data channel file descriptors.
IFX_int32_t	IFX_TAPI_SIG_DETECT_DISABLE	I/O control identifier for this operation
IFX_int32_t	param	The parameter points to a IFX_TAPI_SIG_DETECTION_t structure.

Return Values

Data Type	Description
IFX_int32_t	The return value can be either of the following: • IFX_SUCCESS 0 • IFX_ERROR -1

4.1.12.2 IFX_TAPI_SIG_DETECT_ENABLE

Description

Enables the signal detection for Fax or modem signals.

Attention: It is strongly recommended to use the new event reporting interfaces, see Chapter 4.1.8. This interface is going to be discontinued in a next TAPI release.

Prototype

```
IFX_int32_t ioctl (
        IFX_int32_t fd,
        IFX_TAPI_SIG_DETECT_ENABLE,
        IFX_int32_t param );
```

Parameters

Data Type	Name	Description
IFX_int32_t	fd	File descriptor. It is applicable to data channel file descriptors.
IFX_int32_t	IFX_TAPI_SIG_DETECT_ENABLE	I/O control identifier for this operation
IFX_int32_t	param	The parameter points to a IFX_TAPI_SIG_DETECTION_t structure.

Return Values

Data Type	Description
IFX_int32_t	The return value can be either of the following: • IFX_SUCCESS 0 • IFX_ERROR -1

Remarks

Check the capability list which detection service is available and how many detectors are available Signal events are reported via exception. The information is queried with IFX_TAPI_CH_STATUS_GET. In case of AM detected the driver may automatically switches to modem coefficients after the ending of CED Not every combination is possible. It depends on the under lying device and firmware.

Example

```
IFX_TAPI_SIG_DETECTION_t detect_sig;
IFX_TAPI_CH_STATUS_t param;
IFX_int32_t fd;
```

```
memset(&param, 0, sizeof(IFX_TAPI_CH_STATUS_t));
memset(&detect_sig, 0, sizeof(IFX_TAPI_SIG_DETECTION_t));

detect_sig.sig = IFX_TAPI_SIG_CEDTX | IFX_TAPI_SIG_PHASEREVRX;
ioctl(fd, IFX_TAPI_SIG_DETECT_ENABLE, &detect_sig);
ioctl(fd, IFX_TAPI_CH_STATUS_GET, &param);
if (param.signal & IFX_TAPI_SIG_CED)
{
   printf("Got CED from receive.\n");
}
```

4.1.12.3 IFX_TAPI_TONE_CPTD_START

Description

Start the call progress tone detection based on a previously defined simple tone.

Prototype

```
IFX_int32_t ioctl (
       IFX_int32_t fd,
       IFX_TAPI_TONE_CPTD_START,
       IFX_int32_t param );
```

Parameters

Data Type	Name	Description
IFX_int32_t	fd	File descriptor. It is applicable to data channel file descriptors.
IFX_int32_t	IFX_TAPI_TONE_CPTD_START	I/O control identifier for this operation
IFX_int32_t	param	The parameter points to a IFX_TAPI_TONE_CPTD_t structure.

Return Values

Data Type	Description
IFX_int32_t	The return value can be either of the following: • IFX_SUCCESS 0 • IFX_ERROR -1

Example

```
IFX_TAPI_TONE_CPTD_t param;
IFX_int32_t fd;

memset(&param, 0, sizeof (IFX_TAPI_TONE_CPTD_t));
param.tone = 74;
param.signal = IFX_TAPI_TONE_CPTD_DIRECTION_TX;
```

```
ioctl (fd, IFX_TAPI_TONE_CPTD_START, &param);
```

4.1.12.4 IFX_TAPI_TONE_CPTD_STOP

Description

Stops the call progress tone detection.

Prototype

```
IFX_int32_t ioctl (
        IFX_int32_t fd,
        IFX_TAPI_TONE_CPTD_STOP,
        IFX_int32_t param );
```

Parameters

Data Type	Name	Description
IFX_int32_t	fd	File descriptor. It is applicable to data channel file descriptors.
IFX_int32_t	IFX_TAPI_TONE_CPTD_STOP	I/O control identifier for this operation
IFX_int32_t	param	This interface expects no parameter. It should be set to 0.

Return Values

Data Type	Description
IFX_int32_t	The return value can be either of the following: • IFX_SUCCESS 0 • IFX_ERROR -1

4.1.13 Test Services

4.1.13.1 IFX_TAPI_TEST_HOOKGEN

Prototype

```
void ioctl (
       IFX_int32_t fd,
       IFX_TAPI_TEST_LOOP,
       IFX_int32_t param );
```

Parameters

Data Type	Name	Description
IFX_int32_t	fd	File descriptor. It is applicable to data channel file descriptors.
IFX_int32_t	IFX_TAPI_TEST_LOOP	I/O control identifier for this operation
IFX_int32_t	param	The parameter defines off hook or on hook generation 0_D **IFX_FALSE** Generate on hook 1_D **IFX_TRUE** Generate off hook

Remarks

After switching the hook state, the hook event gets to the hook state machine for validation. Depending on the timing of calling this interface also hook flash and pulse dialing can be verified. The example shows the generation of a flash hook with a timing of 100 ms. The flash hook can be queried afterwards by the ioctl IFX_TAPI_CH_STATUS_GET.

Example

```
 // generate on hook
ret = ioctl(fd, IFX_TAPI_TEST_HOOKGEN, 0);
 // generate off hook for 100 ms
ret = ioctl(fd, IFX_TAPI_TEST_HOOKGEN, 1);
sleep (100);
ret = ioctl(fd, IFX_TAPI_TEST_HOOKGEN, 0);
```

4.1.13.2 IFX_TAPI_TEST_LOOP

Prototype

```
void ioctl (
       IFX_int32_t fd,
       IFX_TAPI_TEST_LOOP,
       IFX_int32_t param );
```

Parameters

Data Type	Name	Description
IFX_int32_t	fd	File descriptor. It is applicable to data channel file descriptors.
IFX_int32_t	IFX_TAPI_TEST_LOOP	I/O control identifier for this operation
IFX_int32_t	param	The parameter points to a IFX_TAPI_TEST_LOOP_t structure.

Remarks

Enables a local test loop in the analog part. The digital voice data is transparently looped back to the network without affecting the downstream tranmission. The reception of local voice (upstream) is disabled. That means, that voice applied to the local phone is not beeing processed, but data sent to the phone can still be heard.

Example

```
IFX_TAPI_TEST_LOOP_t parm;
memset (&param, 0,  sizeof (IFX_TAPI_TEST_LOOP_t));
param.bAnalog = 1;
ret = ioctl(fd, IFX_TAPI_TEST_LOOP, &param);
```

4.1.14 Tone Control Service

All tone services apply to data channels file descriptors except otherwise stated.

Table 55 IO-control Overview of Tone Control Service

Name	Description
IFX_TAPI_TONE_LOCAL_PLAY	Plays a tone, the tone has to be predefined in the tone table.
IFX_TAPI_TONE_NET_PLAY	Plays a tone to the network side, the tone has to be predefined in the tone table.
IFX_TAPI_TONE_STATUS_GET	This service gets the tone playing state.
IFX_TAPI_TONE_STOP	Stop playing of an already started tone.
IFX_TAPI_TONE_TABLE_CFG_SET	Define a tone and adds it to the tone table.

Table 56 Constant Overview of Tone Control Service

Name	Description
IFX_TAPI_TONE_SRC_DEFAULT	Tone is played out on default source.
IFX_TAPI_TONE_SRC_DSP	Tone is played out on DSP, default, if available.
IFX_TAPI_TONE_SRC_TG	Tone is played out on local tone generator in the analog part of the device.
IFX_TAPI_TONE_STEPS_MAX	Maximum tone generation steps, also called cadences.

Table 57 Structure Overview of Tone Control Service

Name	Description
IFX_TAPI_TONE_COMPOSED_t	Structure for definition of composed tones.
IFX_TAPI_TONE_SIMPLE_t	Structure for definition of simple tones.

Table 58 Union Overview of Tone Control Service

Name	Description
IFX_TAPI_TONE_t	Tone descriptor.

Table 59 Enumerator Overview of Tone Control Service

Name	Description
IFX_TAPI_TONE_COMPOSED_t	This chapter define the structure of the tone descriptor.
IFX_TAPI_TONE_FREQ_t	Frequency setting for a cadence step.
IFX_TAPI_TONE_FREQ_t	
IFX_TAPI_TONE_GROUP_t	Tone grouping.
IFX_TAPI_TONE_MODULATION_t	Modulation setting for a cadence step.
IFX_TAPI_TONE_SIMPLE_t	
IFX_TAPI_TONE_t	This chapter define a union for the tone descriptor.
IFX_TAPI_TONE_TG_t	Defines the tone generator usage.
IFX_TAPI_TONE_TYPE_t	Tone types.

4.1.14.1 IFX_TAPI_TONE_LOCAL_PLAY

Description

Start/stop generation of a tone towards local port, the parameter (if greater than 0) gives the tone table index of the tone to be played.

If the parameter is equal to zero, stop current tone generation.

Prototype

```
IFX_int32_t ioctl (
        IFX_int32_t fd,
        IFX_TAPI_TONE_LOCAL_PLAY,
        IFX_int32_t param );
```

Parameters

Data Type	Name	Description
IFX_int32_t	fd	File descriptor. It is applicable to channel file descriptors.
IFX_int32_t	IFX_TAPI_TONE_LOCAL_PLAY	I/O control identifier for this operation
IFX_int32_t	param	The parameter is the index of the tone to the predefined tone table (range 1 - 31) or custom tones added previously (index 32 - 255). Index 0 means tone stop. Using the upper bits modify the default tone playing source:

Return Values

Data Type	Description
IFX_int32_t	The return value can be either of the following: • IFX_SUCCESS 0 • IFX_ERROR -1

Remarks

This can be a predefined, simple or a composed tone. The tone codes are assigned previously on system start. Index 1 - 31 is predefined by the driver and covers the original TAPI.

Example

```
IFX_TAPI_LINE_VOLUME_t param;
IFX_int32_t fd;

/* Play tone index 34 */
ioctl(fd, IFX_TAPI_TONE_LOCAL_PLAY, 34);
```

4.1.14.2 IFX_TAPI_TONE_NET_PLAY

Description

Start/stop generation of a tone towards network, the parameter (if greater than 0) gives the tone table index of the tone to be played.

If the parameter is equal to zero, stop current tone generation.

Prototype

```
IFX_int32_t ioctl (
        IFX_int32_t fd,
        IFX_TAPI_TONE_NET_PLAY,
        IFX_int32_t param );
```

Parameters

Data Type	Name	Description
IFX_int32_t	fd	File descriptor. It is applicable to data channel file descriptors.
IFX_int32_t	IFX_TAPI_TONE_NET_PLAY	I/O control identifier for this operation
IFX_int32_t	param	The parameter is the index of the tone to the predefined tone table (range 1 - 31) or custom tones added previously (index 32 - 255). Index 0 means tone stop. Using the upper bits modify the default tone playing source.

Return Values

Data Type	Description
IFX_int32_t	The return value can be either of the following: • IFX_SUCCESS 0 • IFX_ERROR -1

Remarks

This can be a predefined, simple or a composed tone. The tone codes are assigned previously on system start. Index 1 - 31 is predefined by the driver and covers the original TAPI.

Example

```
/* Play tone index 34 */
ioctl(fd, IFX_TAPI_TONE_NET_PLAY, 34);
```

4.1.14.3 IFX_TAPI_TONE_STATUS_GET

Description

This service gets the tone playing state.

It applies to data channel file descriptors.

Prototype

```
IFX_int32_t ioctl (
        IFX_int32_t fd,
        IFX_TAPI_TONE_STATUS_GET,
        IFX_int32_t* param );
```

Parameters

Data Type	Name	Description
IFX_int32_t	fd	File descriptor
IFX_int32_t	IFX_TAPI_TONE_STATUS_GET	I/O control identifier for this operation
IFX_int32_t*	param	The parameter points to the tone state • 0: no tone is played • 1: tone is played (tone is within on-time) • 2: silence (tone is within off-time)

Return Values

Data Type	Description
IFX_int32_t	The return value can be either of the following: • IFX_SUCCESS 0 • IFX_ERROR -1

Example

```
      int  nToneState;
      /* get the tone state */
     ret = ioctl(fd, IFX_TAPI_TONE_STATUS_GET, &nToneState);
```

4.1.14.4 IFX_TAPI_TONE_STOP

Description

Stop tone generation.

Prototype

```
IFX_int32_t ioctl (
        IFX_int32_t fd,
        IFX_TAPI_TONE_TONE_STOP,
        IFX_int32_t param );
```

Parameters

Data Type	Name	Description
IFX_int32_t	fd	File descriptor. It is applicable to channel file descriptors.
IFX_int32_t	IFX_TAPI_TONE_TONE_STOP	I/O control identifier for this operation
IFX_int32_t	param	Tone table index of the tone to be stopped.

Return Values

Data Type	Description
IFX_int32_t	The return value can be either of the following: • IFX_SUCCESS 0 • IFX_ERROR -1

Example

```
/* Stop playing tone index 34 */
ioctl(fd, IFX_TAPI_TONE_STOP, 34);
```

4.1.14.5 IFX_TAPI_TONE_TABLE_CFG_SET

Description

Configures a tone based on simple or composed tones. The tone is also added to the tone table.

Prototype

```
IFX_int32_t ioctl (
        IFX_int32_t fd,
        IFX_TAPI_TONE_TABLE_CFG_SET,
        IFX_int32_t param );
```

Parameters

Data Type	Name	Description
IFX_int32_t	fd	File descriptor. It is applicable to channel file descriptors.
IFX_int32_t	IFX_TAPI_TONE_TABLE_CFG_SET	I/O control identifier for this operation
IFX_int32_t	param	The parameter points to a IFX_TAPI_TONE_t structure.

Return Values

Data Type	Description
IFX_int32_t	The return value can be either of the following: • IFX_SUCCESS 0 • IFX_ERROR -1

Remarks

A simple tone specifies a tone sequence composed of several single frequency tones or dual frequency tones. The sequence can be transmitted only one time, several times or until the transmission is stopped by client. This interface can add a simple tone to the internal table with maximum 222 entries, starting from 32. At least one cadence must be defined otherwise this interface returns an error. The tone table provides all tone frequencies in 5 Hz steps with a 5% tolerance and are defined in RFC 2833. For composed tones the loop count of each simple tone must be different from 0.

Example

```
IFX_TAPI_CH_INIT_t tapi;
IFX_TAPI_TONE_t tone;
IFX_int32_t fd;

/* Open channel file descriptor for channel 0 */
fd = open("/dev/vin11", O_RDWR, 0x644);

memset(&tapi,0, sizeof(IFX_TAPI_CH_INIT_t));
memset(&tone,0, sizeof(IFX_TAPI_TONE_t));
ioctl(fd, IFX_TAPI_CH_INIT, (IFX_int32_t) &tapi);
tone.simple.format = IFX_TAPI_TONE_TYPE_SIMPLE;
tone.simple.index = 71;
tone.simple.freqA = 480;
tone.simple.freqB = 620;
tone.simple.levelA = -300;
tone.simple.cadence[0] = 2000;
tone.simple.cadence[1] = 2000;
tone.simple.frequencies[0] = IFX_TAPI_TONE_FREQA | IFX_TAPI_TONE_FREQB;
tone.simple.loop = 2;
tone.simple.pause = 200;
ioctl(fd, IFX_TAPI_TONE_TABLE_CFG_SET, (IFX_int32_t) &tone);
memset(&tone, 0, sizeof (IFX_TAPI_TONE_t));
tone.composed.format = IFX_TAPI_TONE_TYPE_COMPOSED;
tone.composed.index = 100;
tone.composed.count = 2;
tone.composed.tones[0] = 71;
tone.composed.tones[1] = 71;
ioctl(fd, IFX_TAPI_TONE_TABLE_CFG_SET, (IFX_int32_t) &tone);
/* Now start playing the simple tone */
ioctl(fd, IFX_TAPI_TONE_LOCAL_PLAY, (IFX_int32_t) 71);
/* Stop playing tone */
ioctl(fd, IFX_TAPI_TONE_STOP, (IFX_int32_t) 71);

/* Close all open fds */
close(fd);
```

4.1.15 Audio Channel Control

All tone services apply to device file descriptors except otherwise stated.

Table 60 IO-control Overview of Tone Control Service

Name	Description
IFX_TAPI_AUDIO_MODE_SET	Selection of audio mode.
IFX_TAPI_AUDIO_MUTE_SET	Mute the audio channel.
IFX_TAPI_AUDIO_ICA_SET	Enable and disable in-call announcement.
IFX_TAPI_AUDIO_RING_START	Start ringing on the audio channel.
IFX_TAPI_AUDIO_RING_STOP	Stop ringing on the audio channel.
IFX_TAPI_AUDIO_ROOM_TYPE_SET	Choose the room type.
IFX_TAPI_AUDIO_VOLUME_SET	Choose the volume level for the audio channel.

Table 61 Enumerator Overview of Tone Control Service

Name	Description
IFX_TAPI_AUDIO_MODE_t	Audio mode.
IFX_TAPI_AUDIO_ROOM_TYPE_t	Room type.

4.1.15.1 IFX_TAPI_AUDIO_MODE_SET

Description

Selection of audio mode.

Prototype

```
IFX_int32_t ioctl (
        IFX_int32_t fd,
        IFX_TAPI_AUDIO_MODE_SET,
        IFX_int32_t param );
```

Parameters

Data Type	Name	Description
IFX_int32_t	fd	File descriptor. It is applicable to device file descriptors.
IFX_int32_t	IFX_TAPI_AUDIO_MODE_SET	I/O control identifier for this operation
IFX_int32_t	param	The parameter is the audio mode to be set:

Return Values

Data Type	Description
IFX_int32_t	The return value can be either of the following: • IFX_SUCCESS 0 • IFX_ERROR -1

4.1.15.2 IFX_TAPI_AUDIO_MUTE_SET

Description

Mute the audio channel.

Prototype

```
IFX_int32_t ioctl (
        IFX_int32_t fd,
        IFX_TAPI_AUDIO_MUTE_SET,
        IFX_operation_t param );
```

Parameters

Data Type	Name	Description
IFX_int32_t	fd	File descriptor. It is applicable to data channel file descriptors.
IFX_int32_t	IFX_TAPI_AUDIO_MUTE_SET	I/O control identifier for this operation
IFX_operation_t	param	The parameter specifies whether to enable or disabling mute.

Return Values

Data Type	Description
IFX_int32_t	The return value can be either of the following: • IFX_SUCCESS 0 • IFX_ERROR -1

4.1.15.3 IFX_TAPI_AUDIO_ICA_SET

Description

Enable and disable in-call announcement.

Prototype

```
IFX_int32_t ioctl (
        IFX_int32_t fd,
        IFX_TAPI_AUDIO_ICA_SET,
        IFX_int32_t param );
```

Parameters

Data Type	Name	Description
IFX_int32_t	fd	File descriptor. It is applicable to data channel file descriptors.

Data Type	Name	Description
IFX_int32_t	IFX_TAPI_AUDIO_ICA_SET	I/O control identifier for this operation
IFX_int32_t	param	The parameter specifies whether to enable or disabling in-call announcement, parameter to be selected from IFX_operation_t.

Return Values

Data Type	Description
IFX_int32_t	The return value can be either of the following: • IFX_SUCCESS 0 • IFX_ERROR -1

4.1.15.4 IFX_TAPI_AUDIO_RING_START

Description

Start ringing on the audio channel.

Prototype

```
IFX_int32_t ioctl (
        IFX_int32_t fd,
        IFX_TAPI_AUDIO_RING_START,
        IFX_int32_t param );
```

Parameters

Data Type	Name	Description
IFX_int32_t	fd	File descriptor. It is applicable to data channel file descriptors.
IFX_int32_t	IFX_TAPI_AUDIO_RING_START	I/O control identifier for this operation
IFX_int32_t	param	Tone table index containing the ring cadence.

Return Values

Data Type	Description
IFX_int32_t	The return value can be either of the following: • IFX_SUCCESS 0 • IFX_ERROR -1

4.1.15.5 IFX_TAPI_AUDIO_RING_STOP

Description

Stop ringing on the audio channel.

Prototype

```
IFX_int32_t ioctl (
```

```
        IFX_int32_t fd,
        IFX_TAPI_AUDIO_RING_STOP,
        IFX_int32_t param );
```

Parameters

Data Type	Name	Description
IFX_int32_t	fd	File descriptor. It is applicable to data channel file descriptors.
IFX_int32_t	IFX_TAPI_AUDIO_RING_STOP	I/O control identifier for this operation
IFX_int32_t	param	Not required.

Return Values

Data Type	Description
IFX_int32_t	The return value can be either of the following: • IFX_SUCCESS 0 • IFX_ERROR -1

4.1.15.6 IFX_TAPI_AUDIO_ROOM_TYPE_SET

Description

Choose the room type.

Prototype

```
IFX_int32_t ioctl (
        IFX_int32_t fd,
        IFX_TAPI_AUDIO_ROOM_TYPE_SET,
        IFX_int32_t param );
```

Parameters

Data Type	Name	Description
IFX_int32_t	fd	File descriptor. It is applicable to data channel file descriptors.
IFX_int32_t	IFX_TAPI_AUDIO_ROOM_TYPE_SET	I/O control identifier for this operation
IFX_int32_t	param	The parameter specifies the room type, to be selected from IFX_TAPI_AUDIO_ROOM_TYPE_t.

Return Values

Data Type	Description
IFX_int32_t	The return value can be either of the following: • IFX_SUCCESS 0 • IFX_ERROR -1

4.1.15.7 IFX_TAPI_AUDIO_VOLUME_SET

Description

Choose the volume level for the audio channel.

Prototype

```
IFX_int32_t ioctl (
        IFX_int32_t fd,
        IFX_TAPI_AUDIO_VOLUME_SET,
        IFX_int32_t param );
```

Parameters

Data Type	Name	Description
IFX_int32_t	fd	File descriptor. It is applicable to data channel file descriptors.
IFX_int32_t	IFX_TAPI_AUDIO_VOLUME_SET	I/O control identifier for this operation
IFX_int32_t	param	Volume level.

Return Values

Data Type	Description
IFX_int32_t	The return value can be either of the following: • IFX_SUCCESS 0 • IFX_ERROR -1

4.2 Type Definition Reference

This chapter contains the basic type definitions, reference of data types and structures of all modules.

4.2.1 Basic Type Definitions

This section describes the basic type definitions such as:

- **IFX_return_t**
- **IFX_boolean_t**
- **IFX_uint8_t**
- **IFX_int8_t**
- **IFX_uint32_t**
- **IFX_int32_t**
- **IFX_uint16_t**
- **IFX_int16_t**
- **IFX_char_t**
- **IFX_void_t**
- **IFX_float_t**
- **IFX_operation_t**

4.2.1.1 IFX_return_t

Description

All the APIs return a Success or a Failure based on their execution Status. The return code is set to IFX_ERROR only if an error occurs, otherwise its value is IFX_SUCCESS.

Prototype

```
typedef enum
{
        IFX_ERROR = -1,
        IFX_SUCCESS = 0
} IFX_return_t;
```

Parameters

Name	Value	Description
IFX_ERROR	-1_D	Operation failed.
IFX_SUCCESS	0_D	Operation was successful.

4.2.1.2 IFX_boolean_t

Description

Definition for true and false.

Prototype

```
typedef enum
{
        IFX_FALSE = 0,
        IFX_TRUE = 1
} IFX_boolean_t;
```

Parameters

Name	Value	Description
IFX_FALSE	0_D	False.
IFX_TRUE	1_D	True.

4.2.1.3 IFX_uint8_t

Prototype

```
typedef unsigned char IFX_uint8_t;
```

Parameters

Data Type	Name	Description
unsigned char	IFX_uint8_t	This is the unsigned char 8-bit datatype.

4.2.1.4 IFX_int8_t

Prototype

```
typedef char IFX_int8_t;
```

Parameters

Data Type	Name	Description
char	IFX_int8_t	This is the char 8-bit datatype.

4.2.1.5 IFX_uint32_t

Prototype

```
typedef unsigned int IFX_uint32_t;
```

Parameters

Data Type	Name	Description
unsigned int	IFX_uint32_t	This is the unsigned int 32-bit datatype.

4.2.1.6 IFX_int32_t

Prototype

```
typedef int IFX_int32_t;
```

Parameters

Data Type	Name	Description
int	IFX_int32_t	This is the int 32-bit datatype.

4.2.1.7 IFX_uint16_t

Prototype

```
typedef unsigned short IFX_uint16_t;
```

Parameters

Data Type	Name	Description
unsigned short	IFX_uint16_t	This is the unsigned short int 16-bit datatype.

4.2.1.8 IFX_int16_t

Prototype

```
typedef short IFX_int16_t;
```

Parameters

Data Type	Name	Description
short	IFX_int16_t	This is the short int 16-bit datatype.

4.2.1.9 IFX_char_t

Prototype

```
typedef char IFX_char_t;
```

Parameters

Data Type	Name	Description
char	IFX_char_t	This is the char 8-bit datatype.

4.2.1.10 IFX_void_t

Prototype

```
typedef void IFX_void_t;
```

Parameters

Data Type	Name	Description
void	IFX_void_t	This is a void datatype.

4.2.1.11 IFX_float_t

Prototype

```
typedef float IFX_float_t;
```

Parameters

Data Type	Name	Description
float	IFX_float_t	This is a float datatype.

4.2.1.12 IFX_operation_t

Description

Definition of enable and disable operation.

Prototype

```
typedef enum
{
        IFX_DISABLE = 0,
        IFX_ENABLE = 1
} IFX_operation_t;
```

Parameters

Name	Value	Description
IFX_DISABLE	0_D	Disable.
IFX_ENABLE	1_D	Enable.

4.2.2 IO-control Reference

This chapter contains the IO-control reference.

Table 62 IO-control Overview of TAPI Interfaces

Name	Description
IFX_TAPI_AUDIO_MODE_SET	Selection of audio mode.
IFX_TAPI_AUDIO_MUTE_SET	Mute the audio channel.
IFX_TAPI_AUDIO_ICA_SET	Enable and disable in-call announcement.
IFX_TAPI_AUDIO_RING_START	Start ringing on the audio channel.
IFX_TAPI_AUDIO_RING_STOP	Stop ringing on the audio channel.
IFX_TAPI_AUDIO_ROOM_TYPE_SET	Choose the room type.
IFX_TAPI_AUDIO_VOLUME_SET	Choose the volume level for the audio channel.
IFX_TAPI_CAP_CHECK	This service checks if a specific capability is supported.
IFX_TAPI_CAP_LIST	This service returns the capability lists.
IFX_TAPI_CAP_NR	This service returns the number of capabilities.
IFX_TAPI_CH_INIT	This service sets the default initialization of device and hardware.
IFX_TAPI_CH_STATUS_GET	Query status information about the phone line.
IFX_TAPI_CID_CFG_SET	Configures the CID transmitter.
IFX_TAPI_CID_RX_DATA_GET	Reads CID Data collected since Caller ID receiver was started.
IFX_TAPI_CID_RX_START	Setup the CID Receiver to start receiving CID Data.
IFX_TAPI_CID_RX_STATUS_GET	Retrieves the current status information of the CID Receiver.

Table 62 IO-control Overview of TAPI Interfaces (cont'd)

Name	Description
IFX_TAPI_CID_RX_STOP	Stop the CID Receiver.
IFX_TAPI_CID_TX_INFO_START	This interfaces transmits CID message.
IFX_TAPI_DEBUG_REPORT_SET	Set the report levels if the driver is compiled with ENABLE_TRACE.
IFX_TAPI_DEC_LEVEL_GET	This service returns the level of the most recently played signal.
IFX_TAPI_DEC_START	Starts the playout of data.
IFX_TAPI_DEC_STOP	Stops the playout of data.
IFX_TAPI_DEC_TYPE_SET	Select a codec for the data channel playout.
IFX_TAPI_DEC_VOLUME_SET	Sets the playout volume.
IFX_TAPI_ENC_FRAME_LEN_GET	This service gets the frame length for the audio packets.
IFX_TAPI_ENC_FRAME_LEN_SET	This service sets the frame length for the audio packets.
IFX_TAPI_ENC_LEVEL_SET	This service returns the level of the most recently recorded signal.
IFX_TAPI_ENC_START	Start recording on the data channel.
IFX_TAPI_ENC_STOP	Stop recording (generating packets) on this channel.
IFX_TAPI_ENC_TYPE_SET	Select a codec for the data channel.
IFX_TAPI_ENC_VAD_CFG_SET	Configures the voice activity detection and silence handling Voice Activity Detection (VAD) is a feature that allows the codec to determine when to send voice data or silence data.
IFX_TAPI_ENC_VOLUME_SET	Sets the recording volume.
IFX_TAPI_EVENT_GET	Get event.
IFX_TAPI_EVENT_ENABLE	Enable event detection.
IFX_TAPI_EVENT_EXT_DTMF	Report DTMF event from application software.
IFX_TAPI_EVENT_EXT_DTMF_CFG	Configure reporting of DTMF event from application software.
IFX_TAPI_EVENT_DISABLE	Disable event detection.
IFX_TAPI_EXCEPTION_GET	This service returns the current exception status.
IFX_TAPI_EXCEPTION_MASK	This service masks the reporting of the exceptions.
IFX_TAPI_JB_CFG_SET	Configures the jitter buffer.
IFX_TAPI_JB_STATISTICS_GET	Reads out jitter buffer statistics.
IFX_TAPI_JB_STATISTICS_RESET	Resets the jitter buffer statistics.
IFX_TAPI_LEC_PCM_CFG_GET	Get the LEC configuration for PCM.
IFX_TAPI_LEC_PCM_CFG_SET	Set the line echo canceller (LEC) configuration for PCM.
IFX_TAPI_LEC_PHONE_CFG_GET	Get the LEC configuration.
IFX_TAPI_LEC_PHONE_CFG_SET	Set the line echo canceller (LEC) configuration.
IFX_TAPI_LINE_FEED_SET	This service sets the line feeding mode.
IFX_TAPI_LINE_HOOK_STATUS_GET	This service reads the hook status from the driver.
IFX_TAPI_LINE_HOOK_VT_SET	Specifies the time for hook, pulse digit and hook flash validation.
IFX_TAPI_LINE_LEVEL_SET	This service enables or disables a high level path of a phone channel.
IFX_TAPI_MAP_DATA_ADD	This interface adds the data channel to an analog phone device.
IFX_TAPI_MAP_DATA_REMOVE	This interface removes a data channel from an analog phone device.

Table 62 IO-control Overview of TAPI Interfaces (cont'd)

Name	Description
IFX_TAPI_MAP_PCM_ADD	This interface adds the PCM channel to an analog phone device.
IFX_TAPI_MAP_PCM_REMOVE	This interface removes the PCM channel to an analog phone device.
IFX_TAPI_MAP_PHONE_ADD	This interface adds the phone channel to another analog phone channel.
IFX_TAPI_MAP_PHONE_REMOVE	This interface removes a phone channel from an analog phone device.
IFX_TAPI_METER_CFG_SET	This service sets the characteristic for the metering service.
IFX_TAPI_METER_START	This service starts the metering.
IFX_TAPI_METER_STOP	This service stops the metering.
IFX_TAPI_PCM_ACTIVATION_GET	This service gets the activation status of the PCM time slots configured for this channel.
IFX_TAPI_PCM_ACTIVATION_SET	This service activate/deactivates the PCM time slots configured for this channel.
IFX_TAPI_PCM_CFG_GET	This service gets the configuration of the PCM interface.
IFX_TAPI_PCM_CFG_SET	This service sets the configuration of the PCM interface.
IFX_TAPI_PCM_VOLUME_SET	Sets the PCM interface volume settings.
IFX_TAPI_PHONE_VOLUME_SET	Sets the speaker phone and microphone volume settings.
IFX_TAPI_PKT_AAL_CFG_SET	This interface configures AAL fields for a new connection.
IFX_TAPI_PKT_AAL_PROFILE_SET	AAL profile configuration.
IFX_TAPI_PKT_EV_OOB_SET	This service controls the DTMF sending mode.
IFX_TAPI_PKT_RTCP_STATISTICS_GET	Retrieves RTCP statistics.
IFX_TAPI_PKT_RTCP_STATISTICS_RESET	Resets the RTCP statistics.
IFX_TAPI_PKT_RTP_CFG_SET	This interface configures RTP and RTCP fields for a new connection.
IFX_TAPI_PKT_RTP_PT_CFG_SET	Change the payload type table.
IFX_TAPI_PULSE_ASCII_GET	This service reads the ASCII character representing the pulse digit out of the receive buffer.
IFX_TAPI_PULSE_GET	This service reads the dial pulse digit out of the receive buffer.
IFX_TAPI_PULSE_READY	This service checks if a pulse digit was received.
IFX_TAPI_RING_CADENCE_HR_SET	This service sets the high resolution ring cadence for the non-blocking ringing services.
IFX_TAPI_RING_CADENCE_SET	This service sets the ring cadence for the non-blocking ringing services.
IFX_TAPI_RING_CFG_GET	This service gets the ring configuration for the non-blocking ringing services.
IFX_TAPI_RING_CFG_SET	This service sets the ring configuration for the non-blocking ringing services.
IFX_TAPI_RING_START	This service starts the non-blocking ringing on the phone line and sends out CID.
IFX_TAPI_RING_STOP	This service stops non-blocking ringing on the phone line which was started before with service IFX_TAPI_RING_START.

Table 62 IO-control Overview of TAPI Interfaces (cont'd)

Name	Description
IFX_TAPI_SIG_DETECT_DISABLE	Disables the signal detection for Fax or modem signals.
IFX_TAPI_SIG_DETECT_ENABLE	Enables the signal detection for Fax or modem signals.
IFX_TAPI_T38_DEMOD_START	This service configures and enables the demodulator for a T.38 fax session.
IFX_TAPI_T38_MOD_START	This service configures and enables the modulator for a T.38 fax session.
IFX_TAPI_T38_STATUS_GET	This service provides the T.38 fax status on query.
IFX_TAPI_T38_STOP	This service disables the T.38 fax data pump and activates the voice path again.
IFX_TAPI_TONE_CPTD_START	Start the call progress tone detection based on a previously defined simple tone.
IFX_TAPI_TONE_CPTD_STOP	Stops the call progress tone detection.
IFX_TAPI_TONE_DTMF_ASCII_GET	This service reads the ASCII character representing the DTMF digit out of the receive buffer.
IFX_TAPI_TONE_DTMF_GET	This service reads the DTMF digit out of the internal receive buffer.
IFX_TAPI_TONE_DTMF_READY_GET	This service checks if a DTMF digit was received.
IFX_TAPI_TONE_LOCAL_PLAY	Plays a tone, which is defined before.
IFX_TAPI_TONE_NET_PLAY	Plays a tone to the network side, which is defined before.
IFX_TAPI_TONE_STATUS_GET	This service gets the tone playing state.
IFX_TAPI_TONE_STOP	Stops playback of the current tone (call progress tone), or cadence.
IFX_TAPI_TONE_TABLE_CFG_SET	Configures a tone based on simple or composed tones.
IFX_TAPI_VERSION_CHECK	Check the supported TAPI interface version.
IFX_TAPI_VERSION_GET	Retrieves the TAPI version string.
IFX_TAPI_WLEC_PCM_CFG_GET	Get the LEC configuration for PCM.
IFX_TAPI_WLEC_PCM_CFG_SET	Set the line echo canceller (LEC) configuration for PCM.
IFX_TAPI_WLEC_PHONE_CFG_GET	Set the line echo canceller (LEC) configuration.
IFX_TAPI_WLEC_PHONE_CFG_SET	Get the LEC configuration.

4.2.3 Constant Reference

This chapter contains the constant reference.

Table 63 Constant Reference for TAPI Interfaces

Name and Description	Value
IFX_TAPI_CID_MSG_LEN_MAX Max number of octets for a CID message element.	50_D
IFX_TAPI_CID_RX_FIFO_SIZE CID RX FIFO Size	10_D
IFX_TAPI_CID_SIZE_MAX Max length for a CID message.	128_D
IFX_TAPI_DTMF_FIFO_SIZE FIFO size for the detected DTMF digits	40_D

Table 63 Constant Reference for TAPI Interfaces (cont'd)

Name and Description	Value
IFX_TAPI_ENC_TYPE_MAX Max number of supported vocoders for the payload table.	00000000_D
IFX_TAPI_EVENT_ALL_CHANNELS Max number of supported vocoders for the payload table.	$FFFF_H$
IFX_TAPI_IOC_MAGIC Magic number for ioctls	0000000_B
IFX_TAPI_LEC_LEN_MAX Maximum length of LEC tail	16_D
IFX_TAPI_LEC_LEN_MIN Minimum length of LEC tail	4_D
IFX_TAPI_LINE_VOLUME_HIGH High volume gain, +24 dB	24_D
IFX_TAPI_LINE_VOLUME_LOW Low volume gain, -24 dB	-24_D
IFX_TAPI_LINE_VOLUME_MEDIUM Medium line volume gain, 0 dB	0_D
IFX_TAPI_LINE_VOLUME_OFF Line volume mute	FF_H
IFX_TAPI_PULSE_FIFO_SIZE FIFO size for the detected pulse digits	20_D
IFX_TAPI_RING_CADENCE_MAX Max number of ring cadences.	40_D
IFX_TAPI_TONE_INDEX_MAX Maximum index value for user defined tones	255_D
IFX_TAPI_TONE_INDEX_MIN Minimum index value for user defined tones	32_D
IFX_TAPI_TONE_SIMPLE_MAX Max number of simple tones part of a composed tone.	7_D
IFX_TAPI_TONE_SRC_DEFAULT Tone is played out on default source.	0_D
IFX_TAPI_TONE_SRC_DSP Tone is played out on DSP, default, if available.	4000_H
IFX_TAPI_TONE_SRC_TG Tone is played out on local tone generator in the analog part of the device.	8000_H
IFX_TAPI_TONE_STEPS_MAX Maximum tone generation steps, also called cadences.	6_D

4.2.4 Structure Reference

This chapter contains the Structure reference.

Table 64 Structure Overview of TAPI Interfaces

Name	Description
IFX_TAPI_CAP_t	Capability structure.
IFX_TAPI_CH_INIT_t	TAPI initialization structure used for IFX_TAPI_CH_INIT.

Table 64 Structure Overview of TAPI Interfaces (cont'd)

Name	Description
IFX_TAPI_CH_STATUS_t	Structure used for IFX_TAPI_CH_STATUS_GET to query the phone status of one or more channels.
IFX_TAPI_CID_ABS_REASON_t	Structure containing CID configuration for ETSI standard using DTMF transmission.
IFX_TAPI_CID_CFG_t	Structure containing CID configuration possibilities.
IFX_TAPI_CID_DTMF_CFG_t	Structure containing the configuration information for DTMF CID.
IFX_TAPI_CID_FSK_CFG_t	Structure containing the configuration information for FSK transmitter and receiver.
IFX_TAPI_CID_MSG_DATE_t	Structure for element types (IFX_TAPI_CID_SERVICE_TYPE_t) with date and time.
IFX_TAPI_CID_MSG_STRING_t	Structure for element types (IFX_TAPI_CID_SERVICE_TYPE_t) with dynamic length (line numbers or names).
IFX_TAPI_CID_MSG_t	Structure containing the CID message type and content.
IFX_TAPI_CID_MSG_TRANSPARENT_t	Structure for element types (IFX_TAPI_CID_SERVICE_TYPE_t) with dynamic length (line numbers or names).
IFX_TAPI_CID_MSG_VALUE_t	Structure for element types (IFX_TAPI_CID_SERVICE_TYPE_t) with one value (length 1).
IFX_TAPI_CID_RX_DATA_t	Structure for Caller ID receiver data
IFX_TAPI_CID_RX_STATUS_t	Structure for Caller ID receiver status.
IFX_TAPI_CID_STD_ETSI_DTMF_t	Structure containing CID configuration for ETSI standard using DTMF transmission.
IFX_TAPI_CID_STD_ETSI_FSK_t	Structure containing CID configuration for ETSI standard using FSK transmission.
IFX_TAPI_CID_STD_NTT_t	Structure containing CID configuration for NTT standard.
IFX_TAPI_CID_STD_SIN_t	Structure containing CID configuration for BT SIN227 standard.
IFX_TAPI_CID_STD_TELCORDIA_t	Structure containing CID configuration for Telcordia standard.
IFX_TAPI_CID_TIMING_t	Structure containing the timing for CID transmission.
IFX_TAPI_DWLD_t	TAPI download structure.
IFX_TAPI_EXCEPTION_BITS_t	Exception bits.
IFX_TAPI_EVENT_t	This structure is reported by an "EVENT_GET" ioctl. For event masking "EVENT_MASK" reusing IFX_TAPI_EVENT_t should be used.
IFX_TAPI_EVENT_DATA_DTMF_t	Information about a DTMF event.
IFX_TAPI_EVENT_DATA_EXT_KEYPAD_t	This structure is used to report a DTMF event to the TAPI from an external software module.
IFX_TAPI_EVENT_DATA_FAX_SIG_t	Information about a fax/modem signal event.
IFX_TAPI_EVENT_DATA_PULSE_t	Information about a pulse event.
IFX_TAPI_EVENT_DATA_RFC2833_t	Information about a tone generation/detection event.
IFX_TAPI_EVENT_DATA_TONE_t	Information about a tone generation/detection event.
IFX_TAPI_EVENT_EXT_DTMF_t	This structure is used to report a DTMF event to the TAPI from an external software module.
IFX_TAPI_EVENT_EXT_DTMF_CFG_t	This structure is used to configure support for the reporting of external DTMF events.

Table 64 Structure Overview of TAPI Interfaces (cont'd)

Name	Description
IFX_TAPI_JB_CFG_t	Structure for jitter buffer configuration used by IFX_TAPI_JB_CFG_SET.
IFX_TAPI_JB_STATISTICS_t	Structure for Jitter Buffer statistics used by ioctl IFX_TAPI_JB_STATISTICS_GET.
IFX_TAPI_LEC_CFG_t	Line echo canceller (LEC) configuration.
IFX_TAPI_LINE_HOOK_VT_t	Structure used for validation times of hook, hook flash and pulse dialing.
IFX_TAPI_LINE_VOLUME_t	Phone speaker phone and microphone volume settings used for ioctl IFX_TAPI_PHONE_VOLUME_SET.
IFX_TAPI_MAP_DATA_t	Phone channel mapping structure used for IFX_TAPI_MAP_DATA_ADD and IFX_TAPI_MAP_DATA_REMOVE.
IFX_TAPI_MAP_PCM_t	Phone channel mapping structure used for IFX_TAPI_MAP_PCM_ADD and IFX_TAPI_MAP_PCM_REMOVE.
IFX_TAPI_MAP_PHONE_t	Phone channel mapping structure used for IFX_TAPI_MAP_PHONE_ADD and IFX_TAPI_MAP_PHONE_REMOVE.
IFX_TAPI_METER_CFG_t	Structure for metering configuration.
IFX_TAPI_PCK_AAL_CFG_t	Structure used for ioctl IFX_TAPI_PKT_AAL_CFG_SET
IFX_TAPI_PCK_AAL_PROFILE_t	AAL profile setup structure used for IFX_TAPI_PKT_AAL_PROFILE_SET.
IFX_TAPI_PCM_CFG_t	Structure for PCM configuration.
IFX_TAPI_PKT_RTCP_STATISTICS_t	Structure for RTCP Statistics.
IFX_TAPI_PKT_RTP_CFG_t	Structure for RTP Configuration.
IFX_TAPI_PKT_RTP_PT_CFG_t	Structure for RTP payload configuration.
IFX_TAPI_RING_CADENCE_t	Structure for ring cadence used in IFX_TAPI_RING_CADENCE_HR_SET.
IFX_TAPI_RING_CFG_t	Ringing configuration structure used for ioctl IFX_TAPI_RING_CFG_SET.
IFX_TAPI_SIG_DETECTION_t	Structure used for enable and disable signal detection.
IFX_TAPI_T38_DEMOD_DATA_t	Structure to setup the demodulator for T.38 fax and used for IFX_TAPI_T38_DEMOD_START.
IFX_TAPI_T38_MOD_DATA_t	Structure to setup the modulator for T.38 fax and used for IFX_TAPI_T38_MOD_START.
IFX_TAPI_T38_STATUS_t	Structure to read the T.38 fax status and used for IFX_TAPI_T38_STATUS_GET.
IFX_TAPI_TEST_LOOP_t	Structure for switching loops for testing.
IFX_TAPI_TONE_COMPOSED_t	Structure for definition of composed tones.
IFX_TAPI_TONE_CPTD_t	Structure used for IFX_TAPI_TONE_CPTD_START and IFX_TAPI_TONE_CPTD_STOP.
IFX_TAPI_TONE_SIMPLE_t	Structure for definition of simple tones.
IFX_TAPI_WLEC_CFG_t	Line echo canceller (LEC) configuration.
IFX_TAPI_VERSION_t	Structure used for the TAPI version support check.

4.2.4.1 IFX_TAPI_CAP_t

Description

Capability structure.

Prototype

```
typedef struct
{
        IFX_char_t desc[80];
        IFX_TAPI_CAP_TYPE_t captype;
        IFX_int32_t cap;
        IFX_int32_t handle;
} IFX_TAPI_CAP_t;
```

Parameters

Data Type	Name	Description
IFX_char_t	desc[80]	Description of the capability.
IFX_TAPI_CAP_TYPE_t	captype	Defines the capability type.
IFX_int32_t	cap	Defines if, what or how many is available. The definition of cap depends on the type. See captype
IFX_int32_t	handle	The number of this capability.

4.2.4.2 IFX_TAPI_CH_INIT_t

Description

TAPI initialization structure used for IFX_TAPI_CH_INIT.

Prototype

```
typedef struct
{
        IFX_uint8_t nMode;
        IFX_uint8_t nCountry;
        IFX_void_t * pProc;
} IFX_TAPI_CH_INIT_t;
```

Parameters

Data Type	Name	Description
IFX_uint8_t	nMode	Channel initialization, to select type of data channel if available (dependent on device type) 0_D **IFX_TAPI_CH_INIT_MODE_DEFAULT** Reserved. This value is supported for compatibility reasons. 1_D **IFX_TAPI_CH_INIT_MODE_VOICE_CODER** Analog channel connected to a packet coder (data channel) with features for signal detection and generation. 2_D **IFX_TAPI_CH_INIT_MODE_PCM_DSP** Analog channel connected to PCM with features for signal detection and generation. 3_D **IFX_TAPI_CH_INIT_MODE_PCM_PHONE** Analog channel connected to PCM without features for signal detection and generation. 255_D **IFX_TAPI_CH_INIT_MODE_NONE** no configuration is done. This is used only for testing
IFX_uint8_t	nCountry	Reserved.
IFX_void_t *	pProc	Pointer to the low-level device initialization struct (for example VMMC_IO_INIT for INCA-IP2). For details please refer to the device specific driver documentation

4.2.4.3 IFX_TAPI_CH_STATUS_t

Description

Structure used for **IFX_TAPI_CH_STATUS_GET** to query the phone status of one or more channels.

Prototype

```
typedef struct
{
        IFX_char_t channels;
        IFX_char_t hook;
        IFX_char_t dialing;
        IFX_char_t digit;
        IFX_int32_t line;
        IFX_uint32_t signal;
         unknown;
        IFX_uint32_t device;
        IFX_uint32_t event;
        IFX_uint32_t error;
```

```
} IFX_TAPI_CH_STATUS_t;
```

Parameters

Data Type	Name	Description
IFX_char_t	channels	Size of the list. If 0 only the status of this channel is queried. If channels not equal to 0 all channels starting with 0 till the value of channels are retrieved. As the return value it defines the channel number
IFX_char_t	hook	Lists the hook status events. 1_H **IFX_TAPI_LINE_HOOK_STATUS_HOOK** Hook event detected (on- or off-hook) 2_H **IFX_TAPI_CH_STATUS_FLASH** Flash hook event detected
IFX_char_t	dialing	Lists the dial status events. 1_H **IFX_TAPI_CH_STATUS_DTMF** DTMF digit detected 2_H **IFX_TAPI_CH_STATUS_PULSE** Pulse digit detected
IFX_char_t	digit	Contains the digit in case of a dialing event.
IFX_int32_t	line	Signals the line status and fault conditions. 1_H **IFX_TAPI_LINE_STATUS_RINGING** PSTN line is ringing 2_H **IFX_TAPI_LINE_STATUS_RINGFINISHED** ringing finished on PSTN line 4_H **IFX_TAPI_LINE_STATUS_FAX** Fax status change 10_H **IFX_TAPI_LINE_STATUS_GNDKEY** Ground key detected 20_H **IFX_TAPI_LINE_STATUS_GNDKEYHIGH** Ground key high detected 40_H **IFX_TAPI_LINE_STATUS_OTEMP** Overtemperature detected 80_H **IFX_TAPI_LINE_STATUS_GNPKEYPOL** Ground key polarity detected 100_H **IFX_TAPI_LINE_STATUS_GR909RES** Cid Receiver event occured 200_H **IFX_TAPI_LINE_STATUS_CIDRX** Caller ID received 400_H **IFX_TAPI_LINE_STATUS_FEEDLOWBATT** Event occurs when the line mode may be switched to IFX_TAPI_LINE_FEED_NORMAL_LOW to save power

Data Type	Name	Description
IFX_uint32_t	signal	Signals the detection of events. The signals/events detected are reported in this field and represented according to the definition of IFX_TAPI_SIG_t. Following description is taken from the definition of IFX_TAPI_SIG_t. Please check it out for further information. 0_H **IFX_TAPI_SIG_NONE** no signal detected 1_H **IFX_TAPI_SIG_DISRX** V.21 Preamble Fax Tone, Digital Identification Signal (DIS), receive path 2_H **IFX_TAPI_SIG_DISTX** V.21 Preamble Fax Tone, Digital Identification Signal (DIS), transmit path 4_H **IFX_TAPI_SIG_DIS** V.21 Preamble Fax Tone in all path, Digital Identification Signal (DIS) 8_H **IFX_TAPI_SIG_CEDRX** V.25 2100 Hz (CED) Modem/Fax Tone, receive path 10_H **IFX_TAPI_SIG_CEDTX** V.25 2100 Hz (CED) Modem/Fax Tone, transmit path 20_H **IFX_TAPI_SIG_CED** V.25 2100 Hz (CED) Modem/Fax Tone in all paths 40_H **IFX_TAPI_SIG_CNGFAXRX** CNG Fax Calling Tone (1100 Hz) receive path 80_H **IFX_TAPI_SIG_CNGFAXTX** CNG Fax Calling Tone (1100 Hz) transmit path 100_H **IFX_TAPI_SIG_CNGFAX** CNG Fax Calling Tone (1100 Hz) in all paths 200_H **IFX_TAPI_SIG_CNGMODRX** CNG Modem Calling Tone (1300 Hz) receive path 400_H **IFX_TAPI_SIG_CNGMODTX** CNG Modem Calling Tone (1300 Hz) transmit path 800_H **IFX_TAPI_SIG_CNGMOD** CNG Modem Calling Tone (1300 Hz) in all paths 1000_H **IFX_TAPI_SIG_PHASEREVRX** Phase reversal detection receive path 2000_H **IFX_TAPI_SIG_PHASEREVTX** Phase reversal detection transmit path 4000_H **IFX_TAPI_SIG_PHASEREV** Phase reversal detection in all paths 8000_H **IFX_TAPI_SIG_AMRX** Amplitude modulation receive path 10000_H **IFX_TAPI_SIG_AMTX** Amplitude modulation transmit path

Data Type	Name	Description
		20000_H**IFX_TAPI_SIG_AM** Amplitude modulation. 40000_H **IFX_TAPI_SIG_TONEHOLDING_ENDRX** Modem tone holding signal stopped receive path 80000_H **IFX_TAPI_SIG_TONEHOLDING_ENDTX** Modem tone holding signal stopped transmit path 100000_H **IFX_TAPI_SIG_TONEHOLDING_END** Modem tone holding signal stopped all paths 200000_H**IFX_TAPI_SIG_CEDENDRX** End of signal CED detection receive path 400000_H**IFX_TAPI_SIG_CEDENDTX** End of signal CED detection transmit path 800000_H**IFX_TAPI_SIG_CEDEND** End of signal CED detection. 1000000_H**IFX_TAPI_SIG_CPTD** Call progress tone detected.
IFX_uint32_t	device	Device specific status information
IFX_uint32_t	event	RFC 2833 event payload information. This field contains the last event (**IFX_TAPI_PKT_EV_NUM_t**), that was received and played out by the device.
IFX_uint32_t	error	Runtime error of type **IFX_TAPI_RUNTIME_ERROR_t**

4.2.4.4 IFX_TAPI_CID_ABS_REASON_t

Description

Structure containing CID configuration for ETSI standard using DTMF transmission.

Prototype

```
typedef struct
{
        IFX_char_t nLength;
        IFX_char_t unavailable[TAPI_CID_MAX_MSG_LEN];
        IFX_char_t private[TAPI_CID_MAX_MSG_LEN];
} IFX_TAPI_CID_ABS_REASON_t;
```

Parameters

Data Type	Name	Description
IFX_char_t	nLength	Length of the coded strings.
IFX_char_t	unavailable[TAPI_CID_MAX_MSG_LEN]	String representing code for unavailable/unknown CLI. Default “00”.
IFX_char_t	private[TAPI_CID_MAX_MSG_LEN]	String representing code for private/withheld CLI. Default “01”.

4.2.4.5 IFX_TAPI_CID_CFG_t

Description

Structure containing CID configuration possibilities.

Prototype

```
typedef struct
{
        IFX_int32_t nStandard;
        IFX_TAPI_CID_STD_TYPE_t cfg;
} IFX_TAPI_CID_CFG_t;
```

Parameters

Data Type	Name	Description
IFX_int32_t	nStandard	Standard used (enumerated in IFX_TAPI_CID_STD_t). Default IFX_TAPI_CID_STD_TELCORDIA.
IFX_TAPI_CID_STD_TYPE_t	cfg	Union of the different standards. Default IFX_TAPI_CID_STD_TELCORDIA_t.

4.2.4.6 IFX_TAPI_CID_DTMF_CFG_t

Description

Structure containing the configuration information for DTMF CID.

Prototype

```
typedef struct
{
        IFX_char_t startTone;
        IFX_char_t stopTone;
        IFX_char_t infoSstartTone;
        IFX_char_t startTone;
        IFX_int32_t digitTime;
        IFX_int32_t interDigitTime;
} IFX_TAPI_CID_DTMF_CFG_t;
```

Parameters

Data Type	Name	Description
IFX_char_t	startTone	Tone id for starting tone. Default is DTMF 'A'.
IFX_char_t	stopTone	Tone id for stop tone. Default is DTMF 'C'.
IFX_char_t	infoSstartTone	Tone id for starting information tone. Default is DTMF 'B'.
IFX_char_t	startTone	Tone id for starting redirection tone. Default is DTMF 'D'.
IFX_int32_t	digitTime	Time for DTMF digit duration. Default is 50 ms.
IFX_int32_t	interDigitTime	Time between DTMF digits in ms. Default is 50 ms.

4.2.4.7 IFX_TAPI_CID_FSK_CFG_t

Description

Structure containing the configuration information for FSK transmitter and receiver.

Prototype

```
typedef struct
{
        IFX_int32_t levelTX;
        IFX_int32_t levelRX;
        IFX_uint32_t seizureTX;
        IFX_uint32_t seizureRX;
        IFX_uint32_t markTXOnhook;
        IFX_uint32_t markTXOffhook;
        IFX_uint32_t markRXOnhook;
        IFX_uint32_t markRXOffhook;
} IFX_TAPI_CID_FSK_CFG_t;
```

Parameters

Data Type	Name	Description
IFX_int32_t	levelTX	Signal level for FSK transmission in 0.1 dB steps. Default -140 (-14 dB).
IFX_int32_t	levelRX	Minimum signal level for FSK reception in 0.1 dB steps. Default -150 (-15 dB).
IFX_uint32_t	seizureTX	Number of seizure bits for FSK transmission. This field is relevant only for on-hook transmission. Default value NTT standard: 0 bits. Other standards: 300 bits.
IFX_uint32_t	seizureRX	Minimum number of seizure bits for FSK reception. This field is relevant only for on-hook transmission. Default value NTT standard: 0 bits. Other standards: 200 bits.

Data Type	Name	Description
IFX_uint32_t	markTXOnhook	Number of mark bits for on-hook FSK transmission. Default value NTT standard: 72 bits. Other standards: 180 bits.
IFX_uint32_t	markTXOffhook	Number of mark bits for off-hook FSK transmission. Default value NTT standard: 72 bits. Other standards: 80 bits.
IFX_uint32_t	markRXOnhook	Minimum number of mark bits for on-hook FSK reception. Default value NTT standard: 50 bits. Other standards: 150 bits.
IFX_uint32_t	markRXOffhook	Minimum number of mark bits for off-hook FSK reception. Default value NTT standard: 50 bits. Other standards: 55 bits.

4.2.4.8 IFX_TAPI_CID_MSG_DATE_t

Description

Structure for element types (IFX_TAPI_CID_SERVICE_TYPE_t) containing date and time information.

Prototype

```
typedef struct
{
        IFX_int32_t elementType;
        IFX_int32_t day;
        IFX_int32_t month;
        IFX_int32_t hour;
        IFX_int32_t mn;
} IFX_TAPI_CID_MSG_DATE_t;
```

Parameters

Data Type	Name	Description
IFX_int32_t	elementType	Element type
IFX_int32_t	day	Day
IFX_int32_t	month	Month
IFX_int32_t	hour	Hour
IFX_int32_t	mn	Minute

4.2.4.9 IFX_TAPI_CID_MSG_STRING_t

Description

Structure for element types (IFX_TAPI_CID_SERVICE_TYPE_t) with dynamic length (line numbers or names).

Prototype

```
typedef struct
{
        IFX_TAPI_CID_SERVICE_TYPE_t elementType;
        IFX_int32_t len;
        IFX_char_t element[TAPI_CID_MAX_MSG_LEN];
} IFX_TAPI_CID_MSG_STRING_t;
```

Parameters

Data Type	Name	Description
IFX_TAPI_CID_SERVICE_TYPE_t	elementType	Element type
IFX_int32_t	len	Length of the message array
IFX_char_t	element[TAPI_CID_MAX_MSG_LEN]	String containing the message element.

4.2.4.10 IFX_TAPI_CID_MSG_t

Description

Structure containing the CID message type and content as well as information about transmission mode. This structure contains all information required by IFX_TAPI_CID_TX_INFO_START to start CID generation.

Prototype

```
typedef struct
{
        IFX_int32_t txMode;
        IFX_int32_t messageType;
        IFX_int32_t nMsgElements;
        IFX_TAPI_CID_MSG_ELEMENT_t* message;
} IFX_TAPI_CID_MSG_t;
```

Parameters

Data Type	Name	Description
IFX_int32_t	txMode	Define the Transmission Mode (enumerated in IFX_TAPI_CID_HOOK_MODE_t). Default IFX_TAPI_CID_HM_ONHOOK.
IFX_int32_t	messageType	Define the Message Type to be displayed (enumerated in IFX_TAPI_CID_MSG_TYPE_t). Default IFX_TAPI_CID_MT_TRANSPARENT.
IFX_int32_t	nMsgElements	Number of elements of the message array.
IFX_TAPI_CID_MSG_ELEMENT_t*	message	Message array.

4.2.4.11 IFX_TAPI_CID_MSG_TRANSPARENT_t

Description

Structure for element types (IFX_TAPI_CID_SERVICE_TYPE_t) with dynamic length (line numbers or names).

Prototype

```
typedef struct
{
        IFX_TAPI_CID_SERVICE_TYPE_t elementType;
        IFX_int32_t len;
        IFX_char_t *data;
} IFX_TAPI_CID_MSG_TRANSPARENT_t;
```

Parameters

Data Type	Name	Description
IFX_TAPI_CID_SERVICE_TYPE_t	elementType	Element type, this must be IFX_TAPI_CID_ST_TRANSPARENT
IFX_int32_t	len	Length of the message buffer
IFX_char_t	*data	Pointer to the message buffer containing the message already coded in the appropriate format and ready to be transmitted

4.2.4.12 IFX_TAPI_CID_MSG_VALUE_t

Description

Structure for element types (IFX_TAPI_CID_SERVICE_TYPE_t) with one value (length 1).

Prototype

```
typedef struct
{
        IFX_TAPI_CID_SERVICE_TYPE_t elementType;
        IFX_uint8_t element;
} IFX_TAPI_CID_MSG_VALUE_t;
```

Parameters

Data Type	Name	Description
IFX_TAPI_CID_SERVICE_TYPE_t	elementType	Element type
IFX_uint8_t	element	Value for the message element

4.2.4.13 IFX_TAPI_CID_RX_DATA_t

Description

Structure for Caller ID receiver data

Prototype

```
typedef struct
{
        IFX_uint8_t data;
        IFX_int32_t nSize;
```

```
} IFX_TAPI_CID_RX_DATA_t;
```

Parameters

Data Type	Name	Description
IFX_uint8_t	data	0_D **IFX_TAPI_CID_RX_SIZE_MAX** Caller Id receiver data.
IFX_int32_t	nSize	Caller Id receiver datasize in bytes.

4.2.4.14 IFX_TAPI_CID_RX_STATUS_t

Description

Structure for Caller ID receiver status.

Prototype

```
typedef struct
{
        IFX_uint8_t nStatus;
        IFX_uint8_t nError;
} IFX_TAPI_CID_RX_STATUS_t;
```

Parameters

Data Type	Name	Description
IFX_uint8_t	nStatus	Caller Id receiver actual status using IFX_TAPI_CID_RX_STATE_t. It can be any of the following values: 0_D **IFX_TAPI_CIDRX_STAT_INACTIVE** CID Receiver is not active. 1_D **IFX_TAPI_CIDRX_STAT_ACTIVE** CID Receiver is active. 2_D **IFX_TAPI_CIDRX_STAT_ONGOING** CID Receiver is just receiving data. 3_D **IFX_TAPI_CIDRX_STAT_DATA_RDY** CID Receiver is completed.
IFX_uint8_t	nError	Caller Id receiver actual error code using IFX_TAPI_CID_RX_ERROR_t. It can be any of the following values: 0_B **IFX_TAPI_CID_RX_ERROR_NONE** No Error during CID Receiver operation. 1_D **IFX_TAPI_CID_RX_ERROR_READ** Reading error during CID Receiver operation.

4.2.4.15 IFX_TAPI_CID_STD_ETSI_DTMF_t

Description

Structure containing CID configuration for ETSI standard using DTMF transmission.

Prototype

```
typedef struct
{
        IFX_TAPI_CID_TIMING_t* pCIDTiming;
        IFX_TAPI_CID_DTMF_CFG_t* pDTMFConf;
        IFX_int32_t nETSIAlertRing;
        IFX_int32_t nETSIAlertNoRing;
        IFX_int32_t nAlertToneOffhook;
        IFX_int32_t nAlertToneOnhook;
        IFX_int32_t ringPulseTime;
        IFX_char_t ackTone;
        IFX_TAPI_CID_ABS_REASON_t* pABSCLICode;
} IFX_TAPI_CID_STD_ETSI_DTMF_t;
```

Parameters

Data Type	Name	Description
IFX_TAPI_CID_TIMING_t*	pCIDTiming	Pointer to a structure containing timing information. If the parameter is not given, IFX_TAPI_CID_TIMING_t default values will be used.
IFX_TAPI_CID_DTMF_CFG_t*	pDTMFConf	Pointer to a structure containing DTMF configuration parameters. If the parameter is not given, IFX_TAPI_CID_DTMF_CFG_t default values will be used.
IFX_int32_t	nETSIAlertRing	Type of ETSI Alert of on-hook services associated to ringing (enumerated in IFX_TAPI_CID_ALERT_ETSI_t). Default IFX_TAPI_CID_ALERT_ETSI_FR.
IFX_int32_t	nETSIAlertNoRing	Type of ETSI Alert of on-hook services not associated to ringing (enumerated in IFX_TAPI_CID_ALERT_ETSI_t). Default IFX_TAPI_CID_ALERT_ETSI_RP.
IFX_int32_t	nAlertToneOffhook	Tone table index for the alert tone to be used. Required for automatic CID/MWI generation. By default TAPI uses an internal tone definition.
IFX_int32_t	nAlertToneOnhook	Tone table index for the alert tone to be used. Required for automatic CID/MWI generation. By default TAPI uses an internal tone definition.
IFX_int32_t	ringPulseTime	Duration of Ring Pulse, in ms, default 500 ms.
IFX_char_t	ackTone	DTMF ACK after CAS, used for offhook transmission. Default DTMF 'D'.
IFX_TAPI_CID_ABS_REASON_t*	pABSCLICode	Pointer to a structure containing the coding for absence reason of calling number.

4.2.4.16 IFX_TAPI_CID_STD_ETSI_FSK_t

Description

Structure containing CID configuration for ETSI standard using FSK transmission.

Prototype

```
typedef struct
{
        IFX_TAPI_CID_TIMING_t* pCIDTiming;
        IFX_TAPI_CID_FSK_CFG_t* pFSKConf;
        IFX_int32_t nETSIAlertRing;
        IFX_int32_t nETSIAlertNoRing;
        IFX_int32_t nAlertToneOffhook;
        IFX_int32_t nAlertToneOnhook;
        IFX_int32_t ringPulseTime;
        IFX_char_t ackTone;
} IFX_TAPI_CID_STD_ETSI_FSK_t;
```

Parameters

Data Type	Name	Description
IFX_TAPI_CID_TIMING_t*	pCIDTiming	Pointer to a structure containing timing information. If the parameter is not given, IFX_TAPI_CID_TIMING_t default values will be used.
IFX_TAPI_CID_FSK_CFG_t *	pFSKConf	Pointer to a structure containing FSK configuration parameters. If the parameter is not given, IFX_TAPI_CID_FSK_CFG_t default values will be used.
IFX_int32_t	nETSIAlertRing	Type of ETSI Alert of on-hook services associated to ringing (enumerated in IFX_TAPI_CID_ALERT_ETSI_t). Default IFX_TAPI_CID_ALERT_ETSI_FR.
IFX_int32_t	nETSIAlertNoRing	Type of ETSI Alert of on-hook services not associated to ringing (enumerated in IFX_TAPI_CID_ALERT_ETSI_t). Default IFX_TAPI_CID_ALERT_ETSI_RP.
IFX_int32_t	nAlertToneOffhook	Tone table index for the alert tone to be used. Required for automatic CID/MWI generation. By default TAPI uses an internal tone definition.
IFX_int32_t	nAlertToneOnhook	Tone table index for the alert tone to be used. Required for automatic CID/MWI generation. By default TAPI uses an internal tone definition.
IFX_int32_t	ringPulseTime	Duration of Ring Pulse, in ms, default 500 ms.
IFX_char_t	ackTone	DTMF ACK after CAS, used for off-hook transmission. Default DTMF is ‘D’.

4.2.4.17 IFX_TAPI_CID_STD_NTT_t

Description

Structure containing CID configuration for NTT standard.

Prototype

```
typedef struct
{
        IFX_TAPI_CID_TIMING_t* pCIDTiming;
        IFX_TAPI_CID_FSK_CFG_t* pFSKConf;
        IFX_int32_t nAlertToneOffhook;
        IFX_int32_t nAlertToneOnhook;
        IFX_int32_t ringPulseTime;
        IFX_int32_t ringPulseTimeLoop;
        IFX_int32_t ringPulseOffTime;
        IFX_int32_t dataOut2incomingSuccessfulTimeout;
} IFX_TAPI_CID_STD_NTT_t;
```

Parameters

Data Type	Name	Description
IFX_TAPI_CID_TIMING_t*	pCIDTiming	Pointer to a structure containing timing information. If the parameter is not given, IFX_TAPI_CID_TIMING_t default values will be used.
IFX_TAPI_CID_FSK_CFG_t*	pFSKConf	Pointer to a structure containing FSK configuration parameters. If the parameter is not given, IFX_TAPI_CID_FSK_CFG_t default values will be used.
IFX_int32_t	nAlertToneOffhook	Tone table index for the alert tone to be used. Required for automatic CID/MWI generation. By default TAPI uses an internal tone definition.
IFX_int32_t	nAlertToneOnhook	Tone table index for the alert tone to be used. Required for automatic CID/MWI generation. By default TAPI uses an internal tone definition.
IFX_int32_t	ringPulseTime	Ring pulse on time (CAR signal), in ms, default 500 ms.
IFX_int32_t	ringPulseTimeLoop	Max number of ring pulses (CAR signals), default 5.
IFX_int32_t	ringPulseOffTime	Ring pulse off time (CAR signal), in ms, default 500 ms.
IFX_int32_t	dataOut2incomingSuccessfulTimeout	Timeout for incoming successful signal to arrive after CID data transmission is completed. Default 7000 ms.

4.2.4.18 IFX_TAPI_CID_STD_SIN_t

Description

Structure containing CID configuration for BT SIN227 standard.

Prototype

```
typedef struct
{
        IFX_TAPI_CID_TIMING_t* pCIDTiming;
        IFX_TAPI_CID_FSK_CFG_t* pFSKConf;
        IFX_int32_t nAlertToneOffhook;
        IFX_int32_t nAlertToneOnhook;
        IFX_char_t ackTone;
} IFX_TAPI_CID_STD_SIN_t;
```

Parameters

Data Type	Name	Description
IFX_TAPI_CID_TIMING_t*	pCIDTiming	Pointer to a structure containing timing information. If the parameter is not given, IFX_TAPI_CID_TIMING_t default values will be used.
IFX_TAPI_CID_FSK_CFG_t*	pFSKConf	Pointer to a structure containing FSK configuration parameters. If the parameter is not given, IFX_TAPI_CID_FSK_CFG_t default values will be used.
IFX_int32_t	nAlertToneOffhook	Tone table index for the alert tone to be used. Required for automatic CID/MWI generation. By default TAPI uses an internal tone definition.
IFX_int32_t	nAlertToneOnhook	Tone table index for the alert tone to be used. Required for automatic CID/MWI generation. By default TAPI uses an internal tone definition.
IFX_char_t	ackTone	DTMF ACK after CAS, used for offhook transmission. Default DTMF 'D'.

4.2.4.19 IFX_TAPI_CID_STD_TELCORDIA_t

Description

Structure containing CID configuration for Telcordia standard.

Prototype

```
typedef struct
{
        IFX_TAPI_CID_TIMING_t* pCIDTiming;
        IFX_TAPI_CID_FSK_CFG_t* pFSKConf;
        IFX_int32_t OSIoffhook;
        IFX_int32_t OSItime;
        IFX_int32_t nAlertToneOffhook;
        IFX_char_t ackTone;
} IFX_TAPI_CID_STD_TELCORDIA_t;
```

Parameters

Data Type	Name	Description
IFX_TAPI_CID_TIMING_t*	pCIDTiming	Pointer to a structure containing timing information. If the parameter is not given, IFX_TAPI_CID_TIMING_t default values will be used.
IFX_TAPI_CID_FSK_CFG_t*	pFSKConf	Pointer to a structure containing FSK configuration parameters. If the parameter is not given, IFX_TAPI_CID_FSK_CFG_t default values will be used.
IFX_int32_t	OSIoffhook	Usage of OSI for offhook transmission. Default IFX_FALSE.
IFX_int32_t	OSItime	Length of the OSI signal in ms. Default 200 ms.

Data Type	Name	Description
IFX_int32_t	nAlertToneOffhook	Tone table index for the alert tone to be used. Required for automatic CID/MWI generation. By default TAPI uses an internal tone definition.
IFX_char_t	ackTone	DTMF ACK after CAS, used for offhook transmission. Default DTMF 'D'.

4.2.4.20 IFX_TAPI_CID_TIMING_t

Description

Structure containing the timing for CID transmission.

Prototype

```
typedef struct
{
        IFX_int32_t beforeData;
        IFX_int32_t dataOut2restoreTimeOnhook;
        IFX_int32_t dataOut2restoreTimeOffhook;
        IFX_int32_t ack2dataOutTime;
        IFX_int32_t cas2ackTime;
        IFX_int32_t afterAckTimeout;
        IFX_int32_t afterFirstRing;
        IFX_int32_t afterRingPulse;
        IFX_int32_t afterDTASOnhook;
        IFX_int32_t afterLineReversal;
        IFX_int32_t afterOSI;
} IFX_TAPI_CID_TIMING_t;
```

Parameters

Data Type	Name	Description
IFX_int32_t	beforeData	Time to wait after AS, in ms, before data transmission. Default 300 ms. Used only for NTT off-hook trasmission.
IFX_int32_t	dataOut2restoreTimeOnhook	Time to wait after data transmission, in ms, for on-hook services. Default 300 ms.
IFX_int32_t	dataOut2restoreTimeOffhook	Time to wait after data transmission, in ms, for off-hook services. Default 60 ms.
IFX_int32_t	ack2dataOutTime	Time to wait after ack detection, in ms, before data transmission. Default 55 ms.
IFX_int32_t	cas2ackTime	Time-out for ack detection, in ms. Default 160 ms.
IFX_int32_t	afterAckTimeout	Time to wait after ACK time-out, in ms. Default 50 ms.
IFX_int32_t	afterFirstRing	Time to wait after first ring, in ms, before starting data transmission. Default 600 ms.
IFX_int32_t	afterRingPulse	Time to wait after ring pulse, in ms, before starting data transmission. Default 600 ms.

Data Type	Name	Description
IFX_int32_t	afterDTASOnhook	Time to wait after DTAS, in ms, before starting data transmission. Default 45 ms.
IFX_int32_t	afterLineReversal	Time to wait after line reversal, in ms, before successive operation[1] is performed. Default 100 ms.
IFX_int32_t	afterOSI	Time to wait after OSI signal, in ms, before data transmission. Default 300 ms.

1) Timing between line reversal and DTAS (ETSI standard) or CAR (NTT standard) signal.

4.2.4.21 IFX_TAPI_DWLD_t

Description

TAPI download structure.

Prototype

```
typedef struct
{
        IFX_TAPI_DWLD_TYPE_t nMode;
        IFX_uint8_t * pBuf;
        IFX_uint32_t nSize;
        IFX_uint16_t nCRC;
} IFX_TAPI_DWLD_t;
```

Parameters

Data Type	Name	Description
IFX_TAPI_DWLD_TYPE_t	nMode	Download type.
IFX_uint8_t *	pBuf	Pointer to the buffer to be downloaded.
IFX_uint32_t	nSize	Size, in bytes, of the buffer to be downloaded.
IFX_uint16_t	nCRC	Return value of the CRC. *Note: Not defined for all downloads.*

4.2.4.22 IFX_TAPI_EXCEPTION_BITS_t

Description

Exception bits.

Prototype

```
typedef struct
{
        IFX_uint32_t reserved2:1;
        IFX_uint32_t eventDetect:1;
        IFX_uint32_t fax_ced_net:1;
        IFX_uint32_t fax_cng_net:1;
        IFX_uint32_t gr909result:1;
        IFX_uint32_t device:1;
        IFX_uint32_t signal:1;
```

```
        IFX_uint32_t otemp:1;
        IFX_uint32_t ground_key_polarity:1;
        IFX_uint32_t ground_key_high:1;
        IFX_uint32_t fault:1;
        IFX_uint32_t fax_supdate:1;
        IFX_uint32_t fax_dis:1;
        IFX_uint32_t fax_ced:1;
        IFX_uint32_t fax_cng:1;
        IFX_uint32_t pulse_digit_ready:1;
        IFX_uint32_t ground_key:1;
        IFX_uint32_t cidrx_supdate:1;
        IFX_uint32_t callerid_ready:1;
        IFX_uint32_t pstn_ring:1;
        IFX_uint32_t flash_hook:1;
        IFX_uint32_t hookstate:1;
        IFX_uint32_t dtmf_ready:1;
} IFX_TAPI_EXCEPTION_BITS_t;
```

Parameters

Data Type	Name	Description
IFX_uint32_t	reserved2:1	
IFX_uint32_t	eventDetect:1	RFC 2833 event playout detection.
IFX_uint32_t	fax_ced_net:1	Reserved.
IFX_uint32_t	fax_cng_net:1	Reserved.
IFX_uint32_t	gr909result:1	GR909 exception.
IFX_uint32_t	device:1	Low-level device exception. The information can be queried from the underlying driver
IFX_uint32_t	signal:1	Signal detected. The information is retrieved with IFX_TAPI_CH_STATUS_GET
IFX_uint32_t	otemp:1	Over temperature detected.
IFX_uint32_t	ground_key_polarity:1	Ground key polarity detected.
IFX_uint32_t	ground_key_high:1	Ground key high detected.
IFX_uint32_t	fault:1	Fault condition detected, query information via IFX_TAPI_CH_STATUS_GET.
IFX_uint32_t	fax_supdate:1	FAX status update, query information via IFX_TAPI_CH_STATUS_GET.
IFX_uint32_t	fax_dis:1	FAX DIS received (no longer supported).
IFX_uint32_t	fax_ced:1	FAX CED received (no longer supported).
IFX_uint32_t	fax_cng:1	FAX CNG received (no longer supported).
IFX_uint32_t	pulse_digit_ready:1	Pulse dial digit is ready.
IFX_uint32_t	ground_key:1	Ground key detected.
IFX_uint32_t	cidrx_supdate:1	Caller id receiver event, query information with IFX_TAPI_CID_RX_STATUS_GET.
IFX_uint32_t	callerid_ready:1	Caller id data is ready (no longer supported).

Data Type	Name	Description
IFX_uint32_t	pstn_ring:1	Reserved.
IFX_uint32_t	flash_hook:1	Flash hook detected.
IFX_uint32_t	hookstate:1	Hook state changed.
IFX_uint32_t	dtmf_ready:1	DTMF digit is ready.

4.2.4.23 IFX_TAPI_EVENT_t

Description

This structure is used by event reporting interfaces.

Prototype

```
typedef struct
{
        IFX_TAPI_EVENT_ID_t id;
        IFX_uint16_t ch;
        IFX_uint16_t tail;
        IFX_TAPI_EVENT_DATA_t data;
} IFX_TAPI_EVENT_t;
```

Parameters

Data Type	Name	Description
IFX_TAPI_EVENT_ID_t	id	Event id.
IFX_uint16_t	ch	Channel where the event occurred.
IFX_uint16_t	tail	This field is used to report whether new events are ready (IFX_TRUE) or not (IFX_FALSE) .
IFX_TAPI_EVENT_DATA_t	data	Data about the individual event.

4.2.4.24 IFX_TAPI_EVENT_DATA_DTMF_t

Description

Information about a DTMF event.

Prototype

```
typedef struct
{
        IFX_uint16_t local:1;
        IFX_uint16_t network:1;
        IFX_uint16_t reserved:6;
        IFX_uint16_t digit:8;
} IFX_TAPI_EVENT_DATA_DTMF_t;
```

Parameters

Data Type	Name	Description
IFX_uint16_t	local:1	Detected from local side.
IFX_uint16_t	network:1	Detected from network side.
IFX_uint16_t	reserved:6	Reserved.
IFX_uint16_t	digit:8	DTMF digit information.

4.2.4.25 IFX_TAPI_EVENT_DATA_EXT_KEYPAD_t

Description

This structure is used to report a DTMF event to the TAPI from an external software module.

Prototype

```
typedef struct
{
        IFX_int8_t key;
        IFX_char_t duration;
} IFX_TAPI_EVENT_DATA_EXT_KEYPAD_t;
```

Parameters

Data Type	Name	Description
IFX_int8_t	key	Key.
IFX_char_t	duration	Duration.

4.2.4.26 IFX_TAPI_EVENT_DATA_FAX_SIG_t

Description

Information about a fax/modem signal event.

Prototype

```
typedef struct
{
        IFX_uint16_t local:1;
        IFX_uint16_t network:1;
        IFX_uint16_t reserved:6;
} IFX_TAPI_EVENT_DATA_FAX_SIG_t;
```

Parameters

Data Type	Name	Description
IFX_uint16_t	local:1	Detected from local side.
IFX_uint16_t	network:1	Detected from network side.
IFX_uint16_t	reserved:6	Reserved.

4.2.4.27 IFX_TAPI_EVENT_DATA_PULSE_t

Description

Information about a pulse event.

Prototype

```
typedef struct
{
        IFX_uint16_t reserved:8;
        IFX_uint16_t digit:8;
} IFX_TAPI_EVENT_DATA_PULSE_t;
```

Parameters

Data Type	Name	Description
IFX_uint16_t	reserved:8	Reserved.
IFX_uint16_t	digit:8	Pulse digit information.

4.2.4.28 IFX_TAPI_EVENT_DATA_RFC2833_t

Description

Information about an RFC2833 event.

Prototype

```
typedef struct
{
        IFX_uint16_t event;
} IFX_TAPI_EVENT_DATA_RFC2833_t;
```

Parameters

Data Type	Name	Description
IFX_uint16_t	event	Event code contained in the RFC2833 frame.

4.2.4.29 IFX_TAPI_EVENT_DATA_TONE_t

Description

Information about a tone generation/detection event.

Prototype

```
typedef struct
{
        IFX_uint16_t local:1;
        IFX_uint16_t network:1;
        IFX_uint16_t reserved:6;
        IFX_uint16_t index:8;
} IFX_TAPI_EVENT_DATA_TONE_t;
```

Parameters

Data Type	Name	Description
IFX_uint16_t	local:1	Detected from local side.
IFX_uint16_t	network:1	Detected from network side.
IFX_uint16_t	reserved:6	Reserved.
IFX_uint16_t	index:8	Tone table index.

4.2.4.30 IFX_TAPI_EVENT_EXT_DTMF_t

Description

This structure is used to report a DTMF event to the TAPI from an external software module.

Prototype

```
typedef struct
{
        IFX_TAPI_EVENT_t event;
        IFX_boolean_t start;
        IFX_uint32_t length;
} IFX_TAPI_EVENT_EXT_DTMF_t;
```

Parameters

Data Type	Name	Description
IFX_TAPI_EVENT_t	event	Event to be reported.
IFX_boolean_t	start	Signal start (IFX_TRUE) or stop (IFX_FALSE) .
IFX_uint32_t	length	DTMF length (optional).

4.2.4.31 IFX_TAPI_EVENT_EXT_DTMF_CFG_t

Description

This structure is used to configure support for the reporting of external DTMF events.

Prototype

```
typedef struct
{
        IFX_operation_t localPlay;
} IFX_TAPI_EVENT_EXT_DTMF_CFG_t;
```

Parameters

Data Type	Name	Description
IFX_operation_t	localPlay	Enable or disable local play of the DTMF tone.

4.2.4.32 IFX_TAPI_JB_CFG_t

Description

Structure for jitter buffer configuration used by **IFX_TAPI_ENC_AGC_CFG_SET**.

Prototype

```
typedef struct
{
        IFX_uint8_t nJbType;
        IFX_char_t nPckAdpt;
        IFX_char_t nLocalAdpt;
        IFX_char_t nScaling;
        IFX_uint16_t nInitialSize;
        IFX_uint16_t nMaxSize;
        IFX_uint16_t nMinSize;
} IFX_TAPI_JB_CFG_t;
```

Parameters

Data Type	Name	Description
IFX_uint8_t	nJbType	Jitter buffer type, value of **IFX_TAPI_JB_TYPE_t**.
IFX_char_t	nPckAdpt	Packet adaptation, value of **IFX_TAPI_JB_PKT_ADAPT_t**
IFX_char_t	nLocalAdpt	Local adaptation, value of **IFX_TAPI_JB_LOCAL_ADAPT_t**. Relevant only for adaptive jitter buffer.
IFX_char_t	nScaling	This factor influences the play out delay. If nPckAdpt is 1, nScaling multiplied by the packet length determines the play out delay. If nPckAdpt is 0, the play out delay is calculated by the multiplication of the estimated network jitter and nScaling. nScaling can be chosen between 1 and 16. Default: 8
IFX_uint16_t	nInitialSize	Initial size of the jitter buffer in timestamps of 125 µs in case of an adaptive jitter buffer.
IFX_uint16_t	nMaxSize	Maximum size of the jitter buffer in timestamps of 125 µs in case of an adaptive jitter buffer.
IFX_uint16_t	nMinSize	Minimum size of the jitter buffer in timestamps of 125 µs in case of an adaptive jitter buffer.

Remarks

This structure may be changed in the future.

4.2.4.33 IFX_TAPI_JB_STATISTICS_t

Description

Structure for Jitter Buffer statistics used by ioctl **IFX_TAPI_JB_STATISTICS_GET**.

Prototype

```
typedef struct
{
        IFX_uint8_t nType;
        IFX_uint32_t nInTime;
        IFX_uint32_t nCNG;
        IFX_uint32_t nBFI;
        IFX_uint16_t nBufSize;
        IFX_uint16_t nMaxBufSize;
        IFX_uint16_t nMinBufSize;
        IFX_uint16_t nMaxDelay;
        IFX_uint16_t nMinDelay;
        IFX_uint16_t nNwJitter;
        IFX_uint16_t nPODelay;
        IFX_uint16_t nMaxPODelay;
        IFX_uint16_t nMinPODelay;
        IFX_uint32_t nPackets;
        IFX_uint16_t nLost;
        IFX_uint16_t nInvalid;
        IFX_uint16_t nDuplicate;
        IFX_uint16_t nLate;
        IFX_uint16_t nEarly;
        IFX_uint16_t nResync;
        IFX_uint32_t nIsUnderflow;
        IFX_uint32_t nIsNoUnderflow;
        IFX_uint32_t nIsIncrement;
        IFX_uint32_t nSkDecrement;
        IFX_uint32_t nDsDecrement;
        IFX_uint32_t nDsOverflow;
        IFX_uint32_t nSid;
} IFX_TAPI_JB_STATISTICS_t;
```

Parameters

Data Type	Name	Description
IFX_uint8_t	nType	Jitter buffer type • 1: fixed • 2: adaptive
IFX_uint32_t	nInTime	Incoming time high word total time in timestamp units for all packets since the start of the connection which could be played out correctly. Not supported anymore
IFX_uint32_t	nCNG	Comfort noise generation. Not supported anymore
IFX_uint32_t	nBFI	Bad frame interpolation. Not supported anymore
IFX_uint16_t	nBufSize	Current jitter buffer size.
IFX_uint16_t	nMaxBufSize	Maximum estimated jitter buffer size.
IFX_uint16_t	nMinBufSize	Minimum estimated jitter buffer size.

Data Type	Name	Description
IFX_uint16_t	nMaxDelay	Maximum packet delay. Not supported anymore
IFX_uint16_t	nMinDelay	Minimum packet delay. Not supported anymore
IFX_uint16_t	nNwJitter	Network jitter value. Not supported anymore
IFX_uint16_t	nPODelay	Play out delay
IFX_uint16_t	nMaxPODelay	Maximum play out delay.
IFX_uint16_t	nMinPODelay	Minimum play out delay.
IFX_uint32_t	nPackets	Received packet number.
IFX_uint16_t	nLost	Lost packets number. Not supported anymore
IFX_uint16_t	nInvalid	Invalid packet number.
IFX_uint16_t	nDuplicate	Duplication packet number. Not supported anymore
IFX_uint16_t	nLate	Late packets number.
IFX_uint16_t	nEarly	Early packets number.
IFX_uint16_t	nResync	Resynchronizations number.
IFX_uint32_t	nIsUnderflow	Total number of injected samples since the beginning of the connection or since the last statistic reset due to jitter buffer underflows.
IFX_uint32_t	nIsNoUnderflow	Total number of injected samples since the beginning of the connection or since the last statistic reset in case of normal jitter buffer operation, which means when there is not a jitter buffer underflow.
IFX_uint32_t	nIsIncrement	Total number of injected samples since the beginning of the connection or since the last statistic reset in case of jitter buffer increments.
IFX_uint32_t	nSkDecrement	Total number of skipped lost samples since the beginning of the connection or since the last statistic reset in case of jitter buffer decrements.
IFX_uint32_t	nDsDecrement	Total number of dropped samples since the beginning of the connection or since the last statistic reset in case of jitter buffer decrements.
IFX_uint32_t	nDsOverflow	Total number of dropped samples since the beginning of the connection or since the last statistic reset in case of jitter buffer overflows.
IFX_uint32_t	nSid	Total number of comfort noise samples since the beginning of the connection or since the last statistic reset.

4.2.4.34 IFX_TAPI_LEC_CFG_t

Description

Line echo canceller (LEC) configuration.

Prototype

```
typedef struct
{
        IFX_TAPI_LEC_TYPE_t nType;
        IFX_char_t nGainIn;
        IFX_char_t nGainOut;
        IFX_char_t nLen;
        IFX_char_t bNlp;
} IFX_TAPI_LEC_CFG_t;
```

Parameters

Data Type	Name	Description
IFX_TAPI_LEC_TYPE_t	nType	LEC type selection.
IFX_char_t	nGainIn	Gain for input. 0_D **IFX_TAPI_LEC_GAIN_OFF** LEC Off 0_D **IFX_TAPI_LEC_GAIN_OFF** LEC Off 1_D **IFX_TAPI_LEC_GAIN_LOW** Low Gain 2_D **IFX_TAPI_LEC_GAIN_MEDIUM** Medium Gain (only supported for VINETIC®) 3_D **IFX_TAPI_LEC_GAIN_HIGH** High Gain
IFX_char_t	nGainOut	Gain for output LEC. 0_D **IFX_TAPI_LEC_GAIN_OFF** LEC Off 1_D **IFX_TAPI_LEC_GAIN_LOW** Low Gain 2_D **IFX_TAPI_LEC_GAIN_MEDIUM** Medium Gain (only supported for VINETIC®) 3_D **IFX_TAPI_LEC_GAIN_HIGH** High Gain
IFX_char_t	nLen	LEC tail length in milli seconds. nLen is currently not supported and can be set to zero
IFX_char_t	bNlp	Switch the NLP on or off. 0_D **TAPI_LEC_NLPDEFAULT** NLP is default 1_D **TAPI_LEC_NLP_ON** NLP is on 2_D **TAPI_LEC_NLP_OFF** NLP is off

4.2.4.35 IFX_TAPI_LINE_HOOK_VT_t

Description

Structure used for validation times of hook, hook flash and pulse dialing.

An example of typical timing

- 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!!! open: only min. time is validated and pre initialized

Prototype

```
typedef struct
{
        IFX_TAPI_LINE_HOOK_VALIDATION_TYPE_t nType;
        IFX_uint32_t nMinTime;
        IFX_uint32_t nMaxTime;
} IFX_TAPI_LINE_HOOK_VT_t;
```

Parameters

Data Type	Name	Description
IFX_TAPI_LINE_HOOK_VALIDATION_TYPE_t	nType	Type of validation time setting.
IFX_uint32_t	nMinTime	Minimum time for validation in ms.
IFX_uint32_t	nMaxTime	Maximum time for validation in ms

4.2.4.36 IFX_TAPI_LINE_VOLUME_t

Description

Phone speaker phone and microphone volume settings used for ioctl IFX_TAPI_PHONE_VOLUME_SET.

Prototype

```
typedef struct
{
        IFX_int32_t nGainRx;
        IFX_int32_t nGainTx;
} IFX_TAPI_LINE_VOLUME_t;
```

Parameters

Data Type	Name	Description
IFX_int32_t	nGainRx	Volume setting for the receiving path, speakerphone The value is given in dB with the range (-24 dB to 24 dB)
IFX_int32_t	nGainTx	Volume setting for the transmitting path, microphone The value is given in dB with the range (-24 dB to 24 dB)

4.2.4.37 IFX_TAPI_MAP_DATA_t

Description

Phone channel mapping structure used for **IFX_TAPI_MAP_DATA_ADD** and **IFX_TAPI_MAP_DATA_REMOVE**.

Prototype

```
typedef struct
{
        IFX_char_t nDstCh;
        IFX_TAPI_MAP_DATA_TYPE_t nChType;
        IFX_TAPI_DATA_MAP_START_STOP_t nRecStart;
        IFX_TAPI_DATA_MAP_START_STOP_t nPlayStart;
} IFX_TAPI_MAP_DATA_t;
```

Parameters

Data Type	Name	Description
IFX_char_t	nDstCh	Phone channel number to which this channel should be mapped. Phone channels numbers start from 0.
IFX_TAPI_MAP_DATA_TYPE_t	nChType	Type of the destination channel. 0_D **IFX_TAPI_MAP_TYPE_DEFAULT** Default selected (phone channel) 1_D **IFX_TAPI_MAP_TYPE_CODER** not supported 2_D **IFX_TAPI_MAP_TYPE_PCM** type is PCM 3_D **IFX_TAPI_MAP_TYPE_PHONE** type is phone channel

Data Type	Name	Description
IFX_TAPI_DATA_MAP_START_STOP_t	nRecStart	Enables or disables the recording service or leaves as it is. 0_D **IFX_TAPI_MAP_DATA_UNCHANGED** Do not modify the status of the recorder 1_D **IFX_TAPI_MAP_DATA_START** Recording is started, same as IFX_TAPI_ENC_START 2_D **IFX_TAPI_MAP_DATA_STOP** Recording is stopped, same as IFX_TAPI_ENC_STOP
IFX_TAPI_DATA_MAP_START_STOP_t	nPlayStart	Enables or disables the play service or leaves as it is. 0_D **IFX_TAPI_MAP_DATA_UNCHANGED** Do not modify the status of the recorder 1_D **IFX_TAPI_MAP_DATA_START** Playing is started, same as IFX_TAPI_ENC_START 2_D **IFX_TAPI_MAP_DATA_STOP** Playing is stopped, same as IFX_TAPI_ENC_STOP

4.2.4.38 IFX_TAPI_MAP_PCM_t

Description

Phone channel mapping structure used for IFX_TAPI_MAP_PCM_ADD and IFX_TAPI_MAP_PCM_REMOVE.

Prototype

```
typedef struct
{
        IFX_uint8_t nDstCh;
        IFX_TAPI_MAP_TYPE_t nChType;
} IFX_TAPI_MAP_PCM_t;
```

Parameters

Data Type	Name	Description
IFX_uint8_t	nDstCh	Channel number to which this channel should be mapped. Channels numbers start from 0.
IFX_TAPI_MAP_TYPE_t	nChType	Type of the destination channel. 0_D **IFX_TAPI_MAP_TYPE_DEFAULT** Default selected (analog phone channel) 1_D **IFX_TAPI_MAP_TYPE_CODER** not supported 2_D **IFX_TAPI_MAP_TYPE_PCM** type is PCM 3_D **IFX_TAPI_MAP_TYPE_PHONE** type is analog phone channel

4.2.4.39 IFX_TAPI_MAP_PHONE_t

Description

Phone channel mapping structure used for IFX_TAPI_MAP_PHONE_ADD and IFX_TAPI_MAP_PHONE_REMOVE.

Prototype

```
typedef struct
{
        IFX_uint8_t nPhoneCh;
        IFX_TAPI_MAP_TYPE_t nChType;
} IFX_TAPI_MAP_PHONE_t;
```

Parameters

Data Type	Name	Description
IFX_uint8_t	nPhoneCh	Phone channel number to which this channel should be mapped. Phone channels numbers start from 0.
IFX_TAPI_MAP_TYPE_t	nChType	Type of the destination channel. 0_D **IFX_TAPI_MAP_TYPE_DEFAULT** Default selected (analog phone channel) 1_D **IFX_TAPI_MAP_TYPE_CODER** not supported 2_D **IFX_TAPI_MAP_TYPE_PCM** type is PCM 3_D **IFX_TAPI_MAP_TYPE_PHONE** type is analog phone channel

4.2.4.40 IFX_TAPI_METER_CFG_t

Description

Structure for metering config.

Prototype

```
typedef struct
{
        IFX_uint8_t mode;
        IFX_uint8_t freq;
        IFX_uint32_t burst_len;
        IFX_uint32_t burst_dist;
        IFX_uint32_t burst_cnt;
} IFX_TAPI_METER_CFG_t;
```

Parameters

Data Type	Name	Description
IFX_uint8_t	mode	Metering mode. 0_D **TAPI_METER_MODE_TTX** TTX mode 1_D **TAPI_METER_MODE_REVPOL** reverse polarity
IFX_uint8_t	freq	Reserved.
IFX_uint32_t	burst_len	Length of metering burst in ms. burst_len must be greater than zero.
IFX_uint32_t	burst_dist	Distance between the metering burst in sec.
IFX_uint32_t	burst_cnt	Defines the number of bursts.

4.2.4.41 IFX_TAPI_PCK_AAL_CFG_t

Description

Structure used for AAL configuration

Prototype

```
typedef struct
{
        IFX_uint16_t nCid;
        IFX_uint16_t nTimestamp;
        IFX_uint16_t nCpsCid;
} IFX_TAPI_PCK_AAL_CFG_t;
```

Parameters

Data Type	Name	Description
IFX_uint16_t	nCid	Connection identifier.

Data Type	Name	Description
IFX_uint16_t	nTimestamp	Start value for the timestamp (resolution of 125 µs).
IFX_uint16_t	nCpsCid	Connection Identifier for CPS events.

4.2.4.42 IFX_TAPI_PCK_AAL_PROFILE_t

Description

AAL profile setup structure.

Prototype

```
typedef struct
{
        IFX_char_t rows;
        IFX_char_t len[10];
        IFX_char_t nUUI[10];
        IFX_char_t codec[10];
} IFX_TAPI_PCK_AAL_PROFILE_t;
```

Parameters7

Data Type	Name	Description
IFX_char_t	rows	Amount of rows to program.
IFX_char_t	len[10]	Length of packet in bytes - 1.
IFX_char_t	nUUI[10]	UUI codepoint range indicator, see IFX_TAPI_PKT_AAL_PROFILE_RANGE_t.
IFX_char_t	codec[10]	Codec as listed for IFX_TAPI_ENC_TYPE_t.

4.2.4.43 IFX_TAPI_PCM_CFG_t

Description

Structure for PCM configuration.

Prototype

```
typedef struct
{
        IFX_uint32_t nTimeslotRX;
        IFX_uint32_t nTimeslotTX;
        IFX_uint32_t nHighway;
        IFX_uint32_t nResolution;
        IFX_uint32_t nRate;
} IFX_TAPI_PCM_CFG_t;
```

Parameters

Data Type	Name	Description
IFX_uint32_t	nTimeslotRX	PCM time slot for the receive direction.
IFX_uint32_t	nTimeslotTX	PCM time slot for the transmit direction.
IFX_uint32_t	nHighway	Defines the PCM highway number which is connected to the channel.
IFX_uint32_t	nResolution	Defines the PCM interface coding. 0_D **IFX_TAPI_PCM_RES_ALAW_8BIT** 8-bit A-law 1_D **IFX_TAPI_PCM_RES_MLAW_8BIT** 8bit µ-law 2_D **IFX_TAPI_PCM_RES_LINEAR_16BIT** 16-bit linear
IFX_uint32_t	nRate	Defines the PCM sample rate in kHz.

4.2.4.44 IFX_TAPI_PKT_RTCP_STATISTICS_t

Description

Structure for RTCP Statistics. It refers to the RFC3550/3551.

Prototype

```
typedef struct
{
        IFX_uint32_t ssrc;
        IFX_uint32_t ntp_sec;
        IFX_uint32_t ntp_frac;
        IFX_uint32_t rtp_ts;
        IFX_uint32_t psent;
        IFX_uint32_t osent;
        IFX_uint32_t rssrc;
        IFX_uint32_t fraction;
        IFX_int32_t lost;
        IFX_uint32_t last_seq;
        IFX_uint32_t jitter;
        IFX_uint32_t lsr;
        IFX_uint32_t dlsr;
} IFX_TAPI_PKT_RTCP_STATISTICS_t;
```

Parameters

Data Type	Name	Description
IFX_uint32_t	ssrc	Sender generating this report.
IFX_uint32_t	ntp_sec	NTP timestamp higher 32 bits.
IFX_uint32_t	ntp_frac	NTP timestamp lower 32 bits.
IFX_uint32_t	rtp_ts	RTP time stamp.
IFX_uint32_t	psent	Sent packet count.

Data Type	Name	Description
IFX_uint32_t	osent	Sent octets count.
IFX_uint32_t	rssrc	Data source.
IFX_uint32_t	fraction	Receivers fraction loss.
IFX_int32_t	lost	Receivers packet lost.
IFX_uint32_t	last_seq	Extended last seq nr received.
IFX_uint32_t	jitter	Receives interarrival jitter.
IFX_uint32_t	lsr	Last sender report.
IFX_uint32_t	dlsr	Delay since last sender report.

4.2.4.45 IFX_TAPI_PKT_RTP_CFG_t

Description

Structure for RTP Configuration.

RFC2833 event payload types (ePT) for the encoder and decoder part are also configured. Parameter “nPlayEvents” and “nEventPlayPT” are used to configure the payload type for the decoder part.

Prototype

```
typedef struct
{
        IFX_uint16_t nSeqNr;
        IFX_uint32_t nSsrc;
        IFX_uint32_t nTimestamp;
        IFX_uint8_t nEvents;
        IFX_uint8_t nPlayEvents;
        IFX_uint8_t nEventPT;
        IFX_uint8_t nEventPlayPT;
} IFX_TAPI_PKT_RTP_CFG_t;
```

Parameters

Data Type	Name	Description
IFX_uint16_t	nSeqNr	Start value for the sequence number.
IFX_uint32_t	nSsrc	Synchronization source value for the voice and SID packets.
IFX_uint32_t	nTimestamp	Reserved.
IFX_uint8_t	nEvents	Defines how to handle RFC 2833 packets in upstream direction. Set parameter as defined in enum IFX_TAPI_PKT_EV_OOB_t.
IFX_uint8_t	nPlayEvents	Defines whether the received RFC 2833 packets have to be played out. Set parameter as defined in enum IFX_TAPI_PKT_EV_OOBPLAY_t.
IFX_uint8_t	nEventPT	Payload type to be used for RFC 2833 frames in encoder direction (upstream).
IFX_uint8_t	nEventPlayPT	Payload type to be used for RFC 2833 frames in decoder direction (downtream).

Remarks

The parameter 'nEventPlayPT' is ignored if the firmware does not support configuration of different payload types for the decoder and encoder.

4.2.4.46 IFX_TAPI_PKT_RTP_PT_CFG_t

Description

Structure for RTP payload type configuration.

Prototype

```
typedef struct
{
        IFX_uint8_t nPTup[MAX_CODECS];
        IFX_uint8_t nPTdown[MAX_CODECS];
} IFX_TAPI_PKT_RTP_PT_CFG_t;
```

Parameters

Data Type	Name	Description
IFX_uint8_t	nPTup[MAX_CODECS]	Table with all payload types, the coder type IFX_TAPI_ENC_TYPE_t is used as index.
IFX_uint8_t	nPTdown[MAX_CODECS]	Table with all payload types, the coder type IFX_TAPI_ENC_TYPE_t is used as index.

4.2.4.47 IFX_TAPI_RING_CADENCE_t

Description

Structure for ring cadence used in IFX_TAPI_RING_CADENCE_HR_SET.

Prototype

```
typedef struct
{
        IFX_char_t data[TAPI_MAX_CADENCE_BYTES];
        IFX_int32_t nr;
        IFX_char_t initial[TAPI_MAX_CADENCE_BYTES];
        IFX_int32_t initialNr;
} IFX_TAPI_RING_CADENCE_t;
```

Parameters

Data Type	Name	Description
IFX_char_t	data[TAPI_MAX_CADENCE_BYTES]	Pointer to data bytes which contain the encoded cadence sequence. One bit represents ring cadence voltage for 50 ms. A maximum of 40 bytes (320 bits) are allowed.
IFX_int32_t	nr	Number of data bits of cadence sequence. A maximum number of 320 data bits is possible which corresponds to a maximum cadence duration of 16 seconds.
IFX_char_t	initial[TAPI_MAX_CADENCE_BYTES]	Pointer to data bytes which contain the encoded cadence sequence for a first non periodic initial ringing. One bit represents ring cadence voltage for 50 ms. A maximum of 40 bytes (320 bits) are allowed.
IFX_int32_t	initialNr	Number of data bits of cadence sequence. A maximum number of 320 data bits is possible which corresponds to a maximum cadence duration of 16 seconds.

4.2.4.48 IFX_TAPI_RING_CFG_t

Description

Ringing configuration structure used for ioctl **IFX_TAPI_RING_CFG_SET**.

Prototype

```
typedef struct
{
        IFX_uint8_t nMode;
        IFX_uint8_t nSubmode;
} IFX_TAPI_RING_CFG_t;
```

Parameters

Data Type	Name	Description
IFX_uint8_t	nMode	Configures the ringing mode. 0_D **INTERNAL_BALANCED** internal balanced 1_D **INTERNAL_UNBALANCED_ROT** internal unbalanced ROT 2_D **INTERNAL_UNBALANCED_ROR** internal unbalanced ROR 3_D **EXTERNAL_IT_CS** external SLIC current sense 4_D **EXTERNAL_IO_CS** external IO current sense
IFX_uint8_t	nSubmode	Configures the ringing submode. 0_D **DC_RNG_TRIP_STANDARD** DC Ring Trip standard 1_D **DC_RNG_TRIP_FAST** DC Ring Trip fast 2_D **AC_RNG_TRIP_STANDARD** AC Ring Trip standard 3_D **AC_RNG_TRIP_FAST** AC Ring Trip fast

4.2.4.49 IFX_TAPI_SIG_DETECTION_t

Description

Structure used for enable and disable signal detection.

Prototype

```
typedef struct
{
        IFX_uint32_t sig;
        IFX_uint32_t sig_ext;
} IFX_TAPI_SIG_DETECTION_t;
```

Parameters

Data Type	Name	Description
IFX_uint32_t	sig	Signals to detect. Can be any combination of IFX_TAPI_SIG_t
IFX_uint32_t	sig_ext	Signals to detect. Can be any combination of IFX_TAPI_SIG_t

4.2.4.50 IFX_TAPI_T38_DEMOD_DATA_t

Description

Structure to setup the demodulator for T.38 fax and used for IFX_TAPI_T38_DEMOD_START.

Prototype

```
typedef struct
{
        IFX_uint8_t nStandard1;
        IFX_uint8_t nStandard2;
        IFX_uint16_t nSigLen;
        IFX_uint16_t nGainRx;
        IFX_uint8_t nEqualizer;
        IFX_uint8_t nTraining;
        IFX_uint16_t nDmbsd;
} IFX_TAPI_T38_DEMOD_DATA_t;
```

Parameters

Data Type	Name	Description
IFX_uint8_t	nStandard1	Selects the standard used for Fax T.38 using IFX_TAPI_T38_STD_t. • 00_H: Silence • 01_H: V.21 • 02_H: V.27/2400 • 03_H: V.27/4800 • 04_H: V.29/7200 • 05_H: V.29/9600 • 06_H: V.17/7200 • 07_H: V.17/9600 • 08_H: V.17/12000
IFX_uint8_t	nStandard2	Selects the alternative standard used for Fax T.38 using IFX_TAPI_T38_STD_t. • 00_H: Silence • 01_H: V.21 • 02_H: V.27/2400 • 03_H: V.27/4800 • 04_H: V.29/7200 • 05_H: V.29/9600 • 06_H: V.17/7200 • 07_H: V.17/9600 • 08_H: V.17/12000 • 09_H: V.17/14400 • FF_H: Use only standard 1.
IFX_uint16_t	nSigLen	Signal duration in ms. Used for the tonal signals (CED, CNG) and silence only. 1 < nSigLen < 4000 [ms]
IFX_uint16_t	nGainRx	Sets the receive gain for the upstream direction.
IFX_uint8_t	nEqualizer	Equalizer configuration flag. • 0: Reset the equalizer • 1: Reuse the equalizer coefficients

Data Type	Name	Description
IFX_uint8_t	nTraining	Training sequence flag, used by V.17 only, ignored in all other cases. • 0: Short training sequence • 1: Long training sequence
IFX_uint16_t	nDmbsd	Level required before the demodulator sends data.

4.2.4.51 IFX_TAPI_T38_MOD_DATA_t

Description

Structure to setup the modulator for T.38 fax and used for IFX_TAPI_T38_MOD_START.

Prototype

```
typedef struct
{
        IFX_uint8_t nStandard;
        IFX_uint16_t nSigLen;
        IFX_uint16_t nGainTx;
        IFX_uint8_t nDbm;
        IFX_uint8_t nTEP;
        IFX_uint8_t nTraining;
        IFX_uint16_t nMobsm;
        IFX_uint16_t nMobrd;
} IFX_TAPI_T38_MOD_DATA_t;
```

Parameters

Data Type	Name	Description
IFX_uint8_t	nStandard	Selects the standard used for Fax T.38. IFX_TAPI_T38_STD_t • 00_H: Silence • 01_H: V.21 • 02_H: V.27/2400 • 03_H: V.27/4800 • 04_H: V.29/7200 • 05_H: V.29/9600 • 06_H: V.17/7200 • 07_H: V.17/9600 • 08_H: V.17/12000 • 09_H: V.17/14400 • $0A_H$: CNG • $0B_H$: CED
IFX_uint16_t	nSigLen	Signal duration in ms. Used for the ton signals (CED, CNG) and silence only. 1 < nSigLen < 4000 [ms]
IFX_uint16_t	nGainTx	Sets the transmit gain for the downstream direction.

Data Type	Name	Description
IFX_uint8_t	nDbm	Desired output signal power in -dBm.
IFX_uint8_t	nTEP	TEP Generation flag, used by V.27, V.29 and V.17 only, ignored in all other cases. • 0: no TEP generation • 1: TEP generation
IFX_uint8_t	nTraining	Training sequence flag, used by V.17 only, ignored in all other cases. • 0: Short training sequence • 1: Long training sequence
IFX_uint16_t	nMobsm	Level required before the modulation starts.
IFX_uint16_t	nMobrd	Level required before the modulation requests more data.

4.2.4.52 IFX_TAPI_T38_STATUS_t

Description

Structure to read the T.38 fax status and used for IFX_TAPI_T38_STATUS_GET.

Prototype

```
typedef struct
{
        IFX_uint8_t nStatus;
        IFX_uint8_t nError;
} IFX_TAPI_T38_STATUS_t;
```

Parameters

Data Type	Name	Description
IFX_uint8_t	nStatus	T.38 fax status, refer to IFX_TAPI_T38_STATUS_t. 0_D **IFX_TAPI_FAX_T38_DP_OFF** Data pump is not active 1_D **IFX_TAPI_FAX_T38_DP_ON** Data pump is active 2_D **IFX_TAPI_FAX_T38_TX_ON** Transmission is active 3_D **IFX_TAPI_FAX_T38_TX_OFF** Transmission is finished.
IFX_uint8_t	nError	T.38 fax error, refer to IFX_TAPI_T38_ERROR_t. 0_D **IFX_TAPI_FAX_T38_NO_ERR** No error occurred 1_D **IFX_TAPI_FAX_T38_ERR** Fax error occurred, the fax data pump should be deactivated 2_D **IFX_TAPI_FAX_T38_MIPS_OVLD** MIPS overload 3_D **IFX_TAPI_FAX_T38_READ_ERR** Error while reading data 4_D **IFX_TAPI_FAX_T38_WRITE_ERR** Error while writing data 5_D **IFX_TAPI_FAX_T38_DATA_ERR** Error while setting up the modulator or demodulator

4.2.4.53 IFX_TAPI_TEST_LOOP_t

Description

Structure for switching loops for testing

Prototype

```
typedef struct
{
       unsigned char bAnalog;
} IFX_TAPI_TEST_LOOP_t;
```

Parameters

Data Type	Name	Description
unsigned char	bAnalog	Switch an analog loop in the device. If switched on, signals that are played to the subscriber are looped back to the receiving side. • 0 IFX_FALSE, Analog loop off • 1 IFX_TRUE, Analog loop on

4.2.4.54 IFX_TAPI_TONE_COMPOSED_t

Description

Structure for definition of composed tones.

Prototype

```
typedef struct
{
        IFX_uint8_t format;
        IFX_uint8_t index;
        IFX_uint8_t loop;
        IFX_uint8_t alternateVoice;
        IFX_uint8_t count;
        IFX_uint8_t tones[TAPI_MAX_SIMPLE_TONES];
} IFX_TAPI_TONE_COMPOSED_t;
```

Parameters

Data Type	Name	Description
IFX_uint8_t	format	Indicate the type of the tone descriptor: 1_D **IFX_TAPI_TONE_TYPE_SIMPLE** the tone descriptor describe a simple tone. 2_D **IFX_TAPI_TONE_TYPE_COMPOSED** the tone descriptor describes a composed tone.
IFX_uint8_t	index	Tone code ID; 0< ID <255.
IFX_uint8_t	loop	Number of times to play the simple tone sequence, 0 for infinite. Maximum 7.
IFX_uint8_t	alternateVoice	Indicate if the voice path is active between the loops.
IFX_uint8_t	count	0_D **IFX_TAPI_TONE_SIMPLE_MAX** Number of simple tones used in the composed tone.
IFX_uint8_t	tones[TAPI_MAX_SIMPLE_TONES]	Simple tones to be used. The simple tones are played in the same order as they are stored in the array. *Note: In order to create composed tones only simple tones with a finite loop count can be used.*

4.2.4.55 IFX_TAPI_TONE_CPTD_t

Description

Structure used for **IFX_TAPI_TONE_CPTD_START** and **IFX_TAPI_TONE_CPTD_STOP**.

Prototype

```
typedef struct
{
        IFX_int32_t tone;
        IFX_int32_t signal;
} IFX_TAPI_TONE_CPTD_t;
```

Parameters

Data Type	Name	Description
IFX_int32_t	tone	The index of the tone to detect. The tone index must be preconfigured in the tone table, before starting tone detection.
IFX_int32_t	signal	The specification of the signal. 1_H **IFX_TAPI_TONE_CPTD_DIRECTION_RX** receive direction of the programmed CPT tone 2_H **IFX_TAPI_TONE_CPTD_DIRECTION_TX** transmit direction of the programmed CPT tone

4.2.4.56 IFX_TAPI_TONE_SIMPLE_t

Description

Struct used to define simple tone characteristics.

Prototype

```
typedef struct
{
        IFX_uint8_t format;
        IFX_uint8_t index;
        IFX_uint8_t loop;
        IFX_int32_t levelA;
        IFX_int32_t levelB;
        IFX_int32_t levelC;
        IFX_int32_t levelD;
        IFX_uint32_t freqA;
        IFX_uint32_t freqB;
        IFX_uint32_t freqC;
        IFX_uint32_t freqD;
        IFX_uint32_t cadence[IFX_TAPI_TONE_STEPS_MAX];
        IFX_uint32_t frequencies[IFX_TAPI_TONE_STEPS_MAX];
        IFX_uint32_t modulation[IFX_TAPI_TONE_STEPS_MAX];
        IFX_uint32_t pause;
} IFX_TAPI_TONE_SIMPLE_t;
```

Parameters

Data Type	Name	Description
IFX_uint8_t	format	Indicate the type of the tone descriptor: 1_D **IFX_TAPI_TONE_TYPE_SIMPLE** the tone descriptor describe a simple tone. 2_D **IFX_TAPI_TONE_TYPE_COMPOSED** the tone descriptor describe a composed tone.
IFX_uint8_t	index	Tone code ID; 0< ID <255.
IFX_uint8_t	loop	Number of times to play the simple tone, 0 < loop < 8.
IFX_int32_t	levelA	Power level for frequency A in 0.1 dB steps; -300 < levelA < 0.
IFX_int32_t	levelB	Power level for frequency B in 0.1 dB steps; -300 < levelB < 0.
IFX_int32_t	levelC	Power level for frequency C in 0.1 dB steps; -300 < levelC < 0.
IFX_int32_t	levelD	Power level for frequency D in 0.1 dB steps; -300 < levelD < 0.
IFX_uint32_t	freqA	Tone frequency A in Hz; 0 <= Hz < 4000.
IFX_uint32_t	freqB	Tone frequency B in Hz; 0 <= Hz < 4000.
IFX_uint32_t	freqC	Tone frequency C in Hz; 0 <= Hz < 4000.
IFX_uint32_t	freqD	Tone frequency D in Hz; 0 <= Hz < 4000.
IFX_uint32_t	cadence[IFX_TAPI_TONE_STEPS_MAX]	$_D$ **IFX_TAPI_TONE_STEPS_MAX** Array defining time duration for each cadence steps, with 2 ms granularity. 0 <= cadence <= 32000. The first cadence[X] = 0 (starting from X=1) in the array indicates that X-1 cadences must be played. A tone with cadence[0]=0 can not be processed!
IFX_uint32_t	frequencies[IFX_TAPI_TONE_STEPS_MAX]	Active frequencies for the cadence steps. More than one frequency can be active in the same cadence step. All active frequencies are summed together. 0_H **IFX_TAPI_TONE_FREQNONE** No frequencies. 1_H **IFX_TAPI_TONE_FREQA** Frequency A is enabled. 2_H **IFX_TAPI_TONE_FREQB** Frequency B is enabled. 3_H **IFX_TAPI_TONE_FREQC** Frequency C is enabled. 4_H **IFX_TAPI_TONE_FREQD** Frequency D is enabled. F_H **IFX_TAPI_TONE_FREQALL** All frequencies are enabled.
IFX_uint32_t	modulation[IFX_TAPI_TONE_STEPS_MAX]	Array containing selection for each cadence steps, whether to enable/disable modulation of frequency A with frequency B. Defined values for each array entry: 0_D **IFX_TAPI_TONE_MODULATION_OFF** Modulation of frequency A with frequency B is disabled. 1_D **IFX_TAPI_TONE_MODULATION_ON** Modulation of frequency A with frequency B is enabled.
IFX_uint32_t	pause	Some tones require an off time at the end of the tone. The offtime will be added to the last used cadence. Therefore the maximum value for offtime is 32000 - “last used cadence” and have the granularity of 2 ms; 0 < 32000 - “last used cadence” < 32000.

4.2.4.57 IFX_TAPI_WLEC_CFG_t

Description

Line echo canceller (LEC) configuration.

Prototype

```
typedef struct
{
        IFX_TAPI_LEC_TYPE_t nType;
        IFX_char_t bNlp;
} IFX_TAPI_WLEC_CFG_t;
```

Parameters

Data Type	Name	Description
IFX_TAPI_LEC_TYPE_t	nType	LEC type selection.
IFX_char_t	bNlp	Switch the NLP on or off. 0_D **TAPI_LEC_NLPDEFAULT** NLP is default 1_D **TAPI_LEC_NLP_ON** NLP is on 2_D **TAPI_LEC_NLP_OFF** NLP is off

4.2.4.58 IFX_TAPI_VERSION_t

Description

Structure used for the TAPI version support check.

Prototype

```
typedef struct
{
        IFX_uint8_t major;
        IFX_uint8_t minor;
} IFX_TAPI_VERSION_t;
```

Parameters

Data Type	Name	Description
IFX_uint8_t	major	Major version number supported
IFX_uint8_t	minor	Minor version number supported

4.2.5 Union Reference

This chapter contains the Union reference.

Table 65 Union Overview of TAPI Interfaces

Name	Description
IFX_TAPI_CID_MSG_ELEMENT_t	CID Message Element
IFX_TAPI_CID_STD_TYPE_t	CID Message Element

Table 65 Union Overview of TAPI Interfaces (cont'd)

Name	Description
IFX_TAPI_EXCEPTION_t	Contains TAPI exception information.
IFX_TAPI_TONE_t	Tone descriptor.

4.2.5.1 IFX_TAPI_CID_MSG_ELEMENT_t

Description

Union of element types

Prototype

```
typedef union
{
        IFX_TAPI_CID_MSG_DATE_t date;
        IFX_TAPI_CID_MSG_STRING_t string;
        IFX_TAPI_CID_MSG_VALUE_t value;
        IFX_TAPI_CID_MSG_TRANSPARENT_t transparent;
} IFX_TAPI_CID_MSG_ELEMENT_t;
```

Parameters

Data Type	Name	Description
IFX_TAPI_CID_MSG_DATE_t	date	Message element including date and time information.
IFX_TAPI_CID_MSG_STRING_t	string	Message element formatted as string.
IFX_TAPI_CID_MSG_VALUE_t	value	Message element formatted as value.
IFX_TAPI_CID_MSG_TRANSPARENT_t	transparent	Message element to be sent with transparent transmission.

4.2.5.2 IFX_TAPI_CID_STD_TYPE_t

Description

Union of the CID configuration structures for different standards.

Prototype

```
typedef union
{
        IFX_TAPI_CID_STD_TELCORDIA_t telcordia;
        IFX_TAPI_CID_STD_ETSI_FSK_t etsiFSK;
        IFX_TAPI_CID_STD_ETSI_DTMF_t etsiDTMF;
        IFX_TAPI_CID_STD_SIN_t sin;
        IFX_TAPI_CID_STD_NTT_t ntt;
} IFX_TAPI_CID_STD_TYPE_t;
```

Parameters

Data Type	Name	Description
IFX_TAPI_CID_STD_TELCORDIA_t	telcordia	Structure defining configuration parameters for Telcordia standard.
IFX_TAPI_CID_STD_ETSI_FSK_t	etsiFSK	Structure defining configuration parameters for ETSI standard, with FSK transmission.
IFX_TAPI_CID_STD_ETSI_DTMF_t	etsiDTMF	Structure defining configuration parameters for ETSI standard, with DTMF transmission.
IFX_TAPI_CID_STD_SIN_t	sin	Structure defining configuration parameters for BT SIN standard.
IFX_TAPI_CID_STD_NTT_t	ntt	Structure defining configuration parameters for NTT standard.

4.2.5.3 IFX_TAPI_EXCEPTION_t

Description

Contains TAPI exception information.

Prototype

```
typedef union
{
        IFX_TAPI_EXCEPTION_BITS_t Bits;
        IFX_uint32_t Status;
} IFX_TAPI_EXCEPTION_t;
```

Parameters

Data Type	Name	Description
IFX_TAPI_EXCEPTION_BITS_t	Bits	Specifies the exception.
IFX_uint32_t	Status	Contains the complete status.

4.2.5.4 IFX_TAPI_EVENT_DATA_t

Description

Union for the possible events to be reported.

Prototype

```
typedef union
{
        IFX_TAPI_EVENT_DATA_PULSE_t pulse;
        IFX_TAPI_EVENT_DATA_DTMF_t dtmf;
        IFX_TAPI_EVENT_DATA_TONE_t tone;
        IFX_TAPI_EVENT_DATA_FAX_SIG_t fax_sig;
        IFX_TAPI_EVENT_DATA_RFC2833_t rfc2833;
        IFX_int16_t value;
        IFX_TAPI_EVENT_DATA_EXT_KEYPAD_t keyinfo;
```

```
} IFX_TAPI_EVENT_DATA_t;
```

Parameters

Data Type	Name	Description
IFX_TAPI_EVENT_DATA_PULSE_t	pulse	Pulse digit information.
IFX_TAPI_EVENT_DATA_DTMF_t	dtmf	DTMF digit information.
IFX_TAPI_EVENT_DATA_TONE_t	tone	Tone generation information.
IFX_TAPI_EVENT_DATA_FAX_SIG_t	fax_sig	Fax/modem signal information.
IFX_TAPI_EVENT_DATA_RFC2833_t	rfc2833	RFC2833 event information.
IFX_int16_t	value	Reserved.
IFX_TAPI_EVENT_DATA_EXT_KEYPAD_t	keyinfo	External keypad event information.

4.2.5.5 IFX_TAPI_TONE_t

Description

This chapter define a union for the tone descriptor.

Prototype

```
typedef union
{
        IFX_TAPI_TONE_SIMPLE_t simple;
        IFX_TAPI_TONE_COMPOSED_t composed;
} IFX_TAPI_TONE_t;
```

Parameters

Data Type	Name	Description
IFX_TAPI_TONE_SIMPLE_t	simple	The parameter points to a IFX_TAPI_TONE_SIMPLE_t structure.
IFX_TAPI_TONE_COMPOSED_t	composed	The parameter points to a IFX_TAPI_TONE_COMPOSED_t structure.

4.2.6 Enumerator Reference

This chapter contains the Enumerator reference.

Table 66 Enumerator Overview of TAPI Interfaces

Name	Description
DEV_ERR	Error codes
IFX_TAPI_AUDIO_MODE_t	Lists the ports for the capability list.

Table 66 Enumerator Overview of TAPI Interfaces (cont'd)

Name	Description
IFX_TAPI_CAP_SIG_DETECT_t	Lists the signal detectors for the capability list.
IFX_TAPI_CAP_TYPE_t	Enumeration used for phone capabilities types.
IFX_TAPI_CH_INIT_COUNTRY_t	Country selection for IFX_TAPI_CH_INIT_t.
IFX_TAPI_CH_INIT_MODE_t	TAPI Initialization modes, selection for target system.
IFX_TAPI_CID_ABSREASON_t	ABSCLI/ABSNAME settings.
IFX_TAPI_CID_ALERT_ETSI_t	List of ETSI Alerts.
IFX_TAPI_CID_HOOK_MODE_t	Caller ID transmission modes.
IFX_TAPI_CID_MSG_TYPE_t	Caller ID message types (defined in ETSI EN 300 659-3).
IFX_TAPI_CID_RX_ERROR_t	CID Receiver Errors.
IFX_TAPI_CID_RX_STATE_t	CID Receiver Status.
IFX_TAPI_CID_SERVICE_TYPE_t	Caller ID Services (defined in ETSI EN 300 659-3).
IFX_TAPI_CID_STD_t	List of CID standards.
IFX_TAPI_CID_VMWI_t	List of VMWI settings.
IFX_TAPI_DATA_MAP_START_STOP_t	Start/Stop information for data channel mapping.
IFX_TAPI_DIALING_STATUS_t	Enumeration for dial status events.
IFX_TAPI_DWLD_TYPE_t	Definition of download type.
IFX_TAPI_ENC_TYPE_t	Possible codecs to select for IFX_TAPI_ENC_TYPE_SET and IFX_TAPI_PCK_AAL_PROFILE_t.
IFX_TAPI_ENC_VAD_t	Enumeration used for ioctl IFX_TAPI_ENC_VAD_CFG_SET.
IFX_TAPI_EVENT_ID_t	Event ID.
IFX_TAPI_EVENT_TYPE_t	Event type.
IFX_TAPI_JB_LOCAL_ADAPT_t	Jitter buffer adoption.
IFX_TAPI_JB_PKT_ADAPT_t	Jitter buffer packet adaptation.
IFX_TAPI_JB_TYPE_t	Jitter buffer type.
IFX_TAPI_LEC_GAIN_t	LEC Gain Levels.
IFX_TAPI_LEC_NLP_t	LEC NLP (Non Linear Processor) Settings.
IFX_TAPI_LEC_TYPE_t	LEC type selection.
IFX_TAPI_LINE_FEED_t	Defines for linefeeding.
IFX_TAPI_LINE_HOOK_STATUS_t	Enumeration for hook status events.
IFX_TAPI_LINE_HOOK_VALIDATION_TYPE_t	Validation types used for structure IFX_TAPI_LINE_HOOK_VT_t.
IFX_TAPI_LINE_LEVEL_t	Specifies the Enable/Disable mode of the high level.
IFX_TAPI_LINE_STATUS_t	Enumeration for phone line status information.
IFX_TAPI_MAP_DATA_TYPE_t	Data channel destination types (conferencing).
IFX_TAPI_MAP_DEC_t	Play out enabling and disabling information.
IFX_TAPI_MAP_ENC_t	Recording enabling and disabling information.
IFX_TAPI_MAP_TYPE_t	Type channel for mapping.
IFX_TAPI_METER_MODE_t	Metering Modes.
IFX_TAPI_PCM_RES_t	PCM resolutions.
IFX_TAPI_PKT_AAL_PROFILE_RANGE_t	Used for IFX_TAPI_PCK_AAL_PROFILE_t in case one coder range.

Table 66 Enumerator Overview of TAPI Interfaces (cont'd)

Name	Description
IFX_TAPI_PKT_EV_OOB_t	Out of band or in band definition.
IFX_TAPI_RING_CFG_MODE_t	Ring Configuration mode.
IFX_TAPI_RING_CFG_SUBMODE_t	Ring Configuration sub-mode.
IFX_TAPI_RUNTIME_ERROR_t	TAPI Runtime Errors.
IFX_TAPI_SIG_t	List the tone detection options.
IFX_TAPI_T38_ERROR_t	T38 Fax errors.
IFX_TAPI_T38_STATUS_t	T38 Fax Datapump states.
IFX_TAPI_T38_STD_t	T38 Fax standards.
IFX_TAPI_TONE_CPTD_DIRECTION_t	Specifies the CPT signal for CPT detection.
IFX_TAPI_TONE_FREQ_t	Frequency setting for a cadence step.
IFX_TAPI_TONE_GROUP_t	Tone grouping.
IFX_TAPI_TONE_MODULATION_t	Modulation setting for a cadence step.
IFX_TAPI_TONE_TG_t	Defines the tone generator usage.
IFX_TAPI_TONE_TYPE_t	Tone types.
IFX_TAPI_WLEC_NLP_t	NLP configuration.
IFX_TAPI_WLEC_TYPE_t	WLEC type configuration.

4.2.6.1 DEV_ERR

Description

Low level driver error codes.

Attention: The error codes listed in this enum are not defined for each Infineon product, for more details please refer to the product specific documentation.

Prototype

```
typedef enum
{
        ERR_OK = 0,
        ERR_CERR = 0x01,
        ERR_CIBX_OF = 0x2,
        ERR_HOST = 0x3,
        ERR_MIPS_OL = 0x4,
        ERR_NO_COBX = 0x5,
        ERR_NO_DATA = 0x6,
        ERR_NO_FIBXMS = 0x7,
        ERR_MORE_DATA = 0x8,
        ERR_NO_MBXEMPTY = 0x9,
        ERR_NO_DLRDY = 0xA,
        ERR_WRONGDATA = 0xB,
        ERR_OBXML_ZERO = 0xC,
        ERR_TEST_FAIL = 0xD,
        ERR_HW_ERR = 0xE,
        ERR_PIBX_OF = 0xF,
        ERR_FUNC_PARM = 0x10,
        ERR_TO_CHSTATE = 0x11,
```

```
        ERR_BUF_UN = 0x12,
        ERR_NO_MEM = 0x13,
        ERR_NOINIT = 0x14,
        ERR_INTSTUCK = 0x15,
        ERR_LT_ON = 0x16,
        ERR_NOPHI = 0x17,
        ERR_EDSP_FAIL = 0x18,
        ERR_FWCRC_FAIL = 0x19,
        ERR_NO_TAPI = 0x1A,
        ERR_SPI = 0x1B,
        ERR_INVALID = 0x1C,
        ERR_GR909 = 0x1D,
        ERR_ACCRC_FAIL = 0x1E,
        ERR_NO_VERSION = 0x1F,
        ERR_DCCRC_FAIL = 0x20,
        ERR_UNKNOWN_VERSION = 0x21,
        ERR_LT_LINE_IS_PDNH = 0x22,
        ERR_LT_UNKNOWN_PARAM = 0x23,
        ERR_CID_TRANSMIT = 0x24,
        ERR_LT_TIMEOUT_LM_OK = 0x25,
        ERR_LT_TIMEOUT_LM_RAMP_RDY = 0x26,
        ERR_PRAM_FW = 0x27,
        ERR_NOFW = 0x28,
        ERR_PHICRC0 = 0x29,
        ERR_ARCDWLD_FAIL = 0x2A,
        ERR_ARCDWLD_BOOT = 0x2B,
        ERR_FWINVALID = 0x2C,
        ERR_NOFWVERS = 0x2D,
        ERR_NOMAXCBX = 0x2E,
        ERR_SIGMOD_NOTEN = 0x2F,
        ERR_SIGCH_NOTEN = 0x30,
        ERR_CODCONF_NOTVALID = 0x31,
        ERR_LT_OPTRES_FAILED = 0x32,
        ERR_NO_FREE_INPUT_SLOT = 0x33,
        ERR_NOTSUPPORTED = 0x34,
        ERR_NORESOURCE = 0x35,
        ERR_WRONG_EVPT = 0x36,
        ERR_CON_INVALID = 0x37,
        ERR_HOSTREG_ACCESS = 0x38,
        ERR_NOPKT_BUFF = 0x39,
        ERR_COD_RUNNING = 0x3A,
        ERR_TONE_PLAYING = 0x3B,
        ERR_INVALID_TONERES = 0x3C,
        ERR_INVALID_SIGSTATE = 0x3D,
        ERR_INVALID_UTGSTATE = 0x3E,
        ERR_CID_RUNNING = 0x3F,
        ERR_UNKNOWN = 0x40,
        ERR_WRONG_CHANNEL_MODE = 0x41,
        ERR_DRVINIT_FAIL = 0x80,
        ERR_DEV_ERR = 0x81
} DEV_ERR_t;
```

Parameters

Name	Value	Description
ERR_OK	0	0x0: no error
ERR_CERR	01_H	Command error, see last command
ERR_CIBX_OF	2_H	Command inbox overflow
ERR_HOST	3_H	Host error
ERR_MIPS_OL	4_H	MIPS overload.
ERR_NO_COBX	5_H	No command data received event within time-out. This error is obsolete, since the driver used a polling mode
ERR_NO_DATA	6_H	No command data received within time-out
ERR_NO_FIBXMS	7_H	Not enough inbox space for writing command
ERR_MORE_DATA	8_H	More data then expected in outbox
ERR_NO_MBXEMPTY	9_H	Mailbox was not empty after time-out. This error occurs while the driver tries to switch the mailbox sizes before and after the firmware download. The time-out is given in the constant WAIT_MBX_EMPTY
ERR_NO_DLRDY	A_H	Download ready event has not occurred
ERR_WRONGDATA	B_H	Register read: expected values do not match
ERR_OBXML_ZERO	C_H	OBXML is zero after COBX-DATA event. This error is obsolete, since the driver is polling the OBXML register, i.e. the COBX-DATA event is not handled anymore in the interrupt routine.
ERR_TEST_FAIL	D_H	Test chip access failed.
ERR_HW_ERR	E_H	Internal EDSP hardware error reported in HWSR1:HW-ERR.
ERR_PIBX_OF	F_H	Mailbox Overflow Error.
ERR_FUNC_PARM	10_H	Invalid parameter in function call
ERR_TO_CHSTATE	11_H	Time-out while waiting on channel status change
ERR_BUF_UN	12_H	Buffer under run in evaluation down streaming
ERR_NO_MEM	13_H	No memory by memory allocation
ERR_NOINIT	14_H	Board previously not initialized
ERR_INTSTUCK	15_H	Interrupts can not be cleared
ERR_LT_ON	16_H	Line testing measurement is running
ERR_NOPHI	17_H	PHI patch was not successfully downloaded. The problem was a chip access problem
ERR_EDSP_FAIL	18_H	EDSP Failures.
ERR_FWCRC_FAIL	19_H	CRC Fail while FW download.
ERR_NO_TAPI	$1A_H$	TAPI not initialized.
ERR_SPI	$1B_H$	Error while using SPI Interface.
ERR_INVALID	$1C_H$	Inconsistent or invalid parameters were provided
ERR_GR909	$1D_H$	No Data to copy to user space for GR909 measurement

Name	Value	Description
ERR_ACCRC_FAIL	$1E_H$	CRC Fail while ALM-DSP download for V1.4.
ERR_NO_VERSION	$1F_H$	Couldn't read out chip version
ERR_DCCRC_FAIL	20_H	CRC Fail in DCCTRL download.
ERR_UNKNOWN_VERSION	21_H	Unknown chip version
ERR_LT_LINE_IS_PDNH	22_H	Linetesting, line is in Power Down High Impedance, measurement not possible.
ERR_LT_UNKNOWN_PARAM	23_H	Linetesting, unknown Parameter.
ERR_CID_TRANSMIT	24_H	Error while sending CID.
ERR_LT_TIMEOUT_LM_OK	25_H	Linetesting, time-out waiting for LM_OK.
ERR_LT_TIMEOUT_LM_RAMP_RDY	26_H	Linetesting, time-out waiting for RAMP_RDY
ERR_PRAM_FW	27_H	Invalid pram fw length
ERR_NOFW	28_H	No firmware specified and not included in driver
ERR_PHICRC0	29_H	PHI CRC is zero.
ERR_ARCDWLD_FAIL	$2A_H$	Embedded Controller download failed.
ERR_ARCDWLD_BOOT	$2B_H$	Embedded Controller boot failed after download.
ERR_FWINVALID	$2C_H$	Firmware binary is invalid.
ERR_NOFWVERS	$2D_H$	Firmware version could not be read, no answer to command.
ERR_NOMAXCBX	$2E_H$	Maximize mailbox failed.
ERR_SIGMOD_NOTEN	$2F_H$	Signaling module not enabled.
ERR_SIGCH_NOTEN	30_H	Signaling channel not enabled.
ERR_CODCONF_NOTVALID	31_H	Coder configuration not valid
ERR_LT_OPTRES_FAILED	32_H	Linetesting, optimum result routine failed.
ERR_NO_FREE_INPUT_SLOT	33_H	No free input found while connecting cod, sig and alm modules.
ERR_NOTSUPPORTED	34_H	Feature or combination not supported
ERR_NORESOURCE	35_H	Resource not available
ERR_WRONG_EVPT	36_H	Event payload type mismatch.
ERR_CON_INVALID	37_H	Connection not valid on remove.
ERR_HOSTREG_ACCESS	38_H	Host register access failure.
ERR_NOPKT_BUFF	39_H	No packet buffers available.
ERR_COD_RUNNING	$3A_H$	At least one parameter is not possible to apply when the coder is running. Event payload types cannot be changed on the fly.
ERR_TONE_PLAYING	$3B_H$	Tone is already played out on this channel.
ERR_INVALID_TONERES	$3C_H$	Tone resource is not capable playing out a certain tone. This error should not occur -> internal mismatch.
ERR_INVALID_SIGSTATE	$3D_H$	Invalid state for switching off signaling modules. Internal error.
ERR_INVALID_UTGSTATE	$3E_H$	Invalid state for switching off signaling modules. Internal error.
ERR_CID_RUNNING	$3F_H$	Cid sending is ongoing in this channel.

Name	Value	Description
ERR_UNKNOWN	40_H	Some internal state occured, that could not be handled. This error should never occur.
ERR_WRONG_CHANNEL_MODE	41_H	Action not supported with this TAPI initialisation mode.
ERR_DRVINIT_FAIL	80_H	Driver initialization failed
ERR_DEV_ERR	81_H	General access error, RDQ bit is always 1

4.2.6.2 IFX_TAPI_AUDIO_MODE_t

Description

Audio modes for the audio channel.

Prototype

```
typedef enum
{
        IFX_TAPI_AUDIO_MODE_DISABLED = 0,
        IFX_TAPI_AUDIO_MODE_HANDSET = 1,
        IFX_TAPI_AUDIO_MODE_HANDSET_OPENL = 2,
        IFX_TAPI_AUDIO_MODE_HEADSET = 3,
        IFX_TAPI_AUDIO_MODE_HEADSET_OPENL = 4,
        IFX_TAPI_AUDIO_MODE_HANDSFREE = 5
} IFX_TAPI_AUDIO_MODE_t;
```

Parameters

Name	Value	Description
IFX_TAPI_AUDIO_MODE_DISABLED	0_D	Audio channel is disabled.
IFX_TAPI_AUDIO_MODE_HANDSET	1_D	Handset mode.
IFX_TAPI_AUDIO_MODE_HANDSET_OPENL	2_D	Handset mode with open listening.
IFX_TAPI_AUDIO_MODE_HEADSET	3_D	Headset mode.
IFX_TAPI_AUDIO_MODE_HEADSET_OPENL	4_D	Headset mode with open listening.
IFX_TAPI_AUDIO_MODE_HANDSFREE	5_D	Hands-free mode.

4.2.6.3 IFX_TAPI_AUDIO_ROOM_TYPE_t

Description

Audio modes.

Prototype

```
typedef enum
{
        IFX_TAPI_AUDIO_ROOM_TYPE_MUFFLED = 1,
        IFX_TAPI_AUDIO_ROOM_TYPE_MEDIUM = 2,
        IFX_TAPI_AUDIO_ROOM_TYPE_ECHOIC = 3
```

```
} IFX_TAPI_AUDIO_ROOM_TYPE_t;
```

Parameters

Name	Value	Description
IFX_TAPI_AUDIO_ROOM_TYPE_MUFFLED	1_D	Muffled room, low echo level.
IFX_TAPI_AUDIO_ROOM_TYPE_MEDIUM	2_D	Medium echo level.
IFX_TAPI_AUDIO_ROOM_TYPE_ECHOIC	3_D	Echoic room, high echo level.

4.2.6.4 IFX_TAPI_CAP_PORT_t

Description

Lists the ports for the capability list.

Prototype

```
typedef enum
{
        IFX_TAPI_CAP_PORT_POTS = 0,
        IFX_TAPI_CAP_PORT_PSTN = 1,
        IFX_TAPI_CAP_PORT_HANDSET = 2,
        IFX_TAPI_CAP_PORT_SPEAKER = 3
} IFX_TAPI_CAP_PORT_t;
```

Parameters

Name	Value	Description
IFX_TAPI_CAP_PORT_POTS	0_D	POTS port available
IFX_TAPI_CAP_PORT_PSTN	1_D	PSTN port available
IFX_TAPI_CAP_PORT_HANDSET	2_D	Handset port available
IFX_TAPI_CAP_PORT_SPEAKER	3_D	Speaker port available

4.2.6.5 IFX_TAPI_CAP_SIG_DETECT_t

Description

Lists the signal detectors for the capability list.

Prototype

```
typedef enum
{
        IFX_TAPI_CAP_SIG_DETECT_CNG = 0,
        IFX_TAPI_CAP_SIG_DETECT_CED = 1,
        IFX_TAPI_CAP_SIG_DETECT_DIS = 2,
        IFX_TAPI_CAP_SIG_DETECT_POWER = 3,
        IFX_TAPI_CAP_SIG_DETECT_CPTD = 4,
        IFX_TAPI_CAP_SIG_DETECT_V8BIS = 5
} IFX_TAPI_CAP_SIG_DETECT_t;
```

Parameters

Name	Value	Description
IFX_TAPI_CAP_SIG_DETECT_CNG	0_D	Signal detection for CNG is available.
IFX_TAPI_CAP_SIG_DETECT_CED	1_D	Signal detection for CED is available.
IFX_TAPI_CAP_SIG_DETECT_DIS	2_D	Signal detection for DIS is available.
IFX_TAPI_CAP_SIG_DETECT_POWER	3_D	Signal detection for line power is available.
IFX_TAPI_CAP_SIG_DETECT_CPTD	4_D	Signal detection for CPT is available.
IFX_TAPI_CAP_SIG_DETECT_V8BIS	5_D	Signal detection for V8.bis is available.

4.2.6.6 IFX_TAPI_CAP_TYPE_t

Description

Enumeration used for phone capabilities types.

Prototype

```
typedef enum
{
        IFX_TAPI_CAP_TYPE_VENDOR = 0,
        IFX_TAPI_CAP_TYPE_DEVICE = 1,
        IFX_TAPI_CAP_TYPE_PORT = 2,
        IFX_TAPI_CAP_TYPE_CODEC = 3,
        IFX_TAPI_CAP_TYPE_DSP = 4,
        IFX_TAPI_CAP_TYPE_PCM = 5,
        IFX_TAPI_CAP_TYPE_CODECS = 6,
        IFX_TAPI_CAP_TYPE_PHONES = 7,
        IFX_TAPI_CAP_TYPE_SIG = 8,
        IFX_TAPI_CAP_TYPE_T38 = 9
} IFX_TAPI_CAP_TYPE_t;
```

Parameters

Name	Value	Description
IFX_TAPI_CAP_TYPE_VENDOR	0_D	Capability type: representation of the vendor.
IFX_TAPI_CAP_TYPE_DEVICE	1_D	Underlying device Capability type: representation of the
IFX_TAPI_CAP_TYPE_PORT	2_D	Capability type: information about available ports
IFX_TAPI_CAP_TYPE_CODEC	3_D	Capability type: vocoder type
IFX_TAPI_CAP_TYPE_DSP	4_D	Capability type: DSP functionality available
IFX_TAPI_CAP_TYPE_PCM	5_D	Capability type: number of PCM modules
IFX_TAPI_CAP_TYPE_CODECS	6_D	Capability type: number of coder modules
IFX_TAPI_CAP_TYPE_PHONES	7_D	Capability type: number of analog interfaces
IFX_TAPI_CAP_TYPE_SIG	8_D	Capability type: number of signaling modules
IFX_TAPI_CAP_TYPE_T38	9_D	Capability type: T.38 support

4.2.6.7 IFX_TAPI_CH_INIT_COUNTRY_t

Description

Country selection for **IFX_TAPI_CH_INIT_t**.

For future purposes, not yet used. If different countries are supported by the implementation, this parameter specifies which one.

Prototype

```
typedef enum
{
        IFX_TAPI_CH_INIT_COUNTRY_DEFAULT = 0,
        IFX_TAPI_CH_INIT_COUNTRY_DE = 1,
        IFX_TAPI_CH_INIT_COUNTRY_US = 2,
        IFX_TAPI_CH_INIT_COUNTRY_UK = 3
} IFX_TAPI_CH_INIT_COUNTRY_t;
```

Parameters

Name	Value	Description
IFX_TAPI_CH_INIT_COUNTRY_DEFAULT	0_D	Default country.
IFX_TAPI_CH_INIT_COUNTRY_DE	1_D	Germany.
IFX_TAPI_CH_INIT_COUNTRY_US	2_D	USA.
IFX_TAPI_CH_INIT_COUNTRY_UK	3_D	United Kingdom.

4.2.6.8 IFX_TAPI_CH_INIT_MODE_t

Description

TAPI Initialization modes, selection for target system.

If different modes are supported by the implementation, this parameter specifies which mode should be set up. The meaning of the mode is dependent on the implementation.

Prototype

```
typedef enum
{
        IFX_TAPI_INIT_MODE_DEFAULT = 0,
        IFX_TAPI_INIT_MODE_VOICE_CODER = 1,
        IFX_TAPI_INIT_MODE_PCM_DSP = 2,
        IFX_TAPI_INIT_MODE_PCM_PHONE = 3,
        IFX_TAPI_INIT_MODE_NONE = 0xFF
} IFX_TAPI_CH_INIT_MODE_t;
```

Parameters

Name	Value	Description
IFX_TAPI_INIT_MODE_DEFAULT	0_D	Default initialization.
IFX_TAPI_INIT_MODE_VOICE_CODER	1_D	Typical VoIP solution. Phone connected to a packet coder (data channel) with DSP features for signal detection
IFX_TAPI_INIT_MODE_PCM_DSP	2_D	Phone to PCM using DSP features for signal detection.
IFX_TAPI_INIT_MODE_PCM_PHONE	3_D	Phone to PCM not using DSP features for signal detection.
IFX_TAPI_INIT_MODE_NONE	FF_H	Phone to PCM connection without DSP features.

4.2.6.9 IFX_TAPI_CID_ABSREASON_t

Description

List of ABSCLI/ABSNAME settings.

Prototype

```
typedef enum
{
        IFX_TAPI_CID_ABSREASON_UNAV = 0x4F,
        IFX_TAPI_CID_ABSREASON_PRIV = 0x50
} IFX_TAPI_CID_ABSREASON_t;
```

Parameters

Name	Value	Description
IFX_TAPI_CID_ABSREASON_UNAV	$4F_H$	Unavailable/Unknown.
IFX_TAPI_CID_ABSREASON_PRIV	50_H	Private.

4.2.6.10 IFX_TAPI_CID_ALERT_ETSI_t

Description

List of ETSI Alerts.

Prototype

```
typedef enum
{
        IFX_TAPI_CID_ALERT_ETSI_FR = 0x0,
        IFX_TAPI_CID_ALERT_ETSI_DTAS = 0x1,
        IFX_TAPI_CID_ALERT_ETSI_RP = 0x2,
        IFX_TAPI_CID_ALERT_ETSI_LRDTAS = 0x3
} IFX_TAPI_CID_ALERT_ETSI_t;
```

Parameters

Name	Value	Description
IFX_TAPI_CID_ALERT_ETSI_FR	0_H	First Ring Burst. *Note: Defined only CID transmission associated with ringing.*
IFX_TAPI_CID_ALERT_ETSI_DTAS	1_H	DTAS.
IFX_TAPI_CID_ALERT_ETSI_RP	2_H	Ring Pulse.
IFX_TAPI_CID_ALERT_ETSI_LRDTAS	3_H	Line Reversal (alias Polarity Reversal) followed by DTAS.

4.2.6.11 IFX_TAPI_CID_HOOK_MODE_t

Description

Caller ID transmission modes.

Prototype

```
typedef enum
{
        IFX_TAPI_CID_HM_ONHOOK = 0x0,
        IFX_TAPI_CID_HM_OFFHOOK = 0x1
} IFX_TAPI_CID_HOOK_MODE_t;
```

Parameters

Name	Value	Description
IFX_TAPI_CID_HM_ONHOOK	0_H	On-hook transmission. Applicable to CID type 1 and MWI
IFX_TAPI_CID_HM_OFFHOOK	1_H	Off-hook transmission. Applicable to CID type 2 and MWI

Remarks

Information required especially for FSK framing.

4.2.6.12 IFX_TAPI_CID_MSG_TYPE_t

Description

Caller ID Message Type defined in ETSI EN 300 659-3.

Prototype

```
typedef enum
{
        IFX_TAPI_CID_MT_TRANSPARENT = 0x00,
        IFX_TAPI_CID_MT_CSUP = 0x80,
        IFX_TAPI_CID_MT_MWI = 0x82,
        IFX_TAPI_CID_MT_AOC = 0x86,
```

```
        IFX_TAPI_CID_MT_SMS = 0x89,
        IFX_TAPI_CID_MT_RES01 = 0xF1,
        IFX_TAPI_CID_MT_RES02 = 0xF2,
        IFX_TAPI_CID_MT_RES03 = 0xF3,
        IFX_TAPI_CID_MT_RES04 = 0xF4,
        IFX_TAPI_CID_MT_RES05 = 0xF5,
        IFX_TAPI_CID_MT_RES06 = 0xF6,
        IFX_TAPI_CID_MT_RES07 = 0xF7,
        IFX_TAPI_CID_MT_RES08 = 0xF8,
        IFX_TAPI_CID_MT_RES09 = 0xF9,
        IFX_TAPI_CID_MT_RES0A = 0xFA,
        IFX_TAPI_CID_MT_RES0B = 0xFB,
        IFX_TAPI_CID_MT_RES0C = 0xFC,
        IFX_TAPI_CID_MT_RES0D = 0xFD,
        IFX_TAPI_CID_MT_RES0E = 0xFE,
        IFX_TAPI_CID_MT_RES0F = 0xFF
} IFX_TAPI_CID_MSG_TYPE_t;
```

Parameters

Name	Value	Description
IFX_TAPI_CID_MT_TRANSPARENT	00_H	Transparent trasmission.
IFX_TAPI_CID_MT_CSUP	80_H	Call Set-up. Corresponds to Caller ID type 1 and type 2.
IFX_TAPI_CID_MT_MWI	82_H	Message Waiting Indicator
IFX_TAPI_CID_MT_AOC	86_H	Advice of Charge
IFX_TAPI_CID_MT_SMS	89_H	Short Message Service
IFX_TAPI_CID_MT_RES01	$F1_H$	Reserved for Network Operator Use
IFX_TAPI_CID_MT_RES02	$F2_H$	Reserved for Network Operator Use
IFX_TAPI_CID_MT_RES03	$F3_H$	Reserved for Network Operator Use
IFX_TAPI_CID_MT_RES04	$F4_H$	Reserved for Network Operator Use
IFX_TAPI_CID_MT_RES05	$F5_H$	Reserved for Network Operator Use
IFX_TAPI_CID_MT_RES06	$F6_H$	Reserved for Network Operator Use
IFX_TAPI_CID_MT_RES07	$F7_H$	Reserved for Network Operator Use
IFX_TAPI_CID_MT_RES08	$F8_H$	Reserved for Network Operator Use
IFX_TAPI_CID_MT_RES09	$F9_H$	Reserved for Network Operator Use
IFX_TAPI_CID_MT_RES0A	FA_H	Reserved for Network Operator Use
IFX_TAPI_CID_MT_RES0B	FB_H	Reserved for Network Operator Use
IFX_TAPI_CID_MT_RES0C	FC_H	Reserved for Network Operator Use
IFX_TAPI_CID_MT_RES0D	FD_H	Reserved for Network Operator Use
IFX_TAPI_CID_MT_RES0E	FE_H	Reserved for Network Operator Use
IFX_TAPI_CID_MT_RES0F	FF_H	Reserved for Network Operator Use

Remarks

Implemented only **IFX_TAPI_CID_MT_TRANSPARENT** (for Caller ID type 1 and type 2) and **IFX_TAPI_CID_MT_MWI** (for Message Waiting).

4.2.6.13 IFX_TAPI_CID_RX_ERROR_t

Description

CID Receiver Errors.

Prototype

```
typedef enum
{
        IFX_TAPI_CID_RX_ERROR_NONE = 0,
        IFX_TAPI_CID_RX_ERROR_READ = 1
} IFX_TAPI_CID_RX_ERROR_t;
```

Parameters

Name	Value	Description
IFX_TAPI_CID_RX_ERROR_NONE	0_D	No Error during CID Receiver operation.
IFX_TAPI_CID_RX_ERROR_READ	1_D	Reading error during CID Receiver operation.

4.2.6.14 IFX_TAPI_CID_RX_STATE_t

Description

CID Receiver Status.

Prototype

```
typedef enum
{
        IFX_TAPI_CIDRX_STAT_INACTIVE = 0,
        IFX_TAPI_CIDRX_STAT_ACTIVE = 1,
        IFX_TAPI_CIDRX_STAT_ONGOING = 2,
        IFX_TAPI_CIDRX_STAT_DATA_RDY = 3
} IFX_TAPI_CID_RX_STATE_t;
```

Parameters

Name	Value	Description
IFX_TAPI_CIDRX_STAT_INACTIVE	0_D	CID Receiver is not active.
IFX_TAPI_CIDRX_STAT_ACTIVE	1_D	CID Receiver is active.
IFX_TAPI_CIDRX_STAT_ONGOING	2_D	CID Receiver is just receiving data.
IFX_TAPI_CIDRX_STAT_DATA_RDY	3_D	CID Receiver is completed.

4.2.6.15 IFX_TAPI_CID_SERVICE_TYPE_t

Description

Caller ID Services (defined in ETSI EN 300 659-3).

Prototype

```
typedef enum
```

```
{
        IFX_TAPI_CID_ST_DATE = 0x01,
        IFX_TAPI_CID_ST_CLI = 0x02,
        IFX_TAPI_CID_ST_CDLI = 0x03,
        IFX_TAPI_CID_ST_ABSCLI = 0x04,
        IFX_TAPI_CID_ST_NAME = 0x07,
        IFX_TAPI_CID_ST_ABSNAME = 0x08,
        IFX_TAPI_CID_ST_VISINDIC = 0x0B,
        IFX_TAPI_CID_ST_MSGIDENT = 0x0D,
        IFX_TAPI_CID_ST_LMSGCLI = 0x0E,
        IFX_TAPI_CID_ST_CDATE = 0x0F,
        IFX_TAPI_CID_ST_CCLI = 0x10,
        IFX_TAPI_CID_ST_CT = 0x11,
        IFX_TAPI_CID_ST_FIRSTCLI = 0x12,
        IFX_TAPI_CID_ST_MSGNR = 0x13,
        IFX_TAPI_CID_ST_FWCT = 0x15,
        IFX_TAPI_CID_ST_USRT = 0x16,
        IFX_TAPI_CID_ST_REDIR = 0x1A,
        IFX_TAPI_CID_ST_CHARGE = 0x20,
        IFX_TAPI_CID_ST_ACHARGE = 0x21,
        IFX_TAPI_CID_ST_DURATION = 0x23,
        IFX_TAPI_CID_ST_NTID = 0x30,
        IFX_TAPI_CID_ST_CARID = 0x31,
        IFX_TAPI_CID_ST_TERMSEL = 0x40,
        IFX_TAPI_CID_ST_DISP = 0x50,
        IFX_TAPI_CID_ST_SINFO = 0x55,
        IFX_TAPI_CID_ST_XOPUSE = 0xE0,
        IFX_TAPI_CID_ST_TRANSPARENT = 0xFF
} IFX_TAPI_CID_SERVICE_TYPE_t;
```

Parameters

Name	Value	Description
IFX_TAPI_CID_ST_DATE	01_H	Date and time presentation
IFX_TAPI_CID_ST_CLI	02_H	Calling line identity (mandatory)
IFX_TAPI_CID_ST_CDLI	03_H	Called line identity
IFX_TAPI_CID_ST_ABSCLI	04_H	Reason for absence of CLI
IFX_TAPI_CID_ST_NAME	07_H	Calling line name
IFX_TAPI_CID_ST_ABSNAME	08_H	Reason for absence of name
IFX_TAPI_CID_ST_VISINDIC	$0B_H$	Visual indicator
IFX_TAPI_CID_ST_MSGIDENT	$0D_H$	Reserved
IFX_TAPI_CID_ST_LMSGCLI	$0E_H$	Reserved
IFX_TAPI_CID_ST_CDATE	$0F_H$	Reserved
IFX_TAPI_CID_ST_CCLI	10_H	Complementary calling line identity
IFX_TAPI_CID_ST_CT	11_H	Call type
IFX_TAPI_CID_ST_FIRSTCLI	12_H	First called line identity
IFX_TAPI_CID_ST_MSGNR	13_H	Number of messages
IFX_TAPI_CID_ST_FWCT	15_H	Type of forwarded call

Name	Value	Description
IFX_TAPI_CID_ST_USRT	16_H	Type of calling user
IFX_TAPI_CID_ST_REDIR	$1A_H$	Number redirection
IFX_TAPI_CID_ST_CHARGE	20_H	Charge
IFX_TAPI_CID_ST_ACHARGE	21_H	Additional charge
IFX_TAPI_CID_ST_DURATION	23_H	Duration of the call
IFX_TAPI_CID_ST_NTID	30_H	Network provider id
IFX_TAPI_CID_ST_CARID	31_H	Carrier identity
IFX_TAPI_CID_ST_TERMSEL	40_H	Selection of terminal function
IFX_TAPI_CID_ST_DISP	50_H	Display information, used as INFO for DTMF
IFX_TAPI_CID_ST_SINFO	55_H	Reserved
IFX_TAPI_CID_ST_XOPUSE	$E0_H$	Extension for operator use
IFX_TAPI_CID_ST_TRANSPARENT	FF_H	Transparent mode

4.2.6.16 IFX_TAPI_CID_STD_t

Description

List of CID standards.

Prototype

```
typedef enum
{
        IFX_TAPI_CID_STD_TELCORDIA = 0x0,
        IFX_TAPI_CID_STD_ETSI_FSK = 0x1,
        IFX_TAPI_CID_STD_ETSI_DTMF = 0x2,
        IFX_TAPI_CID_STD_SIN = 0x3,
        IFX_TAPI_CID_STD_NTT = 0x4
} IFX_TAPI_CID_STD_t;
```

Parameters

Name	Value	Description
IFX_TAPI_CID_STD_TELCORDIA	0_H	Bellcore/Telcordia GR-30-CORE. Using Bell202 FSK coding of CID information.
IFX_TAPI_CID_STD_ETSI_FSK	1_H	ETSI 300-659-1/2/3 V1.3.1. Using V.23 FSK coding to transmit CID information.
IFX_TAPI_CID_STD_ETSI_DTMF	2_H	ETSI 300-659-1/2/3 V1.3.1. Using DTMF transmission of CID information.
IFX_TAPI_CID_STD_SIN	3_H	SIN 227 Issue 3.4. Using V.23 FSK coding of CID information.
IFX_TAPI_CID_STD_NTT	4_H	NTT standard: TELEPHONE SERVICE INTERFACES, Edition 5. Using a modified V.23 FSK coding of CID information.

4.2.6.17 IFX_TAPI_CID_VMWI_t

Description

List of VMWI settings.

Prototype

```
typedef enum
{
        IFX_TAPI_CID_VMWI_DIS = 0x00,
        IFX_TAPI_CID_VMWI_EN = 0xFF
} IFX_TAPI_CID_VMWI_t;
```

Parameters

Name	Value	Description
IFX_TAPI_CID_VMWI_DIS	00_H	Disable VMWI on CPE
IFX_TAPI_CID_VMWI_EN	FF_H	Enable VMWI on CPE

4.2.6.18 IFX_TAPI_DATA_MAP_START_STOP_t

Description

Start/Stop information for data channel mapping.

Prototype

```
typedef enum
{
        IFX_TAPI_MAP_DATA_UNCHANGED = 0,
        IFX_TAPI_MAP_DATA_START = 1,
        IFX_TAPI_MAP_DATA_STOP = 2
} IFX_TAPI_DATA_MAP_START_STOP_t;
```

Parameters

Name	Value	Description
IFX_TAPI_MAP_DATA_UNCHANGED	0_D	Do not modify the status of the recorder.
IFX_TAPI_MAP_DATA_START	1_D	Recording is started.
IFX_TAPI_MAP_DATA_STOP	2_D	Recording is stopped.

4.2.6.19 IFX_TAPI_DEBUG_REPORT_SET_t

Description

Debug report set.

Prototype

```
typedef enum
{
        IFX_TAPI_DEBUG_REPORT_SET_OFF = 0,
        IFX_TAPI_DEBUG_REPORT_SET_LOW = 1,
        IFX_TAPI_DEBUG_REPORT_SET_NORMAL = 2,
        IFX_TAPI_DEBUG_REPORT_SET_HIGH = 3
} IFX_TAPI_DEBUG_REPORT_SET_t;
```

Parameters

Name	Value	Description
IFX_TAPI_DEBUG_REPORT_SET_OFF	0_D	Debug report off.
IFX_TAPI_DEBUG_REPORT_SET_LOW	1_D	Low-level debug report.
IFX_TAPI_DEBUG_REPORT_SET_NORMAL	2_D	Normal level debug report, general information and warnings.
IFX_TAPI_DEBUG_REPORT_SET_HIGH	3_D	High level debug report, only errors.

4.2.6.20 IFX_TAPI_DIALING_STATUS_t

Description

Enumeration for dial status events.

Prototype

```
typedef enum
{
        IFX_TAPI_DIALING_STATUS_DTMF = 0x01,
        IFX_TAPI_DIALING_STATUS_PULSE = 0x02
} IFX_TAPI_DIALING_STATUS_t;
```

Parameters

Name	Value	Description
IFX_TAPI_DIALING_STATUS_DTMF	01_H	DTMF sign detected.
IFX_TAPI_DIALING_STATUS_PULSE	02_H	Pulse digit detected

4.2.6.21 IFX_TAPI_DWLD_TYPE_t

Description

Definition of download type.

Prototype

```
typedef enum
{
        IFX_TAPI_DWLD_TYPE_NONE = 0x0,
        IFX_TAPI_DWLD_TYPE_BBD = 0x1,
        IFX_TAPI_DWLD_TYPE_FW = 0x2
} IFX_TAPI_DWLD_TYPE_t;
```

Parameters

Name	Value	Description
IFX_TAPI_DWLD_TYPE_NONE	0x0	No download.
IFX_TAPI_DWLD_TYPE_BBD	0x1	BBD download type.
IFX_TAPI_DWLD_TYPE_FW	0x2	FW download type.

4.2.6.22 IFX_TAPI_ENC_TYPE_t

Description

Possible codecs to select for **IFX_TAPI_ENC_TYPE_SET** and **IFX_TAPI_PCK_AAL_PROFILE_t**.

Prototype

```
typedef enum
{
        IFX_TAPI_ENC_TYPE_G723_63 = 1,
        IFX_TAPI_ENC_TYPE_G723_53 = 2,
        IFX_TAPI_ENC_TYPE_G728 = 6,
        IFX_TAPI_ENC_TYPE_G729_AB = 7,
        IFX_TAPI_ENC_TYPE_MLAW = 8,
        IFX_TAPI_ENC_TYPE_ALAW = 9,
        IFX_TAPI_ENC_TYPE_G726_16 = 12,
        IFX_TAPI_ENC_TYPE_G726_24 = 13,
        IFX_TAPI_ENC_TYPE_G726_32 = 14,
        IFX_TAPI_ENC_TYPE_G726_40 = 15,
        IFX_TAPI_ENC_TYPE_G729_E = 16,
        IFX_TAPI_ENC_TYPE_ILBC_133 = 17,
        IFX_TAPI_ENC_TYPE_ILBC_152 = 18,
        IFX_TAPI_ENC_TYPE_AMR_4_75 = 21,
        IFX_TAPI_ENC_TYPE_AMR_5_15 = 22,
        IFX_TAPI_ENC_TYPE_AMR_5_9 = 23,
        IFX_TAPI_ENC_TYPE_AMR_6_7 = 24,
        IFX_TAPI_ENC_TYPE_AMR_7_4 = 25,
        IFX_TAPI_ENC_TYPE_AMR_7_95 = 26,
        IFX_TAPI_ENC_TYPE_AMR_10_2 = 27,
        IFX_TAPI_ENC_TYPE_AMR_12_2 = 28,
        IFX_TAPI_ENC_TYPE_G722_64 = 31,
        IFX_TAPI_ENC_TYPE_G7221_24 = 32,
        IFX_TAPI_ENC_TYPE_G7221_32 = 33,
        IFX_TAPI_ENC_TYPE_MAX = 34
} IFX_TAPI_ENC_TYPE_t;
```

Parameters

Name	Value	Description
IFX_TAPI_ENC_TYPE_G723_63	1_D	G723, 6.3 kBit/s.
IFX_TAPI_ENC_TYPE_G723_53	2_D	G723, 5.3 kBit/s.
IFX_TAPI_ENC_TYPE_G728	6_D	G728, 16 kBit/s. Not available!
IFX_TAPI_ENC_TYPE_G729_AB	7_D	G729 A and B (silence compression), 8 kBit/s.
IFX_TAPI_ENC_TYPE_MLAW	8_D	G711 µ-law, 64 kBit/s.
IFX_TAPI_ENC_TYPE_ALAW	9_D	G711 A-law, 64 kBit/s.
IFX_TAPI_ENC_TYPE_G726_16	12_D	G726, 16 kBit/s.
IFX_TAPI_ENC_TYPE_G726_24	13_D	G726, 24 kBit/s.
IFX_TAPI_ENC_TYPE_G726_32	14_D	G726, 32 kBit/s.
IFX_TAPI_ENC_TYPE_G726_40	15_D	G726, 40 kBit/s.

Name	Value	Description
IFX_TAPI_ENC_TYPE_G729_E	16_D	G729 E, 11.8 kBit/s.
IFX_TAPI_ENC_TYPE_ILBC_133	17_D	iLBC, 13.3 kBit/s.
IFX_TAPI_ENC_TYPE_ILBC_152	18_D	iLBC, 15.2 kBit/s.
IFX_TAPI_ENC_TYPE_AMR_4_75	21_D	AMR, 4.75 kBit/s.
IFX_TAPI_ENC_TYPE_AMR_5_15	22_D	AMR, 5.15 kBit/s.
IFX_TAPI_ENC_TYPE_AMR_5_9	23_D	AMR, 5.9 kBit/s.
IFX_TAPI_ENC_TYPE_AMR_6_7	24_D	AMR, 6.7 kBit/s.
IFX_TAPI_ENC_TYPE_AMR_7_4	25_D	AMR, 7.4 kBit/s.
IFX_TAPI_ENC_TYPE_AMR_7_95	26_D	AMR, 7.95 kBit/s.
IFX_TAPI_ENC_TYPE_AMR_10_2	27_D	AMR, 10.2 kBit/s.
IFX_TAPI_ENC_TYPE_AMR_12_2	28_D	AMR, 12.2 kBit/s.
IFX_TAPI_ENC_TYPE_G722_64	31_D	G.722 (wideband), 64 kBit/s.
IFX_TAPI_ENC_TYPE_G7221_24	32_D	G.722.1 (wideband), 24 kBit/s.
IFX_TAPI_ENC_TYPE_G7221_32	33_D	G.722.1 (wideband), 32 kBit/s.
IFX_TAPI_ENC_TYPE_MAX	34_D	Maximum number of Codecs.

4.2.6.23 IFX_TAPI_ENC_VAD_t

Description

Enumeration used for ioctl **IFX_TAPI_ENC_VAD_CFG_SET**.

Prototype

```
typedef enum
{
        IFX_TAPI_ENC_VAD_NOVAD = 0,
        IFX_TAPI_ENC_VAD_ON = 1,
        IFX_TAPI_ENC_VAD_G711 = 2
} IFX_TAPI_ENC_VAD_t;
```

Parameters

Name	Value	Description
IFX_TAPI_ENC_VAD_NOVAD	0_D	No voice activity detection.
IFX_TAPI_ENC_VAD_ON	1_D	Voice activity detection on, in this case also comfort noise and spectral information (nicer noise) is switched on.
IFX_TAPI_ENC_VAD_G711	2_D	Voice activity detection on with comfort noise generation without spectral information.

4.2.6.24 IFX_TAPI_EVENT_ID_t

Description

Event id.

Prototype

```
typedef enum
{
        IFX_TAPI_EVENT_NONE,
        IFX_TAPI_EVENT_FXS_RING,
        IFX_TAPI_EVENT_FXS_RINGBURST_END,
        IFX_TAPI_EVENT_FXS_RINGING_END,
        IFX_TAPI_EVENT_FXS_ONHOOK,
        IFX_TAPI_EVENT_FXS_OFFHOOK,
        IFX_TAPI_EVENT_FXS_FLASH,
        IFX_TAPI_EVENT_FXS_ONHOOK_INT,
        IFX_TAPI_EVENT_FXS_OFFHOOK_INT,
        IFX_TAPI_EVENT_FXO_RING,
        IFX_TAPI_EVENT_FXO_POLARITY,
        IFX_TAPI_EVENT_FXO_BAT,
        IFX_TAPI_EVENT_LT_GR909_RDY,
        IFX_TAPI_EVENT_PULSE_DIGIT,
        IFX_TAPI_EVENT_DTMF_DIGIT,
        IFX_TAPI_EVENT_CID_TX_SEQ_START,
        IFX_TAPI_EVENT_CID_TX_SEQ_END,
        IFX_TAPI_EVENT_CID_TX_INFO_START,
        IFX_TAPI_EVENT_CID_TX_INFO_END,
        IFX_TAPI_EVENT_CID_TX_NOACK_ERR,
        IFX_TAPI_EVENT_CID_TX_RINGCAD_ERR,
        IFX_TAPI_EVENT_CID_TX_ERROR2,
        IFX_TAPI_EVENT_CID_RX_CAS,
        IFX_TAPI_EVENT_CID_RX_FSK_END,
        IFX_TAPI_EVENT_CID_RX_CD,
        IFX_TAPI_EVENT_CID_RX_ERROR_READ,
        IFX_TAPI_EVENT_CID_RX_ERROR1,
        IFX_TAPI_EVENT_CID_RX_ERROR2,
        IFX_TAPI_EVENT_TONE_GEN_END,
        IFX_TAPI_EVENT_TONE_DET_CPT,
        IFX_TAPI_EVENT_FAXMODEM_DIS,
        IFX_TAPI_EVENT_FAXMODEM_CED,
        IFX_TAPI_EVENT_FAXMODEM_PR,
        IFX_TAPI_EVENT_FAXMODEM_AM,
        IFX_TAPI_EVENT_FAXMODEM_CNGFAX,
        IFX_TAPI_EVENT_FAXMODEM_CNGMOD,
        IFX_TAPI_EVENT_FAXMODEM_V21L,
        IFX_TAPI_EVENT_FAXMODEM_V18A,
        IFX_TAPI_EVENT_FAXMODEM_V27,
        IFX_TAPI_EVENT_FAXMODEM_BELL,
        IFX_TAPI_EVENT_FAXMODEM_V22,
        IFX_TAPI_EVENT_FAXMODEM_V22ORBELL,
        IFX_TAPI_EVENT_FAXMODEM_V32AC,
        IFX_TAPI_EVENT_FAXMODEM_V8BIS,
        IFX_TAPI_EVENT_FAXMODEM_HOLD,
        IFX_TAPI_EVENT_RFC2833_EVENT,
        IFX_TAPI_EVENT_T38_ERROR_GEN,
        IFX_TAPI_EVENT_T38_ERROR_OVLD,
```

```
        IFX_TAPI_EVENT_T38_ERROR_READ,
        IFX_TAPI_EVENT_T38_ERROR_WRITE,
        IFX_TAPI_EVENT_T38_ERROR_DATA,
        IFX_TAPI_EVENT_T38_ERROR_SETUP,
        IFX_TAPI_EVENT_FAULT_GENERAL,
        IFX_TAPI_EVENT_FAULT_LINE_GK_POS,
        IFX_TAPI_EVENT_FAULT_LINE_GK_NEG,
        IFX_TAPI_EVENT_FAULT_LINE_GK_LOW,
        IFX_TAPI_EVENT_FAULT_LINE_GK_HIGH,
        IFX_TAPI_EVENT_FAULT_LINE_OVERTEMP
} IFX_TAPI_EVENT_ID_t;
```

Parameters

Name	Value	Description
IFX_TAPI_EVENT_NONE		Reserved
IFX_TAPI_EVENT_FXS_RING		FXS line is ringing
IFX_TAPI_EVENT_FXS_RINGBURST_END		FXS end of a single ring burst
IFX_TAPI_EVENT_FXS_RINGING_END		FXS end of ringing
IFX_TAPI_EVENT_FXS_ONHOOK		Hook event: on-hook
IFX_TAPI_EVENT_FXS_OFFHOOK		Hook event: off-hook
IFX_TAPI_EVENT_FXS_FLASH		Hook event: flash-hook
IFX_TAPI_EVENT_FXS_ONHOOK_INT		Reserved. Hook event: on-hook detected by interrupt
IFX_TAPI_EVENT_FXS_OFFHOOK_INT		Reserved. Hook event: off-hook detected by interrupt
IFX_TAPI_EVENT_FXO_RING		Reserved. FXO line is ringing
IFX_TAPI_EVENT_FXO_POLARITY		Reserved. Polarity reversal
IFX_TAPI_EVENT_FXO_BAT		Reserved. Battery
IFX_TAPI_EVENT_LT_GR909_RDY		GR909 ready
IFX_TAPI_EVENT_PULSE_DIGIT		Pulse Digit
IFX_TAPI_EVENT_DTMF_DIGIT		DTMF tone detected local
IFX_TAPI_EVENT_CID_TX_SEQ_START		Reserved. Start of CID TX Sequence
IFX_TAPI_EVENT_CID_TX_SEQ_END		End of CID TX Sequence
IFX_TAPI_EVENT_CID_TX_INFO_START		Start of CID TX Information
IFX_TAPI_EVENT_CID_TX_INFO_END		End of CID TX Information
IFX_TAPI_EVENT_CID_TX_NOACK_ERR		No acknowledge during CID sequence
IFX_TAPI_EVENT_CID_TX_RINGCAD_ERR		Ring cadence settings error in cid tx
IFX_TAPI_EVENT_CID_TX_ERROR2		Reserved
IFX_TAPI_EVENT_CID_RX_CAS		CID CAS detected (CPE Alert Signal, for CID type 2)
IFX_TAPI_EVENT_CID_RX_FSK_END		CID RX, FSK detection ended
IFX_TAPI_EVENT_CID_RX_CD		Reserved. FSK Carrier Detected
IFX_TAPI_EVENT_CID_RX_ERROR_READ		Error during CID reception
IFX_TAPI_EVENT_CID_RX_ERROR1		Reserved. Error during CID reception
IFX_TAPI_EVENT_CID_RX_ERROR2		Reserved
IFX_TAPI_EVENT_TONE_GEN_END		Tone generator ended
IFX_TAPI_EVENT_TONE_DET_CPT		Tone detect call

Name	Value	Description
IFX_TAPI_EVENT_FAXMODEM_DIS		DIS preamble signal
IFX_TAPI_EVENT_FAXMODEM_CED		2100 Hz (CED) answering tone (ANS)
IFX_TAPI_EVENT_FAXMODEM_PR		Phase reversal
IFX_TAPI_EVENT_FAXMODEM_AM		Amplitude modulation
IFX_TAPI_EVENT_FAXMODEM_CNGFAX		1100 Hz single tone (CNG Fax)
IFX_TAPI_EVENT_FAXMODEM_CNGMOD		1300 Hz single tone (CNG Modem). It can indicate CT, V.18 XCI mark sequence.
IFX_TAPI_EVENT_FAXMODEM_V21L		980 Hz single tone (V.21L mark sequence)
IFX_TAPI_EVENT_FAXMODEM_V18A		1400 Hz single tone (V.18A mark sequence
IFX_TAPI_EVENT_FAXMODEM_V27		1800 Hz single tone (V.27, V.32 carrier)
IFX_TAPI_EVENT_FAXMODEM_BELL		2225 Hz single tone (Bell answering tone)
IFX_TAPI_EVENT_FAXMODEM_V22		2250 Hz single tone (V.22 unscrambled binary ones)
IFX_TAPI_EVENT_FAXMODEM_V22ORBELL		2225 Hz or 2250 Hz single tone, not possible to distinguish
IFX_TAPI_EVENT_FAXMODEM_V32AC		600 Hz + 300 Hz dual tone (V.32 AC)
IFX_TAPI_EVENT_FAXMODEM_V8BIS		1375 Hz + 2002 Hz dual tone (V.8bis initiating segment 1)
IFX_TAPI_EVENT_FAXMODEM_HOLD		Hold caracteristic
IFX_TAPI_EVENT_RFC2833_EVENT		RFC 2833 event
IFX_TAPI_EVENT_T38_ERROR_GEN		Error Generic
IFX_TAPI_EVENT_T38_ERROR_OVLD		Error OVLD
IFX_TAPI_EVENT_T38_ERROR_READ		Error Read
IFX_TAPI_EVENT_T38_ERROR_WRITE		Error Write
IFX_TAPI_EVENT_T38_ERROR_DATA		Error Data
IFX_TAPI_EVENT_T38_ERROR_SETUP		Error Setup
IFX_TAPI_EVENT_FAULT_GENERAL		General System Fault
IFX_TAPI_EVENT_FAULT_LINE_GK_POS		Ground Key, Positive Polarity
IFX_TAPI_EVENT_FAULT_LINE_GK_NEG		Ground Key, Negative Polarity
IFX_TAPI_EVENT_FAULT_LINE_GK_LOW		Ground Key Low
IFX_TAPI_EVENT_FAULT_LINE_GK_HIGH		Ground Key High
IFX_TAPI_EVENT_FAULT_LINE_OVERTEMP		Overtemperature

4.2.6.25 IFX_TAPI_EVENT_GET_RET_t

Description

Return value for event reporting interface.

Prototype

```
typedef enum
{
        IFX_TAPI_EVENT_RET_ERROR = -1,
        IFX_TAPI_EVENT_RET_SUCCESS = 0,
        IFX_TAPI_EVENT_RET_READY = 1
} IFX_TAPI_EVENT_GET_RET_t;
```

Parameters

Name	Value	Description
IFX_TAPI_EVENT_RET_ERROR	-1_D	Error.
IFX_TAPI_EVENT_RET_SUCCESS	0_D	Success.
IFX_TAPI_EVENT_RET_READY	1_D	A new event is ready.

4.2.6.26 IFX_TAPI_EVENT_TYPE_t

Description

Event types.

Prototype

```
typedef enum
{
        IFX_TAPI_EVENT_TYPE_NONE = 0x00000000,
        IFX_TAPI_EVENT_TYPE_FXS = 0x20000000,
        IFX_TAPI_EVENT_TYPE_FXO = 0x21000000,
        IFX_TAPI_EVENT_TYPE_LT = 0x29000000,
        IFX_TAPI_EVENT_TYPE_PULSE = 0x30000000,
        IFX_TAPI_EVENT_TYPE_DTMF = 0x31000000,
        IFX_TAPI_EVENT_TYPE_CID = 0x32000000,
        IFX_TAPI_EVENT_TYPE_TONE_GEN = 0x33000000,
        IFX_TAPI_EVENT_TYPE_TONE_DET = 0x34000000,
        IFX_TAPI_EVENT_TYPE_FAXMODEM_SIGNAL = 0x35000000,
        IFX_TAPI_EVENT_TYPE_CODER = 0x40000000,
        IFX_TAPI_EVENT_TYPE_RFC2833 = 0x43000000,
        IFX_TAPI_EVENT_TYPE_T38 = 0x50000000,
        IFX_TAPI_EVENT_TYPE_LL_DRIVER = 0xE0000000,
        IFX_TAPI_EVENT_TYPE_FAULT_GENERAL = 0xF1000000,
        IFX_TAPI_EVENT_TYPE_FAULT_LINE = 0xF2000000
} IFX_TAPI_EVENT_TYPE_t;
```

Parameters

Name	Value	Description
IFX_TAPI_EVENT_TYPE_NONE	00000000_H	Reserved.
IFX_TAPI_EVENT_TYPE_FXS	20000000_H	Ringing, Hook events.
IFX_TAPI_EVENT_TYPE_FXO	21000000_H	Reserved, not implemented yet. Ringing, polarity reversal, ...
IFX_TAPI_EVENT_TYPE_LT	29000000_H	Reserved, not implemented yet. Linetesting events.
IFX_TAPI_EVENT_TYPE_PULSE	30000000_H	Pulse Digit detected.
IFX_TAPI_EVENT_TYPE_DTMF	31000000_H	DTMF Digit detected.
IFX_TAPI_EVENT_TYPE_CID	32000000_H	Caller ID events.
IFX_TAPI_EVENT_TYPE_TONE_GEN	33000000_H	Tone generation event, for example. Tone generation ended.

Name	Value	Description
IFX_TAPI_EVENT_TYPE_TONE_DET	34000000_H	Tone detection event, for example Call Progress Tones
IFX_TAPI_EVENT_TYPE_FAXMODEM_SIGNAL	35000000_H	Detection of Fax/Modem and V.18 signals.
IFX_TAPI_EVENT_TYPE_CODER	40000000_H	Reserved, not implemented yet. For example vocoder changed
IFX_TAPI_EVENT_TYPE_RFC2833	43000000_H	Reserved, not implemented yet. RFC2833 Frame detected.
IFX_TAPI_EVENT_TYPE_T38	50000000_H	Reserved, not implemented yet. T.38 events.
IFX_TAPI_EVENT_TYPE_LL_DRIVER	$E0000000_H$	Reserved. Events of the low-level driver
IFX_TAPI_EVENT_TYPE_FAULT_GENERAL	$F1000000_H$	General fault.
IFX_TAPI_EVENT_TYPE_FAULT_LINE	$F2000000_H$	For example: Overtemperature, Ground key detected

4.2.6.27 IFX_TAPI_JB_LOCAL_ADAPT_t

Description

Jitter buffer adaptation.

Prototype

```
typedef enum
{
        IFX_TAPI_JB_LOCAL_ADAPT_OFF = 0,
        IFX_TAPI_JB_LOCAL_ADAPT_ON = 1,
        IFX_TAPI_JB_LOCAL_ADAPT_SI_ON = 2
} IFX_TAPI_JB_LOCAL_ADAPT_t;
```

Parameters

Name	Value	Description
IFX_TAPI_JB_LOCAL_ADAPT_OFF	0_D	Local Adaptation OFF. Speech gaps which are detected by the far end side are used for the jitter buffer adaptations.

Name	Value	Description
IFX_TAPI_JB_LOCAL_ADAPT_ON	1_D	Local adaptation ON. Local adaptation on. Jitter buffer adaptation via local and far end speech detection. This means that the jitter buffer adaptation does not need the far end silence compression feature but will use it if the far end side has detected a silence period. Thus a jitter buffer adaptation can occur when the far end side has detected silence or when a speech gap was detected in the downstream direction.
IFX_TAPI_JB_LOCAL_ADAPT_SI_ON	2_D	Local Adaptation ON with Sample interpolation. A jitter buffer adaptation can be done via sample interpolation. The advantage is that not a whole frame has to be interpolated or discarded. Thus no major distortion should be heard.

4.2.6.28 IFX_TAPI_JB_PKT_ADAPT_t

Description

Jitter buffer packet adaptation.

Prototype

```
typedef enum
{
        IFX_TAPI_JB_PKT_ADAPT_RES1 = 0,
        IFX_TAPI_JB_PKT_ADAPT_RES2 = 1,
        IFX_TAPI_JB_PKT_ADAPT_VOICE = 2,
        IFX_TAPI_JB_PKT_ADAPT_DATA = 3
} IFX_TAPI_JB_PKT_ADAPT_t;
```

Parameters

Name	Value	Description
IFX_TAPI_JB_PKT_ADAPT_RES1	0_D	Reserved.
IFX_TAPI_JB_PKT_ADAPT_RES2	1_D	Reserved.
IFX_TAPI_JB_PKT_ADAPT_VOICE	2_D	JB adapted for voice, reduced adjustment speed and packet repetition is off.
IFX_TAPI_JB_PKT_ADAPT_DATA	3_D	JB adapted for data (fax/modem). Reduced adjustment speed and packet repetition is on

4.2.6.29 IFX_TAPI_JB_TYPE_t

Description

Jitter buffer type.

Prototype

```
typedef enum
{
        IFX_TAPI_JB_TYPE_FIXED = 0,
        IFX_TAPI_JB_TYPE_ADAPTIVE = 1
} IFX_TAPI_JB_TYPE_t;
```

Parameters

Name	Value	Description
IFX_TAPI_JB_TYPE_FIXED	0_D	Fixed Jitter Buffer.
IFX_TAPI_JB_TYPE_ADAPTIVE	1_D	Adaptive Jitter Buffer.

4.2.6.30 IFX_TAPI_LEC_GAIN_t

Description

LEC gain levels.

Prototype

```
typedef enum
{
        IFX_TAPI_LEC_GAIN_OFF = 0,
        IFX_TAPI_LEC_GAIN_LOW = 1,
        IFX_TAPI_LEC_GAIN_MEDIUM = 2,
        IFX_TAPI_LEC_GAIN_HIGH = 3
} IFX_TAPI_LEC_GAIN_t;
```

Parameters

Name	Value	Description
IFX_TAPI_LEC_GAIN_OFF	0_D	Turn LEC off.
IFX_TAPI_LEC_GAIN_LOW	1_D	Turn LEC on to low-level.
IFX_TAPI_LEC_GAIN_MEDIUM	2_D	Turn LEC on to normal level.
IFX_TAPI_LEC_GAIN_HIGH	3_D	Turn LEC on to high level.

4.2.6.31 IFX_TAPI_LEC_NLP_t

Description

LEC NLP (Non Linear Processor) settings.

Prototype

```
typedef enum
{
        IFX_TAPI_LEC_NLP_DEFAULT = 0,
        IFX_TAPI_LEC_NLP_ON = 1,
        IFX_TAPI_LEC_NLP_OFF = 2
} IFX_TAPI_LEC_NLP_t;
```

Parameters

Name	Value	Description
IFX_TAPI_LEC_NLP_DEFAULT	0_D	Default NLP on.
IFX_TAPI_LEC_NLP_ON	1_D	NLP on.
IFX_TAPI_LEC_NLP_OFF	2_D	NLP off.

4.2.6.32 IFX_TAPI_LEC_TYPE_t

Description

LEC type selection.

Prototype

```
typedef enum
{
        IFX_TAPI_LEC_TYPE_OFF = 0,
        IFX_TAPI_LEC_NLEC = 1,
        IFX_TAPI_LEC_WLEC = 2
} IFX_TAPI_LEC_TYPE_t;
```

Parameters

Name	Value	Description
IFX_TAPI_LEC_TYPE_OFF	0_D	LEC OFF.
IFX_TAPI_LEC_NLEC	1_D	Near-end LEC.
IFX_TAPI_LEC_WLEC	2_D	Window based LEC (near-end and far-end at the same time).

4.2.6.33 IFX_TAPI_LINE_FEED_t

Description

Defines for line feeding modes.

Prototype

```
typedef enum
{
        IFX_TAPI_LINE_FEED_ACTIVE = 0,
        IFX_TAPI_LINE_FEED_ACTIVE_REV = 1,
        IFX_TAPI_LINE_FEED_STANDBY = 2,
        IFX_TAPI_LINE_FEED_HIGH_IMPEDANCE = 3,
        IFX_TAPI_LINE_FEED_DISABLED = 4,
        IFX_TAPI_LINE_FEED_GROUND_START = 5,
        IFX_TAPI_LINE_FEED_RING_BURST = 8,
        IFX_TAPI_LINE_FEED_RING_PAUSE = 9,
        IFX_TAPI_LINE_FEED_METER = 10
} IFX_TAPI_LINE_FEED_t;
```

Parameters

Name	Value	Description
IFX_TAPI_LINE_FEED_ACTIVE	0_D	Feeding mode for phone used for on-hook or off-hook transmission, with normal polarity.
IFX_TAPI_LINE_FEED_ACTIVE_REV	1_D	Feeding mode for phone used for on-hook or off-hook transmission, with reversed polarity.
IFX_TAPI_LINE_FEED_STANDBY	2_D	Feeding mode to be used if the phone is in standby (on-hook without any transmission active). Off-hook detection is possible in this mode.
IFX_TAPI_LINE_FEED_HIGH_IMPEDANCE	3_D	Switch off the line, the device is only able to test the line.
IFX_TAPI_LINE_FEED_DISABLED	4_D	Switch off the line, no operations possible on the line.
IFX_TAPI_LINE_FEED_GROUND_START	5_D	Use ground start, also known as open tip.
IFX_TAPI_LINE_FEED_RING_BURST	8_D	Reserved, needed for ring internal function.
IFX_TAPI_LINE_FEED_RING_PAUSE	9_D	Reserved, needed for ring internal function.
IFX_TAPI_LINE_FEED_METER	10_D	Reserved, needed for internal function.

4.2.6.34 IFX_TAPI_LINE_HOOK_STATUS_t

Description

Enumeration for hook status events.

Prototype

```
typedef enum
{
        IFX_TAPI_LINE_HOOK_STATUS_HOOK = 0x01,
        IFX_TAPI_LINE_HOOK_STATUS_FLASH = 0x02,
        IFX_TAPI_LINE_HOOK_STATUS_OFFHOOK = 0x04
} IFX_TAPI_LINE_HOOK_STATUS_t;
```

Parameters

Name	Value	Description
IFX_TAPI_LINE_HOOK_STATUS_HOOK	01_H	Hook detected.
IFX_TAPI_LINE_HOOK_STATUS_FLASH	02_H	Hook flash detected.
IFX_TAPI_LINE_HOOK_STATUS_OFFHOOK	04_H	Detected hook event is an off-hook.

4.2.6.35 IFX_TAPI_LINE_HOOK_VALIDATION_TYPE_t

Description

Validation types used for structure **IFX_TAPI_LINE_HOOK_VT_t**.

Prototype

```
typedef enum
{
        IFX_TAPI_LINE_HOOK_VT_OFFHOOK_TIME = 0x0,
        IFX_TAPI_LINE_HOOK_VT_ONHOOK_TIME = 0x1,
        IFX_TAPI_LINE_HOOK_VT_FLASHHOOK_TIME = 0x2,
        IFX_TAPI_LINE_HOOK_VT_DIGITLOW_TIME = 0x4,
        IFX_TAPI_LINE_HOOK_VT_DIGITHIGH_TIME = 0x8,
        IFX_TAPI_LINE_HOOK_VT_INTERDIGIT_TIME = 0x10
} IFX_TAPI_LINE_HOOK_VALIDATION_TYPE_t;
```

Parameters

Name	Value	Description
IFX_TAPI_LINE_HOOK_VT_OFFHOOK_TIME	0_H	Settings for hook validation, if the time matches between nMinTime and nMaxTime an exception is raised.
IFX_TAPI_LINE_HOOK_VT_ONHOOK_TIME	1_H	Settings for hook validation, if the time matches between nMinTime and nMaxTime an exception is raised.
IFX_TAPI_LINE_HOOK_VT_FLASHHOOK_TIME	2_H	Settings for hook flash validation also known as register recall. If the time matches between the time defined in the fields nMinTime and nMaxTime an exception is raised
IFX_TAPI_LINE_HOOK_VT_DIGITLOW_TIME	4_H	Settings for pulse digit low, open loop and make validation. The time must match between the time defined in the fields nMinTime and nMaxTime to recognize it as pulse dialing event
IFX_TAPI_LINE_HOOK_VT_DIGITHIGH_TIME	8_H	Settings for pulse digit high, close loop and break validation. The time must match between the time defined in the fields nMinTime and nMaxTime to recognize it as pulse dialing event
IFX_TAPI_LINE_HOOK_VT_INTERDIGIT_TIME	10_H	Settings for pulse digit pause. The time must match the time defined in the fields nMinTime and nMaxTime to recognize it as pulse dialing event

4.2.6.36 IFX_TAPI_LINE_LEVEL_t

Description

Specifies the whether the “high level” line output mode should be used. Enabling the “high level” mode, it will be possible to have a extremly high output signal level, above +3 dBm. This mode is required for some special howler tones.

Attention: The “high level” mode allow signal levels potentially dangerous for the human ear!

Prototype

```
typedef enum
{
        IFX_TAPI_LINE_LEVEL_NORMAL = 0x0,
        IFX_TAPI_LINE_LEVEL_HIGH = 0x1
} IFX_TAPI_LINE_LEVEL_t;
```

Parameters

Name	Value	Description
IFX_TAPI_LINE_LEVEL_NORMAL	0_H	Line output level for typical operations.
IFX_TAPI_LINE_LEVEL_HIGH	1_H	High level line, suitable for playing howler tones requiring output levels above 3 dBm. This is not suitable for voice calls or other call progress tone detection.

4.2.6.37 IFX_TAPI_LINE_STATUS_t

Description

Enumeration for phone line status information.

Prototype

```
typedef enum
{
        IFX_TAPI_LINE_STATUS_RINGING = 0x1,
        IFX_TAPI_LINE_STATUS_RINGFINISHED = 0x02,
        IFX_TAPI_LINE_STATUS_FAX = 0x04,
        IFX_TAPI_LINE_STATUS_GNDKEY = 0x10,
        IFX_TAPI_LINE_STATUS_GNDKEYHIGH = 0x20,
        IFX_TAPI_LINE_STATUS_OTEMP = 0x40,
        IFX_TAPI_LINE_STATUS_GNDKEYPOL = 0x80,
        IFX_TAPI_LINE_STATUS_GR909RES = 0x100,
        IFX_TAPI_LINE_STATUS_CIDRX = 0x200
} IFX_TAPI_LINE_STATUS_t;
```

Parameters

Name	Value	Description
IFX_TAPI_LINE_STATUS_RINGING	1_H	Line is ringing
IFX_TAPI_LINE_STATUS_RINGFINISHED	02_H	Ringing finished
IFX_TAPI_LINE_STATUS_FAX	04_H	Fax detected -> is replaced by signal
IFX_TAPI_LINE_STATUS_GNDKEY	10_H	Ground key detected
IFX_TAPI_LINE_STATUS_GNDKEYHIGH	20_H	Ground key high detected
IFX_TAPI_LINE_STATUS_OTEMP	40_H	Over temperature detected
IFX_TAPI_LINE_STATUS_GNDKEYPOL	80_H	Ground key polarity detected
IFX_TAPI_LINE_STATUS_GR909RES	100_H	
IFX_TAPI_LINE_STATUS_CIDRX	200_H	Caller id received

4.2.6.38 IFX_TAPI_MAP_DATA_TYPE_t

Description

Data channel destination types (conferencing).

Prototype

```
typedef enum
{
        IFX_TAPI_MAP_DATA_TYPE_DEFAULT = 0,
        IFX_TAPI_MAP_DATA_TYPE_CODER = 1,
        IFX_TAPI_MAP_DATA_TYPE_PCM = 2,
        IFX_TAPI_MAP_DATA_TYPE_PHONE = 3,
        IFX_TAPI_MAP_DATA_TYPE_AUDIO = 4,
        IFX_TAPI_MAP_DATA_TYPE_AUDIO_AUX = 5
} IFX_TAPI_MAP_DATA_TYPE_t;
```

Parameters

Name	Value	Description
IFX_TAPI_MAP_DATA_TYPE_DEFAULT	0_D	Reserved
IFX_TAPI_MAP_DATA_TYPE_CODER	1_D	Map the data channel to a coder module
IFX_TAPI_MAP_DATA_TYPE_PCM	2_D	Map the data channel to a pcm module
IFX_TAPI_MAP_DATA_TYPE_PHONE	3_D	Map the data channel to a phone module
IFX_TAPI_MAP_DATA_TYPE_AUDIO	4_D	Map the data channel to an audio module
IFX_TAPI_MAP_DATA_TYPE_AUDIO_AUX	5_D	Map the data channel to the auxiliary output of the audio module

Remarks

This enum is used to choose the resource type for a mapping ioctl, the applicability on a certain strictly depends on the resource availability on the selected channe. In particular

- **IFX_TAPI_MAP_DATA_TYPE_PHONE** can be used only in IP Phone applications.
- **IFX_TAPI_MAP_DATA_TYPE_AUDIO** and **IFX_TAPI_MAP_DATA_TYPE_AUDIO_AUX** can be used only in IP Phone applications.

4.2.6.39 IFX_TAPI_MAP_DEC_t

Description

Play out enabling and disabling information.

Prototype

```
typedef enum
{
        IFX_TAPI_MAP_DEC_NONE = 0,
        IFX_TAPI_MAP_DEC_START = 1,
        IFX_TAPI_MAP_DEC_STOP = 2
} IFX_TAPI_MAP_DEC_t;
```

Parameters

Name	Value	Description
IFX_TAPI_MAP_DEC_NONE	0_D	Do not modify playing status.
IFX_TAPI_MAP_DEC_START	1_D	Start playing after mapping.
IFX_TAPI_MAP_DEC_STOP	2_D	Stop playing after mapping.

4.2.6.40 IFX_TAPI_MAP_ENC_t

Description

Recording enabling and disabling information.

Prototype

```
typedef enum
{
        IFX_TAPI_MAP_ENC_NONE = 0,
        IFX_TAPI_MAP_ENC_START = 1,
        IFX_TAPI_MAP_ENC_STOP = 2
} IFX_TAPI_MAP_ENC_t;
```

Parameters

Name	Value	Description
IFX_TAPI_MAP_ENC_NONE	0_D	Do not modify recording status.
IFX_TAPI_MAP_ENC_START	1_D	Start recording after mapping.
IFX_TAPI_MAP_ENC_STOP	2_D	Stop recording after mapping.

4.2.6.41 IFX_TAPI_MAP_TYPE_t

Description

Type channel for mapping.

Prototype

```
typedef enum
{
        IFX_TAPI_MAP_TYPE_DEFAULT = 0,
        IFX_TAPI_MAP_TYPE_CODER = 1,
        IFX_TAPI_MAP_TYPE_PCM = 2,
        IFX_TAPI_MAP_TYPE_PHONE = 3,
        IFX_TAPI_MAP_TYPE_AUDIO = 4,
        IFX_TAPI_MAP_TYPE_AUDIO_AUX = 5
} IFX_TAPI_MAP_TYPE_t;
```

Parameters

Name	Value	Description
IFX_TAPI_MAP_TYPE_DEFAULT	0_D	Reserved.
IFX_TAPI_MAP_TYPE_CODER	1_D	Type is a Coder channel.
IFX_TAPI_MAP_TYPE_PCM	2_D	Type is a PCM channel.
IFX_TAPI_MAP_TYPE_PHONE	3	Type is a Analog Phone channel.
IFX_TAPI_MAP_TYPE_AUDIO	4_D	Type is an Audio channel
IFX_TAPI_MAP_TYPE_AUDIO_AUX	5_D	Type is the auxiliary output of the audio channel

Remarks

This enum is used to choose the resource type for a mapping ioctl, the applicability on a certain strictly depends on the resource availability on the selected channe. In particular

- **IFX_TAPI_MAP_TYPE_PHONE** can be used only in IP Phone applications.
- **IFX_TAPI_MAP_TYPE_AUDIO** and **IFX_TAPI_MAP_TYPE_AUDIO_AUX** can be used only in IP Phone applications.

4.2.6.42 IFX_TAPI_METER_MODE_t

Description

Metering modes.

Prototype

```
typedef enum
{
        IIFX_TAPI_METER_MODE_TTX = 0,
        IFX_TAPI_METER_MODE_REVPOL = 1
} IFX_TAPI_METER_MODE_t;
```

Parameters

Name	Value	Description
IIFX_TAPI_METER_MODE_TTX	0_D	Normal TTX mode.
IFX_TAPI_METER_MODE_REVPOL	1_D	Reverse polarity mode.

4.2.6.43 IFX_TAPI_PCM_RES_t

Description

PCM Resolutions.

Prototype

```
typedef enum
{
        IFX_TAPI_PCM_RES_ALAW_8BIT = 0,
        IFX_TAPI_PCM_RES_MLAW_8BIT = 1,
        IFX_TAPI_PCM_RES_LINEAR_16BIT = 2
} IFX_TAPI_PCM_RES_t;
```

Parameters

Name	Value	Description
IFX_TAPI_PCM_RES_ALAW_8BIT	0_D	A-law 8-bit.
IFX_TAPI_PCM_RES_MLAW_8BIT	1_D	µ-law 8-bit
IFX_TAPI_PCM_RES_LINEAR_16BIT	2_D	Linear 16-bit.

4.2.6.44 IFX_TAPI_PKT_AAL_PROFILE_RANGE_t

Description

Used for **IFX_TAPI_PCK_AAL_PROFILE_t** in case one coder range.

The range information is specified as “UUI Codepoint Range” in the specification The ATM Forum, AF-VMOA-0145.000 or ITU-T I.366.2

Prototype

```
typedef enum
{
        IFX_TAPI_PKT_AAL_PROFILE_RANGE_0_15 = 0,
        IFX_TAPI_PKT_AAL_PROFILE_RANGE_0_7 = 1,
        IFX_TAPI_PKT_AAL_PROFILE_RANGE_8_15 = 2,
        IFX_TAPI_PKT_AAL_PROFILE_RANGE_0_3 = 3,
        IFX_TAPI_PKT_AAL_PROFILE_RANGE_4_7 = 4,
        IFX_TAPI_PKT_AAL_PROFILE_RANGE_8_11 = 5,
        IFX_TAPI_PKT_AAL_PROFILE_RANGE_12_15 = 6
} IFX_TAPI_PKT_AAL_PROFILE_RANGE_t;
```

Parameters

Name	Value	Description
IFX_TAPI_PKT_AAL_PROFILE_RANGE_0_15	0_D	One range from 0 to 15.
IFX_TAPI_PKT_AAL_PROFILE_RANGE_0_7	1_D	Range from 0 to 7 for a two range profile entry.
IFX_TAPI_PKT_AAL_PROFILE_RANGE_8_15	2_D	Range from 8 to 15 for a two range profile entry.
IFX_TAPI_PKT_AAL_PROFILE_RANGE_0_3	3_D	Range from 0 to 3 for a four range profile entry.
IFX_TAPI_PKT_AAL_PROFILE_RANGE_4_7	4_D	Range from 4 to 7 for a four range profile entry.
IFX_TAPI_PKT_AAL_PROFILE_RANGE_8_11	5_D	Range from 8 to 11 for a four range profile entry.
IFX_TAPI_PKT_AAL_PROFILE_RANGE_12_15	6_D	Range from 12 to 15 for a four range profile entry.

4.2.6.45 IFX_TAPI_PKT_EV_NUM_t

Description

Out of band or in band definition.

Prototype

```
typedef enum
{
        IFX_TAPI_PKT_EV_NUM_DTMF_0 = 0,
        IFX_TAPI_PKT_EV_NUM_DTMF_1 = 1,
        IFX_TAPI_PKT_EV_NUM_DTMF_2 = 2,
        IFX_TAPI_PKT_EV_NUM_DTMF_3 = 3,
        IFX_TAPI_PKT_EV_NUM_DTMF_4 = 4,
        IFX_TAPI_PKT_EV_NUM_DTMF_5 = 5,
        IFX_TAPI_PKT_EV_NUM_DTMF_6 = 6,
        IFX_TAPI_PKT_EV_NUM_DTMF_7 = 7,
        IFX_TAPI_PKT_EV_NUM_DTMF_8 = 8,
        IFX_TAPI_PKT_EV_NUM_DTMF_9 = 9,
        IFX_TAPI_PKT_EV_NUM_DTMF_STAR = 10,
        IFX_TAPI_PKT_EV_NUM_DTMF_HASH = 11,
        IFX_TAPI_PKT_EV_NUM_ANS = 32,
        IFX_TAPI_PKT_EV_NUM_NANS = 33,
        IFX_TAPI_PKT_EV_NUM_ANSAM = 34,
        IFX_TAPI_PKT_EV_NUM_NANSAM = 35,
        IFX_TAPI_PKT_EV_NUM_CNG = 36,
        IFX_TAPI_PKT_EV_NUM_DIS = 54,
        IFX_TAPI_PKT_EV_NUM_NO_EVENT = 0xFFFFFFFF
} IFX_TAPI_PKT_EV_NUM_t;
```

Parameters

Name	Value	Description
IFX_TAPI_PKT_EV_NUM_DTMF_0	0_D	RFC2833 Event number for DTMF tone 0.
IFX_TAPI_PKT_EV_NUM_DTMF_1	1_D	RFC2833 Event number for DTMF tone 1.
IFX_TAPI_PKT_EV_NUM_DTMF_2	2_D	RFC2833 Event number for DTMF tone 2.
IFX_TAPI_PKT_EV_NUM_DTMF_3	3_D	RFC2833 Event number for DTMF tone 3.
IFX_TAPI_PKT_EV_NUM_DTMF_4	4_D	RFC2833 Event number for DTMF tone 4.
IFX_TAPI_PKT_EV_NUM_DTMF_5	5_D	RFC2833 Event number for DTMF tone 5.
IFX_TAPI_PKT_EV_NUM_DTMF_6	6_D	RFC2833 Event number for DTMF tone 6.
IFX_TAPI_PKT_EV_NUM_DTMF_7	7_D	RFC2833 Event number for DTMF tone 7.
IFX_TAPI_PKT_EV_NUM_DTMF_8	8_D	RFC2833 Event number for DTMF tone 8.
IFX_TAPI_PKT_EV_NUM_DTMF_9	9_D	RFC2833 Event number for DTMF tone 9.
IFX_TAPI_PKT_EV_NUM_DTMF_STAR	10_D	RFC2833 Event number for DTMF tone *.
IFX_TAPI_PKT_EV_NUM_DTMF_HASH	11_D	RFC2833 Event number for DTMF tone #.
IFX_TAPI_PKT_EV_NUM_ANS	32_D	RFC2833 Event number for ANS tone.
IFX_TAPI_PKT_EV_NUM_NANS	33_D	RFC2833 Event number for /ANS tone.
IFX_TAPI_PKT_EV_NUM_ANSAM	34_D	RFC2833 Event number for ANSam tone.
IFX_TAPI_PKT_EV_NUM_NANSAM	35_D	RFC2833 Event number for /ANSam tone.
IFX_TAPI_PKT_EV_NUM_CNG	36_D	RFC2833 Event number for CNG tone.
IFX_TAPI_PKT_EV_NUM_DIS	54_D	RFC2833 Event number for DIS signal.
IFX_TAPI_PKT_EV_NUM_NO_EVENT	$FFFFFFFF_H$	No support RFC event received or no event received.

4.2.6.46 IFX_TAPI_PKT_EV_OOB_t

Description

Out of band or in band definition.

Prototype

```
typedef enum
{
        IFX_TAPI_PKT_EV_OOB_DEFAULT = 0,
        IFX_TAPI_PKT_EV_OOB_NO = 1,
        IFX_TAPI_PKT_EV_OOB_ONLY = 2,
        IFX_TAPI_PKT_EV_OOB_ALL = 3,
        IFX_TAPI_PKT_EV_OOB_BLOCK = 4
} IFX_TAPI_PKT_EV_OOB_t;
```

Parameters

Name	Value	Description
IFX_TAPI_PKT_EV_OOB_DEFAULT	0_D	Reserved.
IFX_TAPI_PKT_EV_OOB_NO	1_D	Transmit only in-band.
IFX_TAPI_PKT_EV_OOB_ONLY	2_D	Transmit only out-of-band.
IFX_TAPI_PKT_EV_OOB_ALL	3_D	Transmit in-band and out-of-band.
IFX_TAPI_PKT_EV_OOB_BLOCK	4_D	Block event transmission: neither in-band nor out-of-band.

4.2.6.47 IFX_TAPI_PKT_EV_OOBPLAY_t

Description

Defines the play out of received RFC2833 event packets.

Prototype

```
typedef enum
{
        IFX_TAPI_PKT_EV_OOBPLAY_DEFAULT = 0,
        IFX_TAPI_PKT_EV_OOBPLAY_PLAY = 1,
        IFX_TAPI_PKT_EV_OOBPLAY_MUTE = 2,
        IFX_TAPI_PKT_EV_OOBPLAY_APT_PLAY = 3
} IFX_TAPI_PKT_EV_OOBPLAY_t;
```

Parameters

Name	Value	Description
IFX_TAPI_PKT_EV_OOBPLAY_DEFAULT	0_D	Device default setting. Not recommended.
IFX_TAPI_PKT_EV_OOBPLAY_PLAY	1_D	All RFC 2833 packets coming from the net are played out. Upstream and downstream RFC 2833 packets have the same payload type.

Name	Value	Description
IFX_TAPI_PKT_EV_OOBPLAY_MUTE	2_D	All RFC 2833 packets coming from the net are muted.
IFX_TAPI_PKT_EV_OOBPLAY_APT_PLAY	3_D	All RFC 2833 packets coming from the net are played out. Upstream and downstream RFC 2833 packets have different payload types.

4.2.6.48 IFX_TAPI_RING_CFG_MODE_t

Description

Ring Configuration Mode.

Prototype

```
typedef enum
{
        IFX_TAPI_RING_CFG_MODE_INT_BALANCED = 0,
        IFX_TAPI_RING_CFG_MODE_INT_UNBALANCED_ROT = 1,
        IFX_TAPI_RING_CFG_MODE_INT_UNBALANCED_ROR = 2,
        IFX_TAPI_RING_CFG_MODE_EXT_IT_CS = 3,
        IFX_TAPI_RING_CFG_MODE_EXT_IO_CS = 4
} IFX_TAPI_RING_CFG_MODE_t;
```

Parameters

Name	Value	Description
IFX_TAPI_RING_CFG_MODE_INT_BALANCED	0_D	Internal balanced.
IFX_TAPI_RING_CFG_MODE_INT_UNBALANCED_ROT	1_D	Internal unbalanced ROT.
IFX_TAPI_RING_CFG_MODE_INT_UNBALANCED_ROR	2_D	Internal unbalanced ROR.
IFX_TAPI_RING_CFG_MODE_EXT_IT_CS	3_D	External SLIC current sense.
IFX_TAPI_RING_CFG_MODE_EXT_IO_CS	4_D	External IO current sense.

4.2.6.49 IFX_TAPI_RING_CFG_SUBMODE_t

Description

Ring Configuration SubMode.

Prototype

```
typedef enum
{
        IFX_TAPI_RING_CFG_SUBMODE_DC_RNG_TRIP_STANDARD = 0,
        IFX_TAPI_RING_CFG_SUBMODE_DC_RNG_TRIP_FAST = 1,
        IFX_TAPI_RING_CFG_SUBMODE_AC_RNG_TRIP_STANDARD = 2,
        IFX_TAPI_RING_CFG_SUBMODE_AC_RNG_TRIP_FAST = 3
} IFX_TAPI_RING_CFG_SUBMODE_t;
```

Parameters

Name	Value	Description
IFX_TAPI_RING_CFG_SUBMODE_DC_RNG_TRIP_STANDARD	0_D	DC Ring Trip standard.
IFX_TAPI_RING_CFG_SUBMODE_DC_RNG_TRIP_FAST	1_D	DC Ring Trip fast.
IFX_TAPI_RING_CFG_SUBMODE_AC_RNG_TRIP_STANDARD	2_D	AC Ring Trip standard.
IFX_TAPI_RING_CFG_SUBMODE_AC_RNG_TRIP_FAST	3_D	AC Ring Trip fast.

4.2.6.50 IFX_TAPI_RUNTIME_ERROR_t

Description

TAPI Runtime Errors.

Prototype

```
typedef enum
{
        IFX_TAPI_RT_ERROR_NONE = 0,
        IFX_TAPI_RT_ERROR_RINGCADENCE_CIDTX = 1,
        IFX_TAPI_RT_ERROR_CIDTX_NOACK = 2
} IFX_TAPI_RUNTIME_ERROR_t;
```

Parameters

Name	Value	Description
IFX_TAPI_RT_ERROR_NONE	0_D	No error.
IFX_TAPI_RT_ERROR_RINGCADENCE_CIDTX	1_D	Ring cadence settings error in cid tx.
IFX_TAPI_RT_ERROR_CIDTX_NOACK	2_D	No acknowledge during CID sequence.

4.2.6.51 IFX_TAPI_SIG_t

Description

List the tone detection options.

Some application maybe not interested whether the signal came from the receive or transmit path. Therefore for each signal a mask exists, that includes receive and transmit path.

Prototype

```
typedef enum
{
        IFX_TAPI_SIG_NONE = 0x0,
        IFX_TAPI_SIG_DISRX = 0x1,
        IFX_TAPI_SIG_DISTX = 0x2,
        IFX_TAPI_SIG_DIS = 0x4,
        IFX_TAPI_SIG_CEDRX = 0x8,
        IFX_TAPI_SIG_CEDTX = 0x10,
        IFX_TAPI_SIG_CED = 0x20,
```

```
        IFX_TAPI_SIG_CNGFAXRX = 0x40,
        IFX_TAPI_SIG_CNGFAXTX = 0x80,
        IFX_TAPI_SIG_CNGFAX = 0x100,
        IFX_TAPI_SIG_CNGMODRX = 0x200,
        IFX_TAPI_SIG_CNGMODTX = 0x400,
        IFX_TAPI_SIG_CNGMOD = 0x800,
        IFX_TAPI_SIG_PHASEREVRX = 0x1000,
        IFX_TAPI_SIG_PHASEREVTX = 0x2000,
        IFX_TAPI_SIG_PHASEREV = 0x4000,
        IFX_TAPI_SIG_AMRX = 0x8000,
        IFX_TAPI_SIG_AMTX = 0x10000,
        IFX_TAPI_SIG_AM = 0x20000,
        IFX_TAPI_SIG_TONEHOLDING_ENDRX = 0x40000,
        IFX_TAPI_SIG_TONEHOLDING_ENDTX = 0x80000,
        IFX_TAPI_SIG_TONEHOLDING_END = 0x100000,
        IFX_TAPI_SIG_CEDENDRX = 0x200000,
        IFX_TAPI_SIG_CEDENDTX = 0x400000,
        IFX_TAPI_SIG_CEDEND = 0x800000,
        IFX_TAPI_SIG_CPTD = 0x1000000,
        IFX_TAPI_SIG_V8BISRX = 0x2000000,
        IFX_TAPI_SIG_V8BISTX = 0x4000000
} IFX_TAPI_SIG_t;
```

Parameters

Name	Value	Description
IFX_TAPI_SIG_NONE	0_H	No signal detected
IFX_TAPI_SIG_DISRX	1_H	V.21 Preamble Fax Tone, Digital identification signal (DIS), receive path.
IFX_TAPI_SIG_DISTX	2_H	V.21 Preamble Fax Tone, Digital identification signal (DIS), transmit path.
IFX_TAPI_SIG_DIS	4_H	V.21 Preamble Fax Tone in all path, Digital identification signal (DIS).
IFX_TAPI_SIG_CEDRX	8_H	V.25 2100 Hz (CED) Modem/Fax Tone, receive path.
IFX_TAPI_SIG_CEDTX	10_H	V.25 2100 Hz (CED) Modem/Fax Tone, transmit path.
IFX_TAPI_SIG_CED	20_H	V.25 2100 Hz (CED) Modem/Fax Tone in all paths.
IFX_TAPI_SIG_CNGFAXRX	40_H	CNG Fax Calling Tone (1100 Hz) receive path.
IFX_TAPI_SIG_CNGFAXTX	80_H	CNG Fax Calling Tone (1100 Hz) transmit path.
IFX_TAPI_SIG_CNGFAX	100_H	CNG Fax Calling Tone (1100 Hz) in all paths.
IFX_TAPI_SIG_CNGMODRX	200_H	CNG Modem Calling Tone (1300 Hz) receive path.
IFX_TAPI_SIG_CNGMODTX	400_H	CNG Modem Calling Tone (1300 Hz) transmit path.
IFX_TAPI_SIG_CNGMOD	800_H	CNG Modem Calling Tone (1300 Hz) in all paths.

Name	Value	Description
IFX_TAPI_SIG_PHASEREVRX	1000_H	Phase reversal detection receive path.
IFX_TAPI_SIG_PHASEREVTX	2000_H	Phase reversal detection transmit path.
IFX_TAPI_SIG_PHASEREV	4000_H	Phase reversal detection in all paths.
IFX_TAPI_SIG_AMRX	8000_H	Amplitude modulation receive path.
IFX_TAPI_SIG_AMTX	10000_H	Amplitude modulation transmit path.
IFX_TAPI_SIG_AM	20000_H	Amplitude modulation.
IFX_TAPI_SIG_TONEHOLDING_ENDRX	40000_H	Modem tone holding signal stopped receive path.
IFX_TAPI_SIG_TONEHOLDING_ENDTX	80000_H	Modem tone holding signal stopped transmit path.
IFX_TAPI_SIG_TONEHOLDING_END	100000_H	Modem tone holding signal stopped all paths.
IFX_TAPI_SIG_CEDENDRX	200000_H	End of signal CED detection receive path.
IFX_TAPI_SIG_CEDENDTX	400000_H	End of signal CED detection transmit path.
IFX_TAPI_SIG_CEDEND	800000_H	End of signal CED detection. This signal also includes information about phase reversals and amplitude modulation, if enabled
IFX_TAPI_SIG_CPTD	1000000_H	Signals a call progress tone detection. This signal is enabled with the interface **IFX_TAPI_TONE_CPTD_START** and stopped with **IFX_TAPI_TONE_CPTD_STOP**. It can not be activate with **IFX_TAPI_SIG_DETECT_ENABLE**
IFX_TAPI_SIG_V8BISRX	2000000_H	Signals the V8bis detection on the receive path.
IFX_TAPI_SIG_V8BISTX	4000000_H	Signals the V8bis detection on the transmit path.

4.2.6.52 IFX_TAPI_T38_ERROR_t

Description

T.38 Fax errors.

Prototype

```
typedef enum
{
        IFX_TAPI_T38_NO_ERR = 0,
        IFX_TAPI_T38_ERR = 1,
        IFX_TAPI_T38_MIPS_OVLD = 2,
        IFX_TAPI_T38_READ_ERR = 3,
        IFX_TAPI_T38_WRITE_ERR = 4,
        IFX_TAPI_T38_DATA_ERR = 5
} IFX_TAPI_T38_ERROR_t;
```

Parameters

Name	Value	Description
IFX_TAPI_T38_NO_ERR	0_D	No Error.
IFX_TAPI_T38_ERR	1_D	Error occurred: Deactivate Datapump.
IFX_TAPI_T38_MIPS_OVLD	2_D	MIPS Overload.
IFX_TAPI_T38_READ_ERR	3_D	Error while reading data.
IFX_TAPI_T38_WRITE_ERR	4_D	Error while writing data.
IFX_TAPI_T38_DATA_ERR	5_D	Error while setting up modulator or demodulator.

4.2.6.53 IFX_TAPI_T38_STATUS_t

Description

T.38 Fax Datapump states.

Prototype

```
typedef enum
{
        IFX_TAPI_T38_DP_OFF = 0x1,
        IFX_TAPI_T38_DP_ON = 0x2,
        IFX_TAPI_T38_TX_ON = 0x4,
        IFX_TAPI_T38_TX_OFF = 0x8
} IFX_TAPI_T38_STATUS_t;
```

Parameters

Name	Value	Description
IFX_TAPI_T38_DP_OFF	1_H	Fax Datapump not active.
IFX_TAPI_T38_DP_ON	2_H	Fax Datapump active.
IFX_TAPI_T38_TX_ON	4_H	Fax transmission is active.
IFX_TAPI_T38_TX_OFF	8_H	Fax transmission is not active.

4.2.6.54 IFX_TAPI_T38_STD_t

Description

T.38 Fax standards.

Prototype

```
typedef enum
{
        IFX_TAPI_T38_STD_V21_STD = 0x1,
        IFX_TAPI_T38_STD_V27_2400_STD = 0x2,
        IFX_TAPI_T38_STD_V27_4800_STD = 0x3,
        IFX_TAPI_T38_STD_V29_7200_STD = 0x4,
        IFX_TAPI_T38_STD_V29_9600_STD = 0x5,
        IFX_TAPI_T38_STD_V17_7200_STD = 0x6,
```

```
        IFX_TAPI_T38_STD_V17_9600_STD = 0x7,
        IFX_TAPI_T38_STD_V17_12000_STD = 0x8,
        IFX_TAPI_T38_STD_V17_14400_STD = 0x9
} IFX_TAPI_T38_STD_t;
```

Parameters

Name	Value	Description
IFX_TAPI_T38_STD_V21_STD	1_H	V.21.
IFX_TAPI_T38_STD_V27_2400_STD	2_H	V.27/2400.
IFX_TAPI_T38_STD_V27_4800_STD	3_H	V.27/4800.
IFX_TAPI_T38_STD_V29_7200_STD	4_H	V.29/7200.
IFX_TAPI_T38_STD_V29_9600_STD	5_H	V.29/9600.
IFX_TAPI_T38_STD_V17_7200_STD	6_H	V.17/7200.
IFX_TAPI_T38_STD_V17_9600_STD	7_H	V.17/9600.
IFX_TAPI_T38_STD_V17_12000_STD	8_H	V.17/12000.
IFX_TAPI_T38_STD_V17_14400_STD	9_H	V.17/14400.

4.2.6.55 IFX_TAPI_TONE_CPTD_DIRECTION_t

Description

Specifies the CPT signal for CPT detection.

Prototype

```
typedef enum
{
        IFX_TAPI_TONE_CPTD_DIRECTION_RX = 0x1,
        IFX_TAPI_TONE_CPTD_DIRECTION_TX = 0x2
} IFX_TAPI_TONE_CPTD_DIRECTION_t;
```

Parameters

Name	Value	Description
IFX_TAPI_TONE_CPTD_DIRECTION_RX	1_H	
IFX_TAPI_TONE_CPTD_DIRECTION_TX	2_H	

4.2.6.56 IFX_TAPI_TONE_FREQ_t

Description

Used for selection of one or more frequencies belonging to a tone cadence.

Prototype

```
typedef enum
{
        IFX_TAPI_TONE_FREQNONE = 0x0,
        IFX_TAPI_TONE_FREQA = 0x1,
        IFX_TAPI_TONE_FREQB = 0x2,
```

```
        IFX_TAPI_TONE_FREQC = 0x4,
        IFX_TAPI_TONE_FREQD = 0x8,
        IFX_TAPI_TONE_FREQALL = 0xF
} IFX_TAPI_TONE_FREQ_t;
```

Parameters

Name	Value	Description
IFX_TAPI_TONE_FREQNONE	0_H	All frequencies are inactive
IFX_TAPI_TONE_FREQA	1_H	Play frequency A
IFX_TAPI_TONE_FREQB	2_H	Play frequency B
IFX_TAPI_TONE_FREQC	4_H	Play frequency C
IFX_TAPI_TONE_FREQD	8_H	Play frequency D
IFX_TAPI_TONE_FREQALL	F_H	Play all frequencies

4.2.6.57 IFX_TAPI_TONE_GROUP_t

Description

Tone grouping.

Prototype

```
typedef enum
{
        IFX_TAPI_TONE_GROUP_NONE = 0,
        IFX_TAPI_TONE_GROUP_AB = 1,
        IFX_TAPI_TONE_GROUP_BC = 2,
        IFX_TAPI_TONE_GROUP_AB_BC = 3
} IFX_TAPI_TONE_GROUP_t;
```

Parameters

Name	Value	Description
IFX_TAPI_TONE_GROUP_NONE	0_D	All tones are sent as single tones
IFX_TAPI_TONE_GROUP_AB	1_D	The frequencies of A and B are sent as dual tone followed by frequency C sent as single tone.
IFX_TAPI_TONE_GROUP_BC	2_D	For group two the frequency A is sent as a single tone followed by frequencies B and C as dual tone.
IFX_TAPI_TONE_GROUP_AB_BC	3_D	The frequencies of A and B are sent as dual tone followed by frequency B and C as dual tone, not yet supported.

4.2.6.58 IFX_TAPI_TONE_MODULATION_t

Description

Modulation setting for a cadence step.

Prototype

```
typedef enum
{
        IFX_TAPI_TONE_MODULATION_OFF = 0,
        IFX_TAPI_TONE_MODULATION_ON = 1
} IFX_TAPI_TONE_MODULATION_t;
```

Parameters

Name	Value	Description
IFX_TAPI_TONE_MODULATION_OFF	0_D	Modulation off for the cadence step
IFX_TAPI_TONE_MODULATION_ON	1_D	Modulation on for the cadence step

4.2.6.59 IFX_TAPI_TONE_TG_t

Description

Defines the tone generator usage.

Prototype

```
typedef enum
{
        IFX_TAPI_TONE_TG1 = 1,
        IFX_TAPI_TONE_TG2 = 2,
        IFX_TAPI_TONE_TGALL = 0xFF
} IFX_TAPI_TONE_TG_t;
```

Parameters

Name	Value	Description
IFX_TAPI_TONE_TG1	1_D	Use tone generator 1.
IFX_TAPI_TONE_TG2	2_D	Use tone generator 2.
IFX_TAPI_TONE_TGALL	FF_H	Use all tone generators.

4.2.6.60 IFX_TAPI_TONE_TYPE_t

Description

Tone types.

Prototype

```
typedef enum
{
        IFX_TAPI_TONE_TYPE_SIMPLE = 1,
        IFX_TAPI_TONE_TYPE_COMPOSED = 2
} IFX_TAPI_TONE_TYPE_t;
```

Parameters

Name	Value	Description
IFX_TAPI_TONE_TYPE_SIMPLE	1_D	Simple tone
IFX_TAPI_TONE_TYPE_COMPOSED	2_D	Composed tone

4.2.6.61 IFX_TAPI_WLEC_NLP_t

Description

NLP configuration.

Prototype

```
typedef enum
{
        IFX_TAPI_WLEC_NLP_DEFAULT = 0,
        IFX_TAPI_WLEC_NLP_ON = 1,
        IFX_TAPI_WLEC_NLP_OFF = 2
} IFX_TAPI_WLEC_NLP_t;
```

Parameters

Name	Value	Description
IFX_TAPI_WLEC_NLP_DEFAULT	0_D	Reserved.
IFX_TAPI_WLEC_NLP_ON	1_D	Enable NLP.
IFX_TAPI_WLEC_NLP_OFF	2_D	Disable NLP.

4.2.6.62 IFX_TAPI_WLEC_TYPE_t

Description

LEC type definition.

Prototype

```
typedef enum
{
        IFX_TAPI_WLEC_TYPE_OFF = 0,
        IFX_TAPI_WLEC_TYPE_NE = 1,
        IFX_TAPI_WLEC_TYPE_NFE = 2
} IFX_TAPI_WLEC_TYPE_t;
```

Parameters

Name	Value	Description
IFX_TAPI_WLEC_TYPE_OFF	0_D	Line echo cancellation disabled..
IFX_TAPI_WLEC_TYPE_NE	1_D	Near-end echo cancellation enabled. .
IFX_TAPI_WLEC_TYPE_NFE	2_D	Near-end and far-end echo cancellation enabled..

References

[1] T.38 Fax Agent Release 1.1 User's Manual Programmer's Reference Rev. 1.0, 2005-09-22

[2] T.38 Protocol Stack Release 1.16 User's Manual Programmer's Reference Rev. 1.0, 2005-09-22

Attention: Please refer to the latest revision of the documents.

Terminology

A	
ACK	Acknowledge
AGC	Automatic Gain Control
ALM	Analog Line Module
API	Application Program Interface
B	
BT	British Telecom
C	
CFG	Configuration
CH	Channel
CID	Caller ID
COD	Coder module
CPE	Customer Premises Equipment
CPT	Call Progress Tone
CTPD	Call Progress Tone Detector
D	
DEC	Decoding
DTMF	Dual Tone Multiple Frequency
E	
ENC	Encoding
ETSI	European Telecommunications Standards Institute
F	
Fd	File descriptor
FSK	Frequency Shift Keying
FW	Firmware
G	
GPIO	General Purpose Input Output
H	
HL	High Level
HW	Hardware
I	
ICA	In-call announcement
IETF	Internet Engineering Task Force
IFX	Infineon
IO	Input/Output
J	
JB	Jitter Buffer
L	
LEC	Line Echo Canceller
LL	Low Level
LR	Line Reversal

LT	Line Testing
M	
MSG	Message
MWI	Message Waiting Indication
N	
NLP	Non Linear Processing
NTT	Nippon Telegraph and Telephone Company
O	
OOB	Out of band
P	
PCM	Pulse Code Modulation
PKT	Packet
POTS	Plain Old Telephone System
PSTN	Public Switched Telephone Network
R	
RFC	IETF Request for Comment
RTCP	Real Time Control Protocol
RTP	Real Time Protocol
RX	Receive
S	
SID	Silence Insertion Descriptor
SIG	Signaling module
SIN	British Telecom Supplier’s Information Note
SSRC	Synchronization source
SW	Software
T	
TAPI	Telephone API
TX	Transmit
U	
UTG	Universal Tone Generator
V	
VAD	Voice Activity Detector
VMWI	Visual Message Waiting Indication
VoIP	Voice over IP

www.infineon.com

Published by Infineon Technologies AG