

VINETIC®

Voice and Internet Enhanced Telephony Interface Circuit

VINETIC® Device Driver

for

VINETIC®-4VIP (PEB 3324), Version 1.4

VINETIC®-4S (PEF 3304), Version 2.1

VINETIC®-4C (PEF 3394), Version 2.2

VINETIC®-4M (PEF 3314), Version 2.2

Preliminary

User's Manual

Driver and API Description

Rev. 1.0

Communication Solutions



Never stop thinking

Edition 2006-03-10

**Published by
Infineon Technologies AG
81726 München, Germany**

**© Infineon Technologies AG 2006.
All Rights Reserved.**

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.

Page	Subjects (major changes since last revision)

Trademarks

ABM®, ACE®, AOP®, Arcoff®, ASM®, ASP®, BlueMoon®, BlueNIX®, ConverGate®, C166®, DUALFALC®, DuSLIC®, ELIC®, EPIC®, FALC®, GEMINAX®, IDEC®, INCA®, IOM®, Ipat®-2, IPVD®, Isac®, ITAC®, IWE®, IWORX®, M-GOLD®, MUSAC®, MuSLIC®, OCTALFALC®, OCTAT®, POTSWIRE®, QUADFALC®, QUAT®, SCOUT®, SCT®, SEROCCO®, S-GOLD®, SICAT®, SICOFI®, SIDEC®, SIEGET®, SLICOFI®, SMARTI®, SOCRATES®, VDSLite®, VINETIC®, 10BaseS® are registered trademarks of Infineon Technologies AG.

DIGITAPE™, EasyPort™, E-GOLD™, E-GOLDlite™, S-GOLDlite™, S-GOLD2™, S-GOLD3™, VINAX™, WildPass™, 10BaseV™, 10BaseVX™ are trademarks of Infineon Technologies AG.

Microsoft® and Visio® are registered trademarks of Microsoft Corporation. Linux® is a registered trademark of Linus Torvalds. FrameMaker® is a registered trademark of Adobe Systems Incorporated. APOXI® is a registered trademark of Comneon GmbH & Co. OHG. PrimeCell®, RealView®, ARM® are registered trademarks of ARM Limited. OakDSPCore®, TeakLite® DSP Core, OCEM® are registered trademarks of ParthusCeva Inc.

IndoorGPS™, GL-20000™, GL-LN-22™ are trademarks of Global Locate. ARM926EJ-S™, ADS™, Multi-ICE™ are trademarks of ARM Limited.

Table of Contents

1	Development Environment Setup	19
1.1	General Introduction to the Device Driver	19
1.2	Files	19
1.3	Device Nodes	19
1.3.1	Linux	19
1.3.2	VxWorks	19
1.4	Compilation	20
1.5	Interface Names and Backwards Compatibility	21
1.6	Linetesting Support	22
2	First Steps	23
2.1	File Descriptors	23
2.1.1	Device Nodes	23
2.1.2	Phone and Data Channel Resources	24
2.2	Initialization	26
2.2.1	Basic Device Driver Init	26
2.2.2	TAPI Channel initialization	26
2.2.2.1	VoIP Initialization (default)	26
2.2.2.2	PCM with DSP Initialization	27
2.2.2.3	PCM with Analog TAPI Initialization	27
2.2.3	Initialization Example	28
2.3	Line Feeding Modes	30
2.4	Manual Battery Switching	31
2.4.1	Battery Switching Implementation	31
2.4.2	Battery Switching Programming Example	31
2.5	Event Detection	33
2.5.1	Hook State	34
2.5.2	DTMF and Pulse Digits Detection	35
2.5.3	Masking Events	36
2.5.4	Configure Hook Validation Timing	36
2.6	Establishing Connections	38
2.6.1	Internal Call	38
2.6.2	Voice Call with External Party	40
2.6.3	Conferencing	41
2.6.3.1	Initialization	41
2.6.3.2	Conference with an Internal and an External Party	42
2.6.3.3	Conference with Two External Parties	43
3	Feature Description	45
3.1	Channel Configuration	45
3.1.1	Phone Volume	45
3.1.2	PCM	46
3.1.2.1	Example of System using PCM Interface	47
3.1.3	LEC	48
3.1.4	Jitter Buffer	48
3.1.5	RTP	49
3.1.5.1	RTP Payload Type Tables	49
3.1.5.2	RTP Session Parameters	50
3.1.6	AAL2	50
3.2	Encoder/Decoder Control	51

3.3	RTCP and Jitter Buffer Statistics	52
3.4	Tone API	52
3.4.1	Tone Table	53
3.4.2	Playing Tones	54
3.4.3	Defining Simple Tones	54
3.4.4	Defining Composed Tones	55
3.4.5	Predefined Tones	56
3.4.6	Generating Tones with High-level Output	57
3.4.7	Generating Special Tones	58
3.4.8	Call Progress Tone Detection	59
3.5	Ringling and Caller ID Support	60
3.5.1	Introduction to Ringling and Caller ID Features	60
3.5.1.1	Information on Caller ID	60
3.5.2	CID Configuration	62
3.5.3	Ringling Support	65
3.5.3.1	Configure Ringling Type	65
3.5.3.2	Configure Ring Cadence	66
3.5.3.3	Start / Stop Ringling	66
3.5.4	CID TX	67
3.5.4.1	Prepare CID Message	67
3.5.4.2	Examples of CID TX	68
3.5.5	CID RX	72
3.6	Fax/Modem Support	72
3.6.1	Detect Fax/Modem Signals	73
3.6.2	Pass-Through Mode	73
3.6.3	T.38 Data-Pump Mode	75
3.7	Metering	75
4	Line Testing Services	77
4.1	Line Testing	77
4.1.1	Line Testing System Integration	77
4.1.1.1	Board Specific Constants	78
4.1.1.2	Board Specific IO Pin Mapping	79
4.1.1.3	Board Specific Voltages (optional)	79
4.2	Line Testing Measurements	80
4.3	Network Linetesting Code Setup	80
4.4	GR-909 Measurements	81
5	Device Driver Services	83
5.1	Polling Mode	83
5.1.1	Polling and Interleaved Polling	83
5.1.2	Buffer Handling	84
5.1.3	Driver Polling Interface	84
5.2	General-Purpose IOs	85
6	TAPI Interfaces	87
6.1	ioctl commands of TAPI Interfaces	87
6.1.1	CID Features Service	88
6.1.1.1	IFX_TAPI_CID_CFG_SET	89
6.1.1.2	IFX_TAPI_CID_RX_DATA_GET	90
6.1.1.3	IFX_TAPI_CID_RX_START	91
6.1.1.4	IFX_TAPI_CID_RX_STATUS_GET	92
6.1.1.5	IFX_TAPI_CID_RX_STOP	93

6.1.1.6	IFX_TAPI_CID_TX_INFO_START	94
6.1.1.7	IFX_TAPI_CID_TX_SEQ_START	95
6.1.2	Connection Control Service	96
6.1.2.1	IFX_TAPI_DEC_LEVEL_GET	97
6.1.2.2	IFX_TAPI_DEC_START	98
6.1.2.3	IFX_TAPI_DEC_STOP	99
6.1.2.4	IFX_TAPI_DEC_TYPE_SET	99
6.1.2.5	IFX_TAPI_DEC_VOLUME_SET	100
6.1.2.6	IFX_TAPI_ENC_FRAME_LEN_GET	101
6.1.2.7	IFX_TAPI_ENC_FRAME_LEN_SET	102
6.1.2.8	IFX_TAPI_ENC_LEVEL_SET	102
6.1.2.9	IFX_TAPI_ENC_START	103
6.1.2.10	IFX_TAPI_ENC_STOP	104
6.1.2.11	IFX_TAPI_ENC_TYPE_SET	105
6.1.2.12	IFX_TAPI_ENC_VAD_CFG_SET	105
6.1.2.13	IFX_TAPI_ENC_VOLUME_SET	106
6.1.2.14	IFX_TAPI_JB_CFG_SET	107
6.1.2.15	IFX_TAPI_JB_STATISTICS_GET	108
6.1.2.16	IFX_TAPI_JB_STATISTICS_RESET	109
6.1.2.17	IFX_TAPI_MAP_DATA_ADD	109
6.1.2.18	IFX_TAPI_MAP_DATA_REMOVE	110
6.1.2.19	IFX_TAPI_MAP_PCM_ADD	111
6.1.2.20	IFX_TAPI_MAP_PCM_REMOVE	112
6.1.2.21	IFX_TAPI_MAP_PHONE_ADD	113
6.1.2.22	IFX_TAPI_MAP_PHONE_REMOVE	114
6.1.2.23	IFX_TAPI_PKT_AAL_CFG_SET	115
6.1.2.24	IFX_TAPI_PKT_AAL_PROFILE_SET	115
6.1.2.25	IFX_TAPI_PKT_RTCP_STATISTICS_GET	117
6.1.2.26	IFX_TAPI_PKT_RTCP_STATISTICS_RESET	117
6.1.2.27	IFX_TAPI_PKT_RTP_CFG_SET	118
6.1.2.28	IFX_TAPI_PKT_RTP_PT_CFG_SET	119
6.1.3	Dial Service	121
6.1.3.1	IFX_TAPI_PKT_EV_OOB_SET	121
6.1.3.2	IFX_TAPI_PULSE_ASCII_GET	122
6.1.3.3	IFX_TAPI_PULSE_GET	123
6.1.3.4	IFX_TAPI_PULSE_READY	124
6.1.3.5	IFX_TAPI_TONE_DTMF_ASCII_GET	125
6.1.3.6	IFX_TAPI_TONE_DTMF_GET	125
6.1.3.7	IFX_TAPI_TONE_DTMF_READY_GET	127
6.1.4	Fax T.38 Service	129
6.1.4.1	IFX_TAPI_T38_DEMOD_START	129
6.1.4.2	IFX_TAPI_T38_MOD_START	130
6.1.4.3	IFX_TAPI_T38_STATUS_GET	131
6.1.4.4	IFX_TAPI_T38_STOP	132
6.1.5	Initialization Service	134
6.1.5.1	IFX_TAPI_CH_INIT	134
6.1.6	Metering Service	136
6.1.6.1	IFX_TAPI_METER_CFG_SET	136
6.1.6.2	IFX_TAPI_METER_START	137
6.1.6.3	IFX_TAPI_METER_STOP	138
6.1.7	Miscellaneous Services	139

6.1.7.1	IFX_TAPI_CAP_CHECK	139
6.1.7.2	IFX_TAPI_CAP_LIST	141
6.1.7.3	IFX_TAPI_CAP_NR	142
6.1.7.4	IFX_TAPI_CH_STATUS_GET	143
6.1.7.5	IFX_TAPI_DEBUG_REPORT_SET	145
6.1.7.6	IFX_TAPI_EXCEPTION_GET	146
6.1.7.7	IFX_TAPI_EXCEPTION_MASK	147
6.1.7.8	IFX_TAPI_VERSION_CHECK	148
6.1.7.9	IFX_TAPI_VERSION_GET	148
6.1.8	Operation Control Service	149
6.1.8.1	IFX_TAPI_LEC_PCM_CFG_GET	150
6.1.8.2	IFX_TAPI_LEC_PCM_CFG_SET	150
6.1.8.3	IFX_TAPI_LEC_PHONE_CFG_SET	151
6.1.8.4	IFX_TAPI_LEC_PHONE_CFG_GET	152
6.1.8.5	IFX_TAPI_LINE_FEED_SET	153
6.1.8.6	IFX_TAPI_LINE_HOOK_STATUS_GET	153
6.1.8.7	IFX_TAPI_LINE_HOOK_VT_SET	154
6.1.8.8	IFX_TAPI_LINE_LEVEL_SET	155
6.1.8.9	IFX_TAPI_PCM_VOLUME_SET	156
6.1.8.10	IFX_TAPI_PHONE_VOLUME_SET	157
6.1.9	PCM Service	159
6.1.9.1	IFX_TAPI_PCM_ACTIVATION_GET	159
6.1.9.2	IFX_TAPI_PCM_ACTIVATION_SET	160
6.1.9.3	IFX_TAPI_PCM_CFG_GET	160
6.1.9.4	IFX_TAPI_PCM_CFG_SET	161
6.1.10	Ringing Service	163
6.1.10.1	IFX_TAPI_RING_CADENCE_HR_SET	163
6.1.10.2	IFX_TAPI_RING_CADENCE_SET	164
6.1.10.3	IFX_TAPI_RING_CFG_GET	165
6.1.10.4	IFX_TAPI_RING_CFG_SET	166
6.1.10.5	IFX_TAPI_RING_START	166
6.1.10.6	IFX_TAPI_RING_STOP	167
6.1.11	Signal Detection Service	169
6.1.11.1	IFX_TAPI_SIG_DETECT_DISABLE	169
6.1.11.2	IFX_TAPI_SIG_DETECT_ENABLE	170
6.1.11.3	IFX_TAPI_TONE_CPTD_START	171
6.1.11.4	IFX_TAPI_TONE_CPTD_STOP	172
6.1.12	Test Services	173
6.1.12.1	IFX_TAPI_TEST_HOOKGEN	173
6.1.12.2	IFX_TAPI_TEST_LOOP	173
6.1.13	Tone Control Service	175
6.1.13.1	IFX_TAPI_TONE_LOCAL_PLAY	176
6.1.13.2	IFX_TAPI_TONE_NET_PLAY	177
6.1.13.3	IFX_TAPI_TONE_STATUS_GET	177
6.1.13.4	IFX_TAPI_TONE_STOP	178
6.1.13.5	IFX_TAPI_TONE_TABLE_CFG_SET	179
6.2	Type Definition Reference	181
6.2.1	Basic Type Definitions	181
6.2.1.1	IFX_return_t	181
6.2.1.2	IFX_boolean_t	181
6.2.1.3	IFX_uint8_t	182

6.2.1.4	IFX_int8_t	182
6.2.1.5	IFX_uint32_t	182
6.2.1.6	IFX_int32_t	182
6.2.1.7	IFX_uint16_t	183
6.2.1.8	IFX_int16_t	183
6.2.1.9	IFX_char_t	183
6.2.1.10	IFX_void_t	183
6.2.1.11	IFX_float_t	183
6.2.2	IO-control Reference	184
6.2.3	Constant Reference	186
6.2.4	Structure Reference	188
6.2.4.1	IFX_TAPI_CAP_t	189
6.2.4.2	IFX_TAPI_CH_INIT_t	190
6.2.4.3	IFX_TAPI_CH_STATUS_t	191
6.2.4.4	IFX_TAPI_CID_ABS_REASON_t	195
6.2.4.5	IFX_TAPI_CID_CFG_t	196
6.2.4.6	IFX_TAPI_CID_DTMF_CFG_t	196
6.2.4.7	IFX_TAPI_CID_FSK_CFG_t	197
6.2.4.8	IFX_TAPI_CID_MSG_DATE_t	198
6.2.4.9	IFX_TAPI_CID_MSG_STRING_t	198
6.2.4.10	IFX_TAPI_CID_MSG_t	199
6.2.4.11	IFX_TAPI_CID_MSG_TRANSPARENT_t	199
6.2.4.12	IFX_TAPI_CID_MSG_VALUE_t	200
6.2.4.13	IFX_TAPI_CID_RX_DATA_t	200
6.2.4.14	IFX_TAPI_CID_RX_STATUS_t	201
6.2.4.15	IFX_TAPI_CID_STD_ETSI_DTMF_t	201
6.2.4.16	IFX_TAPI_CID_STD_ETSI_FSK_t	202
6.2.4.17	IFX_TAPI_CID_STD_NTT_t	203
6.2.4.18	IFX_TAPI_CID_STD_SIN_t	204
6.2.4.19	IFX_TAPI_CID_STD_TELCORDIA_t	205
6.2.4.20	IFX_TAPI_CID_TIMING_t	205
6.2.4.21	IFX_TAPI_EXCEPTION_BITS_t	206
6.2.4.22	IFX_TAPI_JB_CFG_t	208
6.2.4.23	IFX_TAPI_JB_STATISTICS_t	209
6.2.4.24	IFX_TAPI_LEC_CFG_t	211
6.2.4.25	IFX_TAPI_LINE_HOOK_VT_t	212
6.2.4.26	IFX_TAPI_LINE_VOLUME_t	213
6.2.4.27	IFX_TAPI_MAP_DATA_t	213
6.2.4.28	IFX_TAPI_MAP_PCM_t	214
6.2.4.29	IFX_TAPI_MAP_PHONE_t	215
6.2.4.30	IFX_TAPI_METER_CFG_t	216
6.2.4.31	IFX_TAPI_PCK_AAL_CFG_t	216
6.2.4.32	IFX_TAPI_PCK_AAL_PROFILE_t	217
6.2.4.33	IFX_TAPI_PCM_CFG_t	217
6.2.4.34	IFX_TAPI_PKT_RTCP_STATISTICS_t	218
6.2.4.35	IFX_TAPI_PKT_RTP_CFG_t	219
6.2.4.36	IFX_TAPI_PKT_RTP_PT_CFG_t	220
6.2.4.37	IFX_TAPI_RING_CADENCE_t	220
6.2.4.38	IFX_TAPI_RING_CFG_t	221
6.2.4.39	IFX_TAPI_SIG_DETECTION_t	222
6.2.4.40	IFX_TAPI_T38_DEMOD_DATA_t	222

6.2.4.41	IFX_TAPI_T38_MOD_DATA_t	223
6.2.4.42	IFX_TAPI_T38_STATUS_t	225
6.2.4.43	IFX_TAPI_TEST_LOOP_t	225
6.2.4.44	IFX_TAPI_TONE_COMPOSED_t	226
6.2.4.45	IFX_TAPI_TONE_CPTD_t	227
6.2.4.46	IFX_TAPI_TONE_SIMPLE_t	227
6.2.4.47	IFX_TAPI_VERSION_t	229
6.2.5	Union Reference	229
6.2.5.1	IFX_TAPI_CID_MSG_ELEMENT_t	229
6.2.5.2	IFX_TAPI_CID_STD_TYPE_t	230
6.2.5.3	IFX_TAPI_EXCEPTION_t	230
6.2.5.4	IFX_TAPI_TONE_t	231
6.2.6	Enumerator Reference	231
6.2.6.1	IFX_TAPI_CAP_PORT_t	233
6.2.6.2	IFX_TAPI_CAP_SIG_DETECT_t	233
6.2.6.3	IFX_TAPI_CAP_TYPE_t	234
6.2.6.4	IFX_TAPI_CH_INIT_COUNTRY_t	235
6.2.6.5	IFX_TAPI_CH_INIT_MODE_t	235
6.2.6.6	IFX_TAPI_CID_ABSREASON_t	236
6.2.6.7	IFX_TAPI_CID_ALERT_ETSI_t	236
6.2.6.8	IFX_TAPI_CID_HOOK_MODE_t	237
6.2.6.9	IFX_TAPI_CID_MSG_TYPE_t	237
6.2.6.10	IFX_TAPI_CID_RX_ERROR_t	238
6.2.6.11	IFX_TAPI_CID_RX_STATE_t	239
6.2.6.12	IFX_TAPI_CID_SERVICE_TYPE_t	239
6.2.6.13	IFX_TAPI_CID_STD_t	241
6.2.6.14	IFX_TAPI_CID_VMWI_t	241
6.2.6.15	IFX_TAPI_DATA_MAP_START_STOP_t	242
6.2.6.16	IFX_TAPI_DEBUG_REPORT_SET_t	242
6.2.6.17	IFX_TAPI_DIALING_STATUS_t	243
6.2.6.18	IFX_TAPI_ENC_TYPE_t	243
6.2.6.19	IFX_TAPI_ENC_VAD_t	244
6.2.6.20	IFX_TAPI_JB_LOCAL_ADAPT_t	245
6.2.6.21	IFX_TAPI_JB_PKT_ADAPT_t	245
6.2.6.22	IFX_TAPI_JB_TYPE_t	246
6.2.6.23	IFX_TAPI_LEC_GAIN_t	246
6.2.6.24	IFX_TAPI_LEC_NLP_t	247
6.2.6.25	IFX_TAPI_LEC_TYPE_t	247
6.2.6.26	IFX_TAPI_LINE_HOOK_STATUS_t	248
6.2.6.27	IFX_TAPI_LINE_FEED_t	248
6.2.6.28	IFX_TAPI_LINE_HOOK_VALIDATION_TYPE_t	250
6.2.6.29	IFX_TAPI_LINE_LEVEL_t	251
6.2.6.30	IFX_TAPI_LINE_STATUS_t	251
6.2.6.31	IFX_TAPI_MAP_DATA_TYPE_t	252
6.2.6.32	IFX_TAPI_MAP_DEC_t	252
6.2.6.33	IFX_TAPI_MAP_ENC_t	253
6.2.6.34	IFX_TAPI_MAP_TYPE_t	253
6.2.6.35	IFX_TAPI_METER_MODE_t	254
6.2.6.36	IFX_TAPI_PCM_RES_t	254
6.2.6.37	IFX_TAPI_PKT_AAL_PROFILE_RANGE_t	255
6.2.6.38	IFX_TAPI_PKT_EV_NUM_t	255

6.2.6.39	IFX_TAPI_PKT_EV_OOB_t	256
6.2.6.40	IFX_TAPI_RING_CFG_MODE_t	257
6.2.6.41	IFX_TAPI_RING_CFG_SUBMODE_t	257
6.2.6.42	IFX_TAPI_RUNTIME_ERROR_t	258
6.2.6.43	IFX_TAPI_SIG_t	258
6.2.6.44	IFX_TAPI_T38_ERROR_t	260
6.2.6.45	IFX_TAPI_T38_STATUS_t	261
6.2.6.46	IFX_TAPI_T38_STD_t	261
6.2.6.47	IFX_TAPI_TONE_CPTD_DIRECTION_t	262
6.2.6.48	IFX_TAPI_TONE_FREQ_t	262
6.2.6.49	IFX_TAPI_TONE_GROUP_t	263
6.2.6.50	IFX_TAPI_TONE_MODULATION_t	264
6.2.6.51	IFX_TAPI_TONE_TG_t	264
6.2.6.52	IFX_TAPI_TONE_TYPE_t	265
6.3	Mapping to Old TAPI Names	265
6.3.1	Mapping of ioctl Interfaces	265
6.3.2	Mapping of Constants	268
6.3.3	Mapping of Structures	269
6.3.4	Mapping of Unions	270
6.3.5	Mapping of Enumerations	270
7	Device Driver Interfaces	273
7.1	ioctl commands of the Device Driver Interface	273
7.1.1	Basic Interface	274
7.1.1.1	FIO_VINETIC_DOW_PHI	275
7.1.1.2	FIO_VINETIC_DOWNFPI	275
7.1.1.3	FIO_VINETIC_DOWNLOAD_CRAM	276
7.1.1.4	FIO_VINETIC_DRV_CTRL	277
7.1.1.5	FIO_VINETIC_INIT	277
7.1.1.6	FIO_VINETIC_LASTERR	278
7.1.1.7	FIO_VINETIC_RCMD	279
7.1.1.8	FIO_VINETIC_REPORT_SET	280
7.1.1.9	FIO_VINETIC_RSC	280
7.1.1.10	FIO_VINETIC_RUNTIME_TRACE_SET	281
7.1.1.11	FIO_VINETIC_VERS	282
7.1.1.12	FIO_VINETIC_WCMD	282
7.1.1.13	FIO_VINETIC_WSC	283
7.1.2	Driver Initialization Interface	284
7.1.2.1	FIO_VINETIC_BASICDEV_INIT	284
7.1.2.2	FIO_VINETIC_DEV_RESET	285
7.1.3	Driver Kernel Interfaces	286
7.1.4	GPIO Interface	287
7.1.4.1	FIO_VINETIC_DEV_GPIO_CFG	287
7.1.4.2	FIO_VINETIC_DEV_GPIO_SET	288
7.1.4.3	FIO_VINETIC_GPIO_RESERVE	288
7.1.4.4	FIO_VINETIC_GPIO_CONFIG	289
7.1.4.5	FIO_VINETIC_GPIO_SET	289
7.1.4.6	FIO_VINETIC_GPIO_GET	290
7.1.4.7	FIO_VINETIC_GPIO_RELEASE	290
7.1.5	Line Testing Interface	292
7.1.5.1	FIO_VINETIC_GR909_RESULT	293
7.1.5.2	FIO_VINETIC_GR909_START	294

7.1.5.3	FIO_VINETIC_GR909_STOP	294
7.1.5.4	FIO_VINETIC_OFFSET_CALIBRATION	295
7.1.5.5	FIO_VINETIC_LT_CURRENT	295
7.1.5.6	FIO_VINETIC_LT_VOLTAGE	296
7.1.5.7	FIO_VINETIC_LT_RESISTANCE	296
7.1.5.8	FIO_VINETIC_LT_CAPACITANCE	297
7.1.5.9	FIO_VINETIC_LT_IMPEDANCE	298
7.1.5.10	FIO_VINETIC_LT_AC_MEASUREMENTS	298
7.1.5.11	FIO_VINETIC_LT_AC_DIRECT	299
7.1.5.12	FIO_VINETIC_LT_RESISTANCE_RC	299
7.1.5.13	FIO_VINETIC_LT_CONFIG	300
7.1.6	Polling Interface	301
7.1.6.1	FIO_VINETIC_POLL_EVT	301
7.1.6.2	FIO_VINETIC_POLL_DEV_ADD	302
7.2	Function Reference	302
7.2.1	VINETIC_GpioConfig	303
7.2.2	VINETIC_GpioGet	303
7.2.3	VINETIC_GpioIntMask	304
7.2.4	VINETIC_GpioRelease	304
7.2.5	VINETIC_GpioReserve	305
7.2.6	VINETIC_GpioSet	305
7.2.7	VINETIC_IrqPollConf	306
7.2.8	VINETIC_OpenKernel	306
7.2.9	VINETIC_Poll_Down	307
7.2.10	VINETIC_Poll_DownIntlv	307
7.2.11	VINETIC_Poll_Events	308
7.2.12	VINETIC_Poll_Up	308
7.2.13	VINETIC_Poll_UpIntlv	309
7.2.14	VINETIC_ReleaseKernel	309
7.3	Type Definition Reference	311
7.3.1	IO-control Reference	311
7.3.2	Constant Reference	312
7.3.3	Structure Reference	313
7.3.3.1	VINETIC_BasicDeviceInit_t	314
7.3.3.2	VINETIC_GPIO_CONFIG	315
7.3.3.3	VINETIC_IO_CRAM	315
7.3.3.4	VINETIC_IO_CRAM_REG_CFG	316
7.3.3.5	VINETIC_IO_DRVCTRL	317
7.3.3.6	VINETIC_IO_EMBDCTRL	317
7.3.3.7	VINETIC_IO_GPIO_CONTROL	318
7.3.3.8	VINETIC_IO_GR909	318
7.3.3.9	VINETIC_IO_INIT	319
7.3.3.10	VINETIC_IO_LT_AC_DIRECT	321
7.3.3.11	VINETIC_IO_LT_AC_LM_FILTER	321
7.3.3.12	VINETIC_IO_LT_AC_MEASUREMENTS	322
7.3.3.13	VINETIC_IO_LT_AC_TG	323
7.3.3.14	VINETIC_IO_LT_CAPACITANCE	323
7.3.3.15	VINETIC_IO_LT_CONFIG	324
7.3.3.16	VINETIC_IO_LT_LL_CONFIG	324
7.3.3.17	VINETIC_IO_LT_CURRENT	325
7.3.3.18	VINETIC_IO_LT_IMPEDANCE	325

7.3.3.19	VINETIC_IO_LT_RESISTANCE	326
7.3.3.20	VINETIC_IO_LT_RESISTANCE_RC	327
7.3.3.21	VINETIC_IO_LT_RESULT	327
7.3.3.22	VINETIC_IO_LT_RESULT_CONFIG	328
7.3.3.23	VINETIC_IO_LT_VOLTAGE	328
7.3.3.24	VINETIC_IO_MB_CMD	329
7.3.3.25	VINETIC_IO_OFFSET_CALIBRATION	329
7.3.3.26	VINETIC_IO_OFFSET_VALUE	330
7.3.3.27	VINETIC_IO_READ_SC	330
7.3.3.28	VINETIC_IO_TAPI_LT_CONFIG	331
7.3.3.29	VINETIC_IO_USR	331
7.3.3.30	VINETIC_IO_VERSION	332
7.3.3.31	VINETIC_IO_WRITE_SC	333
7.3.4	Enumerator Reference	333
7.3.4.1	DEV_ERR	334
7.3.4.2	GR909_RESULT_PARAM	338
7.3.4.3	GR909_TEST_RESULT	340
7.3.4.4	GR909_TESTS	340
7.3.4.5	VIN_ACCESS	341
7.3.4.6	VINETIC_GPIO_MODE	342
7.3.4.7	VINETIC_IO_CHIP_MAJOR_REVISION	343
7.3.4.8	VINETIC_IO_CHIP_REVISION	344
7.3.4.9	VINETIC_IO_CHIP_TYPE	344
7.3.4.10	VINETIC_IO_CRAM_FORMAT	345
7.3.4.11	VINETIC_IO_LT_AC_LM_FILTER_TYPE	345
7.3.4.12	VINETIC_IO_LT_AC_LM_SELECT	346
7.3.4.13	VINETIC_IO_LT_AC_MODE	346
7.3.4.14	VINETIC_IO_LT_AC_TG_STATE	347
7.3.4.15	VINETIC_IO_LT_AC_TH_FILTER	347
7.3.4.16	VINETIC_IO_LT_BP_QUAL	348
7.3.4.17	VINETIC_IO_LT_CAPACITANCE_MODE	348
7.3.4.18	VINETIC_IO_LT_CONFIG_MODE	349
7.3.4.19	VINETIC_IO_LT_CURRENT_MODE	349
7.3.4.20	VINETIC_IO_LT_CURRENT_SUBMODE	350
7.3.4.21	VINETIC_IO_LT_IMPEDANCE_MODE	350
7.3.4.22	VINETIC_IO_LT_IMPEDANCE_SUBMODE	351
7.3.4.23	VINETIC_IO_LT_RESISTANCE_MODE	351
7.3.4.24	VINETIC_IO_LT_RS	352
7.3.4.25	VINETIC_IO_LT_TG	352
7.3.4.26	VINETIC_IO_LT_VOLTAGE_MODE	352
7.3.4.27	VINETIC_IO_LT_VOLTAGE_SUBMODE	353
7.3.4.28	VINETIC_IO_OFFSET_CALIBRATION_MODE	353
8	Line Testing Interfaces	355
8.1	Function Reference	355
8.1.1	Ifxphone_LT_AC_Gain_Tracking	356
8.1.2	Ifxphone_LT_AC_Idle_Noise	357
8.1.3	Ifxphone_LT_AC_Level	358
8.1.4	Ifxphone_LT_AC_Level_Measurement	359
8.1.5	Ifxphone_LT_AC_SNR	361
8.1.6	Ifxphone_LT_AC_Transhybrid	362
8.1.7	Ifxphone_LT_Capacitance	363

8.1.8	Ifxphone_LT_Charge	364
8.1.9	Ifxphone_LT_Current	365
8.1.10	Ifxphone_LT_Discharge	366
8.1.11	Ifxphone_LT_GR909_Config	367
8.1.12	Ifxphone_LT_GR909_GetResults	367
8.1.13	Ifxphone_LT_GR909_Start	368
8.1.14	Ifxphone_LT_Impedance	368
8.1.15	Ifxphone_LT_Network	370
8.1.16	Ifxphone_LT_Network_Ground	371
8.1.17	Ifxphone_LT_Network_Tip_Ring	373
8.1.18	Ifxphone_LT_Resistance	374
8.1.19	Ifxphone_LT_Resistance_RC	376
8.1.20	Ifxphone_LT_Voltage	377
8.1.21	TAPI_LT_Configure	379
8.2	Type Definition Reference	380
8.2.1	Structure Reference	380
8.2.1.1	IFX_LT_GR909_CFG_t	380
8.2.1.2	IFX_LT_GR909_FEMF_t	381
8.2.1.3	IFX_LT_GR909_HPT_t	381
8.2.1.4	IFX_LT_GR909_RESULT_t	382
8.2.1.5	IFX_LT_GR909_RFT_t	382
8.2.1.6	IFX_LT_GR909_RIT_t	383
8.2.1.7	IFX_LT_GR909_ROH_t	383
8.2.1.8	IFXPHONE_LTEST	384
8.2.1.9	IFXPHONE_LTEST_CHARGE_t	384
8.2.1.10	IFXPHONE_LTEST_DISCHARGE_t	385
8.2.1.11	IFXPHONE_LTEST_NET_GND	386
8.2.1.12	IFXPHONE_LTEST_NET	386
8.2.1.13	IFXPHONE_LTEST_NET_TIP_RING	388
8.2.2	Enumerator Reference	389
8.2.2.1	IFX_LT_GR909_MASK_t	389
8.2.2.2	LTEST_ERROR	390
8.2.2.3	LTEST_MODE	391
8.2.2.4	LTEST_SIGNAL_TYPE	392
8.2.2.5	LTEST_SUBMODE	392
8.2.2.6	LTEST_VALUES	394

List of Figures

Figure 1	Example of File Descriptors and Resources (1)	25
Figure 2	Example of File Descriptors and Resources (2)	25
Figure 3	Module Connection for VoIP TAPI Initialization (default)	27
Figure 4	Module Connection for PCM DSP TAPI Initialization	27
Figure 5	Module Connection for PCM Analog TAPI Initialization	28
Figure 6	Sequence of Line Feeding Modes	30
Figure 7	Simple Internal Call.	39
Figure 8	Two Independent VoIP Calls	40
Figure 9	External and Internal Conference	42
Figure 10	External Conference.	44
Figure 11	Connection of a DuSLIC via PCM Interface	47
Figure 12	Simple and composed tones	53
Figure 13	Overview of the Configuration structure	63
Figure 14	Schematic Showing the Circuit between VINETIC and Tip/Ring	79
Figure 15	Polling Mode Packet and Event handling	83

List of Tables

Table 1	VINETIC® Files	19
Table 2	Configuration Options for Linux and VxWorks	20
Table 3	Device Nodes for a System including two VINETIC® devices	23
Table 4	Supported TAPI Initialization Modes	26
Table 5	Tone Table - Predefined Tones	56
Table 6	Caller ID Types	60
Table 7	Caller ID Generation - On-hook CID (type 1)	61
Table 8	Caller ID Generation - On-hook MWI	61
Table 9	Caller ID Generation - Off-hook CID (type 2) and off-hook MWI	61
Table 10	Caller ID Generation - Abbreviations	61
Table 11	Relevant Timing Applicable to the Different Standards	63
Table 12	Caller ID/MWI Type Selection	67
Table 13	TAPI Interface Overview	87
Table 14	IO-control Overview of CID Features	88
Table 15	Structure Overview of CID Features	88
Table 16	Union Overview of CID Features	88
Table 17	Enumerator Overview of CID Features	89
Table 18	IO-control Overview of Connection Control Service	96
Table 19	Structure Overview of Connection Control Service	97
Table 20	Enumerator Overview of Connection Control Service	97
Table 21	IO-control Overview of Dial Service	121
Table 22	IO-control Overview of Fax T.38 Service	129
Table 23	Structure Overview of Fax T.38 Service	129
Table 24	Enumerator Overview of Fax T.38 Service	129
Table 25	IO-control Overview of Initialization Service	134
Table 26	Structure Overview of Initialization Service	134
Table 27	Enumerator Overview of Initialization Service	134
Table 28	IO-control Overview of Metering Service	136
Table 29	Structure Overview of Metering Service	136
Table 30	IO-control Overview of Miscellaneous Services	139
Table 31	Structure Overview of Miscellaneous Services	139
Table 32	Union Overview of Miscellaneous Services	139
Table 33	Enumerator Overview of Miscellaneous Service	139
Table 34	IO-control Overview of Operation Control Service	149
Table 35	Structure Overview of Operation Control Service	150
Table 36	IO-control Overview of PCM Service	159
Table 37	IO-control Overview of Ringing Service	163
Table 38	Structure Overview of Ringing Service	163
Table 39	IO-control Overview of Signal Detection Service	169
Table 40	Structure Overview of Signal Detection Service	169
Table 41	Enumerator Overview of Signal Detection Service	169
Table 42	IO-control Overview of Tone Control Service	175
Table 43	Constant Overview of Tone Control Service	175
Table 44	Structure Overview of Tone Control Service	175
Table 45	Union Overview of Tone Control Service	175
Table 46	Enumerator Overview of Tone Control Service	175
Table 47	IO-control Overview of TAPI Interfaces	184
Table 48	Constant Reference for TAPI Interfaces	187
Table 49	Structure Overview of TAPI Interfaces	188

Table 50	Union Overview of TAPI Interfaces	229
Table 51	Enumerator Overview of TAPI Interfaces	231
Table 52	Mapping to old names of TAPI ioctl Interfaces	265
Table 53	Mapping to old names of TAPI Constants	268
Table 54	Mapping to old names of TAPI Structures	269
Table 55	Mapping to old names of TAPI Unions	270
Table 56	Mapping to old names of TAPI Enumerations	270
Table 57	Device Driver Interface Overview	273
Table 58	IO-control Overview of Basic Interface	274
Table 59	Constant Overview of Basic Interface	274
Table 60	Structure Overview of Basic Interface	274
Table 61	Enumerator Overview of Basic Interface	275
Table 62	IO-control Overview of Driver Initialization Interface	284
Table 63	Structure Reference of Driver Initialization Interface	284
Table 64	Enumerator Overview of Driver Initialization Interface	284
Table 65	Enumerator Overview of Driver Kernel Interface	286
Table 66	Function Overview of Driver Kernel Interface	286
Table 67	IO-control Overview of GPIO Interface	287
Table 68	Structure Overview of GPIO Interface	287
Table 69	IO-control Overview of Line Testing Interface	292
Table 70	Structure Overview of Line Testing Interface	292
Table 71	Enumerator Overview of Line Testing Interface	293
Table 72	IO-control Overview of Polling Interface	301
Table 73	Constant Overview of Polling Interface	301
Table 74	Function Overview of Polling Interface	301
Table 75	Function Overview of Non TAPI Interfaces	302
Table 76	IO-control Overview of Non TAPI Interfaces	311
Table 77	Constant Overview of Non TAPI Interfaces	312
Table 78	Constant Reference for Non TAPI Interfaces	312
Table 79	Structure Overview of Device Driver Interfaces	313
Table 80	Enumerator Overview of Non TAPI Interfaces	333
Table 81	Function Overview of Line Testing Interfaces	355
Table 82	Structure Overview of Line Testing Interfaces	380
Table 83	Enumerator Overview of Line Testing Interfaces	389

Preface

This document describes the VINETIC® device driver structure and usage. If not otherwise specified, the description in this document applies to the following devices:

VINETIC®-4VIP (PEB 3324), Version 1.4

VINETIC®-4S (PEF 3304), Version 2.1

VINETIC®-4C (PEF 3394), Version 2.2

VINETIC®-4M (PEF 3314), Version 2.2

This document replaces the following documents:

VINETIC® Prel. User's Manual Software Description Driver Rev. 4.0, 2004-12-08; VINETIC® (PEB/PEF 33xy) Version 1.4 and 2.1/2.2 Application Note Telephony API (TAPI) V2.3 Rev. 16, 2005-04-13

To simplify matters, the following synonyms are used:

VINETIC®:

Synonym used for the system consisting of codec versions VINETIC®-4VIP (PEB 3324), Version 1.4, VINETIC®-4S (PEF 3304), Version 2.1, VINETIC®-4M (PEF 3314), Version 2.2 and VINETIC®-4C (PEF 3394), Version 2.2.

SLIC:

Synonym used for the SLIC-E (PEB 4265) Version 1.2/Version 2.1, SLIC-S (PEB 4264) Version 1.2, SLIC-P (PEB 4266) Version 1.2, SLIC-LCP (PEB 4262) Version 1.1 and SLIC-DC (PEF 4268) Version 1.1/Version 1.2.

Attention: The TSLIC-S (PEB 4364) and TSLIC-E (PEB 4365) chips are dual channel versions of the SLIC-S (PEB 4264) and SLIC-E (PEF 4265) with identical technical specifications for each channel. Therefore whenever SLIC-S or SLIC-E are mentioned in the specification, also TSLIC-S or TSLIC-E can be deployed.

Organization of this Document

This document is divided into 8 chapters. It is organized as follows:

- **Chapter 1, Development Environment Setup**
Overview of the VINETIC® device driver package. 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 remaining TAPI functionality.
- **Chapter 4, Line Testing Services**
Description of the line testing services.
- **Chapter 5, Device Driver Services**
Description of important device driver services-
- **Chapter 6, TAPI Interfaces**
Reference for the VINETIC® TAPI interfaces.
- **Chapter 7, Device Driver Interfaces**
Reference for the VINETIC® Device Driver interfaces.
- **Chapter 8, Line Testing Interfaces**
Reference for the VINETIC® Line Testing interfaces.

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.

IMPORTANT NOTE

This device driver version provides a completely redesigned interface. The ioctls have been rearranged and grouped hierarchically. For more details please refer to [Chapter 1.5](#).

1 Development Environment Setup

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

After an overview of the contents of the device driver distribution package, the details of the device driver interface to the system are discussed. Finally, this chapter shows how to compile the VINETIC® device driver for different configurations.

1.1 General Introduction to the Device Driver

Since the introduction of the VINETIC® device driver 1.0, the system- and board-specific code has been isolated. It is no longer required to modify the VINETIC® device driver to support a particular system/board. The integration of the VINETIC® device driver into the target system is done through the so-called “board driver”, keeping the VINETIC® device driver board-independent. The introduction of the board driver allows a smooth integration of new releases of the VINETIC® device driver while keeping the board- and OS-specific code untouched. For details, please refer to [13].

1.2 Files

This chapter lists the files of the VINETIC® device driver you will have to include in your application, and describes the contents.

Table 1 VINETIC® Files

Filename	Description
vinetic_io.h	VINETIC®-specific ioctl interface
drv_tapi_io.h	TAPI ioctl interface

1.3 Device Nodes

The system uses device nodes to access the VINETIC® from the application. Different device nodes are used to access either the device or a single channel.

1.3.1 Linux

If the Linux kernel includes support for the device file system, the VINETIC® device driver will create the required device nodes automatically on insmod. Otherwise, the device nodes must be created manually using the `mknod` command. For example, with VINETIC®:

```
mknod /dev/vin10 c 230 10
mknod /dev/vin11 c 230 11
mknod /dev/vin12 c 230 12
mknod /dev/vin13 c 230 13
mknod /dev/vin14 c 230 14
```

In this example, the “major” number 230 was chosen for the VINETIC® device, while the “minor” number is used to identify the different channels.

Attention: Currently VINETIC® device driver uses the same device nodes and major number as the VINETIC® family device driver. A later release will support the mixed use of these two.

1.3.2 VxWorks

VxWorks® has no automatic installation procedure for drivers. Although it is possible to load a module dynamically (if required).

To integrate the driver into a VxWorks system follow the following steps:

- Insert the project “drv_vinetic.wpj” file to the workspace

- Call the function `Vinetic_DeviceDriverInit()` somewhere from the BSP or the application code
- Link the driver archive `drv_vinetic.a` to the image using the image build tab - Macro - `EXTRA_MODULES` and add the path and file name

1.4 Compilation

This chapter describes how to compile the VINETIC® device driver for Linux and VxWorks. For Linux, the GNU toolchain (autoconf, automake) is used.

If the toolchain is already in place, the path of the cross-compiler should be included in the `PATH`. 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 2](#) and then
- Execute `make` or `make install`.

For VxWorks, the Tornado project files are required which are supplied in the distribution package.

To retrieve the device driver sources from the distribution package and to obtain the execution rights and directory structure, use the following command or equivalent on your system. It will extract all sources to a subdirectory with the name of the distribution package.

```
tar xvzf drv_vinetic-1.x.x.tar.gz
```

Table 2 Configuration Options for Linux and VxWorks

Configuration options	Linux	VxWork	Supported	Usage
VxWorks support	N/A	-DVXWORKS	All	Optional
Debug features	--enable-debug	-DDEBUG	All	Optional
TAPI interface support	Implicitly selected	-DTAPI	All	Required
TAPI Linetesting	--enable-lt	-DTAPI_LT	All	Optional
TAPI GR909	--enable-lt	-DTAPI_GR909	All (except VINETIC®-4M, VINETIC®-4C, VINETIC®-4S Version 2.1 and Version 2.2)	Optional
TAPI Voice services	--enable-voice	-DTAPI_VOICE	All (except VINETIC®-4S)	Required
Trace features	--enable-trace	-DENABLE_TRACE	All	Optional
Register access trace features	--enable-trace	-DRUNTIME_TRACE	All	Optional
SPI host access support	--enable-spi	-DVIN_SPI	All	Optional
Polling mode support	N/A	-DVIN_POLL	All	Optional
Support for VINETIC® V1.x devices	--enable-v1	-DVINETIC_V14_SUPPORT		Mandatory (enable-v1 OR enable-v2 OR both)

Table 2 Configuration Options for Linux and VxWorks (cont'd)

Configuration options	Linux	VxWork	Supported	Usage
Support for VINETIC® V2.x devices	--enable-v2	-DVINETIC_V21_SUPPORT		Mandatory (enable-v1 OR enable-v1 OR both)
Use default VINETIC® firmware	--enable-incl-bin	- DVIN_DEFAULT_FW	All (except VINETIC®-4S)	Optional
Fax support	--enable-fax	-DTAPI_FAX_T38	All (except VINETIC®-4S)	Optional
DTMF services	--enable-dtmf	-DTAPI_DTMF	All (except VINETIC®-4S)	Optional
CID support	--enable-cid	-DTAPI_CID	All (except VINETIC®-4S)	Optional
Maximum number of VINETIC® devices	--with-max-devices=<value>	-DVINETIC_MAX_DEVICES=<value>	All	
Enable /proc filesystem	--enable-proc	n/a	All	Optional
Name of development platform	--build=<value>	n/a	All	Optional. For example: --build=i686-pc-linux
Name of target platform: prefix of the cross compiler name	--host=<value>	n/a	All	Required For example --host=powerpc-linux
Path to kernel header files	--enable-kernelincl=<value>	n/a	All	Required For example: --enable-kernelincl=/home/\${USER}/linux/include
Enable compiler warnings	--enable-warnings	n/a	All	Optional
Path to install directory	--prefix=<value>	n/a	All	Optional

1.5 Interface Names and Backwards Compatibility

The device driver TAPI interfaces have been renamed¹⁾ with a hierarchical organization of the interfaces.

Although we encourage our customers to use the new interface, backward compatibility is ensured: in order to use the old interface names, compile flag -DTAPI_OLD_IO must be used, or a corresponding define before the include of drv_tapi_io.h must be used. A complete list of old and new names is listed in [Chapter 6.3](#).

1) According to guidelines provided by the Network Processing Forum.

New features will be provided only in accordance to the redesigned interface. Code samples in this document use the redesigned interface.

1.6 Linetesting Support

Linetesting, including GR-909 support, is included in separate libraries to allow floating point calculations of the results, which is not possible on driver level for some operating systems. To enable linetesting and GR-909 support in the device driver, configure option `--enable-lt` or the equivalent define shall be used. See also [Table 2](#) for the compiler options.

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 usage of the driver interfaces.

The arguments described in this chapters are

- Role of the file descriptors, [Chapter 2.1](#)
- Initialization of device driver and channel, [Chapter 2.2](#)
- Set-up line modes, [Chapter 2.3](#)
- Reporting of events to the application software, [Chapter 2.5](#)
- Establishment of connections and conferences, [Chapter 2.6](#)

2.1 File Descriptors

Communication between application software and the driver is achieved through system calls performed on specific file descriptors. For the VINETIC® driver, two types of file descriptors are defined: device-specific and channel-specific.

As the name suggests, the device-specific 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.

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 VINETIC® driver can handle one or more devices in one board, several device nodes are specified to give the programmer the ability to address a specific device and its channels.

2.1.1 Device Nodes

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

The device-specific nodes are of the format “/dev/vinX0”, where X corresponds to the particular VINETIC® device. These device nodes are used for controlling the VINETIC® device as a whole. For example, in a system with two VINETIC®, /dev/vin10 addresses VINETIC® number 1 and /dev/vin20 addresses VINETIC® number 2.

The channel-specific nodes are of the format “/dev/vinXY”, addressing VINETIC® number X, channel Y. For example, in a system with two VINETIC®, /dev/vin11 addresses VINETIC® number 1/channel 1 and /dev/vin23 addresses VINETIC® number 2/channel 3.

A specific channel-file descriptor must be opened for read and write access, implementing packet upstream and packet downstream. Exceptions are not reported via the channel-file descriptors, because operating systems such as VxWorks do not support them. Exceptions are reported on the device-specific node, not per channel. The status information is requested per channel or for multiple channels (using ioctl [IFX_TAPI_CH_STATUS_GET](#)).

Table 3 Device Nodes for a System including two VINETIC® devices¹⁾

Device Node	VINETIC® Number	Addressed Resource Set	Included Resources
/dev/vin10	Device 1	Entire device 1	VINETIC® chip
/dev/vin11	Device 1	Of device 1, channel 1	ALM0, SIG0, COD0, PCM0 resources
/dev/vin12	Device 1	Of device 1, channel 2	ALM1, SIG1, COD1, PCM1 resources
/dev/vin13	Device 1	Of device 1, channel 3	SIG2, COD2, PCM2 resources
/dev/vin14	Device 1	Of device 1, channel 4	SIG3, COD3, PCM3 resources
/dev/vin20	Device 2	Entire device 2	VINETIC® chip

Table 3 Device Nodes for a System including two VINETIC® devices¹⁾ (cont'd)

Device Node	VINETIC® Number	Addressed Resource Set	Included Resources
/dev/vin21	Device 2	Of device 2, channel 1	ALM0, SIG0, COD0, PCM0 resources
/dev/vin22	Device 2	Of device 2, channel 2	ALM1, SIG1, COD1, PCM1 resources
/dev/vin23	Device 2	Of device 2, channel 3	SIG2, COD2, PCM2 resources
/dev/vin24	Device 2	Of device 2, channel 4	SIG3, COD3, PCM3 resources

1) In this example we have two VINETIC® V2.1 devices, each of them includes two analog channels (ALM) while there are four VoIP channels (alias data channel, SIG and COD).

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.

PCM and analog module (alias ALM) resources belong to phone channels while SIG and COD resources belong to data channels.

Usage of data channels and phone channels allows effective usage of the VINETIC® resources: the modular architecture allows decoupling 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 2](#)).

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 6](#), in the reference description of each TAPI command, states whether a command must be applied to a data or phone channel.

Examples in [Figure 1](#) and [Figure 2](#) show two possible ways 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 1](#), 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).

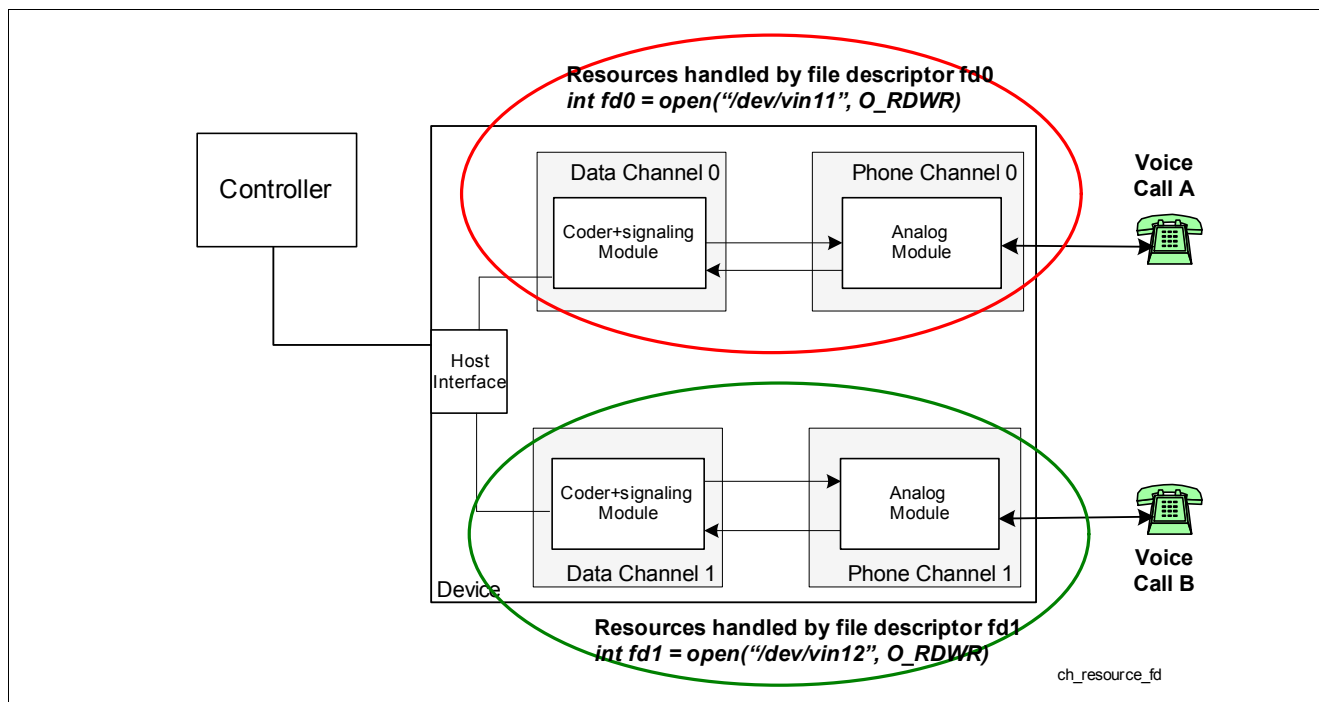


Figure 1 Example of File Descriptors and Resources (1)

In the example shown in [Figure 2](#), 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 2.6](#).

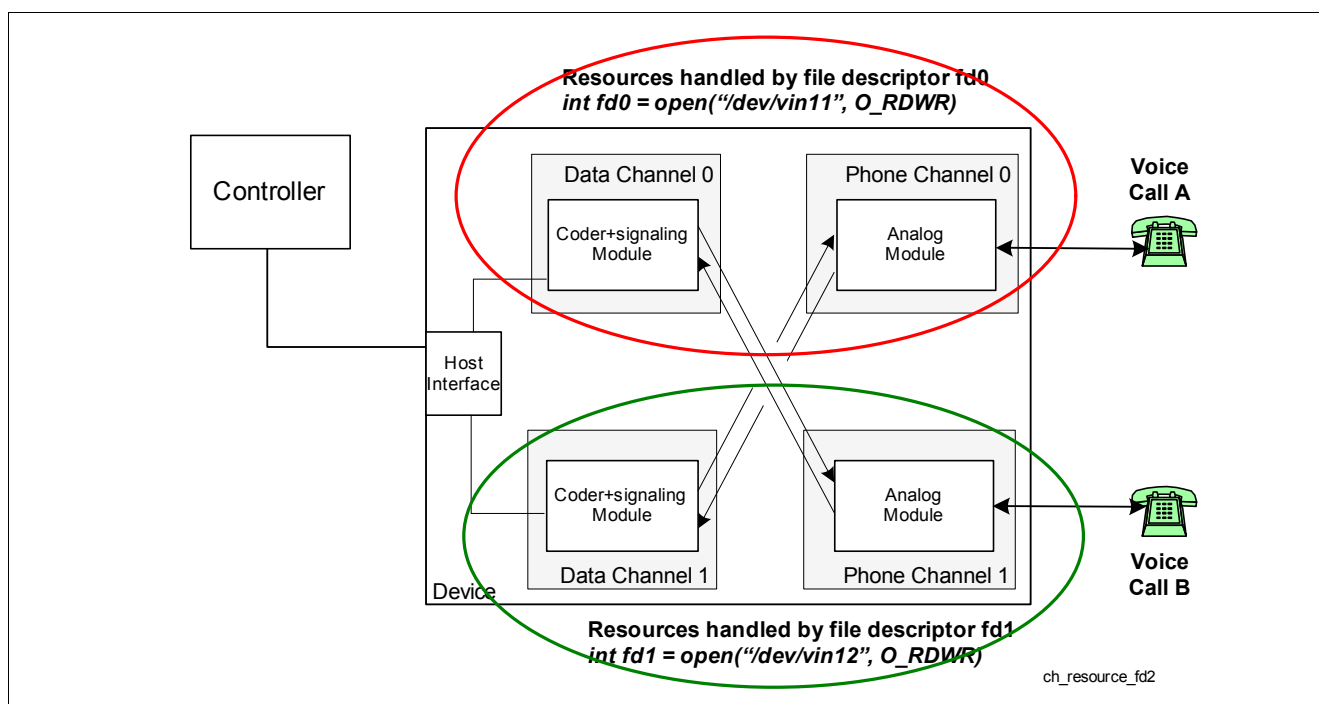


Figure 2 Example of File Descriptors and Resources (2)

2.2 Initialization

The foundation for all subsequent operations is the successful initialization of the device driver and the VINETIC®. 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 VINETIC® devices in the system. Configuration of VINETIC® base address and the interrupt line to attach its interrupt handler to, is done using driver ioctl **FIO_VINETIC_BASICDEV_INIT**; usage is shown in the code example in [Chapter 2.2.3](#).

2.2.2 TAPI Channel initialization

The channel initialization¹⁾ 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: TAPI channel type, valid value for VINETIC® is **IFX_TAPI_CH_INIT_MODE_VOICE_CODER** (**IFX_TAPI_CH_INIT_MODE_DEFAULT** has the same effect). This configuration should be done for each channel.
- pProc: a pointer to device-specific initialization details (of type **VINETIC_IO_INIT**) taking optionally as parameters the binaries for firmware and CRAM coefficients
 - Firmware download: issued only one time after device reset.
 - CRAM coefficients: these are dependent on SLIC/DAA and line type; it is possible to download a different set of CRAM coefficients per channel. This operation is defined only for channels including an analog port.

IFX_TAPI_CH_INIT service supports three different initialization modes, assuming that the mode is supported by the selected VINETIC® device. These modes are:

- **IFX_TAPI_CH_INIT_MODE_VOICE_CODER**
- **IFX_TAPI_CH_INIT_MODE_PCM_DSP**
- **IFX_TAPI_CH_INIT_MODE_PCM_PHONE**

The following chapters list the basic initialization of the modules in detail depending on the chosen initialization mode. The configuration of the initialization mode is performed in the source file `drv_vinetic_basic.c`.

The [Table 20](#) lists all possible and default modes for the different devices.

Table 4 Supported TAPI Initialization Modes

Device	IFX_TAPI_CH_INIT_MODE_VOICE_CODER	IFX_TAPI_CH_INIT_MODE_PCM_DSP	IFX_TAPI_CH_INIT_MODE_PCM_PHONE
VINETIC®-4VIP	x (default)	x	x
VINETIC®-4M	x (default)	x	x
VINETIC®-4C	-	x (default)	x
VINETIC®-4S	-	-	x (default)

2.2.2.1 VoIP Initialization (default)

The VINETIC® device is initialized to a default configuration by the TAPI service **IFX_TAPI_CH_INIT** in case of initialization mode **IFX_TAPI_CH_INIT_MODE_VOICE_CODER**. The following device configuration can be chosen and is selected with **TAPI_DEFAULT**:

- VINETIC®-4VIP Version 1.4
- VINETIC®-4M Version 2.2

¹⁾ This chapter replaces the chapter 4.4 of User's Manual Software Description Driver (Version 0.11.8) Rev. 4.0.

Figure 1 depicts the signal connection of the coder, signaling and analog line module for channel 0 after the VoIP default initialization. The signal connection is performed via the signal array.

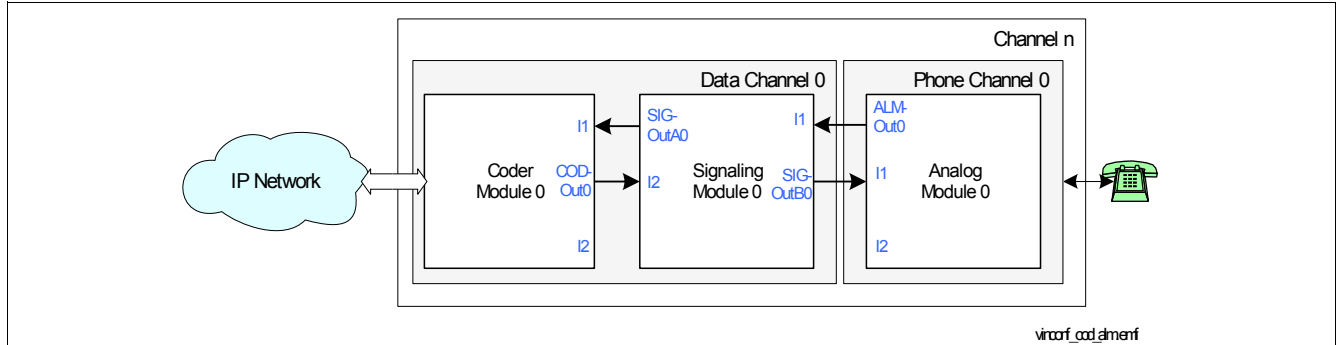


Figure 3 Module Connection for VoIP TAPI Initialization (default)

2.2.2.2 PCM with DSP Initialization

VINETIC® can be initialized by the TAPI service **IFX_TAPI_CH_INIT** with initialization mode **IFX_TAPI_CH_INIT_MODE_PCM_DSP** to a configuration supporting an analog via DSP to PCM connection. The DSP is able to detect signals (see **IFX_TAPI_SIG_DETECT_ENABLE**).

By default, the **IFX_TAPI_CH_INIT_MODE_PCM_DSP** is selected for the following devices:

- VINETIC®-4C Version 2.2

This initialization mode can also be applied for the following devices:

- VINETIC®-4VIP Version 1.4
- VINETIC®-4M Version 2.2

Figure 2 depicts the signal connection of the PCM, signaling and analog line module for channel 0 after the **IFX_TAPI_CH_INIT_MODE_PCM_DSP** mode initialization. The signal connection is performed via the signal array.

Attention: This initialization does not support dynamically module connection (see Chapter 6.4).

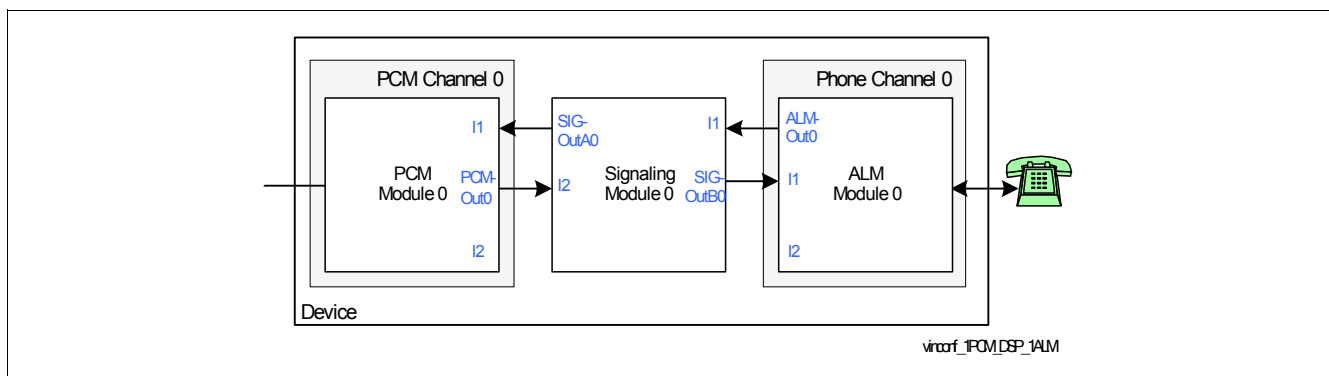


Figure 4 Module Connection for PCM DSP TAPI Initialization

2.2.2.3 PCM with Analog TAPI Initialization

The VINETIC® can be initialized with the TAPI service **IFX_TAPI_CH_INIT** and with initialization mode **IFX_TAPI_CH_INIT_MODE_PCM_PHONE**. The mode **IFX_TAPI_CH_INIT_MODE_PCM_PHONE** supports an analog to PCM connection. No signal detection, including DTMF digits, is possible with this initialization. By default, the **IFX_TAPI_CH_INIT_MODE_PCM_PHONE** is selected for the following devices

- VINETIC®-4S Version 2.2

Figure 3 depicts the signal connection of the PCM, and analog line module for channel 0 after the **IFX_TAPI_CH_INIT_MODE_PCM_PHONE** mode initialization. The signal connection is performed via the signal array.

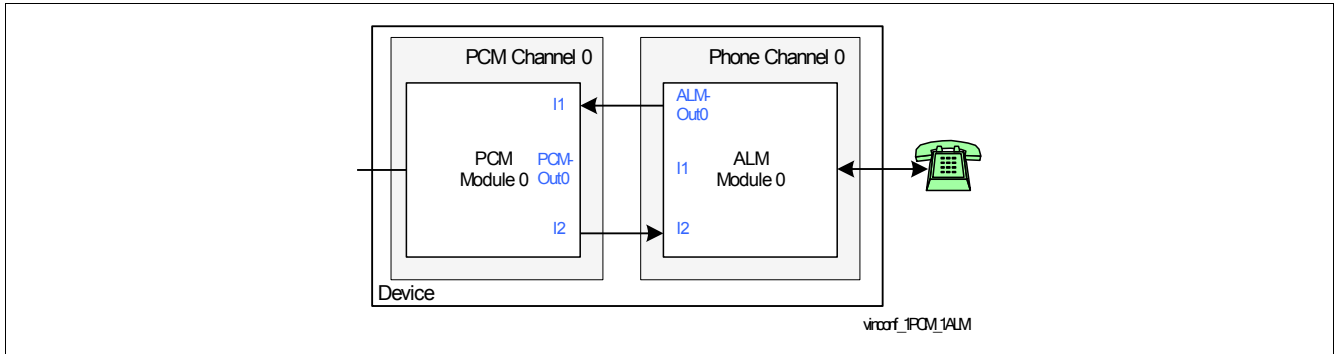


Figure 5 Module Connection for PCM Analog TAPI Initialization

2.2.3 Initialization Example

```

/* Include binaries files for firmware */
#include "edspPRAMfw_RTP_xx_xx_xx.c"
#include "edspDRAMfw_RTP_xx_xx_xx.c"
#include "vinetic_coefficients.by"

/* Drivers interfaces needed headers */
#include "ifx_types.h"
#include "vinetic_io.h"
#include "drv_tapi_io.h"

/* Constants, enums, defines */

/* Number of data channels */
static const IFX_int32_t DATA_CH_CNT = 4;

/* Number of phone channels */
static const IFX_int32_t PHONE_CH_CNT = 2;

IFX_return_t Init()
{
    VINETIC_BasicDeviceInit_t binit;
    IFX_TAPI_CH_INIT_t init;
    VINETIC_IO_INIT vinit;
    VINETIC_IO_VERSION ioCmd;

    IFX_int32_t ret, err;
    IFX_int32_t ch;
    IFX_int32_t fd[DATA_CH_CNT];
    IFX_char_t dev_nodes[20];

```

```

/* Before VINETIC system must be initialized */

/* Initialize TAPI part, chip */

/* Set the firmware pointer from the included file */
memset(&vinit, 0, sizeof(VINETIC_IO_INIT));
vinit.pPRAMfw = (IFX_uint8_t *) vinetic_fw_pram;
vinit.pDRAMfw = (IFX_uint8_t *) vinetic_fw_dram;
vinit.dram_size = sizeof(vinetic_fw_dram);
vinit.pram_size = sizeof(vinetic_fw_pram);
/* set the coefficient download pointer */
vinit.pCram = (IFX_uint8_t *) bbd_buf;
vinit.cram_size = sizeof(bbd_buf); /* in bytes */

/* Set the pointer to the VINETIC dev specific init structure */
memset(&init, 0, sizeof(IFX_TAPI_CH_INIT_t));
init.pProc = (IFX_void_t *) &vinit;
/* Init for VoIP applications */
init.nMode = IFX_TAPI_INIT_MODE_VOICE_CODER;

/* Initialize all channels of one device */
/* Note: the binaries are downloaded only once */
for (ch = 0; ch < DATA_CH_CNT; ch++)
{
    sprintf(dev_nodes, "/dev/vin1%d", ch + 1);
    fd[ch] = open(dev_nodes, O_RDWR, 0x644);
    if (0 >= fd[ch])
    {
        printf("err: open channel %d failed\n", ch);
        return IFX_ERROR;
    }
    ioctl(fd[ch], IFX_TAPI_CH_INIT, (IFX_uint32_t *) &init);
    if (PHONE_CH_CNT > ch)
    {
        /* Set linefeed to standby mode */
        ret = ioctl(fd[ch], IFX_TAPI_LINE_FEED_SET,
                    (IFX_int32_t) IFX_TAPI_LINE_FEED_STANDBY);

        /* Set line in active mode */
        ret = ioctl(fd[ch], IFX_TAPI_LINE_FEED_SET,
                    (IFX_int32_t) IFX_TAPI_LINE_FEED_ACTIVE);
    }
} /* for */

/* Initialize the version structure */
memset((IFX_void_t *) &ioCmd, 0, sizeof(VINETIC_IO_VERSION));
err = ioctl(fdcfg, FIO_VINETIC_VERS, (IFX_uint32_t*) &ioCmd);
if (err == 0)
{
    if ((0xff != ioCmd.nEdspVers) && (0xff != ioCmd.nEdspIntern))
    {

```

```

        printf(" EDSP Firmware Version: %d.%02d\n",
               (int) ioCmd.nEdspVers,
               (int) ioCmd.nEdspIntern);
    }
}
/* Close all open fds */
for (ch = 0; ch < DATA_CH_CNT; ch++)
{
    close(fd[ch]);
}
return IFX_SUCCESS;
}

```

2.3 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.

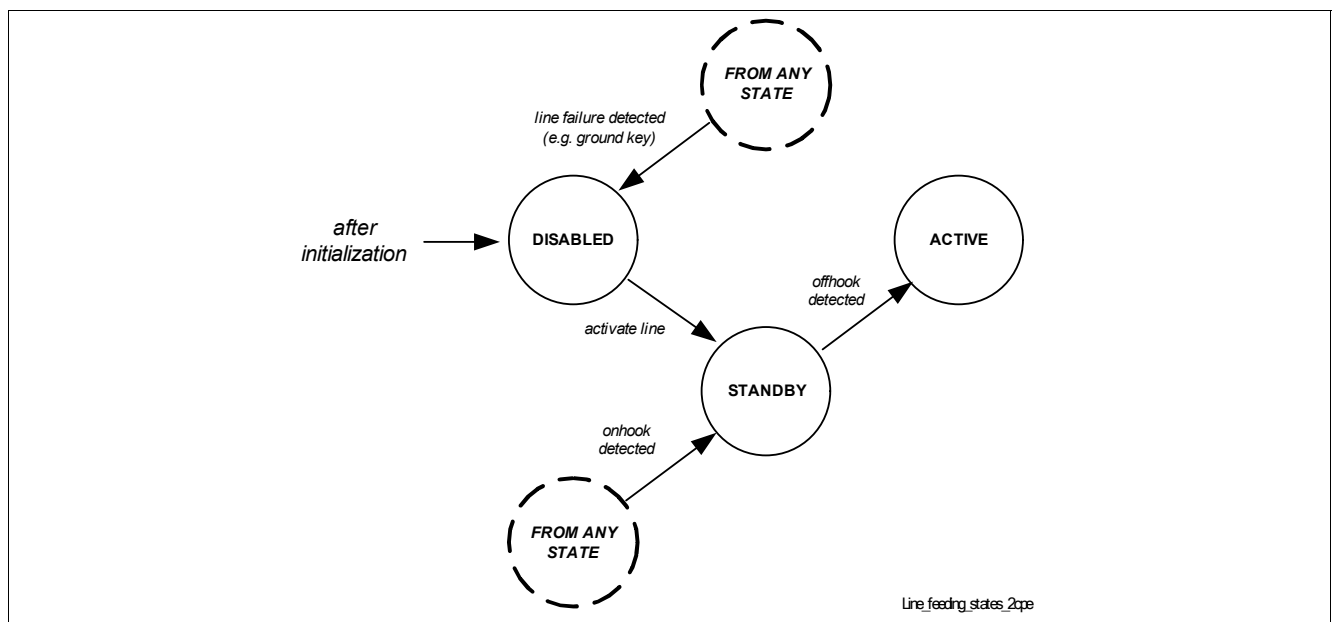


Figure 6 Sequence of Line Feeding Modes

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;
IFX_int32_t fd;

/* 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 Manual Battery Switching

In Active mode, the selected battery voltage V_{BAT} has the strongest influence on power dissipation. The most efficient way to reduce short-loop power dissipation is to use a second battery voltage of lower value (V_{BATL}) whenever line resistance is small enough.

VINETIC® provides a mechanism which can be used as indication for the battery switching. A threshold for the voltage at Tip/Ring can be set for generating an interrupt. This threshold can be user programmed via VINETICOS, coefficient "Tip Ring Voltage Threshold".

2.4.1 Battery Switching Implementation

To support the manual battery switching, the driver enables the Falling-Edge interrupt for VRTLIM bit. The interrupt is generated when the VRTLIM bit goes from 1 to 0. In parked mode, when the phone is On-hook, this bit is 1. When the phone goes Off-hook and the line is switched to normal mode the bit goes to 0 if the voltage is below the limit. This means it can be switched to low feeding mode.

By default the event is not reported to the application. If the event should be reported to the application the exception mask bit **feedLowBatt** must be set to 0. Setting this bit to 1 will deactivate the feature.

The event is reported via **IFX_TAPI_CH_STATUS_GET**. When low power feeding mode can be used the bit **IFX_TAPI_LINE_STATUS_FEEDLOWBATT** is set in the line field of **IFX_TAPI_CH_STATUS_t**.

The threshold is programmable via **FIO_VINETIC_DOWNLOAD_CRAM**.

2.4.2 Battery Switching Programming Example

The following example enables the manual battery switching for analog channel 0. It waits for a hook event and switches the line to normal feeding mode after enabling the feeding voltage monitor. After getting the **IFX_TAPI_LINE_STATUS_FEEDLOWBATT** event it switches to low feeding mode.

Example

```

TAPI_EXCEPTION param;
IFX_TAPI_CH_STATUS_t status[1];
int width, fdcfg, fd[1];
fd_set rfd;

param.Status = 0;
FD_ZERO (&rfd);
/* open device file descriptor */
fdcfg = open (/dev/vin10, O_RDWR);

```

```

FD_SET (fdcfg, &rfd);
width = fdcfg;
/* open channel file descriptor for channel 0 */
fd[0] = open (/dev/vin11, O_RDWR);
FD_SET (fd[0], &rfd);
if (width < fd[0])
    width = fd[0];
ioctl (fd, IFX_TAPI_CH_INIT, 0);
/* feedLowBatt is made after init */
/* set the line mode to on hook */
ioctl (fd, IFXPHONE_SET_LINEFEED, IFX_TAPI_LINE_FEED_STANDBY);

/* now wait for events or data */
select(width + 1, &rfd, NULL, NULL, NULL);
/* select woke up from exception on fdcfg */
if (FD_ISSET(fdcfg, &rfd))
{
    /* exception occurred */
    status[0].channels = 1;
    /* get the status of the channel */
    ioctl(fdcfg, IFXPHONE_PHONE_STATUS, status);
    if (status[0].hook & TAPI_HOOKSTATUS_OFFHOOK)
    {
        /* hook arrived. Now unmask the battery monitor */
        param.Bits.feedLowBatt = 0;
        ioctl (fd, IFXPHONE_MASK_EXCEPTION, param);
        ioctl (fd, IFXPHONE_SET_LINEFEED, IFX_TAPI_LINE_FEED_ACTIVE);
        /* when switching the line a feedLowBatt event occurs if the low feed
           voltage is sufficient */
    }
}
FD_SET (fd[0], &rfd);
select(width + 1, &rfd, NULL, NULL, NULL);
/* select woke up from exception on fdcfg */
if (FD_ISSET(fdcfg, &rfd))
{
    /* exception occurred */
    status[0].channels = 1;
    /* get the status of the channel */
    ioctl(fdcfg, IFXPHONE_PHONE_STATUS, status);
    if (status[0].line & TAPI_LINESTATUS_FEEDLOWBATT)
    {
        ioctl (fd, IFXPHONE_SET_LINEFEED, TAPI_LINEFEED_NORMAL_LOW);
    }
}
/* Now mask the battery monitor to avoid multiple events when pulse dialing happens */
param.Bits.feedLowBatt = 1;
ioctl (fd, IFXPHONE_MASK_EXCEPTION, param);
...
/* after the call switch back to parked */
ioctl (fd, IFXPHONE_SET_LINEFEED, TAPI_LINEFEED_PARKED);

```

2.5 Event Detection

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 specific file descriptor per device. The application can sleep on the file descriptor with the system call “select” provided by many operating systems (Linux, VxWorks,...). The status information is requested per channel or for multiple channels with **IFX_TAPI_CH_STATUS_GET**.

Example - Detect Digit is an example of digit detection (DTMF or pulse), **Example - Event Reporting** a select waiting on one fd for exceptions and on two others for data. In one call, the status information for both channels is queried from the driver.

The following chapters give more details on events detection.

- **Chapter 2.5.1**, hook-state detection.
- **Chapter 2.5.2**, DTMF/pulse digit detection.
- **Chapter 2.5.3**, masking events.

Example - Detect Digit

```
/* This example shows the call of the status information of one channel */
IFX_TAPI_CH_STATUS_t status[1];
IFX_int32_t fd;

/* Get the status only for channel 0 ("/dev/vin11") */
while (1)
{
    /* Only query this channel */
    status[0].channels = 0;
    ioctl(fd, IFX_TAPI_CH_STATUS_GET, status);
    /* Check whether a digit has been detected */
    if ((status[0].dialing & IFX_TAPI_DIALING_STATUS_DTMF) ||
        (status[0].dialing & IFX_TAPI_DIALING_STATUS_PULSE))
    {
        printf("Digit pressed\n");
    }
}
```

Example - Event Reporting

```
IFX_TAPI_CH_STATUS_t status[4];
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], &rfd);
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], &rfd);
if (width < fd[1])
{
    width = fd[1];
}

/* Now wait for events and data */
ret = select(width + 1, &rfd, IFX_NULL, IFX_NULL, IFX_NULL);
/* Select woke up from exception on fd_dev or data on fd[x] */
if (FD_ISSET(fd_dev, &rfd))
{
    /* an exception occurred, get status of two channels */
    status[0].channels = 2;
    ioctl(fd_dev, IFX_TAPI_CH_STATUS_GET, (IFX_int32_t) &status);
    for (i = 0; i < 2; i++)
    {
        if (status[i].hook || status[i].dialing ||
            status[i].line || status[i].digit || status[i].signal)
        {
            printf("Exception\n");
        }
    }
}

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

```

2.5.1 Hook State

The line hook state can be checked using command **IFX_TAPI_LINE_HOOK_STATUS_GET** on the file descriptor associated with the phone resource.

A hook change generates an event, reported by **IFX_TAPI_CH_STATUS_GET** (see also [Chapter 2.5](#)).

Example - Hook Status

```

IFX_int32_t hookStatus = 0;

ioctl(fd, IFX_TAPI_LINE_HOOK_STATUS_GET, (IFX_int32_t) &hookStatus);

/* Now variable hookStatus contains hook status */

```

```
switch (hookStatus)
{
    case 0:
        /* Line is on hook */
        printf("ON HOOK\n");
        break;
    case 1:
        /* Line is on hook */
        printf("OFF HOOK\n");
        break;
    default:
        /* Undefined line hook status */
        break;
}
```

2.5.2 DTMF and Pulse Digits Detection

Detection of DTMF and/or pulse digits from a local subscriber are also reported through [IFX_TAPI_CH_STATUS_GET](#). This interface tells whether a pulse or a DTMF digit was detected, and with the same command the digit information can also be obtained.

An alternative way of detecting (for example DTMF) digits is by polling with command [IFX_TAPI_TONE_DTMF_READY_GET](#). This interface reports whether a DTMF digit has been detected. If a DTMF digit has been detected, the application software can read the digit with [IFX_TAPI_TONE_DTMF_GET](#). [IFX_TAPI_TONE_DTMF_ASCII_GET](#) can be used to read the digit in ASCII format. A similar approach can be used for detection of pulse digits through interfaces [IFX_TAPI_PULSE_READY](#), [IFX_TAPI_PULSE_GET](#) and [IFX_TAPI_PULSE_ASCII_GET](#).

Example - Detect Pulse Digit alternative

```
IFX_int32_t bReady, digit, fd;

while (1)
{
    /* Check the pulse dialing status */
    ioctl(fd, IFX\_TAPI\_PULSE\_READY, &bReady);
    if (1 == bReady)
    {
        /* Digit arrived */
        ioctl(fd, IFX\_TAPI\_PULSE\_GET, &digit);
        printf("Pulse digit %d\n", digit);
    }
    else
    {
        /* No digit in the buffer */
    }
}
```

Example - DTMF Digit alternative

```
IFX_int32_t bReady, digit, fd;
```

```
while (1)
{
    /* Check the DTMF dialing status */
    ioctl(fd, IFX_TAPI_TONE_DTMF_READY_GET, &bReady);
    if (1 == bReady)
    {
        /* Digit arrived */
        ioctl(fd, IFX_TAPI_TONE_DTMF_GET, &digit);
        printf("DTMF digit %d\n", digit);
    }
    else
    {
        /* No digit in the buffer */
    }
}
```

2.5.3 Masking Events

Interface **IFX_TAPI_EXCEPTION_MASK** is used to mask TAPI events which should not be reported to the application software. Parameter can be a **IFX_uint32_t** to be used as bitmask.

It is recommended that union **IFX_TAPI_EXCEPTION_t** is used as a parameter; the programming sequence should be as follows:

- Event is masked by setting a 1 to the bit position addressed by bit (defined in structure **IFX_TAPI_EXCEPTION_BITS_t**)
- Execute **IFX_TAPI_EXCEPTION_MASK** command using Status field.

Example - Masking Events

```
/* Mask reporting of exceptions */
IFX_TAPI_EXCEPTION_t exceptionMask;
IFX_int32_t fd;

memset(&exceptionMask, 0, sizeof(exceptionMask));
/* Mask dtmf ready and hook state events */
exceptionMask.Bits.dtmf_ready = 0;
exceptionMask.Bits.hookstate = 0;

/* Now mask the selected exceptions */
ioctl(fd, IFX_TAPI_EXCEPTION_MASK, exceptionMask.Status);
```

2.5.4 Configure Hook Validation Timing

Using ioctl **IFX_TAPI_LINE_HOOK_VT_SET** it is possible to configure detection timing for the parameters

- Onhook: minimum time
- Offhook: 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)

To be noted that for some parameters only minimum time can be configured.

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.nMaxTime = 200;
ioctl(fd, IFX_TAPI_LINE_HOOK_VT_SET, (IFX_int32_t) &param);

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

2.6 Establishing Connections

Several connection types are supported; any phone connected to an ALM/PCM port can be connected to another ALM/PCM port or to a VoIP party. Three-party conferencing is also possible considering any combination of ports. This chapter introduces the API required for linking the VINETIC® ports, and describes how to establish different type of calls and three-party conferences.

The APIs required for linking the available ports are

- **IFX_TAPI_MAP_PHONE_ADD** and **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. These interfaces are relevant for scenarios where a telephone is directly connected¹⁾ to the VINETIC®.
- **IFX_TAPI_MAP_DATA_ADD** and **IFX_TAPI_MAP_DATA_REMOVE**, to be used for adding/removing a link between a data resource (it means coder and signaling) to a phone channel (ALM or PCM). These interfaces are relevant for establishing a VoIP call with a phone attached to the analog port or to the PCM port. Furthermore, linking a data resource to given port (ALM or PCM) makes it possible to detect/generate tones and signals. For typical applications, each phone channel used should be linked to one dedicated data channel.
- **IFX_TAPI_MAP_PCM_ADD** and **IFX_TAPI_MAP_PCM_REMOVE**, to be used for adding/removing a link between a PCM port to other PCM port or an analog port. These interfaces are relevant for scenarios where a voice connection is going through the VINETIC® PCM interface.

2.6.1 Internal Call

Internal calls can be established simply connecting the two phone channels using interface **IFX_TAPI_MAP_PHONE_ADD**.

Figure 7 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).

1) Connected via SLIC or DAA device.

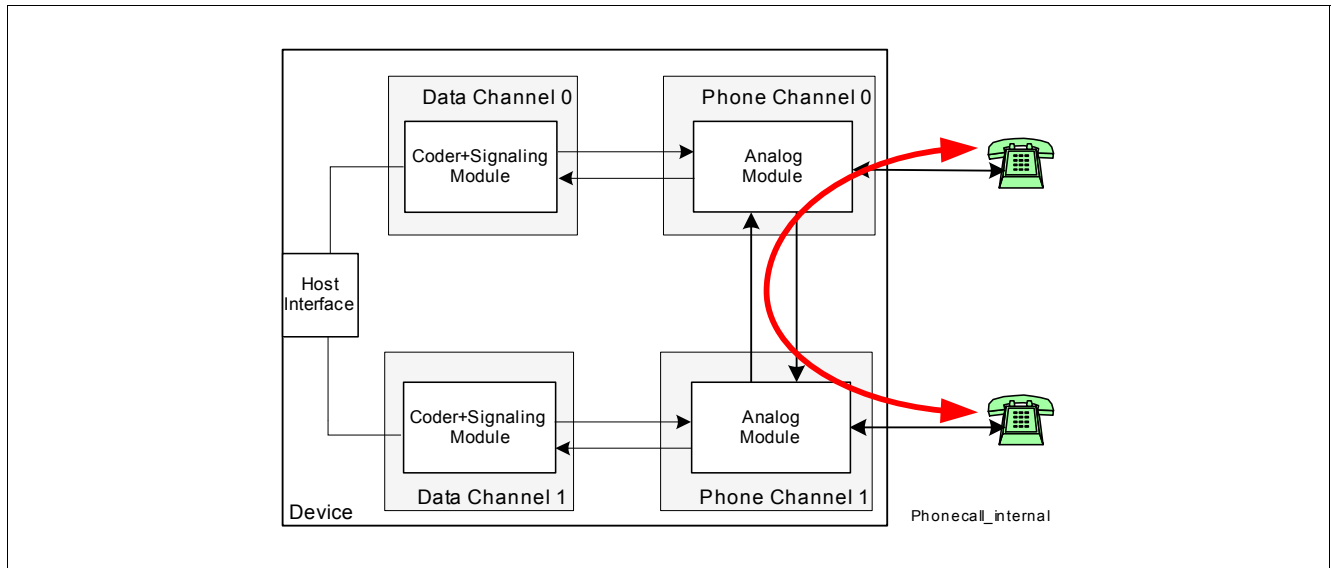


Figure 7 Simple Internal Call

Example - Internal Call

```

IFX_TAPI_CH_INIT_t InitCh1, InitCh2;
IFX_TAPI_MAP_PHONE_t param;
IFX_int32_t fd0, fd1;

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 */
InitCh1.nMode = IFX_TAPI_INIT_MODE_VOICE_CODER;
InitCh2.nMode = IFX_TAPI_INIT_MODE_VOICE_CODER;
ioctl(fd0, IFX_TAPI_CH_INIT, &Init);
ioctl(fd1, IFX_TAPI_CH_INIT, &Init);

/* Now connect the two channels, as in Figure 4 */
/* Connect phone channel 1 (nPhoneCh=1) to phone channel 0 (fd0) */
param.nPhoneCh = 1;
ioctl(fd0, IFX_TAPI_MAP_PHONE_ADD, &param);

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

```

2.6.2 Voice Call with External Party

Figure 8 depicts the usage of the resources for two VoIP calls in parallel.

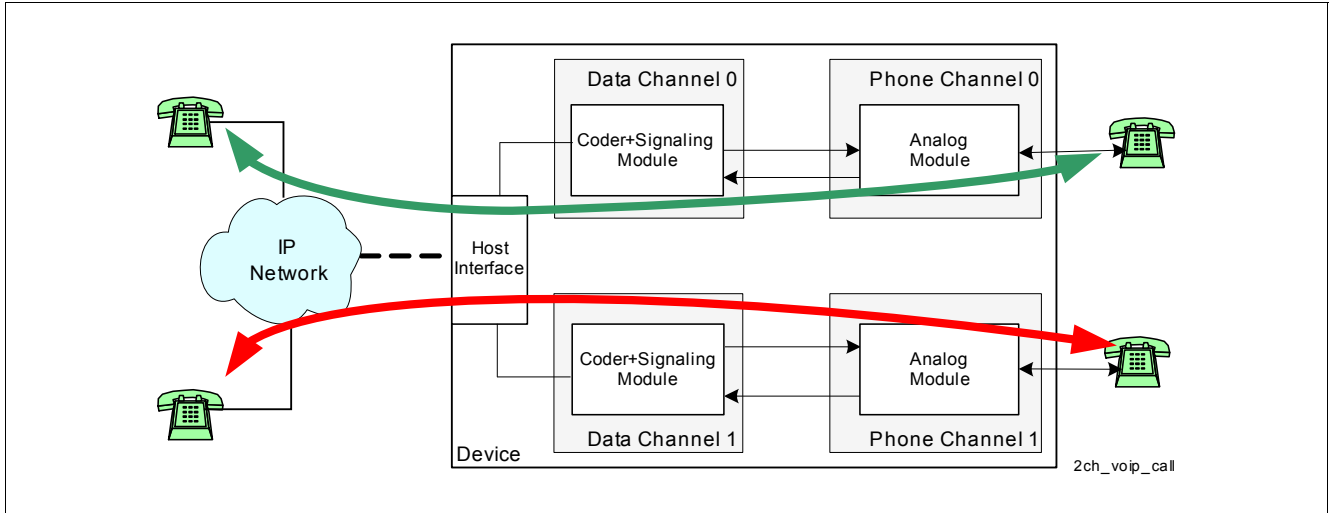


Figure 8 Two Independent VoIP Calls

Example - Use Default Resource Assignment

```
IFX_TAPI_CH_INIT_t InitCh1, InitCh2;
IFX_int32_t fd0, fd1;

memset(&InitCh1, 0, sizeof(IFX_TAPI_CH_INIT_t));
memset(&InitCh2, 0, sizeof(IFX_TAPI_CH_INIT_t));

/* Initialize the channels for VoIP applications */
/* Each channel contains one analog and one data resource */
InitCh1.nMode = IFX_TAPI_INIT_MODE_VOICE_CODER;
InitCh2.nMode = IFX_TAPI_INIT_MODE_VOICE_CODER;
ioctl(fd0, IFX_TAPI_CH_INIT, &InitCh1);
ioctl(fd1, IFX_TAPI_CH_INIT, &InitCh2);
/* Now the two channels are initialized as in Figure 8 */
/* For a complete channel initialization please refer to Chapter 2.2.2 example */
/* It is possible to start all call services
```

Example - Explicit Resource Assignment

```
IFX_TAPI_CH_INIT_t InitCh1, InitCh2;
IFX_int32_t fd0, fd1;
IFX_TAPI_MAP_DATA_t datamap;

memset(&InitCh1, 0, sizeof(IFX_TAPI_CH_INIT_t));
memset(&InitCh2, 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, &InitCh1);
ioctl(fd1, IFX_TAPI_CH_INIT, &InitCh2);

/* Now the two channels are initialized as in Figure 5 */
/* For a complete channel initialization please refer to */
/* Chapter 2.2.2 example */
/* 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;
ioctl(fd0, IFX_TAPI_MAP_DATA_REMOVE, &datamap);
/* Remove connection between phone channel 1 (nDstCh=1) and */
/* data channel 1 (fd1) */
datamap.nDstCh = 1;
ioctl(fd1, IFX_TAPI_MAP_DATA_REMOVE, &datamap);
/* Now phone and data resources are unmapped */
/* Now application wants to map the resources as in Figure 5 */
/* Connect phone channel 0 (nDstCh=0) to data channel 0 (fd0) */
datamap.nDstCh = 0;
ioctl(fd0, IFX_TAPI_MAP_DATA_ADD, &datamap);
/* Connect phone channel 1 (nDstCh=1) to data channel 1 (fd1) */
datamap.nDstCh = 1;
ioctl(fd1, IFX_TAPI_MAP_DATA_ADD, &datamap);
/* Now phone and data resources are mapped as per Figure 8 */

```

2.6.3 Conferencing

This chapter lists the conferencing services with some scenarios. Conferencing is supported by the services **IFX_TAPI_MAP_DATA_ADD**, **IFX_TAPI_MAP_DATA_REMOVE**, **IFX_TAPI_MAP_PHONE_ADD** and **IFX_TAPI_MAP_PHONE_REMOVE**. These services map data or phone channels to other channels.

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 (chip) internal calls.

External calls are established using a data channel. Internal calls can also be established connecting phone channels internally in the chip.

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

2.6.3.1 Initialization

If using TAPI conferencing with resource management, the data channels should be disconnected on start up. To simplify the example, return values are not handled.

```

IFX_TAPI_MAP_DATA_t datamap;
IFX_TAPI_CH_INIT_t Init;
IFX_TAPI_CAP_t cap;
IFX_int32_t fd0, fd1;

```

```

/* Initialize the device and all channels. It sets up the configuration */
/* to default (nMode = TAPI_DEFAULT, country = default). */
memset(&Init, 0, sizeof(IFX_TAPI_CH_INIT_t));

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

/* Check coder channel amount */
cap.capttype = 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 */

```

2.6.3.2 Conference with an Internal and an External Party

Continuing the setup from [Chapter 2.6.1](#), an external party can be added (see [Figure 9](#)). 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.

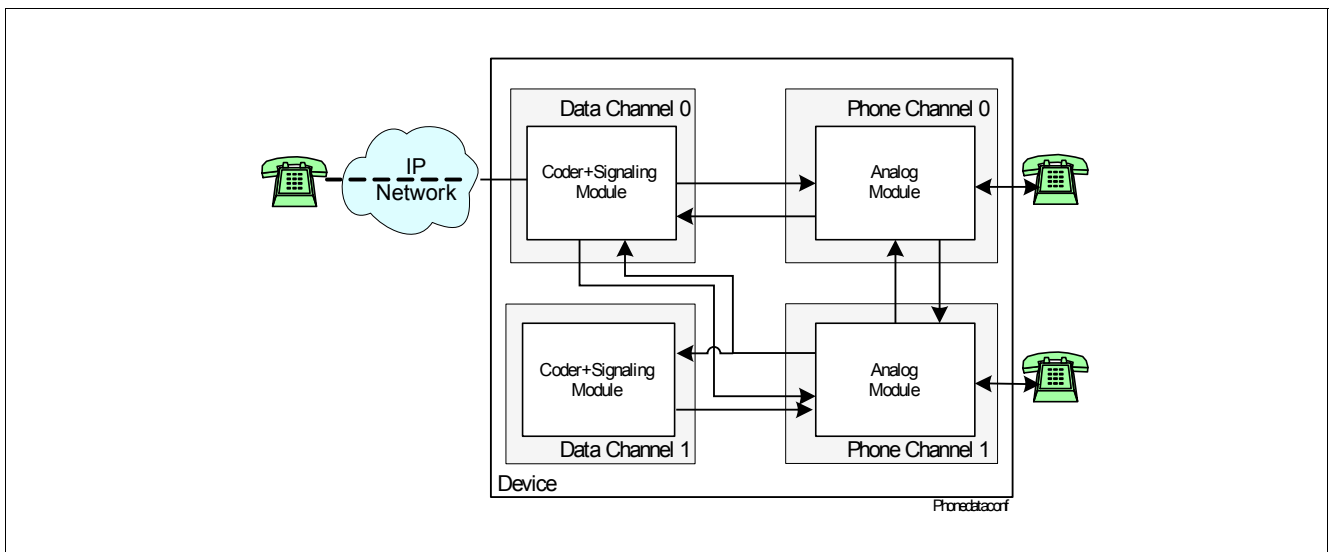


Figure 9 External and Internal Conference

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;
IFX_int32_t fd0, fd1;

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

/* Add phone 0 to phone 1 */
ioctl(fd1, IFX_TAPI_MAP_PHONE_ADD, &phonemap);

```

```
phonemap.nPhoneCh = 0;
/* Add phone 0 to data 0 */
ioctl(fd0, IFX_TAPI_MAP_DATA_ADD, &datamap);
datamap.nDstCh = 1;
/* Add phone 1 to data 0 */
ioctl(fd0, IFX_TAPI_MAP_DATA_ADD, &datamap);
ioctl(fd0, IFX_TAPI_ENC_START, 0);
```

Removing the Connection

To get back the setup as after the initialization described in [Chapter 2.6.3.1](#), the connections are removed with the following commands:

```
IFX_TAPI_MAP_PHONE_t phonemap;
IFX_TAPI_MAP_DATA_t datamap;
IFX_int32_t fd0, fd1;

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

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

2.6.3.3 Conference with Two External Parties

Starting from the setup from [Chapter 2.6.3.1](#), two external parties can be added to the local phone channel 0 (see [Figure 10](#)). In this case, data channels 0 and 1 are mapped to the phone channel 0. The connection between the data channels must not be configured, but is done automatically.

```
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;
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);
```

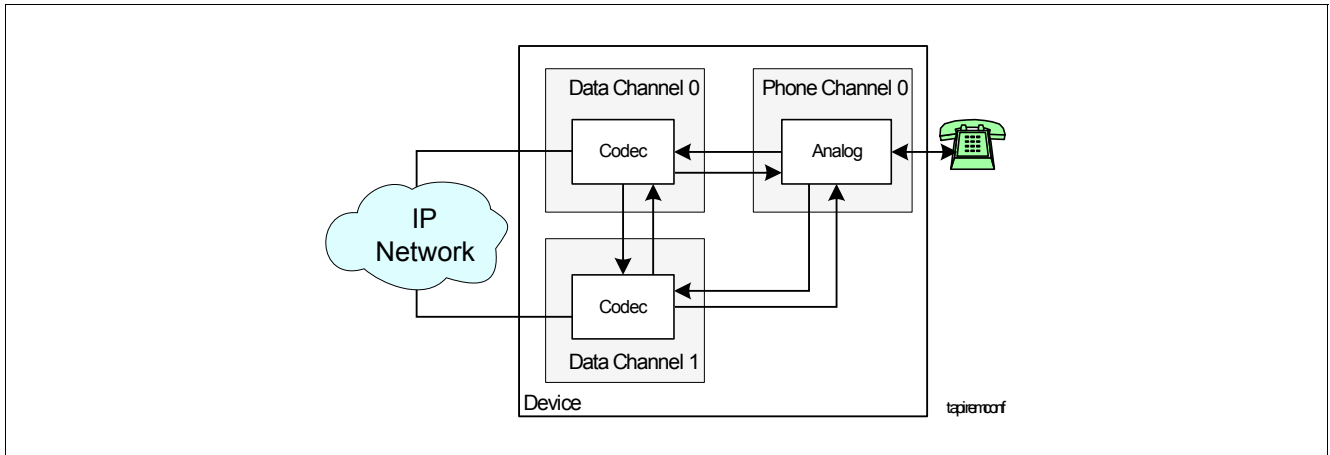


Figure 10 External Conference

Removing the Connection

To get back the setup as after the initialization described in [Chapter 2.6.3.1](#), the connections are removed with the following commands:

```
IFX_TAPI_MAP_DATA_t datamap;
IFX_int32_t fd0, fd1;

memset(&datamap, 0, sizeof(IFX_TAPI_MAP_DATA_t));
ioctl(fd0, IFX_TAPI_ENC_STOP, 0);
ioctl(fd1, IFX_TAPI_ENC_STOP, 0);
datamap.nDstCh = 0;
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

- Channel configuration, [Chapter 3.1](#)
- Configuration and control of encoder/decoder, [Chapter 3.2](#)
- Connection statistics retrieval, [Chapter 3.3](#)
- Tone management (play, detect call progress tones, etc), [Chapter 3.4](#)
- Caller ID and ringing support, [Chapter 3.5](#)
- Fax/modem support, [Chapter 3.6](#)
- Metering support, [Chapter 3.7](#)

3.1 Channel Configuration

This chapter describes configuration options for a VINETIC® channel, in particular how to set up and/or use:

- Input/output gains (volume), [Chapter 3.1.1](#)
- Line Echo Cancellor (LEC), [Chapter 3.1.3](#)
- PCM interface, [Chapter 3.1.2](#)
- Jitter Buffer parameters, [Chapter 3.1.4](#)
- RTP parameters, [Chapter 3.1.5](#)
- AAL2 parameters, [Chapter 3.1.6](#)

3.1.1 Phone Volume

It is possible to tune input (RX) and output (TX) gains independently for the analog phone port¹⁾ using the interface [IFX_TAPI_PHONE_VOLUME_SET](#) for the specific channel fd.

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 CRAM 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).
- 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 CRAM 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 CRAM 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.

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

Example - Volume

```

/* Configure RX and TX gains */
IFX_TAPI_LINE_VOLUME_t volume;
IFX_int32_t fd;

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

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

/* Values can vary between -24dB .. +24dB */
/* Set the input gain to, for example, -2 dB */
volume.nGainRx = 24;
/* 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 PCM

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

Each VINETIC® channel 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 VINETIC® side: the device will start transmitting/receiving on the programmed time slots and highway.

The PCM interface can be connected to an ALM using **IFX_TAPI_MAP_PCM_ADD** and to a coder (COD) using **IFX_TAPI_MAP_DATA_ADD**, for more details on establishing connection between ALM, PCM and COD please refer to [Chapter 2.6](#).

Attention: Activation of LEC in the PCM interface might be required; see also [Chapter 3.1.3](#).

Example - Program PCM Interface

```

IFX_TAPI_PCM_CFG_t pcmConf;
IFX_int32_t activated;
IFX_int32_t fd;

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

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;

pcmConf.nTimeslotTX);

```



```
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 - test it */
```

3.1.2.1 Example of System using PCM Interface

The example in [Figure 11](#) shows some possible flows for a CPE system including a VINETIC® and a DuSLIC-S chip set (only codec + SLIC, DSP functionality not provided).

- Flow A: Classical VINETIC® VoIP call, not involving PCM.
- Flow B: Call switched between VINETIC® 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 VINETIC® resources.

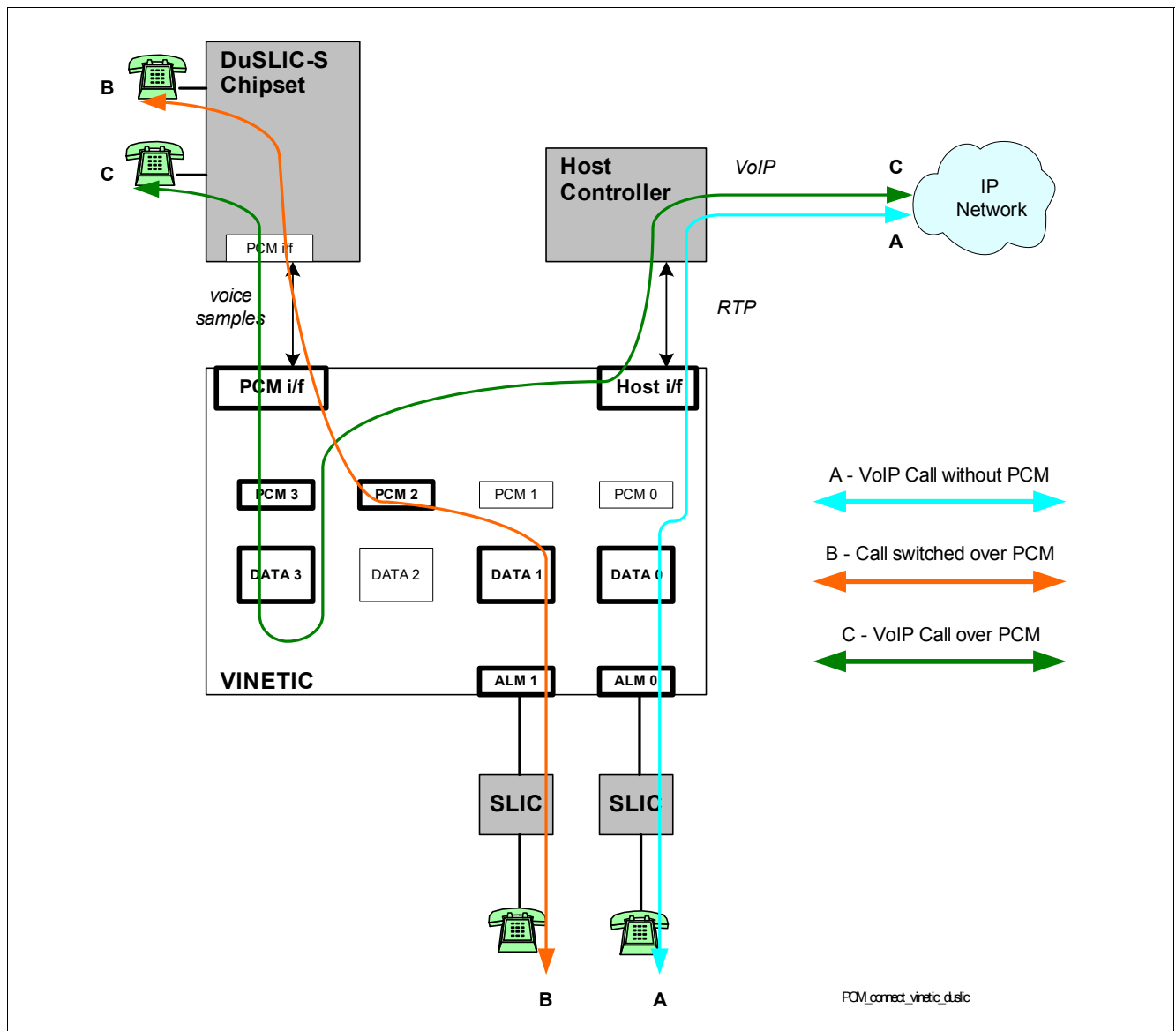


Figure 11 Connection of a DuSLIC via PCM Interface

3.1.3 LEC

The LEC services are used to choose the LEC type (near-end plus far-end or near-end only) 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_LEC_PHONE_CFG_SET**) and in the PCM interface (**IFX_TAPI_LEC_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_LEC_CFG_t** struct.

In systems where the analog telephone is directly connected to VINETIC®+SLIC (for example flow A in **Figure 11**), 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 11**), the LEC must¹⁾ be activated in the PCM interface.

Enabling VINETIC® LEC allows cancellation of near-end echo (nType=**IFX_TAPI_LEC_NLEC**).

In order to ensure that LEC is active, nGainIn and nGainOut parameters must be set with value **IFX_TAPI_LEC_GAIN_MEDIUM**.

The example shows how to use LEC in the mixed scenario of **Figure 11**.

Example - LEC Configuration

```
/* Activate LEC for PCM Port */

IFX_TAPI_LEC_CFG_t lecConf;

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

/* Select LEC Type */;
lecConf.nType = IFX_TAPI_LEC_NLEC;

/* Activate LEC */;
lecConf.nGainTX = IFX_TAPI_LEC_GAIN_MEDIUM;
lecConf.nGainRX = IFX_TAPI_LEC_GAIN_MEDIUM;

/* Activate NLP */;
lecConf.bNlp = IFX_TAPI_LEC_NLP_ON;

ioctl(fd, IFX_TAPI_LEC_PCM_CFG_SET, (IFX_int32_t) &lecConf);
```

3.1.4 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;
IFX_int32_t fd;
```

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

```

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

memset (&jbCfgVoice, 0, sizeof(IFX_TAPI_JB_CFG_t));
/* Adaptive JB */
jbCfgVoice.nJbType = IFX_TAPI_JB_TYPE_FIXED;
/* Optimization for voice */
jbCfgVoice.nPckAdpt = IFX_TAPI_JB_PKT_ADAPT_VOICE;
jbCfgVoice.nScaling = 0x16;
/* Initial JB size 90 ms = 0x2D0 * 125 µs */
jbCfgVoice.nInitialSize = 0xDFF;
/* Minimum JB size 10 ms = 0x50 * 125 µs */
jbCfgVoice.nMinSize = 0xFF;
/* Maximum JB size 180 ms = 0x5A0 * 125 µs */
jbCfgVoice.nMaxSize = 0xFFF;
ioctl(fd, IFX_TAPI_JB_CFG_SET, (IFX_int32_t) &jbCfgVoice);

```

3.1.5 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 3.1.5.1](#).
- [IFX_TAPI_PKT_RTP_CFG_SET](#) to configure RTP session parameters, see [Chapter 3.1.5.2](#).

3.1.5.1 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.

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.

Example - RTP Payload Types Tables

```

/* Configure RTP Payload Type tables */
/* 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_PT_CFG_t rtpPTConf;
IFX_int32_t fd;

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

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;
/* 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);

```

3.1.5.2 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, initial value for timestamp, payload type for event packets (RFC2833 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

```

/* Configure RTP Payload Type tables */
/* 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;
IFX_int32_t fd;

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

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

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

/* Enable in-band and out-of-band (RFC 2833) event transmission */
rtpConf.nEvents = IFX_TAPI_PKT_EV_OOB_ONLY;
/* Configure payload type for event packets (RFC2833) */
rtpConf.nEventPT = 18;
ioctl(fd, IFX_TAPI_PKT_RTP_CFG_SET, (IFX_int32_t) &rtpConf);

```

3.1.6 AAL2

AAL2 support is given by interface **IFX_TAPI_PKT_AAL_CFG_SET** to configure connection identifiers, start value for the timestamp and interface **IFX_TAPI_PKT_AAL_PROFILE_SET** to configure packet length, UUI codepoint range (**IFX_TAPI_PKT_AAL_PROFILE_RANGE_t**) and encoding type (**IFX_TAPI_ENC_TYPE_t**).

Example - AAL2 Configuration

```

IFX_TAPI_PCK_AAL_CFG_t aalCfg;
IFX_TAPI_PCK_AAL_PROFILE_t aalProfile;

memset(&aalCfg, 0, sizeof (IFX_TAPI_PCK_AAL_CFG_t));
memset(&aalProfile, 0, sizeof (IFX_TAPI_PCK_AAL_PROFILE_t));

aalCfg.nCid = 0x10;
aalCfg.nTimestamp = 0x0;
aalCfg.nCspCid = 0x20;
ioctl(fd, IFX_TAPI_PKT_AAL_CFG_SET, (IFX_int32_t) &aalCfg);

ioctl(fd0, IFX_TAPI_PKT_AAL_PROFILE_SET, (IFX_int32_t) &aalProfile);

```

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 (or AAL2) frames. Before starting encoding, packetization session parameters have to be set (for RTP see also [Chapter 3.1.5](#), for AAL2 see also [Chapter 3.1.6](#)); otherwise, random RTP/AAL2 parameters will apply!

In a similar fashion, decoding of the RTP/AAL2 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 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 clean 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_PKT_RTCP_STATISTICS_t jbStat;
IFX_int32_t fd;

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

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 tones¹⁾, it is possible to define tones with more complex cadences and combinations of frequencies.

Structure of this chapter

- **Chapter 3.4.1**, usage of the Tone Table.
- **Chapter 3.4.2**, play tones.
- **Chapter 3.4.3**, definition of simple tones.
- **Chapter 3.4.4**, definition of composed tones.
- **Chapter 3.4.5**, predefined tones.
- **Chapter 3.4.6**, play tones with high output level.
- **Chapter 3.4.7**, play special tones.
- **Chapter 3.4.8**, call progress tone detection.

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

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 tones¹⁾. 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:

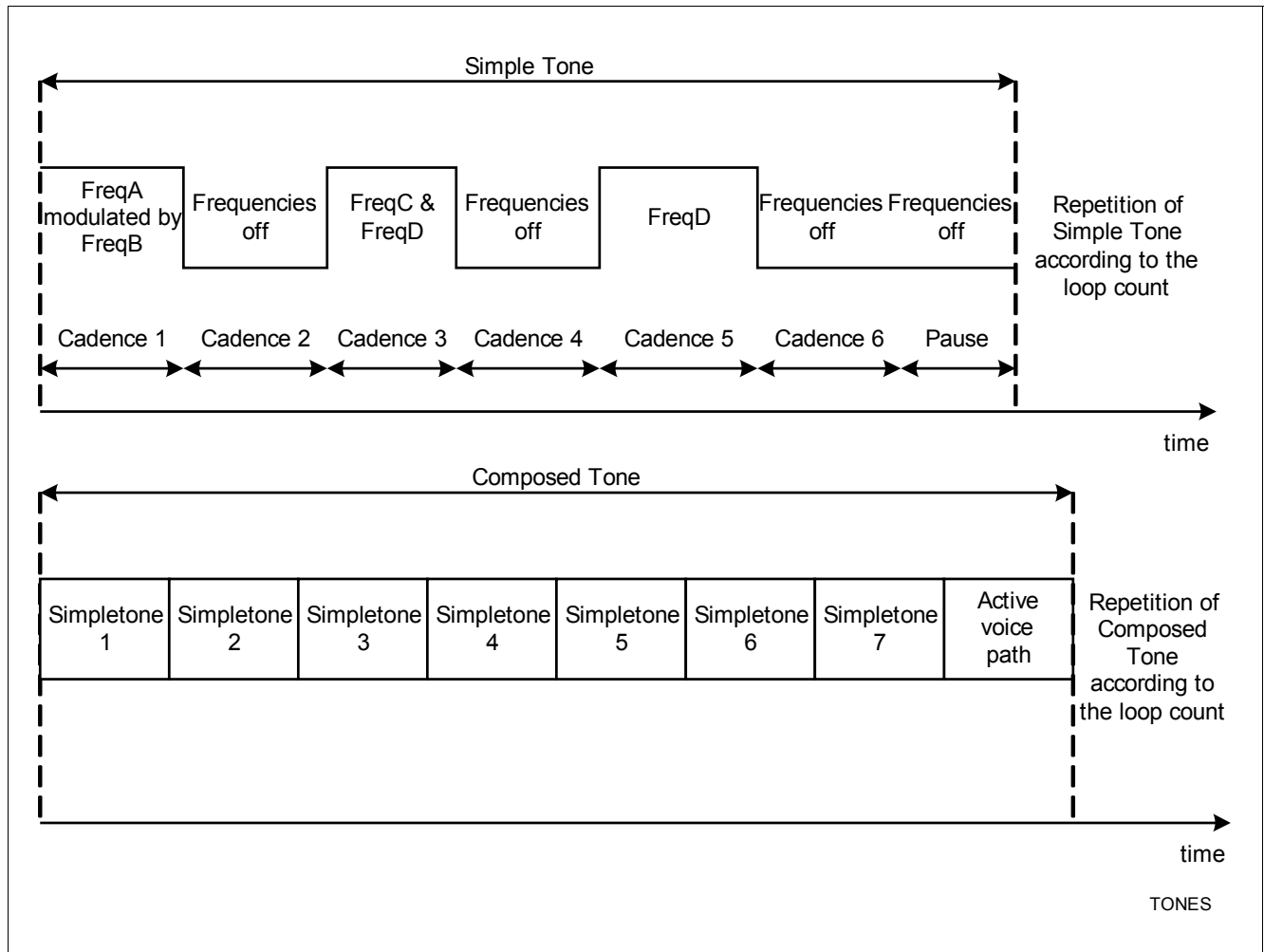


Figure 12 Simple and composed tones

As depicted in [Figure 12](#), 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.

¹⁾ Example of dial tone, busy tone and ringing tone and other tones often used in telephony.

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

```
IFX_int32_t fd;

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

/* 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;
IFX_int32_t fd;

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

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;
IFX_int32_t fd;

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

/* 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 5 reports the predefined tones entries in the tone table.

Table 5 **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)

Table 5 **Tone Table - Predefined Tones (cont'd)**

Index	Frequency 1	Level1 [dB]	Frequency 2	Level 2 [dB]	Note
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
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

Some howler tones require relative high levels (BT howler tone requires up to +15 dBm) in order to enable the VINETIC® to 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 “[Phone Volume](#)” on Page 45).

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 VINETIC® devices.

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;
IFX_int32_t fd;

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

```

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 VINETIC® system 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 the VINETIC®. 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 encoding¹⁾ of the special tone.

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

3.4.8 Call Progress Tone Detection

The VINETIC® 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 channel status (**IFX_TAPI_CH_STATUS_GET**).

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);

/* Applications waits for CPTD event */

/* Event will be reported by IFX_TAPI_CH_STATUS_GET */

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

3.5 Ringing and Caller ID Support

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

- Introduction to CID and Ringing support, [Chapter 3.5.1](#)
- CID Configuration, [Chapter 3.5.2](#).
- Ringing, [Chapter 3.5.3](#).
- CID Transmission, [Chapter 3.5.4](#)
- CID Reception, [Chapter 3.5.5](#)

3.5.1 Introduction to 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 6](#); the various types are supported¹⁾ according to Telcordia, ETSI, BT and NTT standards.

Table 6 **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” of the VINETIC® system ([Chapter 3.5.2](#)). It allows selection among CID standards supported by VINETIC®, 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 3.5.4](#)) or CID reception ([Chapter 3.5.5](#)) 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 3.5.3](#))

3.5.1.1 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 7](#) (on-hook transmission associated with ringing), [Table 8](#) (on-hook transmission not associated with ringing), [Table 9](#) (off-hook transmission) and [Table 10](#) (abbreviations used).

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

Table 7 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	Short off-hook	FSK	LR - Periodical ringing

Table 8 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
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 9 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 10 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.

Table 10 Caller ID Generation - Abbreviations (cont'd)

Abbreviation	Definition	Note
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

3.5.2 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 13](#) 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 3.5.3](#).

Please refer to the following coding examples on [Page 64](#).

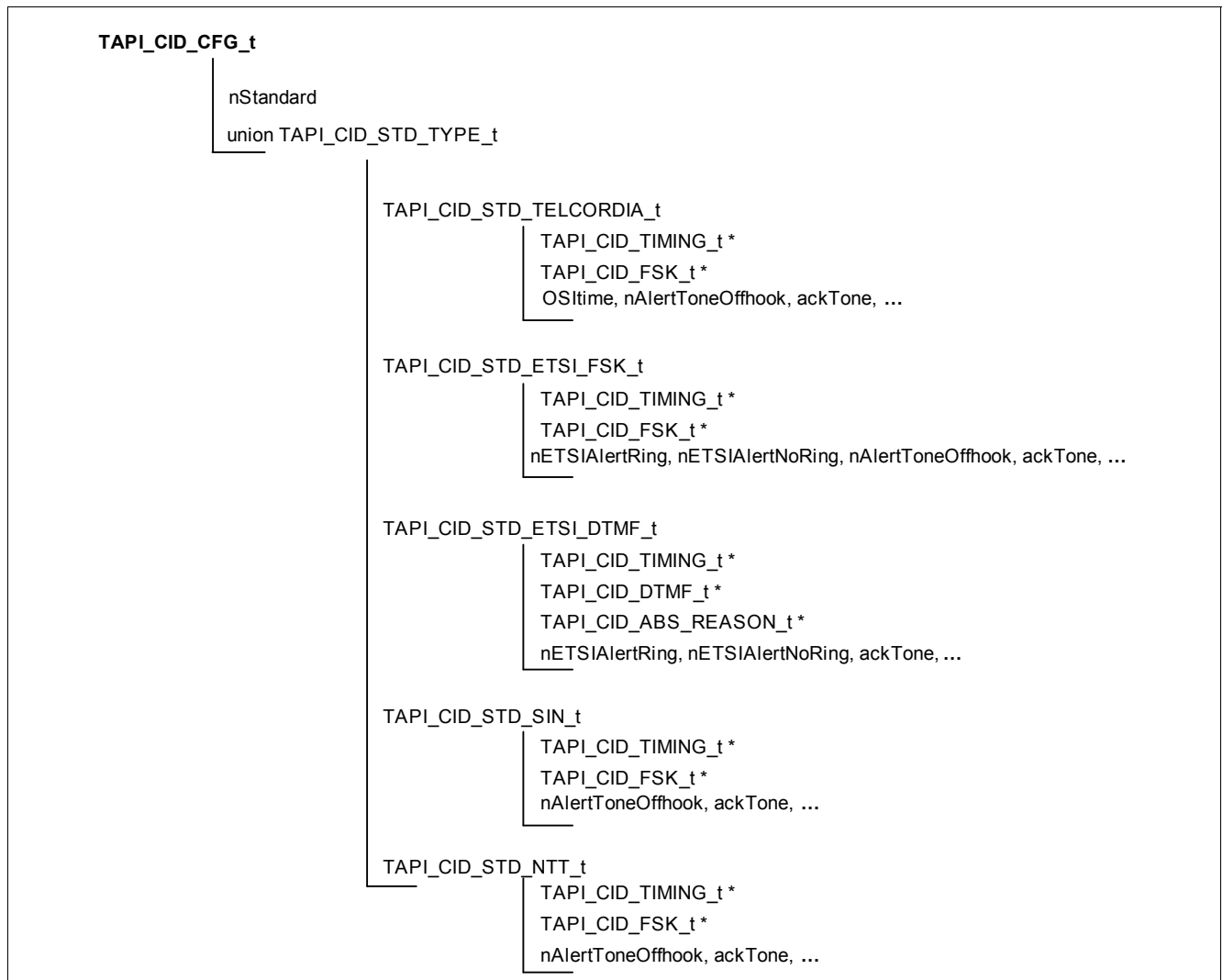


Figure 13 Overview of the Configuration structure

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 11](#) reports which [IFX_TAPI_CID_TIMING_t](#) parameters apply to each CID transmission standard. For a description of each timing parameter, please refer to the struct description.

Table 11 Relevant Timing Applicable to the Different Standards

Timing	Telcordia /Bellcore	ETSI	SIN (BT)	NTT
beforeData	Not applicable	Not applicable	Not applicable	Off-hook
dataOut2restoreTimeOnhook	Not applicable	On-hook ¹⁾	On-hook	On-hook
dataOut2restoreTimeOffhook	Off-hook	Off-hook	Off-hook	Off-hook
ack2dataOutTime	Off-hook	Off-hook	Off-hook	Not applicable
cas2ackTime	Off-hook	Off-hook	Off-hook	Not applicable
afterAckTimeout	Off-hook	Off-hook	Off-hook	Not applicable

Table 11 Relevant Timing Applicable to the Different Standards (cont'd)

Timing	Telcordia /Bellcore	ETSI	SIN (BT)	NTT
afterFirstRing	On-hook associated with ringing	On-hook associated with ringing ²⁾	Not applicable	Not applicable
afterRingPulse	Not applicable	On-hook ³⁾	Not applicable	Not applicable
afterDTASOnhook	Not applicable	On-hook ⁴⁾	On-hook	Not applicable
afterLineReversal	Not applicable	On-hook ⁵⁾	On-hook	Not applicable
afterOSI	On-hook	Not applicable	Not applicable	Not applicable

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

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;
IFX_int32_t fd;

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

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;
IFX_int32_t fd;

/* open channel file descriptor for channel 0 */

```

```
fd = open("/dev/vin11", O_RDWR, 0x644);

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);
```

3.5.3 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).

3.5.3.1 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)
```

3.5.3.2 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 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).

3.5.3.3 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;
IFX_int32_t fd;

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

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);
```

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

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

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

3.5.4 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 3.5.4.1 describes how to prepare the CID message, and **Chapter 3.5.4.2** gives some examples of CID transmission.

3.5.4.1 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.

Fields `txMode` and `msgType` determine CID type; see also **Table 12** 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” activated the `msgType` is **IFX_TAPI_CID_MT_TRANSPARENT**. 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 3.5.4.2**.

Table 12 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](#)
- Transparent transmission, [IFX_TAPI_CID_MT_TRANSPARENT](#)

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](#)

3.5.4.2 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;
IFX_int32_t fd;

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

/* 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];
IFX_int32_t fd;

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

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_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];
IFX_int32_t fd;

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

```



```
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.messageType = 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;
IFX_int32_t fd;

/* 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};

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

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.messageType = 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);
```

3.5.5 CID RX

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.

For Caller ID reception, it is necessary to distinguish between DTMF and FSK transmission.

CID receiver for FSK transmission can be implemented using the following steps:

- Start CID receiver with **IFX_TAPI_CID_RX_START**
- Wait for the **IFX_TAPI_LINE_STATUS_CIDRX** event, i.e. sleep on select()
- Poll the CID receiver status with **IFX_TAPI_CID_RX_STATUS_GET**; this service reports receiver status (active/inactive), reception progress (ongoing, data ready) and errors during CID reception.

In case of DTMF transmission, the application software has to activate DTMF detection on the line and accumulate the single DTMF digits (see also [Chapter 2.5.2](#)).

Example - CID Receiver (FSK)

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

IFX_TAPI_CID_RX_STATUS_t cidRxStatus;
IFX_TAPI_CID_RX_DATA_t cidRxData;
IFX_int32_t fd;
IFX_return_t ret;

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

memset(&cidRxStatus, 0, sizeof (IFX_TAPI_CID_RX_STATUS_t));
memset(&cidRxData, 0, sizeof (IFX_TAPI_CID_RX_DATA_t));
/* Configure CID Standard and if required seizure and mark bits */
/* Start CID RX Engine */
ret = ioctl(fd, IFX_TAPI_CID_RX_START, 0);

/*wait for IFX_TAPI_LINE_STATUS_CIDRX event (sleep on select)*/

/* 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 */
```

3.6 Fax/Modem Support

This chapter describes TAPI support for fax/modem transmission

- Detection of Fax/Modem signals; see [Chapter 3.6.1](#) for details.
- Pass-through mode (also called transparent mode) is supported for both fax and modem; see [Chapter 3.6.2](#) for details.
- Fax relay mode: control T.38 data pump¹⁾; see [Chapter 3.6.3](#) for details.

3.6.1 Detect Fax/Modem Signals

Interface [IFX_TAPI_SIG_DETECT_ENABLE](#) makes it possible to select in-band fax/modem signals (originated by local subscriber and/or coming inband and originated by far end) that should be detected by the firmware; the event will be reported by [IFX_TAPI_CH_STATUS_GET](#).

To disable signal detection, [IFX_TAPI_SIG_DETECT_DISABLE](#) shall be used.

Example - Detect Fax/Modem Signals

```
IFX_TAPI_SIG_DETECTION_t startSig;
IFX_TAPI_SIG_DETECTION_t stopSig;
IFX_int32_t fd;

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

memset(&startSig, 0, sizeof (IFX_TAPI_SIG_DETECTION_t));
memset(&stopSig, 0, sizeof (IFX_TAPI_SIG_DETECTION_t));

/* Example: start detection of Fax DIS and CNG for Fax and Modem */
startSig.sig = IFX_TAPI_SIG_DISRX | IFX_TAPI_SIG_CNGFAXRX
              | IFX_TAPI_SIG_CNGMODRX;
ioctl(fd, IFX_TAPI_SIG_DETECT_ENABLE, (IFX_int32_t) &startSig);
/* Event will be reported through IFX_TAPI_TONE_STATUS_GET, signal field */
/* After detection of start signal: a fax/modem connection is going to */
/* start */
/* Disable detection of the start signals */
ioctl(fd, IFX_TAPI_SIG_DETECT_DISABLE, (IFX_int32_t) &startSig);
/* Now start detection of fax/modem stop using holding signal */
stopSig.sig = IFX_TAPI_SIG_TONEHOLDING_END;
ioctl(fd, IFX_TAPI_SIG_DETECT_ENABLE, (IFX_int32_t) &stopSig);
/* Event will be reported through IFX_TAPI_TONE_STATUS_GET, signal field */
/* After detection of stop signal or onhook : the fax/modem connection */
/* ended */
/* Disable detection of the stop signal */
ioctl(fd, IFX_TAPI_SIG_DETECT_DISABLE, (IFX_int32_t) &stopSig);
```

3.6.2 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.

¹⁾ The T.38 data pump runs in VINETIC® firmware and complements the T.38 protocol implementation running on top of TAPI.

- **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: **IFX_TAPI_SIG_DETECT_ENABLE** is used to detect the end of fax/modem transmission, and as soon as the signal 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_LEC_CFG_t lecCfgVoice, lecCfgData;
IFX_TAPI_ENC_TYPE_t encTypeVoice, encTypeData;
IFX_int32_t fd;

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

memset(&jbCfgVoice, 0, sizeof (IFX_TAPI_JB_CFG_t));
memset(&jbCfgData, 0, sizeof (IFX_TAPI_JB_CFG_t));
memset(&lecCfgVoice, 0, sizeof (IFX_TAPI_LEC_CFG_t));
memset(&lecCfgData, 0, sizeof (IFX_TAPI_LEC_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 */
/* lecCfgVoice 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  $\mu$ s */
jbCfgVoice.nInitialSize = 0x02D0;
/* Minimum JB size 10 ms = 0x50 * 125  $\mu$ s */
jbCfgVoice.nMinSize = 0x50;
/* Maximum JB size 180 ms = 0x5A0 * 125  $\mu$ s */
jbCfgVoice.nMaxSize = 0x5A0;

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

/* Disable LEC */
lecCfgData.nGainOut = IFX_TAPI_LEC_GAIN_OFF;
lecCfgData.nGainIn = IFX_TAPI_LEC_GAIN_OFF;

```

```
lecCfgData.bNlp = IFX_TAPI_LEC_NLP_OFF;  
ioctl(fd, IFX_TAPI_LEC_PHONE_CFG_SET, (IFX_int32_t) &lecCfgData);
```

3.6.3 T.38 Data-Pump Mode

TAPI includes interfaces for controlling the T.38 data pump, implemented in the VINETIC® firmware.

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 frames¹⁾ with the same interfaces defined for packetized voice: the read interface to get modulated data-pump frames from VINETIC®, and write for sending fax frames to the demodulator.

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 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 (using respectively **IFX_TAPI_ENC_STOP, **IFX_TAPI_DEC_STOP** and **IFX_TAPI_LEC_PHONE_CFG_SET**). Deactivation of fax/modem signal detectors (using **IFX_TAPI_SIG_DETECT_DISABLE**) is also recommended; optionally a tone- holding detector might be enabled to detect the end of fax transmission.**

3.7 Metering

Activation and deactivation of metering is done using respectively **IFX_TAPI_METER_START** and **IFX_TAPI_METER_STOP** on the phone channel file descriptor. Configuration of the metering parameters (such as metering burst length, distance between bursts) is done with **IFX_TAPI_METER_CFG_SET**.

Example - Metering

```
/* Configure Metering Support */  
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, (IFX_int32_t) &Metering);  
  
/* Start Metering on the line */  
ioctl(fd, IFX_TAPI_METER_START, (IFX_int32_t) 0);  
  
/* After a while ... */  
  
/* Stop Metering on the line */  
ioctl(fd, IFX_TAPI_METER_STOP, (IFX_int32_t) 0);
```

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.

4 Line Testing Services

The topics described in this chapters are

- Line Testing, [Chapter 4.1](#)
- Line Testing Measurement, [Chapter 4.2](#)
- Network Linetesting Code Setup, [Chapter 4.3](#)
- Usage of GR-909 tests, [Chapter 4.4](#)

4.1 Line Testing

In contrast to all other TAPI services, the line testing services are not realized as *ioctl* calls to the device driver. To avoid floating point operation at device driver level these services are implemented as normal function calls within task (application) level.

If you decide not to use floating point arithmetic on your embedded target, you can do the calculations on a separate controller and forward all parameters received from the *ioctl*. The library code in *lib_tapi_lt* and *lib_tapi_vinetic* has to be ported partly to the separate controller.

Please note that Line Testing is board specific by its nature. The current implementation reduces porting efforts to a minimum, please refer to [Chapter 4.3](#) for details.

4.1.1 Line Testing System Integration

This chapter describes, how to integrate the line testing library into a system. The whole porting splits up into three parts.

- Defining board specific resistors, voltage dividers, etc.
- Defining which io pins are connected to Tip, Ring and Battery Voltages
- (optional) registering a function which controls the external analog multiplexer for battery voltages.

All these settings are done in a project specific application file, in the case of the EASY 334 eval board this file is named *prj_globals.c*, which will be used as reference. The configuration has to be done before executing the first measurement. An example is shown below. The following sections describe the details.

Example

```
void InitTAPI_BoardCoefficients(void)
{
    /* Board specific coefficients /
       see schematic in lib_tapi_lt_vinetic.h */
    TAPI_BoardCoefficients.RIT1      = 510.0; /* Ohm */
    TAPI_BoardCoefficients.RIT2      = 680.0; /* Ohm */
    TAPI_BoardCoefficients.RIL       = 1600.0; /* Ohm */
    TAPI_BoardCoefficients.RS        = 30.0; /* Ohm */
    TAPI_BoardCoefficients.RP         = 20.0; /* Ohm */
    TAPI_BoardCoefficients.RL        = 0.0; /* Ohm */
    TAPI_BoardCoefficients.RCS       = 0.0; /* Ohm */
    TAPI_BoardCoefficients.CS         = 15.0; /* nF */
    TAPI_BoardCoefficients.VoltDiv_TipRing.R1= 10000.0; /* kOhm */
    TAPI_BoardCoefficients.VoltDiv_TipRing.R2= 47.0; /* kOhm */
    TAPI_BoardCoefficients.VoltDiv_VBATH.R1 = 1500.0; /* kOhm */
    TAPI_BoardCoefficients.VoltDiv_VBATH.R2 = 10.0; /* kOhm */
    TAPI_BoardCoefficients.VoltDiv_VBATL.R1 = 1500.0; /* kOhm */
    TAPI_BoardCoefficients.VoltDiv_VBATL.R2 = 10.0; /* kOhm */
    TAPI_BoardCoefficients.VoltDiv_VBATR.R1 = 1500.0; /* kOhm */
    TAPI_BoardCoefficients.VoltDiv_VBATR.R2 = 10.0; /* kOhm */
}
```

```

TAPI_BoardCoefficients.VoltDiv_VHR.R1      = 1000.0; /* kOhm */
TAPI_BoardCoefficients.VoltDiv_VHR.R2      = 10.0; /* kOhm */
TAPI_BoardCoefficients.VoltDiv_VSLD1.R1     = 0.0; /* kOhm */
TAPI_BoardCoefficients.VoltDiv_VSLD1.R2     = 0.0; /* kOhm */
TAPI_BoardCoefficients.VoltDiv_VSLD2.R1     = 0.0; /* kOhm */
TAPI_BoardCoefficients.VoltDiv_VSLD2.R2     = 0.0; /* kOhm */
TAPI_BoardCoefficients.VoltDiv_User1.R1     = 0.0; /* kOhm */
TAPI_BoardCoefficients.VoltDiv_User1.R2     = 0.0; /* kOhm */
TAPI_BoardCoefficients.VoltDiv_User2.R1     = 0.0; /* kOhm */
TAPI_BoardCoefficients.VoltDiv_User2.R2     = 0.0; /* kOhm */
TAPI_BoardCoefficients.VoltDiv_User3.R1     = 0.0; /* kOhm */
TAPI_BoardCoefficients.VoltDiv_User3.R2     = 0.0; /* kOhm */
TAPI_BoardCoefficients.VoltDiv_User4.R1     = 0.0; /* kOhm */
TAPI_BoardCoefficients.VoltDiv_User4.R2     = 0.0; /* kOhm */

/* SLIC specific coefficients / see SLIC datasheet */
TAPI_BoardCoefficients.slic.KIL              = 100.0;
TAPI_BoardCoefficients.slic.KIT              = 50.0;
TAPI_BoardCoefficients.slic.KITPDNR          = 5.0;

/* Select io pins for the different measurements /
   see explanation in lib_tapi_lt_vinetic.h for details */
TAPI_BoardCoefficients.LT_RING_TIP_REVERSE = TRUE;
TAPI_BoardCoefficients.LT_MODE_RING_TIP    =
    VINETIC_IO_LT_VOLTAGE_MODE_IO4_IO3;
TAPI_BoardCoefficients.LT_MODE_TIP_GROUND  =
    VINETIC_IO_LT_VOLTAGE_MODE_IO4;
TAPI_BoardCoefficients.LT_MODE_RING_GROUND =
    VINETIC_IO_LT_VOLTAGE_MODE_IO3;
TAPI_BoardCoefficients.LT_BATTERY          =
    VINETIC_IO_LT_VOLTAGE_MODE_IO2;
/* set function pointer to select the battery voltage
   if your board is able to measure additional voltages
   via an external mux or set to NULL */
TAPI_BoardCoefficients.setAnalogMux        = setAnalogMux;

/* pass the coefficients and function pointer to the TAPI */
setTAPIBoardCoefficients (&TAPI_BoardCoefficients);
}

```

4.1.1.1 Board Specific Constants

The first part of the function shown in the above example is setting board specific coefficients. The specific values for your design can be easily taken from the individual schematic or the SLIC data sheet. The schematic in [Figure 14](#) will guide you to figure out the right values. An updated schematic might be available in `lib_tapi_lt_vinetic.h`.

The voltage dividers to measure the voltage at Tip and Ring have to be mapped to `TAPI_BoardCoefficients.VoltDiv_TipRing.R1` and `R2`. Additional voltage dividers can be specified. For details please refer to [Chapter 4.1.1.3](#).

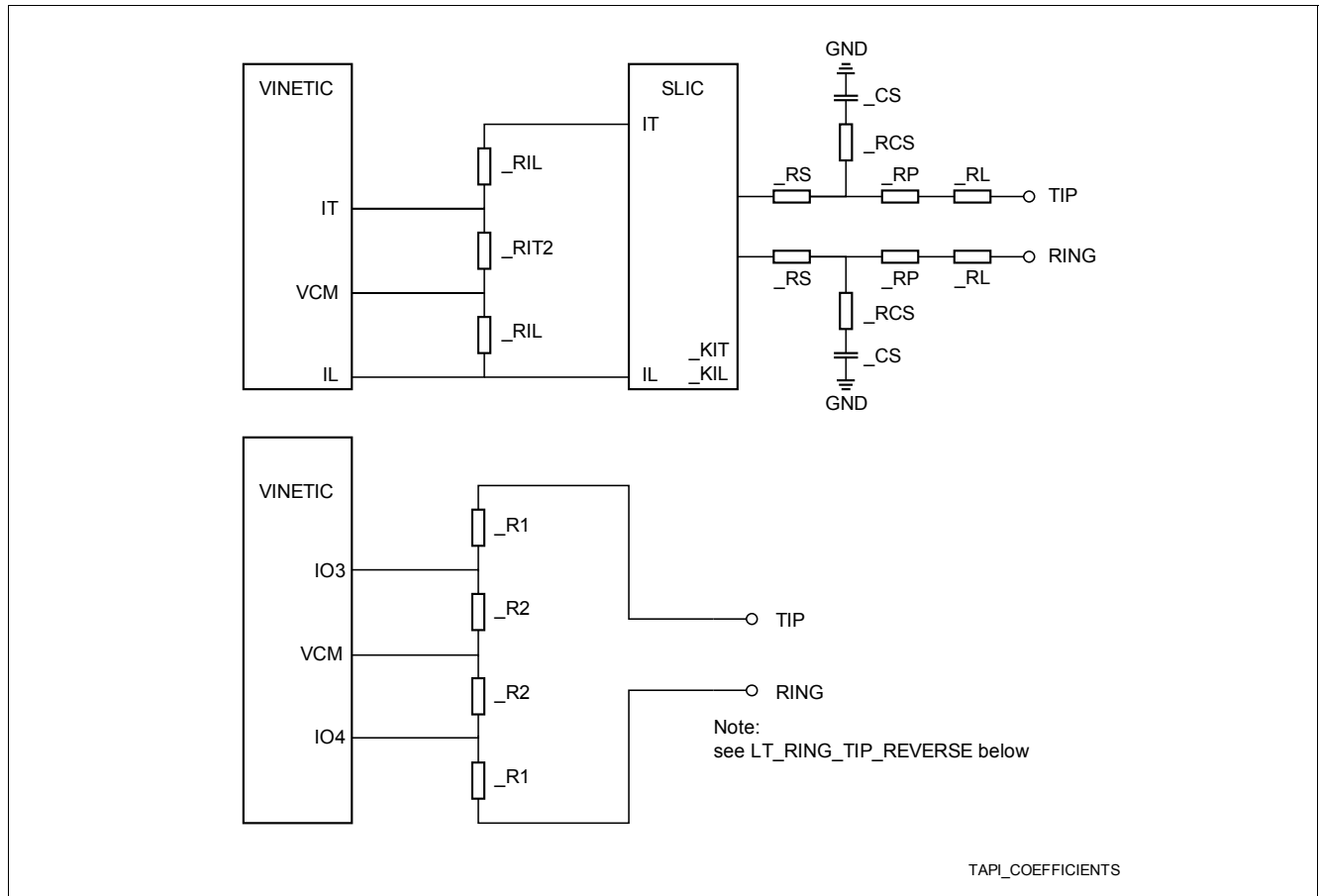


Figure 14 Schematic Showing the Circuit between VINETIC and Tip/Ring

4.1.1.2 Board Specific IO Pin Mapping

Depending on the individual board layout, Tip and Ring as well as the board specific voltages are mapped to specific io pins. To tell the device driver on which pin a certain voltage has to be measured, a couple of mappings are required. If for example Tip and Ring are mapped to IO3 and IO4, LT_MODE_RING_TIP has to be set to VINETIC_IO_LT_VOLTAGE_MODE_IO4_IO3.

To ensure polarity correctness of differential voltages, LT_RING_TIP_REVERSE has to be set to "TRUE", if Tip and Ring are connected to the IOs reversed to [Figure 14](#).

4.1.1.3 Board Specific Voltages (optional)

To measure board specific voltages (for example battery voltages) an external analog multiplexer might be used. There are numerous ways to control the multiplexer, reaching from GPIO pins to additional programmable logic. The TAPI library needs to configure the multiplexer to measure the desired (battery) voltage. Therefore a function pointer has to be registered in the TAPI_BoardCoefficients structure.

If the same io pin is used to measure the voltage at Tip or Ring and the additional board specific voltages, the external multiplexer has to be initialized by the application at boot time - before the first measurement is done. The TAPI library will switch back to this state after each board specific voltage measurement, i.e. it will call the callback function with the parameter TAPI_LT_RESET.

A template function to configure the external multiplexer is shown below.

For each board specific voltage, a set of voltage divider coefficients (the values of the two resistors) has to be configured. [Chapter 4.1.1.1](#) covers the details.

Example

```
int setAnalogMux(int fd, TAPI_LT_BATTERY bat)
{
    int err = SUCCESS;

    switch (bat)
    {
        case TAPI_LT_RESET:
        case TAPI_LT_USER_MODE1:
            break;
        case TAPI_LT_BATTERY_VBATH:
            break;
        case TAPI_LT_BATTERY_VBATL:
            break;
        case TAPI_LT_BATTERY_VHR:
            break;
        case TAPI_LT_BATTERY_VSLD1:
            break;
        case TAPI_LT_BATTERY_VSLD2:
            break;
        default:
            err = ERR;
    }
    return err;
}
```

4.2 Line Testing Measurements

This description is intended for the provided functions implement network linetesting algorithms applicable both over the tip/ring as well as tip/gnd and ring/gnd. The measurement uses an equivalent circuit to describe the local loop. This equivalent circuit consists of capacitance C_{ring} in series with resistance R_{ring} , together in parallel with resistance $R_{tipring}$. The measurement determines the values of these three components using the following described interface.

4.3 Network Linetesting Code Setup

In order to setup the driver sources for compiling the network linetesting functionality a Perl script has been provided. The script namely creates Look Up Tables (LUT) which contain a set of DC voltage measurement values versus LMRES register values for all VINETIC® devices supported by the measurement. Since no floating point calculations is possible in kernel context the linetesting algorithms rely on LUTs. Setting up of the LUT depends on R1 and R2 voltage divider resistances which are design specific components, and these have to be provided as input parameters to the script. Default tables exist in the code for $R1=10\text{ M}\Omega$ and $R2=47\text{ k}\Omega$. See below for an example of running the script on a Linux shell and Windows command prompt respectively.

Example - Perl Script Execution on Linux Shell

```
~/drv_vinetic-1.1.6/vip>python lmres_to_vin.py
Invalid number of arguments!
Usage: lmres_to_vin.py [R1] [R2]
where,
    R1 = voltage divider resistor 1 in kOhm
```

```
R2 = voltage divider resistor 2 in kOhm
~/dev_vinetic-1.1.6/vip>python lmres_ro_vin.py 10000 47
~/dev_vinetic-1.1.6/vip>
```

Example - Perl Script Execution on Windows Command Prompt

```
C:\drv_vinetic-1.1.6\vip>lmres_to_vin.py
Invalid number of arguments!
Usage: lmres_to_vin.py [R1] [R2]
where,
    R1 = voltage divider resistor 1 in kOhm
    R2 = voltage divider resistor 2 in kOhm
C:\dev_vinetic-1.1.6\vip>python lmres_ro_vin.py 10000 47
```

4.4 GR-909 Measurements

The first step is to define a subset of measurements to be executed and start them using [Ifxphone_LT_GR909_Start\(\)](#). In the example below all 5 measurements are selected. The second parameter selects the powerline frequency (i.e. EU like (50 Hz) or US like (60 Hz)). Now the thread is put to sleep using the `select()` mechanism.

Once the measurement is completed on driver level, the driver will wakeup the thread again and the application can retrieve the results using [Ifxphone_LT_GR909_GetResults\(\)](#).

To get also the individual results of the measurements, details of the system specific voltage divider must be configured using [Ifxphone_LT_GR909_Config\(\)](#) (not part of this example).

Example Code for GR909

```
#include "lib_tapi_lt_gr909.h"

/* measurement result */
#define GR909_RESULT(str,res) \
    printf ("%s : %s\n\r", (str), \
        (((res) == IFX_TRUE) ? "Passed\0" : "Failed\0"));

IFX_TAPI_CH_STATUS_t status;
IFX_LT_GR909_RESULT_t res;

fd_set          rfds;

/* setup control fd for select */
FD_ZERO (&rfds);
FD_SET (ch_fd, &rfds);

ret = Ifxphone_LT_GR909_Start (ch_fd, IFX_TRUE,
    (IFX_LT_GR909_HPT_MASK | IFX_LT_GR909_FEMF_MASK |
    IFX_LT_GR909_RFT_MASK | IFX_LT_GR909_ROH_MASK |
    IFX_LT_GR909_RIT_MASK));
/* error handling */

if (select(ch_fd + 1, &rfds, NULL, NULL, 0) > 0)
{
    /* control device has events ? */
    if (FD_ISSET (ch_fd, &rfds))
```

```

{
    memset (&status, 0,  sizeof(status));
    status.channels = 0;
    /* get the status of all four channels */
    if (ioctl(ch_fd, IFX_TAPI_CH_STATUS_GET, (int)&status) == IFX_ERROR)
    {
        printf ("gr909: error occurred at status reading\n");
        return IFX_FALSE;
    }
    if (status.line & IFX_TAPI_LINE_STATUS_GR909RES)
    {
        memset (&res, 0, sizeof (res));
        ret = Ifxphone_LT_GR909_GetResults (fd_line, &res);
        /* error handling */

        /* print some traces */
        printf ("gr909: valid results = 0x%lX\n\r", res.valid_mask);
    }
    /* Hazardous Potential Tests (HPT) */
    if (res.valid_mask & IFX_LT_GR909_HPT_MASK)
    {
        GR909_RESULT("\n\rHazardous Potential Tests (HPT)\0",
                     res.hpt.b_result);
    }
    /* Foreign electromotive Force Tests (FEMT) */
    if (res.valid_mask & IFX_LT_GR909_FEMF_MASK)
    {
        GR909_RESULT("\n\rForeign electromotive Force Tests (FEMT)\0",
                     res.femf.b_result);
    }
    /* Resistive Fault Tests (RFT) */
    if (res.valid_mask & IFX_LT_GR909_RFT_MASK)
    {
        GR909_RESULT("\n\rResistive Fault Tests (RFT)\0",
                     res.rft.b_result);
    }
    /* Receiver Offhook Tests (ROH) */
    if (res.valid_mask & IFX_LT_GR909_ROH_MASK)
    {
        GR909_RESULT("\n\rReceiver Offhook Tests (ROH)\0",
                     res.roh.b_result);
    }
    /* Ringer Impedance Tests (RIT) */
    if (res.valid_mask & IFX_LT_GR909_RIT_MASK)
    {
        GR909_RESULT("\n\rRinger Impedance Tests (RIT)\0",
                     res.rit.b_result);
    }
}
}

```

5 Device Driver Services

This chapter describes important device driver services.

- Usage of polling mode, [Chapter 5.1](#).
- Usage of programmable I/Os (GPIOs), [Chapter 5.2](#).

5.1 Polling Mode

The driver can be configured to use a polling based packet and event handling in order to reduce the overall interrupt context switching times on multi channel systems. In this case the application has to ensure exact periodical cycles (for example 5 or 10 ms) for the data and event handling to prevent mailbox overflows. [Figure 15](#) shows the general concept.

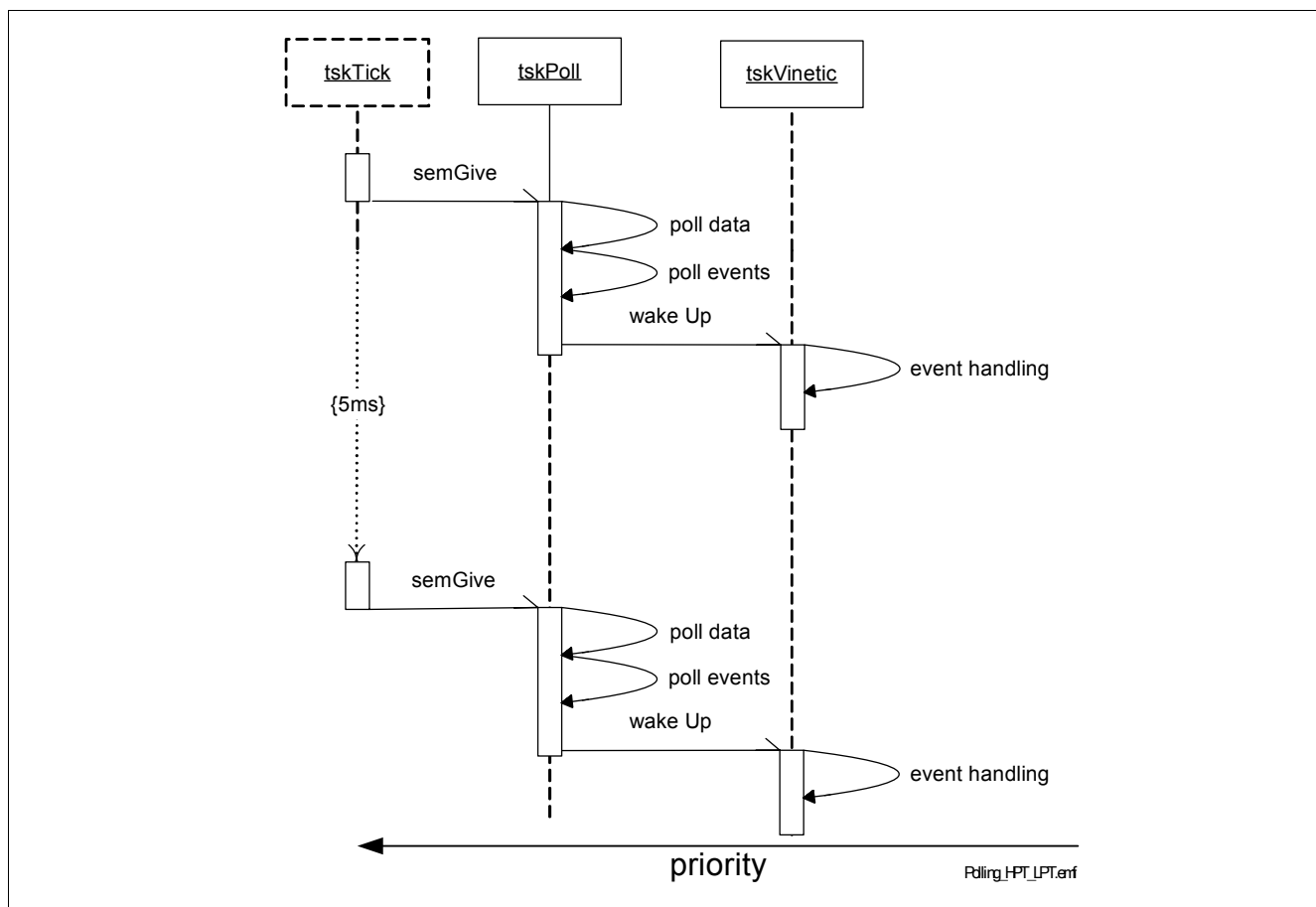


Figure 15 Polling Mode Packet and Event handling

5.1.1 Polling and Interleaved Polling

The VINETIC® device driver provides two flavors to poll voice packets: a simple polling mode as well as an interleaved polling mode.

In case of the simple polling mode, a device is polled for reading or writing packets and events until the mailbox is empty or all packets are sent. After this, the next device is handled.

The interleaved polling mode uses the chip recovery time in which the mailbox cannot be accessed to poll data from other devices. This mode allows an optimal throughput for systems with at least 4 VINETIC® chips (version 1.x), as the routine introduces additional handling overhead in the software.

An independent set of functions for upstream and downstream data is provided for each flavor.

5.1.2 Buffer Handling

To transport voice packets between the driver and the application without the need to copy, an array of pointers to buffers is passed to the driver and vice versa. This concept works well with VxWorks (flat memory) and is not provided for Linux, due to kernel / user space differentiation (virtual memory).

Contrary to the rest of the driver, the polling interface is not channel based. In downstream direction the application will pass a list of all arrived packets (regardless for which channel or device) with one function call to the driver and will retrieve packets from all channels and devices in upstream direction. The desired channel number is encoded inside each packet header.

Each voice packet is stored in a separate buffer. For efficient buffer handling, the application has to provide a set of functions to retrieve or return buffers. In upstream direction the driver takes an empty buffer from the buffer pool for each arriving packet, while in downstream direction the driver will pass the buffer back to the pool after it was sent to the VINETIC®. This can be done using either `lib_bufferpool` which is provided by Infineon (located on the tool CD), or a customer implementation, which provides a compatible interface.

Interface description is done in [Chapter 5.1.3](#).

5.1.3 Driver Polling Interface

To use the driver polling interface, the driver has to be configured accordingly. The following code example shows how to do this. Details regarding the buffer handling are described in [Chapter 5.1.2](#).

Example - Configure Device Driver for Polling

```
VINETIC_IO_DRVCTRL vinPoll;

memset (&vinPoll, 0, sizeof (VINETIC_IO_DRVCTRL));
vinPoll.bPoll = TRUE;
/* configure function pointers the driver will use to retrieve
   from or return buffers to the pool and initialize the buffer
   pool */
vinPoll.pBufferPool =
    (void*)bufferPoolInit (sizeof(PACKET), 250, 0);
vinPoll.getBuf = bufferPoolGet;
vinPoll.putBuf = bufferPoolPut;
ret = ioctl (gFd0, FIO_VINETIC_DRV_CTRL, (int)&vinPoll);
```

After this is done, the timer and polling tasks have to be started. A simplified example of the polling task is shown below. The buffer array has to be sized according to the number of channels in the system. It has to provide enough pointers to take the maximum number of packets that can be read in one cycle. This depends on the number of channels and the maximum number of packets the coder can produce in the polling cycle.

Example - Polling Usage

```
/* Access the driver with a periodical task to
   handle packets in interleaved mode and events */
IFX_int32_t bufElementNum = 0;
IFX_void_t *bufferArray [100] = {0};

void task_5msPeriodical ()
{
    /* downstream */
    VINETIC_Poll_DownIntlv (*bufElementNum, bufferArray);
    /* upstream */
    VINETIC_Poll_UpIntlv (bufElementNum, bufferArray);
}
```

```

/* event handling */
eventHandling ();
}

```

5.2 General-Purpose IOs

The VINETIC® device driver's GPIO module allows access to the VINETIC® GPIO pins¹⁾ from user and kernel mode. To avoid concurrent access, the GPIO pin needs to be reserved (**FIO_VINETIC_GPIO_RESERVE** or **VINETIC_GpioReserve**) before it can be used. Any subsequent try to reserve this pin will fail until it is released explicitly (**FIO_VINETIC_GPIO_RELEASE** or **VINETIC_GpioRelease**).

Any of the GPIO pins can be configured as input or output (**FIO_VINETIC_GPIO_CONFIG** or **VINETIC_GpioConfig**) and the according value can be set²⁾ (**FIO_VINETIC_GPIO_SET** or **VINETIC_GpioSet**) or read out (**FIO_VINETIC_GPIO_GET** or **VINETIC_GpioGet**) using the provided interface.

The GPIO ioctl interface is accessed via the device file descriptor (e.g. /dev/vin10, etc.), the structure **VINETIC_IO_GPIO_CONTROL** is used to configure the pins.

In kernel mode, the calling software needs to pass the address of the device (for the GPIOs) structure on reservation. From this point, the calling party uses the IO handle for subsequent operations.

In addition to the basic input/output operation, the kernel mode supports using interrupt capabilities of the GPIOs to register a callback function. The function **VINETIC_GpioIntMask** can be used to enable and disable the interrupt after registering the callback.

For kernel mode, the structure **VINETIC_GPIO_CONFIG** is used to configure the pins.

Example ioctl Interfaces

Configure pin 0..3 as input and pin 4..7 as output. Pin 4 and 5 should be switched on, 6 and 7 off.

```

/* Configure pin 0..3 as input and pin 4..7 as output. */
/* Pin 4 and 5 should be switched on, 6 and 7 off. */
VINETIC_IO_GPIO_CONTROL gpio;
IFX_int32_t fd_dev;
IFX_return_t err;

memset(&gpio, 0, sizeof(gpio));
/* open the control file descriptor */
fd_dev = open("/dev/vin10", O_RDWR);

/* Reserve the pins, for exclusive access */
/* Select pins: 0..7 */
gpio.nGpio = 0x00FF;
err = ioctl(fd_dev, FIO_VINETIC_GPIO_RESERVE, (IFX_int32_t) &gpio);
/* now gpio contains the iohandle required for subsequent accesses(!) */

/* Configure pin 0..3 as input, 4..7 as output */
/* Select pins 0..7 */
gpio.nGpio = 0x00FF;
/* '1' is for output, '0' for input in the position corresponding to the pin */
gpio.nMask = 0x00F0;
err = ioctl(fd_dev, FIO_VINETIC_GPIO_CONFIG, (IFX_int32_t) &gpio);

```

1) Not available in all packages.

2) Only if the GPIO is configured as output.

```

/* Pins 4 and 5 are high : '1' in bit position 4 and 5 */
/* Pins 6 and 7 are low: '0' in bit position 5 and 7 */
gpio.nGpio = 0x0030;
gpio.nMask = 0x00F0;
err = ioctl(fd_dev, FIO_VINETIC_GPIO_SET, (IFX_int32_t) &gpio);

/* read back the status of all the pins (input and output) */
gpio.nMask = 0x00FF;
err = ioctl(fd_dev, FIO_VINETIC_GPIO_GET, (IFX_int32_t) &gpio);
err = ioctl(fd_dev, FIO_VINETIC_GPIO_RELEASE, (IFX_int32_t) &gpio);

```

Example of Kernel Interfaces

This example reserves 5 GPIO pins, sets pin 1,2 and 5 to value 1 and gets the value of pin 4.

```

/* Initializes the corresponding driver instance */
VINETIC_GPIO_CONFIG ioCfg;
IFX_int32_t ctx, i, hd;
IFX_uint16_t val;

/* Get the device handle with a open from kernel space. */
/* Device 0 and channel 1*/
ctx = VINETIC_OpenKernel(0, 1);
/* reserve 5 pins and get the handle */
hd = VINETIC_GpioReserve(ctx, 0x1F);
ioCfg.nMode = GPIO_MODE_OUTPUT;
ioCfg.nGpio = 0x13;
ioCfg.callback = callback;
/* configure pins 1,2 and 5 as output */
VINETIC_GpioConfig(hd, &ioCfg);
ioCfg.nMode = GPIO_MODE_INPUT;
ioCfg.nGpio = 0x8;
/* configure pin 4 as input */
VINETIC_GpioConfig(hd, &ioCfg);
/* set the value of pin 4 to 1 */
VINETIC_GpioSet(hd, 0x10, 0x10);
/* set the value of pins 1 and 2 to 1 */
VINETIC_GpioSet(hd, 0x03, 0x03);
/* get the value of pin 4 */
VINETIC_GpioGet(hd, &val, 0x08);
/* Release GPIO pin resource, can use also VINETIC_GpioRelease() */
VINETIC_ReleaseKernel(hd);

```

6 TAPI Interfaces

This section describes all TAPI interfaces.

- [Chapter 6.1](#) provides a reference for the ioctl services, grouped by functionality.
- [Chapter 6.2](#) provides the reference for all types defined by TAPI, including
 - [Chapter 6.2.1](#), basic type definition
 - [Chapter 6.2.2](#), summary of all ioctl
 - [Chapter 6.2.3](#), constant reference
 - [Chapter 6.2.4](#), struct reference
 - [Chapter 6.2.5](#), union reference
 - [Chapter 6.2.6](#), enum reference

6.1 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 13 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
Operation Control Service	Operation Control Service
PCM Service	PCM Service
Ringing Service	Ringing Service
Signal Detection Service	Signal Detection Service
Test Services	Test Service
Tone Control Service	Tone Control Service

6.1.1 CID Features Service

Caller ID support.

Table 14 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 whole CID sequence generation.

Table 15 Structure Overview of CID Features

Name	Description
IFX_TAPI_CID_ABS_REASON_t	ABSLI/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 16 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 17 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_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_MSG_TYPE_t	Caller ID Message Type 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_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_CID_VMWI_t	List of VMWI settings.

6.1.1.1 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_TAPI_CID_CFG_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_CID_CFG_SET	I/O control identifier for this operation
IFX_TAPI_CID_CFG_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);
```

6.1.1.2 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_TAPI_CID_RX_DATA_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_TAPI_CID_RX_DATA_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);

```

6.1.1.3 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	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

This command must be sent so that the driver can start collecting CID Data. Only type 1 CID FSK is supported.

Example

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

6.1.1.4 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_TAPI_CID_RX_STATUS_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_TAPI_CID_RX_STATUS_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" );
}

```

6.1.1.5 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);

```

6.1.1.6 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_TAPI_CID_MSG_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_TAPI_CID_MSG_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);

```

6.1.1.7 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_TAPI_CID_MSG_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_TAPI_CID_MSG_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.

6.1.2 Connection Control Service

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

Table 18 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 a connection between a data channel and an analog phone channel or PCM channel.
IFX_TAPI_MAP_DATA_REMOVE	This interface removes a connection between a data channel and an analog phone channel or PCM channel.
IFX_TAPI_MAP_PCM_ADD	This interface adds a connection between a PCM channel and another PCM channel or an analog phone channel.
IFX_TAPI_MAP_PCM_REMOVE	This interface removes a connection between a PCM channel and another PCM channel or an analog phone channel.
IFX_TAPI_MAP_PHONE_ADD	This interface adds a connection between an analog phone channel and another analog phone channel or an PCM channel.
IFX_TAPI_MAP_PHONE_REMOVE	This interface removes a connection between an analog phone channel and another analog phone channel or an PCM channel.
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 19 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 20 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.

6.1.2.1 IFX_TAPI_DEC_LEVEL_GET

Description

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

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) */

```

6.1.2.2 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);
```

6.1.2.3 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);
```

6.1.2.4 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_TAPI_ENC_TYPE_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_TAPI_ENC_TYPE_t	param	IFX_TAPI_ENC_TYPE_t The parameter specifies the codec, default G711.

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

6.1.2.5 IFX_TAPI_DEC_VOLUME_SET

Description

Sets the decoding playout volume.

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);
```

6.1.2.6 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);
```

6.1.2.7 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);
```

6.1.2.8 IFX_TAPI_ENC_LEVEL_SET

Description

This service sets the encoding level.

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) */

```

6.1.2.9 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);
```

6.1.2.10 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);
```

6.1.2.11 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_TAPI_ENC_TYPE_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_TAPI_ENC_TYPE_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);
```

6.1.2.12 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_TAPI_ENC_VAD_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_TAPI_ENC_VAD_t	param	The following values are valid: 0 _D IFX_TAPI_ENC_VAD_NOVAD , voice activity detection not active 1 _D IFX_TAPI_ENC_VAD_ON , voice activity detection switched on 2 _D IFX_TAPI_ENC_VAD_G711 , voice activity detection switched on and comfort noise generation activated, applies to G.711

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);
```

6.1.2.13 IFX_TAPI_ENC_VOLUME_SET

Description

Sets the encoding volume.

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);
```

6.1.2.14 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_TAPI_JB_CFG_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_TAPI_JB_CFG_t*	param	The parameter points to a IFX_TAPI_JB_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

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);
```

6.1.2.15 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_TAPI_JB_STATISTICS_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_TAPI_JB_STATISTICS_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);
```

6.1.2.16 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);
```

6.1.2.17 IFX_TAPI_MAP_DATA_ADD

Description

This interface adds a connection between a data channel and an analog phone channel or PCM channel.

Prototype

```
IFX_int32_t ioctl (
    IFX_int32_t fd,
    IFX_TAPI_MAP_DATA_ADD,
    IFX_TAPI_MAP_DATA_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_TAPI_MAP_DATA_t*	param	The parameter points to a IFX_TAPI_MAP_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

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);

```

6.1.2.18 IFX_TAPI_MAP_DATA_REMOVE

Description

This interface removes a connection between a data channel and an analog phone channel or PCM channel.

Prototype

```

IFX_int32_t ioctl (
    IFX_int32_t fd,
    IFX_TAPI_MAP_DATA_REMOVE,
    IFX_TAPI_MAP_DATA_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_TAPI_MAP_DATA_t*	param	The parameter points to a IFX_TAPI_MAP_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

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 */

```

6.1.2.19 IFX_TAPI_MAP_PCM_ADD

Description

This interface adds a connection between a PCM channel and another PCM channel or an analog phone channel.

Prototype

```

IFX_int32_t ioctl (
    IFX_int32_t fd,
    IFX_TAPI_MAP_PCM_ADD,
    IFX_TAPI_MAP_PCM_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_TAPI_MAP_PCM_t*	param	The parameter points to a IFX_TAPI_MAP_PCM_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\_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);

```

6.1.2.20 IFX_TAPI_MAP_PCM_REMOVE

Description

This interface removes a connection between a PCM channel and another PCM channel or an analog phone channel.

Prototype

```

IFX_int32_t ioctl (
    IFX_int32_t fd,
    IFX_TAPI_MAP_PCM_REMOVE,
    IFX_TAPI_MAP_PCM_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_TAPI_MAP_PCM_t*	param	The parameter points to a IFX_TAPI_MAP_PCM_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_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);

```

6.1.2.21 IFX_TAPI_MAP_PHONE_ADD
Description

This interface adds a connection between an analog phone channel and another analog phone channel or an PCM channel.

Prototype

```

IFX_int32_t ioctl (
    IFX_int32_t fd,
    IFX_TAPI_MAP_PHONE_ADD,
    IFX_TAPI_MAP_PHONE_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_TAPI_MAP_PHONE_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. 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);
```

6.1.2.22 IFX_TAPI_MAP_PHONE_REMOVE

Description

This interface removes a connection between an analog phone channel and another analog phone channel or an PCM channel.

Prototype

```
IFX_int32_t ioctl (
    IFX_int32_t fd,
    IFX_TAPI_MAP_PHONE_REMOVE,
    IFX_TAPI_MAP_PHONE_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_TAPI_MAP_PHONE_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);
```

6.1.2.23 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_TAPI_PCK_AAL_CFG_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_TAPI_PCK_AAL_CFG_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)
```

6.1.2.24 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_TAPI_PCK_AAL_PROFILE_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_PROFILE_SET	I/O control identifier for this operation
IFX_TAPI_PCK_AAL_PROFILE_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);
```

6.1.2.25 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_TAPI_PKT_RTCP_STATISTICS_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_TAPI_PKT_RTCP_STATISTICS_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

The structure refers to the RFC 3550 / 3551. Not all fields may be supported by the TAPI, but must be filled by the calling application.

Example

```
IFX_TAPI_PKT_RTCP_STATISTICS_t param;
IFX_int32_t fd;

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

6.1.2.26 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);
```

6.1.2.27 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_TAPI_PKT_RTP_CFG_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_TAPI_PKT_RTP_CFG_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);

```

6.1.2.28 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_TAPI_PKT_RTP_PT_CFG_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_TAPI_PKT_RTP_PT_CFG_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[6] = 0x5;
param.nPTdown[12] = 0x4;
ioctl(fd, IFX_TAPI_PKT_RTP_PT_CFG_SET, &param);
```

6.1.3 Dial Service

Contains services for dialing.

Table 21 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.

6.1.3.1 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_TAPI_PKT_EV_OOB_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_TAPI_PKT_EV_OOB_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);
```

6.1.3.2 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);
```

6.1.3.3 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);
```

6.1.3.4 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 */
}

```

6.1.3.5 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);

```

6.1.3.6 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);
```

6.1.3.7 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 */  
}
```

6.1.4 Fax T.38 Service

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

Also the data interface changes. So the FAX data is sent and received via the read and write interface of the driver, once the or has been called successfully. A call to switches the driver and the device back to voice processing mode and the functions read and write will service voice data (RTP or AAL packets). All fax services apply to data channels except otherwise stated.

Table 22 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 23 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 24 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.

6.1.4.1 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_TAPI_T38_DEMOD_DATA_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_DEMOD_START	I/O control identifier for this operation
IFX_TAPI_T38_DEMOD_DATA_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);

```

6.1.4.2 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_TAPI_T38_MOD_DATA_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_MOD_START	I/O control identifier for this operation
IFX_TAPI_T38_MOD_DATA_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);

```

6.1.4.3 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_TAPI_T38_STATUS_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_STATUS_GET	I/O control identifier for this operation
IFX_TAPI_T38_STATUS_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);

```

6.1.4.4 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);
```

6.1.5 Initialization Service

This service sets the default initialization of device and hardware.

Table 25 IO-control Overview of Initialization Service

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

Table 26 Structure Overview of Initialization Service

Name	Description
IFX_TAPI_CH_INIT_t	TAPI initialization structure used for IFX_TAPI_CH_INIT .

Table 27 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.

6.1.5.1 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_TAPI_CH_INIT_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_TAPI_CH_INIT_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);
```


6.1.6 Metering Service

Contains services for metering.

All metering services apply to phone channels except otherwise stated.

Table 28 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 29 Structure Overview of Metering Service

Name	Description
IFX_TAPI_METER_CFG_t	Structure for metering config.

6.1.6.1 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_TAPI_METER_CFG_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_TAPI_METER_CFG_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);

```

6.1.6.2 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);

```

6.1.6.3 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);
```

6.1.7 Miscellaneous Services

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

Table 30 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 <code>ENABLE_TRACE</code> .
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 31 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 32 Union Overview of Miscellaneous Services

Name	Description
IFX_TAPI_EXCEPTION_t	Contains TAPI exception information.

Table 33 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.

6.1.7.1 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_TAPI_CAP_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_TAPI_CAP_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.capttype = 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.capttype = 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.capttype = 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);
```

6.1.7.2 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_TAPI_CAP_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_TAPI_CAP_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);

```

6.1.7.3 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);
```

6.1.7.4 IFX_TAPI_CH_STATUS_GET

Description

Query status information about the phone line.

Prototype

```
IFX_int32_t ioctl (
    IFX_int32_t fd,
    IFX_TAPI_CH_STATUS_GET,
    IFX_TAPI_CH_STATUS_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_TAPI_CH_STATUS_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, &rfd, NULL, NULL, NULL);
/* Select woke up from exception on fd_dev or data on fd[x] */
if (FD_ISSET(fd_dev, &rfd))
{
    /* 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], &rfd))
    {
        printf("Handle data\n");
    } /* for */
}

/* close all open fds */
close(fd_dev);
close(fd[0]);
close(fd[1]);
```

6.1.7.5 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.

Data Type	Name	Description
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

6.1.7.6 IFX_TAPI_EXCEPTION_GET

Description

This service returns the current exception status.

Prototype

```
IFX_void_t ioctl (
    IFX_int32_t fd,
    IFX_TAPI_EXCEPTION_GET,
    IFX_TAPI_EXCEPTION_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_TAPI_EXCEPTION_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");
}
```

6.1.7.7 IFX_TAPI_EXCEPTION_MASK

Description

This service masks the reporting of the exceptions.

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 device 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_int32_t nException;

/* Mask reporting of exceptions */
ioctl(fd, IFX\_TAPI\_EXCEPTION\_MASK, nException);
```

6.1.7.8 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_TAPI_VERSION_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_TAPI_VERSION_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");
}
```

6.1.7.9 IFX_TAPI_VERSION_GET

Description

Retrieves the TAPI version string.

Prototype

```
IFX_int32_t ioctl (
    IFX_int32_t fd,
    IFX_TAPI_VERSION_GET,
    IFX_char_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_char_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);
```

6.1.8 Operation Control Service

Table 34 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.
IFX_TAPI_PCM_VOLUME_SET	Sets the PCM interface volume settings.
IFX_TAPI_PHONE_VOLUME_SET	Sets the speaker phone and microphone volume settings.

Table 35 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 .

6.1.8.1 IFX_TAPI_LEC_PCM_CFG_GET

Description

Get the LEC configuration for PCM.

Prototype

```
IFX_int32_t ioctl (
    IFX_int32_t fd,
    IFX_TAPI_LEC_PCM_CFG_GET,
    IFX_TAPI_LEC_CFG_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_TAPI_LEC_CFG_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

6.1.8.2 IFX_TAPI_LEC_PCM_CFG_SET

Description

Set the line echo canceller (LEC) configuration for PCM.

Prototype

```
IFX_int32_t ioctl (
    IFX_int32_t fd,
    IFX_TAPI_LEC_PCM_CFG_SET,
    IFX_TAPI_LEC_CFG_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_TAPI_LEC_CFG_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);

```

6.1.8.3 IFX_TAPI_LEC_PHONE_CFG_SET

Description

Set the line echo canceller (LEC) configuration.

Prototype

```

IFX_int32_t ioctl (
    IFX_int32_t fd,
    IFX_TAPI_LEC_PHONE_CFG_SET_t,
    IFX_TAPI_LEC_CFG_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_LEC_PHONE_CFG_SET_t	I/O control identifier for this operation
IFX_TAPI_LEC_CFG_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);

```

6.1.8.4 IFX_TAPI_LEC_PHONE_CFG_GET

Description

Get the LEC configuration.

Prototype

```

IFX_int32_t ioctl (
    IFX_int32_t fd,
    IFX_TAPI_LEC_PHONE_CFG_GET,
    IFX_TAPI_LEC_CFG_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_TAPI_LEC_CFG_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

6.1.8.5 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_SET .

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);
```

6.1.8.6 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;  
}
```

6.1.8.7 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_TAPI_LINE_HOOK_VT_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_TAPI_LINE_HOOK_VT_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);
```

6.1.8.8 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 . 0 _D IFX_TAPI_LINE_LEVEL_NORMAL , disable the high level path. 1 _D IFX_TAPI_LINE_LEVEL_HIGH , enable the high level path.

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);
```

6.1.8.9 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_TAPI_LINE_VOLUME_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_TAPI_LINE_VOLUME_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);

```

6.1.8.10 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_TAPI_LINE_VOLUME_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_TAPI_LINE_VOLUME_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);

```

6.1.9 PCM Service

Contains services for PCM configuration. Apply to phone channels except otherwise stated.

Table 36 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.

6.1.9.1 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");
}
```

6.1.9.2 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);
```

6.1.9.3 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_TAPI_PCM_CFG_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_TAPI_PCM_CFG_t*	param	Reserved.

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);
```

6.1.9.4 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_TAPI_PCM_CFG_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_TAPI_PCM_CFG_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);

```

6.1.10 Ringing Service

Table 37 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 38 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 .

6.1.10.1 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_TAPI_RING_CADENCE_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_TAPI_RING_CADENCE_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);

```

6.1.10.2 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);
```

6.1.10.3 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_TAPI_RING_CFG_t* param );
```

Parameters

Data Type	Name	Description
IFX_int32_t	fd	Pointer to a file descriptor
IFX_int32_t	IFX_TAPI_RING_CFG_GET	I/O control identifier for this operation
IFX_TAPI_RING_CFG_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

6.1.10.4 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_TAPI_RING_CFG_t* param );
```

Parameters

Data Type	Name	Description
IFX_int32_t	fd	Pointer to a file descriptor
IFX_int32_t	IFX_TAPI_RING_CFG_SET	I/O control identifier for this operation
IFX_TAPI_RING_CFG_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)
```

6.1.10.5 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.

6.1.10.6 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);
```

6.1.11 Signal Detection Service

Table 39 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 40 Structure Overview of Signal Detection Service

Name	Description
IFX_TAPI_SIG_DETECTION_t	Structure used for enable and disable signal detection.

Table 41 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.

6.1.11.1 IFX_TAPI_SIG_DETECT_DISABLE

Description

Disables the signal detection for Fax or modem signals.

Prototype

```
IFX_int32_t ioctl (
    IFX_int32_t fd,
    IFX_TAPI_SIG_DETECT_DISABLE,
    IFX_TAPI_SIG_DETECTION_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_TAPI_SIG_DETECTION_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

6.1.11.2 IFX_TAPI_SIG_DETECT_ENABLE

Description

Enables the signal detection for Fax or modem signals.

Prototype

```
IFX_int32_t ioctl (
    IFX_int32_t fd,
    IFX_TAPI_SIG_DETECT_ENABLE,
    IFX_TAPI_SIG_DETECTION_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_TAPI_SIG_DETECTION_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 switch to modem coefficients after the ending of CED. Not every combination is possible. It depends on the underlying 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");
}
```

6.1.11.3 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_TAPI_TONE_CPTD_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_TAPI_TONE_CPTD_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);
```

6.1.11.4 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

6.1.12 Test Services

6.1.12.1 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);
```

6.1.12.2 IFX_TAPI_TEST_LOOP

Prototype

```
void ioctl (
    IFX_int32_t fd,
    IFX_TAPI_TEST_LOOP,
    IFX_TAPI_TEST_LOOP_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_TAPI_TEST_LOOP_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 transmission. The reception of local voice (upstream) is disabled. That means, that voice applied to the local phone is not being processed, but data sent to the phone can still be heard.

Example

```
IFX\_TAPI\_TEST\_LOOP\_t parm;
memset (&parm, 0,  sizeof (IFX\_TAPI\_TEST\_LOOP\_t));
parm.bAnalog = 1;
ret = ioctl(fd, IFX\_TAPI\_TEST\_LOOP, &parm);
```

6.1.13 Tone Control Service

All tone services apply to data channels file descriptors except otherwise stated.

Table 42 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 43 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 44 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 45 Union Overview of Tone Control Service

Name	Description
IFX_TAPI_TONE_t	Tone descriptor.

Table 46 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.

6.1.13.1 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);
```

6.1.13.2 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);
```

6.1.13.3 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);
```

6.1.13.4 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);
```

6.1.13.5 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_TAPI_TONE_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_TAPI_TONE_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 on 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);

```

6.2 Type Definition Reference

This chapter contains the basic type definitions, reference of data types and structures of all modules.

6.2.1 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](#)

6.2.1.1 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.

6.2.1.2 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.

6.2.1.3 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.

6.2.1.4 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.

6.2.1.5 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.

6.2.1.6 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.

6.2.1.7 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.

6.2.1.8 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.

6.2.1.9 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.

6.2.1.10 IFX_void_t

Prototype

```
typedef void IFX_void_t;
```

Parameters

Data Type	Name	Description
void	IFX_void_t	This is a void datatype.

6.2.1.11 IFX_float_t

Prototype

```
typedef float IFX_float_t;
```


Parameters

Data Type	Name	Description
float	IFX_float_t	This is a float datatype.

6.2.2 IO-control Reference

This chapter contains the IO-control reference.

Table 47 IO-control Overview of TAPI Interfaces

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_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.
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_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.

Table 47 IO-control Overview of TAPI Interfaces (cont'd)

Name	Description
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 a connection between a data channel and an analog phone channel or PCM channel.
IFX_TAPI_MAP_DATA_REMOVE	This interface removes a connection between a data channel and an analog phone channel or PCM channel.
IFX_TAPI_MAP_PCM_ADD	This interface adds a connection between a PCM channel and another PCM channel or an analog phone channel.
IFX_TAPI_MAP_PCM_REMOVE	This interface removes a connection between a PCM channel and another PCM channel or an analog phone channel.
IFX_TAPI_MAP_PHONE_ADD	This interface adds a connection between an analog phone channel and another analog phone channel or a PCM channel.
IFX_TAPI_MAP_PHONE_REMOVE	This interface removes a connection between an analog phone channel and another analog phone channel or a PCM channel.
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.

Table 47 IO-control Overview of TAPI Interfaces (cont'd)

Name	Description
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.
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.

6.2.3 Constant Reference

This chapter contains the constant reference.

Table 48 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
IFX_TAPI_ENC_TYPE_MAX Max number of supported vocoders for the payload table.	00000000 _D
IFX_TAPI_IOC_MAGIC Magic number for ioctl's	00000000 _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

6.2.4 Structure Reference

This chapter contains the Structure reference.

Table 49 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 .
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_ENC_AGC_CFG_t	Structure used for IFX_TAPI_ENC_AGC_CFG_SET .
IFX_TAPI_EXCEPTION_BITS_t	Exception bits.
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 .

Table 49 Structure Overview of TAPI Interfaces (cont'd)

Name	Description
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 used for composed tone configuration.
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 used for simple tone configuration.
IFX_TAPI_TONE_SIMPLE_t	Structure for definition of simple tones.
IFX_TAPI_VERSION_t	Structure used for the TAPI version support check.

6.2.4.1 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. 0 _D IFX_TAPI_CAP_TYPE_VENDOR , string representation of the vendor 1 _D IFX_TAPI_CAP_TYPE_DEVICE , string representation of the underlying device 2 _D IFX_TAPI_CAP_TYPE_PORT , gives information about the available ports, see IFX_TAPI_CAP_PORT_t for port types 3 _D IFX_TAPI_CAP_TYPE_CODEC , gives information about the available codecs as listed in IFX_TAPI_ENC_TYPE_t 4 _D IFX_TAPI_CAP_TYPE_DSP , 1 if DSP is available, otherwise 0 5 _D IFX_TAPI_CAP_TYPE_PCM , cap defines how many pcm channels are available 6 _D IFX_TAPI_CAP_TYPE_CODECS , cap defines how many codec channels are available 7 _D IFX_TAPI_CAP_TYPE_PHONES , cap defines how many phone channels are available 9 _D IFX_TAPI_CAP_TYPE_SIG , signal detectors out of IFX_TAPI_CAP_SIG_DETECT_t
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.

6.2.4.2 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 , default initialization see device driver documentation. This value is used 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	Country selection. For future purposes, not yet used
IFX_void_t *	pProc	Low-level Device initialization pointer. Pointing to a VINETIC_IO_INIT structure.

6.2.4.3 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 signal_ext;
        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.

Data Type	Name	Description
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 occurred 200_H IFX_TAPI_LINE_STATUS_CIDRX, Caller ID received 400_H IFX_TAPI_LINE_STATUS_FEEDLOWBAT, 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	signal_ext	Reserved.
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

6.2.4.4 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".

6.2.4.5 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 .

6.2.4.6 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.

6.2.4.7 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. Default 300 bits, default NTT 0 bits.
IFX_uint32_t	seizureRX	Minimum number of seizure bits for FSK reception. Default 300 bits, default NTT 0 bits.
IFX_uint32_t	markTXOnhook	Number of mark bits for on-hook FSK transmission. Default 180 bits, default NTT 72 bits.
IFX_uint32_t	markTXOffhook	Number of mark bits for off-hook FSK transmission. Default 80 bits, default NTT 72 bits.
IFX_uint32_t	markRXOnhook	Minimum number of mark bits for on-hook FSK reception. Default 150 bits, default NTT 50 bits.
IFX_uint32_t	markRXOffhook	Minimum number of mark bits for off-hook FSK reception. Default 55 bits, default NTT 50 bits.

6.2.4.8 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

6.2.4.9 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.

6.2.4.10 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.

6.2.4.11 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 .

Data Type	Name	Description
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.

6.2.4.12 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.

6.2.4.13 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.

6.2.4.14 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.

6.2.4.15 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 nETSIAlertrRing;
    IFX_int32_t nETSIAlertrNoRing;
    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.

6.2.4.16 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_TAPI_CID_STD_ETSI_FSK_t;

```

```

    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'.

6.2.4.17 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_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	Duration of a ring pulse (CAR signal), in ms, default 500 ms.
IFX_int32_t	ringPulseTimeLoop	Max number of ring pulses (CAR signals), default 5.

6.2.4.18 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.

Data Type	Name	Description
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'.

6.2.4.19 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.
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'.

6.2.4.20 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 transmission.
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 100 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 500 ms.
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.

6.2.4.21 IFX_TAPI_EXCEPTION_BITS_t
Description

Exception bits.

Prototype

```
typedef struct
{
    IFX_uint32_t reserved2;
    IFX_uint32_t eventDetect;
    IFX_uint32_t fax_ced_net;
    IFX_uint32_t fax_cng_net;
    IFX_uint32_t gr909result;
    IFX_uint32_t device;
    IFX_uint32_t signal;
    IFX_uint32_t otemp;
    IFX_uint32_t ground_key_polarity;
    IFX_uint32_t ground_key_high;
    IFX_uint32_t fault;
    IFX_uint32_t fax_supdate;
    IFX_uint32_t fax_dis;
    IFX_uint32_t fax_ced;
    IFX_uint32_t fax_cng;
    IFX_uint32_t pulse_digit_ready;
    IFX_uint32_t ground_key;
    IFX_uint32_t cidrx_supdate;
    IFX_uint32_t callerid_ready;
    IFX_uint32_t pstn_ring;
    IFX_uint32_t flash_hook;
    IFX_uint32_t hookstate;
    IFX_uint32_t dtmf_ready;
} IFX_TAPI_EXCEPTION_BITS_t;
```

Parameters

Data Type	Name	Description
IFX_uint32_t	reserved2	
IFX_uint32_t	eventDetect	RFC 2833 event playout detection.
IFX_uint32_t	fax_ced_net	Reserved.
IFX_uint32_t	fax_cng_net	Reserved.
IFX_uint32_t	gr909result	GR909 exception.
IFX_uint32_t	device	Low-level device exception. The information can be queried from the underlying driver
IFX_uint32_t	signal	Signal detected. The information is retrieved with IFX_TAPI_CH_STATUS_GET
IFX_uint32_t	otemp	Over temperature detected.
IFX_uint32_t	ground_key_polarity	Ground key polarity detected.
IFX_uint32_t	ground_key_high	Ground key high detected.
IFX_uint32_t	fault	Fault condition detected, query information via IFX_TAPI_CH_STATUS_GET .

Data Type	Name	Description
IFX_uint32_t	fax_supdate	FAX status update, query information via IFX_TAPI_CH_STATUS_GET .
IFX_uint32_t	fax_dis	FAX DIS received (no longer supported).
IFX_uint32_t	fax_ced	FAX CED received (no longer supported).
IFX_uint32_t	fax_cng	FAX CNG received (no longer supported).
IFX_uint32_t	pulse_digit_ready	Pulse dial digit is ready.
IFX_uint32_t	ground_key	Ground key detected.
IFX_uint32_t	cidrx_supdate	Caller id receiver event, query information with IFX_TAPI_CID_RX_STATUS_GET .
IFX_uint32_t	callerid_ready	Caller id data is ready (no longer supported).
IFX_uint32_t	pstn_ring	Reserved.
IFX_uint32_t	flash_hook	Flash hook detected.
IFX_uint32_t	hookstate	Hook state changed.
IFX_uint32_t	dtmf_ready	DTMF digit is ready.

6.2.4.22 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.

Data Type	Name	Description
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 0 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.

6.2.4.23 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.
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.

Data Type	Name	Description
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.

6.2.4.24 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 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

6.2.4.25 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

6.2.4.26 IFX_TAPI_LINE_VOLUME_t
Description

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)

6.2.4.27 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. 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
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_UNCHANGE D , 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_UNCHANGE D , 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

6.2.4.28 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

6.2.4.29 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

6.2.4.30 IFX_TAPI_METER_CFG_t

Description

Structure for metering configuration.

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	Metering frequency. 0 _D TAPI_METER_FREQ_DEFAULT , device default 1 _D TAPI_METER_FREQ_12_KHZ , 12 kHz 2 _D TAPI_METER_FREQ_16_KHZ , 16 kHz
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.

6.2.4.31 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.
IFX_uint16_t	nTimestamp	Start value for the timestamp (resolution of 125 µs).
IFX_uint16_t	nCpsCid	Connection Identifier for CPS events.

6.2.4.32 IFX_TAPI_PCK_AAL_PROFILE_t
Description

AAL profile setup structure used for IFX_TAPI_PKT_AAL_PROFILE_SET.

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 .

6.2.4.33 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.

6.2.4.34 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.

6.2.4.35 IFX_TAPI_PKT_RTP_CFG_t

Description

Structure for RTP Configuration.

Prototype

```
typedef struct
{
    IFX_uint16_t nSeqNr;
    IFX_uint32_t nSsrc;
    IFX_uint32_t nTimestamp;
    IFX_uint8_t nEvents;
    IFX_uint8_t nEventPT;
} IFX_TAPI_PKT_RTP_CFG_t;
```

Parameters

Data Type	Name	Description
IFX_uint16_t	nSeqNr	Start value for the sequence nr.
IFX_uint32_t	nSsrc	Synchronization source value for the voice and SID packets.
IFX_uint32_t	nTimestamp	Start value for the timestamp (resolution of 125 µs).
IFX_uint8_t	nEvents	Defines how events are transmitted. 0 _D IFX_TAPI_PKT_EV_OOB_DEFAULT , Default, implementation dependent. Not recommended. 1 _D IFX_TAPI_PKT_EV_OOB_NO , No events, only voice 2 _D IFX_TAPI_PKT_EV_OOB_ONLY , No voice, only events 3 _D IFX_TAPI_PKT_EV_OOB_ALL , Events and voice
IFX_uint8_t	nEventPT	Event payload type.

6.2.4.36 IFX_TAPI_PKT_RTP_PT_CFG_t

Description

Structure for RTP payload configuration.

Prototype

```
typedef struct
{
    IFX_uint8_t nPT[MAX_CODECS];
} IFX_TAPI_PKT_RTP_PT_CFG_t;
```

Parameters

Data Type	Name	Description
IFX_uint8_t	nPT[MAX_CODECS]	Table with all payload types, the coder type IFX_TAPI_ENC_TYPE_t is used as index.

6.2.4.37 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.

Data Type	Name	Description
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.

6.2.4.38 IFX_TAPI_RING_CFG_t

Description

Ringing configuration structure used for [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

6.2.4.39 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 .

6.2.4.40 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 < 100 [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
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.

6.2.4.41 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 < 100 [ms]
IFX_uint16_t	nGainTx	Sets the transmit gain for the downstream direction.
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.

6.2.4.42 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.

6.2.4.43 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

6.2.4.44 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. <i>Note: In order to create composed tones only simple tones with a finite loop count can be used.</i>

6.2.4.45 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

6.2.4.46 IFX_TAPI_TONE_SIMPLE_t

Description

Structure 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; $0 \leq Hz < 4000$.
IFX_uint32_t	freqB	Tone frequency B in Hz; $0 \leq Hz < 4000$.
IFX_uint32_t	freqC	Tone frequency C in Hz; $0 \leq Hz < 4000$.
IFX_uint32_t	freqD	Tone frequency D in Hz; $0 \leq Hz < 4000$.
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. $0 \leq cadence \leq 32000$. 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 a pause at the end of the cadence. The pause will be added to the last used cadence. Therefore the maximum value for pause is $32000 - \text{"last used cadence"}$ and have the granularity of 2 ms; $0 < 32000 - \text{"last used cadence"} < 32000$.

6.2.4.47 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

6.2.5 Union Reference

This chapter contains the Union reference.

Table 50 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
IFX_TAPI_EXCEPTION_t	Contains TAPI exception information.
IFX_TAPI_TONE_t	Tone descriptor.

6.2.5.1 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.

6.2.5.2 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.

6.2.5.3 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.

6.2.5.4 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.

6.2.6 Enumerator Reference

This chapter contains the Enumerator reference.

Table 51 Enumerator Overview of TAPI Interfaces

Name	Description
IFX_return_t	Error codes.
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_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	ABSLI/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.

Table 51 Enumerator Overview of TAPI Interfaces (cont'd)

Name	Description
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_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_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 line feeding.
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.
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.

Table 51 Enumerator Overview of TAPI Interfaces (cont'd)

Name	Description
IFX_TAPI_TONE_TG_t	Defines the tone generator usage.
IFX_TAPI_TONE_TYPE_t	Tone types.

6.2.6.1 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

6.2.6.2 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.

6.2.6.3 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

6.2.6.4 IFX_TAPI_CH_INIT_COUNTRY_t

Description

Country selection for [IFX_TAPI_CH_INIT_t](#).

Reserved.

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.

6.2.6.5 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

Name	Value	Description
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.

6.2.6.6 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.

6.2.6.7 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. <i>Note: Defined only CID transmission associated with ringing.</i>
IFX_TAPI_CID_ALERT_ETSI_DTAS	1 _H	DTAS.

Name	Value	Description
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.

6.2.6.8 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.

6.2.6.9 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 transmission.
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).

6.2.6.10 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.

6.2.6.11 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.

6.2.6.12 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.
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.

Name	Value	Description
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.

6.2.6.13 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.

6.2.6.14 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

6.2.6.15 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.

6.2.6.16 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.

6.2.6.17 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

6.2.6.18 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_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.
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.

6.2.6.19 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.

6.2.6.20 IFX_TAPI_JB_LOCAL_ADAPT_t

Description

Jitter buffer adaption.

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.
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 doesn't 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.

6.2.6.21 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

6.2.6.22 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.

6.2.6.23 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.

Name	Value	Description
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.

6.2.6.24 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.

6.2.6.25 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).

6.2.6.26 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.

6.2.6.27 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_NORMAL_AUTO = 6,
    IFX_TAPI_LINE_FEED_REVERSED_AUTO = 7,
    IFX_TAPI_LINE_FEED_NORMAL_LOW = 8,
    IFX_TAPI_LINE_FEED_REVERSED_LOW = 9,
    IFX_TAPI_LINE_FEED_RING_BURST = 10,
    IFX_TAPI_LINE_FEED_RING_PAUSE = 11,
    IFX_TAPI_LINE_FEED_METER = 12,
    IFX_TAPI_LINE_FEED_ACTIVE_LOW = 13,
    IFX_TAPI_LINE_FEED_ACTIVE_BOOSTED = 14,
    IFX_TAPI_LINE_FEED_ACT_TESTIN = 15,
    IFX_TAPI_LINE_FEED_DISABLED_RESISTIVE_SWITCH = 16,
    IFX_TAPI_LINE_FEED_PARKED_REVERSED = 17,
    IFX_TAPI_LINE_FEED_ACT_RES_NORMAL = 18,
    IFX_TAPI_LINE_FEED_ACT_RES_REVERSED = 19,
    IFX_TAPI_LINE_FEED_ACT_TEST = 20
}
```

```
} 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_NORMAL_AUTO	6 _D	Thresholds for automatic battery switch are set via coefficient settings.
IFX_TAPI_LINE_FEED_REVERSED_AUTO	7 _D	Thresholds for automatic battery switch are set via coefficient settings reversed.
IFX_TAPI_LINE_FEED_NORMAL_LOW	8 _D	Feeding mode for phone off hook with low battery to save power.
IFX_TAPI_LINE_FEED_REVERSED_LOW	9 _D	Feeding mode for phone off hook with low battery to save power and reserved polarity.
IFX_TAPI_LINE_FEED_RING_BURST	10 _D	Reserved, needed for ring internal function.
IFX_TAPI_LINE_FEED_RING_PAUSE	11 _D	Reserved, needed for ring internal function.
IFX_TAPI_LINE_FEED_METER	12 _D	Reserved, needed for internal function.
IFX_TAPI_LINE_FEED_ACTIVE_LOW	13 _D	Reserved, special test mode.
IFX_TAPI_LINE_FEED_ACTIVE_BOOSTED	14 _D	Reserved, special test mode.
IFX_TAPI_LINE_FEED_ACT_TESTIN	15 _D	Active Test (ACT-TI) mode, valid for SLIC-LCP and S-MAX SLIC.
IFX_TAPI_LINE_FEED_DISABLED_RESISTIVE_SWITCH	16 _D	Reserved, special mode for S-MAX Slic.
IFX_TAPI_LINE_FEED_PARKED_REVERSED	17 _D	Reserved, Power down resistance = on hook with hook detection.
IFX_TAPI_LINE_FEED_ACT_RES_NORMAL	18 _D	Active High Resistive (ACTH-R), valid for SLIC-E.
IFX_TAPI_LINE_FEED_ACT_RES_REVERSED	19 _D	Active High Resistive (ACTH-R), with reversed polarity, valid for SLIC-E.
IFX_TAPI_LINE_FEED_ACT_TEST	20 _D	Active Test (ACT-T) mode, valid for SLIC-LCP and S-MAX SLIC.

6.2.6.28 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.

6.2.6.29 IFX_TAPI_LINE_LEVEL_t

Description

Specifies whether the “high level” line output mode should be used. Enabling the “high level” mode, it will be possible to have an extremely 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.

6.2.6.30 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.

Name	Value	Description
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	GR909 results available
IFX_TAPI_LINE_STATUS_CIDRX	200 _H	Caller id received.

6.2.6.31 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_t;
```

Parameters

Name	Value	Description
IFX_TAPI_MAP_DATA_TYPE_DEFAULT	0 _D	Default.
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.

6.2.6.32 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.

6.2.6.33 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.

6.2.6.34 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_t;
```

Parameters

Name	Value	Description
IFX_TAPI_MAP_TYPE_DEFAULT	0 _D	Default. It depends on the device and configures the best applicable
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.

6.2.6.35 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.

6.2.6.36 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.

Name	Value	Description
IFX_TAPI_PCM_RES_MLAW_8BIT	1 _D	μ-law 8-bit
IFX_TAPI_PCM_RES_LINEAR_16BIT	2 _D	Linear 16-bit.

6.2.6.37 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.

6.2.6.38 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.

6.2.6.39 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_t;
```

Parameters

Name	Value	Description
IFX_TAPI_PKT_EV_OOB_DEFAULT	0 _D	Default (Transmit in band and out of band).
IFX_TAPI_PKT_EV_OOB_NO	1 _D	Transmit only out of band.
IFX_TAPI_PKT_EV_OOB_ONLY	2 _D	Transmit only in band.
IFX_TAPI_PKT_EV_OOB_ALL	3 _D	Transmit in band and out of band.

6.2.6.40 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.

6.2.6.41 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.

6.2.6.42 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.

6.2.6.43 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.

Name	Value	Description
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.
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.

6.2.6.44 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.

6.2.6.45 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.

6.2.6.46 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.

6.2.6.47 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	Call progress tone detection, RX direction.
IFX_TAPI_TONE_CPTD_DIRECTION_TX	2 _H	Call progress tone detection, TX direction.

6.2.6.48 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

6.2.6.49 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.

Name	Value	Description
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.

6.2.6.50 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.

6.2.6.51 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.

6.2.6.52 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.

6.3 Mapping to Old TAPI Names

This chapter gives the mapping of the new TAPI names to the old names. For details about selection between old and new names please refer to [Chapter 1.5](#).

6.3.1 Mapping of ioctl Interfaces

The following table gives the mapping of the new names of the ioctl commands to the old names.

Table 52 Mapping to old names of TAPI ioctl Interfaces

Old Name	New Name	Remarks
Initialization Service		
IFXPHONE_INIT	IFX_TAPI_CH_INIT	
Operation Service		
IFXPHONE_GETLECCONF	IFX_TAPI_LEC_PHONE_CFG_GET	
IFXPHONE_GETLECCONF_PCM	IFX_TAPI_LEC_PCM_CFG_GET	
IFXPHONE_HIGH_LEVEL	IFX_TAPI_LINE_LEVEL_SET	
IFXPHONE_LECCONF	IFX_TAPI_LEC_PHONE_CFG_SET	
IFXPHONE_LECCONF_PCM	IFX_TAPI_LEC_PCM_CFG_SET	
IFXPHONE_PCM_VOLUME	IFX_TAPI_PCM_VOLUME_SET	
IFXPHONE_SET_LINEFEED	IFX_TAPI_LINE_FEED_SET	
IFXPHONE_VALIDATION_TIME	IFX_TAPI_LINE_HOOK_VT_SET	
IFXPHONE_VOLUME	IFX_TAPI_PHONE_VOLUME_SET	
PHONE_HOOKSTATE	IFX_TAPI_LINE_HOOK_STATUS_GET	
Metering Service		
IFXPHONE_METER_CHAR	IFX_TAPI_METER_CFG_SET	
IFXPHONE_METER_START	IFX_TAPI_METER_START	
IFXPHONE_METER_STOP	IFX_TAPI_METER_STOP	
Tone Control Service		

Table 52 Mapping to old names of TAPI ioctl Interfaces (cont'd)

Old Name	New Name	Remarks
IFXPHONE_SET_TONE_LEVEL		Not supported anymore.
PHONE_BUSY		Not supported anymore.
PHONE_CPT_STOP	IFX_TAPI_TONE_STOP	
PHONE_DIALTONE		Not supported anymore.
PHONE_GET_TONE_OFF_TIME		Not supported anymore.
PHONE_GET_TONE_ON_TIME		Not supported anymore.
PHONE_GET_TONE_STATE	IFX_TAPI_TONE_STATUS_GET	
PHONE_PLAY_TONE	IFX_TAPI_TONE_LOCAL_PLAY	
PHONE_RINGBACK		Not supported anymore.
PHONE_SET_TONE_OFF_TIME		Not supported anymore.
PHONE_SET_TONE_ON_TIME		Not supported anymore.
Dial Service		
IFXPHONE_GET_PULSE	IFX_TAPI_PULSE_GET	
IFXPHONE_GET_PULSE_ASCII	IFX_TAPI_PULSE_ASCII_GET	
IFXPHONE_PULSE_READY	IFX_TAPI_PULSE_READY	
PHONE_DTMF_OOB	IFX_TAPI_PKT_EV_OOB_SET	
PHONE_DTMF_READY	IFX_TAPI_TONE_DTMF_READY_GET	
PHONE_GET_DTMF	IFX_TAPI_TONE_DTMF_GET	
PHONE_GET_DTMF_ASCII	IFX_TAPI_TONE_DTMF_ASCII_GET	
Signal Detection Service		
IFXPHONE_CPTD_START	IFX_TAPI_TONE_CPTD_START	
IFXPHONE_CPTD_STOP	IFX_TAPI_TONE_CPTD_STOP	
IFXPHONE_DISABLE_SIGDETECT	IFX_TAPI_SIG_DETECT_DISABLE	
IFXPHONE_ENABLE_SIGDETECT	IFX_TAPI_SIG_DETECT_ENABLE	
CID Features		
IFXPHONE_CIDCONF	IFX_TAPI_CID_CFG_SET	Important functional change.
IFXPHONE_CID2_CONF		Not supported anymore.
IFXPHONE_CID2_GETCONF		Not supported anymore.
IFXPHONE_CID2_SEND		Not supported anymore.
IFXPHONE_CID_INFO_TX	IFX_TAPI_CID_TX_INFO_START	
IFXPHONE_CID_SEQ_TX	IFX_TAPI_CID_TX_SEQ_START	
IFXPHONE_CIDRX_DATA	IFX_TAPI_CID_RX_DATA_GET	
IFXPHONE_CIDRX_START	IFX_TAPI_CID_RX_START	
IFXPHONE_CIDRX_STATUS	IFX_TAPI_CID_RX_STATUS_GET	
IFXPHONE_CIDRX_STOP	IFX_TAPI_CID_RX_STOP	
IFXPHONE_DTMFCID_CONF		Not supported anymore.
IFXPHONE_GETCIDCONF		Not supported anymore.
Connection Control Service		
IFXPHONE_AAL_CONF	IFX_TAPI_PKT_AAL_CFG_SET	
IFXPHONE_AAL_PROFILE	IFX_TAPI_PKT_AAL_PROFILE_SET	
IFXPHONE_DATA_ADD	IFX_TAPI_MAP_DATA_ADD	

Table 52 Mapping to old names of TAPI ioctl Interfaces (cont'd)

Old Name	New Name	Remarks
IFXPHONE_DATA_REMOVE	IFX_TAPI_MAP_DATA_REMOVE	
IFXPHONE_GET_FRAME	IFX_TAPI_ENC_FRAME_LEN_GET	
IFXPHONE_JB_CONF	IFX_TAPI_JB_CFG_SET	
IFXPHONE_JB_RESET	IFX_TAPI_JB_STATISTICS_RESET	
IFXPHONE_JB_STATISTICS	IFX_TAPI_JB_STATISTICS_GET	
IFXPHONE_PCM_ADD	IFX_TAPI_MAP_PCM_ADD	
IFXPHONE_PCM_REMOVE	IFX_TAPI_MAP_PCM_REMOVE	
IFXPHONE_PHONE_ADD	IFX_TAPI_MAP_PHONE_ADD	
IFXPHONE_PHONE_REMOVE	IFX_TAPI_MAP_PHONE_REMOVE	
IFXPHONE_RTP_CONF	IFX_TAPI_PKT_RTP_CFG_SET	
IFXPHONE_RTP_PTCONF	IFX_TAPI_PKT_RTP_PT_CFG_SET	
IFXPHONE_RTCP_RESET	IFX_TAPI_PKT_RTCP_STATISTICS_RESET	
IFXPHONE_RTCP_STATISTICS	IFX_TAPI_PKT_RTCP_STATISTICS_GET	
PHONE_FRAME	IFX_TAPI_ENC_FRAME_LEN_SET	
PHONE_PLAY_CODEC	IFX_TAPI_DEC_TYPE_SET	
PHONE_PLAY_LEVEL	IFX_TAPI_DEC_LEVEL_GET	
PHONE_PLAY_START	IFX_TAPI_DEC_START	
PHONE_PLAY_STOP	IFX_TAPI_DEC_STOP	
PHONE_PLAY_VOLUME	IFX_TAPI_DEC_VOLUME_SET	
PHONE_REC_CODEC	IFX_TAPI_ENC_TYPE_SET	
PHONE_REC_LEVEL	IFX_TAPI_ENC_LEVEL_SET	
PHONE_REC_START	IFX_TAPI_ENC_START	
PHONE_REC_STOP	IFX_TAPI_ENC_STOP	
PHONE_REC_VOLUME	IFX_TAPI_ENC_VOLUME_SET	
PHONE_VAD	IFX_TAPI_ENC_VAD_CFG_SET	
Miscellaneous Service		
IFXPHONE_GET_VERSION	IFX_TAPI_VERSION_GET	
IFXPHONE_MASK_EXCEPTION	IFX_TAPI_EXCEPTION_MASK	
IFXPHONE_PHONE_STATUS	IFX_TAPI_CH_STATUS_GET	
IFXPHONE_REPORT_SET	IFX_TAPI_DEBUG_REPORT_SET	
IFXPHONE_VERSION_CHECK	IFX_TAPI_VERSION_CHECK	
PHONE_CAPABILITIES	IFX_TAPI_CAP_NR	
PHONE_CAPABILITIES_CHECK	IFX_TAPI_CAP_CHECK	
PHONE_CAPABILITIES_LIST	IFX_TAPI_CAP_LIST	
PHONE_EXCEPTION	IFX_TAPI_EXCEPTION_GET	
Ringing Service		
IFXPHONE_RING_CADENCE_HIGH_RES	IFX_TAPI_RING_CADENCE_HR_SET	
IFXPHONE_RING_CONFIG	IFX_TAPI_RING_CFG_SET	

Table 52 Mapping to old names of TAPI ioctl Interfaces (cont'd)

Old Name	New Name	Remarks
IFXPHONE_RING_GET_CONFIG	IFX_TAPI_RING_CFG_GET	
PHONE_MAXRINGS		Not supported anymore.
PHONE_RING		Not supported anymore.
PHONE_RING_CADENCE	IFX_TAPI_RING_CADENCE_SET	
PHONE_RING_START	IFX_TAPI_RING_START	
PHONE_RING_STOP	IFX_TAPI_RING_STOP	
PCM Service		
IFXPHONE_GET_PCM_ACTIVATION	IFX_TAPI_PCM_ACTIVATION_GET	
IFXPHONE_SET_PCM_ACTIVATION	IFX_TAPI_PCM_ACTIVATION_SET	
IFXPHONE_GET_PCM_CONFIG	IFX_TAPI_PCM_CFG_GET	
IFXPHONE_SET_PCM_CONFIG	IFX_TAPI_PCM_CFG_SET	
Fax T.38 Service		
IFXPHONE_FAX_SETDEMODO	IFX_TAPI_T38_DEMOD_START	
IFXPHONE_FAX_SETMOD	IFX_TAPI_T38_MOD_START	
IFXPHONE_FAX_STATUS	IFX_TAPI_T38_STATUS_GET	
IFXPHONE_FAX_DISABLE	IFX_TAPI_T38_STOP	

6.3.2 Mapping of Constants

The following table gives the mapping of the new names of TAPI constants to the old names.

Table 53 Mapping to old names of TAPI Constants

Old Name	New Name	Remarks
MAX_CODECS	IFX_TAPI_ENC_TYPE_MAX	
TAPI_CID_MAX_MSG_LEN	IFX_TAPI_CID_MSG_LEN_MAX	
TAPI_CID_RX_FIFO_SIZE	IFX_TAPI_CID_RX_FIFO_SIZE	
TAPI_DIAL_FIFO_SIZE	IFX_TAPI_PULSE_FIFO_SIZE	
TAPI_DTMF_FIFO_SIZE	IFX_TAPI_DTMF_FIFO_SIZE	
TAPI_IOC_MAGIC	IFX_TAPI_IOC_MAGIC	
TAPI_LEC_MAX_LEN	IFX_TAPI_LEC_LEN_MAX	
TAPI_LEC_MIN_LEN	IFX_TAPI_LEC_LEN_MIN	
TAPI_MAX_CADENCE_BYTES	IFX_TAPI_RING_CADENCE_MAX	
TAPI_MAX_CID_PARAMETER	IFX_TAPI_CID_SIZE_MAX	
TAPI_MAX_SIMPLE_TONES	IFX_TAPI_TONE_SIMPLE_MAX	
TAPI_PHONE_VOLCONF_HIGH	IFX_TAPI_LINE_VOLUME_HIGH	
TAPI_PHONE_VOLCONF_LOW	IFX_TAPI_LINE_VOLUME_LOW	
TAPI_PHONE_VOLCONF_MEDIUM	IFX_TAPI_LINE_VOLUME_MEDIUM	
TAPI_PHONE_VOLCONF_OFF	IFX_TAPI_LINE_VOLUME_OFF	
TAPI_TONE_MAX_INDEX	IFX_TAPI_TONE_INDEX_MAX	
TAPI_TONE_MAX_STEPS	IFX_TAPI_TONE_STEPS_MAX	
TAPI_TONE_MIN_INDEX	IFX_TAPI_TONE_INDEX_MIN	
TAPI_TONE_SRC_DEFAULT	IFX_TAPI_TONE_SRC_DEFAULT	

Table 53 Mapping to old names of TAPI Constants (cont'd)

Old Name	New Name	Remarks
TAPI_TONE_SRC_DSP	IFX_TAPI_TONE_SRC_DSP	
TAPI_TONE_SRC_TG	IFX_TAPI_TONE_SRC_TG	

6.3.3 Mapping of Structures

The following table gives the mapping of the new names of TAPI structures to the old names.

Table 54 Mapping to old names of TAPI Structures

Old Name	New Name	Remarks
EXCEPTION_BITS	IFX_TAPI_EXCEPTION_BITS_t	
TAPI_AAL_CONF	IFX_TAPI_PCK_AAL_CFG_t	
TAPI_AAL_PROFILE	IFX_TAPI_PCK_AAL_PROFILE_t	
TAPI_CID_CONF_t	IFX_TAPI_CID_CFG_t	
TAPI_CID_CONFIG	IFX_TAPI_CID_CFG_t	
TAPI_CID_DTMF_CONF	IFX_TAPI_CID_DTMF_CFG_t	
TAPI_CID_FSK_CONF	IFX_TAPI_CID_FSK_CFG_t	
TAPI_CID_INFO	IFX_TAPI_CID_MSG_t	
TAPI_CID_MSG_DATE	IFX_TAPI_CID_MSG_DATE_t	
TAPI_CID_MSG_STRING	IFX_TAPI_CID_MSG_STRING_t	
TAPI_CID_TIMING	IFX_TAPI_CID_TIMING_t	
TAPI_CID2_CONFIG		Not supported anymore.
TAPI_CID2_DEFAULT_CONFIG		Not supported anymore.
TAPI_CID2_NTT_CONFIG		Not supported anymore.
TAPI_CIDRX_DATA	IFX_TAPI_CID_RX_DATA_t	
TAPI_CIDRX_STATUS	IFX_TAPI_CID_RX_STATUS_t	
TAPI_COMPOSED_TONE	IFX_TAPI_TONE_COMPOSED_t	
TAPI_DATA_MAPPING	IFX_TAPI_MAP_DATA_t	
TAPI_DTMFCID		Not supported anymore.
TAPI_FAX_DEMODDATA	IFX_TAPI_T38_DEMOD_DATA_t	
TAPI_FAX_MODDATA	IFX_TAPI_T38_MOD_DATA_t	
TAPI_FAX_STATUS	IFX_TAPI_T38_STATUS_t	
TAPI_INIT	IFX_TAPI_CH_INIT_t	
TAPI_JB_CONF	IFX_TAPI_JB_CFG_t	
TAPI_JB_DATA	IFX_TAPI_JB_STATISTICS_t	
TAPI_LECCONF	IFX_TAPI_LEC_CFG_t	
TAPI_METER_CONFIG	IFX_TAPI_METER_CFG_t	
TAPI_PCM_CONFIG	IFX_TAPI_PCM_CFG_t	
TAPI_PCM_MAPPING	IFX_TAPI_MAP_PCM_t	
TAPI_PHONE_CAPABILITY	IFX_TAPI_CAP_t	
TAPI_PHONE_CID	IFX_TAPI_CID_MSG_t	
TAPI_PHONE_CPTD	IFX_TAPI_TONE_CPTD_t	
TAPI_PHONE_MAPPING	IFX_TAPI_MAP_PHONE_t	

Table 54 Mapping to old names of TAPI Structures (cont'd)

Old Name	New Name	Remarks
TAPI_PHONE_SIGDETECTION	IFX_TAPI_SIG_DETECTION_t	
TAPI_PHONE_STATUS	IFX_TAPI_CH_STATUS_t	
TAPI_PHONE_VERSION	IFX_TAPI_VERSION_t	
TAPI_PHONE_VOLUME	IFX_TAPI_LINE_VOLUME_t	
TAPI_RING_CADENCE	IFX_TAPI_RING_CADENCE_t	
TAPI_RING_CONFIG	IFX_TAPI_RING_CFG_t	
TAPI_RTCP_DATA	IFX_TAPI_PKT_RTCP_STATISTICS_t	
TAPI_RTP_CONF	IFX_TAPI_PKT_RTP_CFG_t	
TAPI_RTP_PT_CONF	IFX_TAPI_PKT_RTP_PT_CFG_t	
TAPI_SIMPLE_TONE	IFX_TAPI_TONE_SIMPLE_t	
TAPI_TONE_COMPOSED	IFX_TAPI_TONE_COMPOSED_t	
TAPI_TONE_SIMPLE	IFX_TAPI_TONE_SIMPLE_t	
TAPI_VALIDATION_TIMES	IFX_TAPI_LINE_HOOK_VT_t	

6.3.4 Mapping of Unions

The following table gives the mapping of the new names of TAPI Unions to the old names.

Table 55 Mapping to old names of TAPI Unions

Old Name	New Name	Remarks
TAPI_CID2_STD_CONFIG		Not supported anymore.
TAPI_EXCEPTION	IFX_TAPI_EXCEPTION_t	
TAPI_TONE	IFX_TAPI_TONE_t	

6.3.5 Mapping of Enumerations

Table 56 Mapping to old names of TAPI Enumerations

Old Name	New Name	Remarks
errorCode	IFX_return_t	
PCM_CONFIG_RES	IFX_TAPI_PCM_RES_t	
RING_CONFIG_MODE	IFX_TAPI_RING_CFG_MODE_t	
RING_CONFIG_SUBMODE	IFX_TAPI_RING_CFG_SUBMODE_t	
TAPI_AAL_PROFILE_RANGE	IFX_TAPI_PKT_AAL_PROFILE_RANGE_t	
TAPI_CID_ABSREASON	IFX_TAPI_CID_ABSREASON_t	
TAPI_CID_APPEARANCE		Not supported anymore.
TAPI_CID_ETSI_ALERT	IFX_TAPI_CID_ALERT_ETSI_t	
TAPI_CID_HOOK_MODE	IFX_TAPI_CID_HOOK_MODE_t	
TAPI_CID_LEVEL		
TAPI_CID_MESSAGE_TYPE	IFX_TAPI_CID_MSG_TYPE_t	
TAPI_CID_MODE		
TAPI_CID_SERVICE_TYPE	IFX_TAPI_CID_SERVICE_TYPE_t	

Table 56 Mapping to old names of TAPI Enumerations (cont'd)

Old Name	New Name	Remarks
TAPI_CID_SPEC		
TAPI_CID_STD	IFX_TAPI_CID_STD_t	
TAPI_CID_VMWI	IFX_TAPI_CID_VMWI_t	
TAPI_CID2_STD		
TAPI_CIDRX_ERR	IFX_TAPI_CID_RX_ERROR_t	
TAPI_CIDRX_STAT	IFX_TAPI_CID_RX_STATE_t	
TAPI_COUNTRY	IFX_TAPI_CH_INIT_COUNTRY_t	
TAPI_CPT_SIGNAL	IFX_TAPI_TONE_CPTD_DIRECTION_t	
TAPI_DATA_MAPPING_START_STOP	IFX_TAPI_DATA_MAP_START_STOP_t	
TAPI_DATA_MAPTYPE	IFX_TAPI_MAP_DATA_TYPE_t	
TAPI_DIALING	IFX_TAPI_DIALING_STATUS_t	
TAPI_FAX_T38_ERROR	IFX_TAPI_T38_ERROR_t	
TAPI_FAX_T38_STATUS	IFX_TAPI_T38_STATUS_t	
TAPI_FAX_T38_STD	IFX_TAPI_T38_STD_t	
TAPI_HIGH_LEVEL	IFX_TAPI_LINE_LEVEL_t	
TAPI_HOOKSTATUS	IFX_TAPI_LINE_HOOK_STATUS_t	
TAPI_INIT_MODES	IFX_TAPI_CH_INIT_MODE_t	
TAPI_LEC_GAIN	IFX_TAPI_LEC_GAIN_t	
TAPI_LINEFEED	IFX_TAPI_LINE_FEED_t	
TAPI_LINESTATUS	IFX_TAPI_LINE_STATUS_t	
TAPI_LOCAL_ADAPT	IFX_TAPI_JB_LOCAL_ADAPT_t	
TAPI_MAPPLAY	IFX_TAPI_MAP_DEC_t	
TAPI_MAPREC	IFX_TAPI_MAP_ENC_t	
TAPI_MAPTYPE	IFX_TAPI_MAP_TYPE_t	
TAPI_METER_FREQ		Not supported anymore.
TAPI_METER_MODE		Not supported anymore.
TAPI_OOB	IFX_TAPI_PKT_EV_OOB_t	
TAPI_PHONE_CAP	IFX_TAPI_CAP_TYPE_t	
TAPI_PHONE_CAP_SIGDETECT	IFX_TAPI_CAP_SIG_DETECT_t	
TAPI_PHONE_CODEC	IFX_TAPI_ENC_TYPE_t	
TAPI_PHONE_PORTS	IFX_TAPI_CAP_PORT_t	
TAPI_RTPEV		Not supported anymore, use IFX_TAPI_PKT_EV_OOB_t
TAPI_SIGNAL	IFX_TAPI_SIG_t	
TAPI_TONE_FREQ	IFX_TAPI_TONE_FREQ_t	
TAPI_TONE_MODULATION	IFX_TAPI_TONE_MODULATION_t	
TAPI_TONE_TG	IFX_TAPI_TONE_TG_t	
TAPI_TONE_TYPES	IFX_TAPI_TONE_TYPE_t	
TAPI_VAD	IFX_TAPI_ENC_VAD_t	

Table 56 Mapping to old names of TAPI Enumerations (cont'd)

Old Name	New Name	Remarks
TAPI_VALIDATION_TYPES	IFX_TAPI_LINE_HOOK_VALIDATION_T YPE_t	
TONE_GROUP	IFX_TAPI_TONE_GROUP_t	

This section describes the mapping of new names of enumerations to the old names.

7 Device Driver Interfaces

This section describes device specific interfaces, also called device driver interfaces.

7.1 ioctl commands of the Device Driver Interface

This chapter describes all device driver interfaces. The ioctl commands are explained by mentioning the return values for each function. The organization is as follows:

Table 57 Device Driver Interface Overview

Name	Description
Basic Interface	Basic Interface.
Driver Initialization Interface	Driver Initialization Interface.
Driver Kernel Interfaces	Kernel Interface for GPIO/IO pin handling.
GPIO Interface	Control the device and channel specific IO pins.
Line Testing Interface	Interfaces for low-level line testing functions.
Polling Interface	Polling Interface.

7.1.1 Basic Interface

Basic VINETIC® access routines as command read and write and initialization.

Table 58 IO-control Overview of Basic Interface

Name	Description
FIO_VINETIC_DOW_PHI	PHI download using structure VINETIC_IO_INIT .
FIO_VINETIC_DOWNFPI	FPI download, only for internal use.
FIO_VINETIC_DOWNLOAD_CRAM	Download cram coefficients to one or all VINETIC® channels.
FIO_VINETIC_DRV_CTRL	Driver configuration.
FIO_VINETIC_INIT	VINETIC® initialization.
FIO_VINETIC_LASTERR	Get the last occurred error.
FIO_VINETIC_RCMD	Read command.
FIO_VINETIC_REPORT_SET	Set the driver report levels if the driver is compiled with ENABLE_TRACE .
FIO_VINETIC_RSC	Read short command.
FIO_VINETIC_RUNTIME_TRACE_SET	Driver runtime tracing of register accesses.
FIO_VINETIC_VERS	Read relevant version information.
FIO_VINETIC_WCMD	Write a VINETIC® command.
FIO_VINETIC_WSC	Write short command.

Table 59 Constant Overview of Basic Interface

Name	Description
DC_DWLD	Flag for VINETIC_IO_INIT to perform a DC control download in case of V1.4.
FW_AUTODWLD	Flag for VINETIC_IO_INIT to send the auto download command in case of V1.4.
NO_AC_DWLD	Flag for VINETIC_IO_INIT to avoid AC download in case of V1.4.
NO_BCONF	Flag for VINETIC_IO_INIT to avoid no board configuration.
NO_CRAM_DWLD	Flag for VINETIC_IO_INIT to avoid CRAM download in case of V1.4.
NO_EDSP_START	Flag for VINETIC_IO_INIT to avoid EDSP start.
NO_FW_DWLD	Flag for VINETIC_IO_INIT to avoid firmware download.
NO_PHI_DWLD	Flag for VINETIC_IO_INIT to avoid PHI download in case of V1.4.
VINETIC_IO_CRAM_DATA_SIZE	Maximum size of CRAM download buffer.

Table 60 Structure Overview of Basic Interface

Name	Description
VINETIC_IO_CRAM	Structure for CRAM download with the V1.4 and the V2.x format.
VINETIC_IO_CRAM_REG_CFG	CRAM download related setting of register values (used for activation).
VINETIC_IO_DRVCTRL	Driver global configuration structure.

Table 61 Enumerator Overview of Basic Interface

Name	Description
VINETIC_IO_CRAM_FORMAT	CRAM file format version.

7.1.1.1 FIO_VINETIC_DOW_PHI

Description

PHI download using structure VINETIC_IO_PHI.

Prototype

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

Parameters

Data Type	Name	Description
IFX_int32_t	fd	Pointer to a file descriptor.
IFX_int32_t	FIO_VINETIC_DOW_PHI	I/O control identifier for this operation.
IFX_int32_t	param	

Return Values

Data Type	Description
IFX_void_t	No return value

7.1.1.2 FIO_VINETIC_DOWNFPI

Description

FPI download, only for internal use.

Prototype

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

Parameters

Data Type	Name	Description
IFX_int32_t	fd	Pointer to a file descriptor.
IFX_int32_t	FIO_VINETIC_DOWNFPI	I/O control identifier for this operation.
IFX_int32_t	param	

Return Values

Data Type	Description
IFX_void_t	No return value.

7.1.1.3 FIO_VINETIC_DOWNLOAD_CRAM
Description

Download cram coefficients to one or all VINETIC® channels.

Prototype

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

Parameters

Data Type	Name	Description
IFX_int32_t	fd	Pointer to a file descriptor.
IFX_int32_t	FIO_VINETIC_DOWNLOAD_CRAM	I/O control identifier for this operation.
IFX_int32_t	param	The parameter is a pointer to a VINETIC_IO_CRAM structure.

Return Values

Data Type	Description
IFX_void_t	No return value.

Example

```
VINETIC\_IO\_CRAM ioCram;
unsigned int nCh;
int err = IFX_SUCCESS;

memset(&ioCram, 0, sizeof (VINETIC\_IO\_CRAM));

ioCram.bBroadcast = 1;
ioCram.nFormat = ;
ioCram.nStartAddr = VINETIC_COP_START_ADDRESS;
ioCram.nLength = sizeof (VINETIC_COEFF) / 2;
ioCram.nCRC = VINETIC_CRAM_CRC;
ioCram.bcr1.nData = VINETIC_COEFF_BCR[0][0];
ioCram.bcr1.nMask = VINETIC_COEFF_BCR[0][1];
ioCram.bcr2.nData = VINETIC_COEFF_BCR[1][0];
ioCram.bcr2.nMask = VINETIC_COEFF_BCR[1][1];
ioCram.tstr2.nData = VINETIC_COEFF_TSTR2[0][0];
ioCram.tstr2.nMask = VINETIC_COEFF_TSTR2[0][1];
```

```
memcpy (ioCram.aData[VINETIC_IO_CRAM_DATA_SIZE, VINETIC_COEFF, sizeof
(VINETIC_COEFF));
err = ioctl (fd, FIO_VINETIC_DOWNLOAD_CRAM, (INT) &ioCram);
```

7.1.1.4 FIO_VINETIC_DRV_CTRL

Description

Driver configuration.

Prototype

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

Parameters

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

Return Values

Data Type	Description
IFX_void_t	No return value.

7.1.1.5 FIO_VINETIC_INIT

Description

VINETIC® Initialization.

Prototype

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

Parameters

Data Type	Name	Description
IFX_int32_t	fd	File descriptor.

Data Type	Name	Description
IFX_int32_t	FIO_VINETIC_INIT	I/O control identifier for this operation
IFX_int32_t	param	If the driver is compiled with TAPI support the interface IFX_TAPI_CH_INIT should be used. This interface expects a pointer to a VINETIC_IO_INIT structure.

Return Values

Data Type	Description
IFX_void_t	No return value.

7.1.1.6 FIO_VINETIC_LASTERR
Description

Get the last occurred error.

Prototype

```
IFX_void_t ioctl (
    IFX_int32_t fd,
    FIO_VINETIC_LASTERR,
    IFX_int32_t *param );
```

Parameters

Data Type	Name	Description
IFX_int32_t	fd	File descriptor.
IFX_int32_t	FIO_VINETIC_LASTERR	I/O control identifier for this operation.
IFX_int32_t	*param	The error codes are enumerated in DEV_ERR .

Return Values

Data Type	Description
IFX_void_t	No return value.

Example

```
IFX_int32_t fd_dev;
IFX_int32_t lasterr;
IFX_return_t ret;

/* Open control file descriptor */
fd_dev = open("/dev/vin10", O_RDWR, 0x644);

ret = ioctl(fd_dev, FIO\_VINETIC\_LASTERR, (IFX_int32_t) &lasterr);
printf("Last error = 0x%08x (%d), %d\n", lasterr, lasterr, ret);

/* Close all open fds */
```

```
close(fd_dev);
```

7.1.1.7 FIO_VINETIC_RCMD

Description

Read command.

Prototype

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

Parameters

Data Type	Name	Description
IFX_int32_t	fd	File descriptor.
IFX_int32_t	FIO_VINETIC_RCMD	I/O control identifier for this operation.
IFX_int32_t	param	The parameter points to a VINETIC_IO_MB_CMD structure. This interface is only for debugging and testing purposes. The use of this interface may disturb the driver operation, as it provides full chip access.

Return Values

Data Type	Description
IFX_void_t	No return value.

Example

```
VINETIC\_IO\_MB\_CMD ioCmd;
IFX_int32_t err;
IFX_int32_t fd_dev;
IFX_int32_t ch = 0;

/* open control file descriptor */
fd_dev = open("/dev/vin10", O_RDWR, 0x644);

ioCmd.cmd1 = 0x8300 | ch;
/* Read OPMODE_CUR */
ioCmd.cmd2 = 0x2101;
err = ioctl(fd_dev, FIO\_VINETIC\_RCMD, (IFX_int32_t) &ioCmd);
printf("VINETIC cmd is: cmd1 %d, cmd2 %d\n", ioCmd.cmd1, ioCmd.cmd2);

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


7.1.1.8 FIO_VINETIC_REPORT_SET

Description

Set the driver report levels if the driver is compiled with `ENABLE_TRACE`.

Prototype

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

Parameters

Data Type	Name	Description
IFX_int32_t	fd	Pointer to a file descriptor
IFX_int32_t	FIO_VINETIC_REPORT_SET	I/O control identifier for this operation.
IFX_int32_t	param	Valid arguments are 0: off 1: low, high output 2: normal, general information and warnings 3: high, only errors are reported.

Return Values

Data Type	Description
IFX_void_t	No return value.

Remarks

Valid arguments are

0: off

1: low, high output

2: normal, general information and warnings

3: high, only errors are reported

7.1.1.9 FIO_VINETIC_RSC

Description

Read short command.

Prototype

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

Parameters

Data Type	Name	Description
IFX_int32_t	fd	Pointer to a file descriptor.
IFX_int32_t	FIO_VINETIC_RSC	I/O control identifier for this operation.
IFX_int32_t	param	The parameter points to a VINETIC_IO_READ_SC structure. This interface is only for debugging and testing purposes. The use of this interface may disturb the driver operation, as it provides full chip access.

Return Values

Data Type	Description
IFX_void_t	No return value.

Example

```

VINETIC_IO_READ_SC ioCmd;
memset (&ioCmd, 0,  sizeof (ioCmd));
ioCmd.nCmd = 0xC100;
ret = ioctl (fd, FIO_VINETIC_RSC, (INT) &ioCmd);
if (ioCmd.pData[256][0] == 0x1FFF) return 1;

```

7.1.1.10 FIO_VINETIC_RUNTIME_TRACE_SET
Description

Driver runtime tracing of register accesses.

Prototype

```

IFX_void_t ioctl (
    IFX_int32_t fd,
    FIO_VINETIC_RUNTIME_TRACE_SET,
    IFX_int32_t param );

```

Parameters

Data Type	Name	Description
IFX_int32_t	fd	Pointer to a file descriptor.
IFX_int32_t	FIO_VINETIC_RUNTIME_TRACE_SET	I/O control identifier for this operation.
IFX_int32_t	param	

Return Values

Data Type	Description
IFX_void_t	No return value.

7.1.1.11 FIO_VINETIC_VERS

Description

Read relevant version information.

Prototype

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

Parameters

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

Return Values

Data Type	Description
IFX_void_t	No return value.

7.1.1.12 FIO_VINETIC_WCMD

Description

Write a VINETIC® command.

Prototype

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

Parameters

Data Type	Name	Description
IFX_int32_t	fd	File descriptor.
IFX_int32_t	FIO_VINETIC_WCMD	I/O control identifier for this operation.
IFX_int32_t	param	The parameter points to a VINETIC_IO_MB_CMD structure. This interface is only for debugging and testing purposes. The use of this interface may disturb the driver operation, as it provides full chip access.

Return Values

Data Type	Description
IFX_void_t	No return value.

7.1.1.13 FIO_VINETIC_WSC

Description

Write short command.

Prototype

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

Parameters

Data Type	Name	Description
IFX_int32_t	fd	Pointer to a file descriptor.
IFX_int32_t	FIO_VINETIC_WSC	I/O control identifier for this operation.
IFX_int32_t	param	The parameter points to a VINETIC_IO_WRITE_SC structure This interface is only for debugging and testing purposes. The use of this interface may disturb the driver operation as it provides full chip access.

Return Values

Data Type	Description
IFX_void_t	No return value.

7.1.2 Driver Initialization Interface

Interfaces for the Driver Initialization.

Table 62 IO-control Overview of Driver Initialization Interface

Name	Description
FIO_VINETIC_BASICDEV_INIT	Initialize VINETIC® Device driver information for one chip.
FIO_VINETIC_DEV_RESET	Reset VINETIC® Device driver internal structure for one chip.

Table 63 Structure Reference of Driver Initialization Interface

Name	Description
VINETIC_IO_INIT	Structure used for device initialization

Table 64 Enumerator Overview of Driver Initialization Interface

Name	Description
VIN_ACCESS	VINETIC® Access Modes.

7.1.2.1 FIO_VINETIC_BASICDEV_INIT

Description

As the driver doesn't support board dependent implementations anymore, this interface is to call at very first to initialize the VINETIC® driver with the chip parameters passed the pointer to the structure [VINETIC_BasicDeviceInit_t](#). No chip access will be done until this basic initialization is successful.

Prototype

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

Parameters

Data Type	Name	Description
IFX_int32_t	fd	File descriptor.
IFX_int32_t	FIO_VINETIC_BASICDEV_INIT	I/O control identifier for this operation.
IFX_int32_t	param	Use structure VINETIC_BasicDeviceInit_t .

Return Values

Data Type	Description
IFX_void_t	No return value.

7.1.2.2 FIO_VINETIC_DEV_RESET

Description

Reset VINETIC® Device driver internal structure for one chip.

Prototype

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

Parameters

Data Type	Name	Description
IFX_int32_t	fd	File descriptor.
IFX_int32_t	FIO_VINETIC_DEV_RESET	I/O control identifier for this operation.
IFX_int32_t	param	

Return Values

Data Type	Description
IFX_void_t	No return value.

7.1.3 Driver Kernel Interfaces

Kernel Interface for GPIO/IO pin handling.

Table 65 Enumerator Overview of Driver Kernel Interface

Name	Description
VINETIC_GPIO_MODE	GPIO pin configuration modes.

Table 66 Function Overview of Driver Kernel Interface

Name	Description
VINETIC_GpioConfig	Configure a VINETIC® IO or GPIO pin.
VINETIC_GpioGet	Read the value from a VINETIC® IO or GPIO pin.
VINETIC_GpioIntMask	Set the interrupt enable mask.
VINETIC_GpioRelease	Release a VINETIC® IO or GPIO pin resource.
VINETIC_GpioReserve	Reserve a VINETIC® IO or GPIO pin resource.
VINETIC_GpioSet	Set the value of a VINETIC® IO or GPIO pin.
VINETIC_OpenKernel	Open the device from Kernel mode.
VINETIC_ReleaseKernel	Release a VINETIC® IO or GPIO pin resource.

7.1.4 GPIO Interface

Control the device and channel specific IO pins.

Table 67 IO-control Overview of GPIO Interface

Name	Description
FIO_VINETIC_DEV_GPIO_CFG	Configure device GPIO pins 0.
FIO_VINETIC_DEV_GPIO_SET	Set GPIO pins values.
FIO_VINETIC_GPIO_RESERVE	Reserve GPIO pins for use.
FIO_VINETIC_GPIO_CONFIG	Configure GPIO pins.
FIO_VINETIC_GPIO_SET	Set GPIO pin values.
FIO_VINETIC_GPIO_GET	Get GPIO pin values.
FIO_VINETIC_GPIO_RELEASE	Set GPIO pin values.

Table 68 Structure Overview of GPIO Interface

Name	Description
VINETIC_GPIO_CONFIG	GPIO Configuration Structure.
VINETIC_IO_GPIO_CONTROL	GPIO Control Structure.

7.1.4.1 FIO_VINETIC_DEV_GPIO_CFG

Description

Configure device GPIO pins.

Prototype

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

Parameters

Data Type	Name	Description
IFX_int32_t	fd	File descriptor.
IFX_int32_t	FIO_VINETIC_DEV_GPIO_CFG	I/O control identifier for this operation.
IFX_int32_t	param	

Return Values

Data Type	Description
IFX_void_t	No return value.

7.1.4.2 FIO_VINETIC_DEV_GPIO_SET

Description

Set GPIO pins values.

Prototype

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

Parameters

Data Type	Name	Description
IFX_int32_t	fd	File descriptor.
IFX_int32_t	FIO_VINETIC_DEV_GPIO_SET	I/O control identifier for this operation.
IFX_int32_t	param	

Return Values

Data Type	Description
IFX_void_t	No return value.

7.1.4.3 FIO_VINETIC_GPIO_RESERVE

Description

Reserve GPIO pins.

Prototype

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

Parameters

Data Type	Name	Description
IFX_int32_t	fd	File descriptor.
IFX_int32_t	FIO_VINETIC_GPIO_RESERVE	I/O control identifier for this operation.
IFX_int32_t	param	Use structure VINETIC_IO_GPIO_CONTROL .

Return Values

Data Type	Description
IFX_void_t	No return value.

7.1.4.4 FIO_VINETIC_GPIO_CONFIG
Description

Configure GPIO pins.

Prototype

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

Parameters

Data Type	Name	Description
IFX_int32_t	fd	File descriptor.
IFX_int32_t	FIO_VINETIC_GPIO_CONFIG	I/O control identifier for this operation.
IFX_int32_t	param	Use structure VINETIC_IO_GPIO_CONTROL .

Return Values

Data Type	Description
IFX_void_t	No return value.

7.1.4.5 FIO_VINETIC_GPIO_SET
Description

Set GPIO pin values.

Prototype

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

Parameters

Data Type	Name	Description
IFX_int32_t	fd	File descriptor.
IFX_int32_t	FIO_VINETIC_GPIO_SET	I/O control identifier for this operation.
IFX_int32_t	param	Use structure VINETIC_IO_GPIO_CONTROL .

Return Values

Data Type	Description
IFX_void_t	No return value.

7.1.4.6 FIO_VINETIC_GPIO_GET

Description

Get GPIO pin values.

Prototype

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

Parameters

Data Type	Name	Description
IFX_int32_t	fd	File descriptor.
IFX_int32_t	FIO_VINETIC_GPIO_GET	I/O control identifier for this operation.
IFX_int32_t	param	Use structure VINETIC_IO_GPIO_CONTROL .

Return Values

Data Type	Description
IFX_void_t	No return value.

7.1.4.7 FIO_VINETIC_GPIO_RELEASE

Description

Set GPIO pin values.

Prototype

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

Parameters

Data Type	Name	Description
IFX_int32_t	fd	File descriptor.
IFX_int32_t	FIO_VINETIC_GPIO_RELEASE	I/O control identifier for this operation.
IFX_int32_t	param	Use structure VINETIC_IO_GPIO_CONTROL .

Return Values

Data Type	Description
IFX_void_t	No return value.

7.1.5 Line Testing Interface

Interfaces for low-level line testing functions.

Table 69 IO-control Overview of Line Testing Interface

Name	Description
FIO_VINETIC_GR909_RESULT	Read Results of a GR909 Measurement.
FIO_VINETIC_GR909_START	Start a GR909 Measurement.
FIO_VINETIC_GR909_STOP	Stop a GR909 Measurement.
FIO_VINETIC_LT_AC_DIRECT	Linetesting AC measurement (most parameters configurable).
FIO_VINETIC_LT_AC_MEASUREMENTS	Linetesting AC measurements (PCM4-like).
FIO_VINETIC_LT_CAPACITANCE	Linetesting capacitance measurement.
FIO_VINETIC_LT_CONFIG	Linetesting configuration.
FIO_VINETIC_LT_CURRENT	Linetesting current measurement.
FIO_VINETIC_LT_IMPEDANCE	Linetesting impedance measurement.
FIO_VINETIC_LT_RESISTANCE	Linetesting resistance measurement.
FIO_VINETIC_LT_RESISTANCE_RC	Linetesting resistance RC measurement.
FIO_VINETIC_LT_VOLTAGE	Linetesting voltage measurement.
FIO_VINETIC_OFFSET_CALIBRATION	Offset calibration.

Table 70 Structure Overview of Line Testing Interface

Name	Description
VINETIC_IO_LT_AC_DIRECT	Linetesting AC Direct Measurement (all parameters of the measurement can be configured by the application).
VINETIC_IO_LT_AC_LM_FILTER	Linetesting AC Direct Measurement Level Meter Filter Configuration.
VINETIC_IO_LT_AC_MEASUREMENTS	Linetesting AC Measurement.
VINETIC_IO_LT_AC_TG	Linetesting AC Direct Measurement Tone Generator Configuration.
VINETIC_IO_LT_CAPACITANCE	Linetesting Capacitance Measurement.
VINETIC_IO_LT_CONFIG	Linetesting Configuration.
VINETIC_IO_LT_CURRENT	Linetesting Current Measurement.
VINETIC_IO_LT_IMPEDANCE	Linetesting Impedance Measurement.
VINETIC_IO_LT_RESISTANCE	Linetesting Resistance Measurement.
VINETIC_IO_LT_RESISTANCE_RC	Linetesting Resistance Measurement via RC network.
VINETIC_IO_LT_RESULT	Linetesting Measurement Result.
VINETIC_IO_LT_RESULT_CONFIG	Linetesting Measurement Result Configuration.
VINETIC_IO_LT_VOLTAGE	Linetesting Voltage Measurement.
VINETIC_IO_OFFSET_CALIBRATION	Offset Calibration.
VINETIC_IO_OFFSET_VALUE	Offset Calibration Results.
VINETIC_IO_GR909	GR909 IO Structure.

Table 71 Enumerator Overview of Line Testing Interface

Name	Description
GR909_RESULT_PARAM	Measurement results parameters alignment in result buffer.
GR909_TEST_RESULT	GR909 Measurements status values.
GR909_TESTS	GR909 possible measurements.
VINETIC_IO_LT_AC_LM_FILTER_TYPE	Linetesting AC Direct Measurement Level Meter Filter Structure.
VINETIC_IO_LT_AC_LM_SELECT	Linetesting AC Direct Measurement Input Selection.
VINETIC_IO_LT_AC_MODE	Linetesting AC Measurement Mode Selection.
VINETIC_IO_LT_AC_TG_STATE	Linetesting AC Direct Measurement Tone Generator State.
VINETIC_IO_LT_AC_TH_FILTER	Linetesting AC Direct Measurement Transhybrid Filter Configuration.
VINETIC_IO_LT_BP_QUAL	Bandpass quality selection (VINETIC 1.x only).
VINETIC_IO_LT_CAPACITANCE_MODE	Linetesting Capacitance Measurement Mode Selection.
VINETIC_IO_LT_CONFIG_MODE	Linetesting Configuration Mode.
VINETIC_IO_LT_CURRENT_MODE	Linetesting Current Measurement Mode Selection.
VINETIC_IO_LT_CURRENT_SUBMODE	Linetesting Current Measurement Submode Selection.
VINETIC_IO_LT_IMPEDANCE_MODE	Linetesting Impedance Measurement Mode Selection.
VINETIC_IO_LT_IMPEDANCE_SUBMODE	Linetesting Impedance Measurement Submode Selection.
VINETIC_IO_LT_RESISTANCE_MODE	Linetesting Resistance Measurement Mode Selection.
VINETIC_IO_LT_RS	Linetesting Magnitude Processing.
VINETIC_IO_LT_TG	Linetesting AC Direct Measurement Tone Generator Selection.
VINETIC_IO_LT_VOLTAGE_MODE	Linetesting Voltage Measurement Mode Selection.
VINETIC_IO_LT_VOLTAGE_SUBMODE	Linetesting Voltage Measurement Submode Selection.
VINETIC_IO_OFFSET_CALIBRATION_MODE	Offset Calibration Mode Selection.

7.1.5.1 FIO_VINETIC_GR909_RESULT

Description

Read Results of a GR909 Measurement.

Prototype

```
IFX_void_t ioctl1 (
    IFX_int32_t fd,
    VINETIC_IO_GR909_RESULT,
    IFX_int32_t param );
```

Parameters

Data Type	Name	Description
IFX_int32_t	fd	Pointer to a file descriptor.

Data Type	Name	Description
IFX_int32_t	VINETIC_IO_GR909_RESULT	I/O control identifier for this operation.
IFX_int32_t	param	Parameter is a pointer to FIO_VINETIC_GR909_RESULT structure.

Return Values

Data Type	Description
IFX_void_t	No return value.

7.1.5.2 FIO_VINETIC_GR909_START

Description

Start a GR909 Measurement.

Prototype

```
IFX_void_t ioctl (
    IFX_int32_t fd,
    VINETIC_IO_GR909_START,
    IFX_int32_t param );
```

Parameters

Data Type	Name	Description
IFX_int32_t	fd	Pointer to a file descriptor.
IFX_int32_t	VINETIC_IO_GR909_START	I/O control identifier for this operation.
IFX_int32_t	param	Parameter is a pointer to FIO_VINETIC_GR909_START structure.

Return Values

Data Type	Description
IFX_void_t	No return value.

7.1.5.3 FIO_VINETIC_GR909_STOP

Description

Stop a GR909 Measurement.

Prototype

```
IFX_void_t ioctl (
    IFX_int32_t fd,
    FIO_VINETIC_GR909_STOP,
    IFX_int32_t param );
```

Parameters

Data Type	Name	Description
IFX_int32_t	fd	Pointer to a file descriptor.
IFX_int32_t	FIO_VINETIC_GR909_STOP	I/O control identifier for this operation.
IFX_int32_t	param	

Return Values

Data Type	Description
IFX_void_t	No return value.

7.1.5.4 FIO_VINETIC_OFFSET_CALIBRATION
Description

Offset calibration.

Prototype

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

Parameters

Data Type	Name	Description
IFX_int32_t	fd	Pointer to a file descriptor.
IFX_int32_t	VINETIC_IO_OFFSET_CALIBRATION	I/O control identifier for this operation.
IFX_int32_t	param	Parameter is a pointer to a FIO_VINETIC_OFFSET_CALIBRATION structure

Return Values

Data Type	Description
IFX_void_t	No return value.

7.1.5.5 FIO_VINETIC_LT_CURRENT
Description

Linetesting current measurement.

Prototype

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



```
VINETIC_IO_LT_CURRENT,  
IFX_int32_t param );
```

Parameters

Data Type	Name	Description
IFX_int32_t	fd	Pointer to a file descriptor.
IFX_int32_t	VINETIC_IO_LT_CURRENT	I/O control identifier for this operation.
IFX_int32_t	param	Parameter is a pointer to a FIO_VINETIC_LT_CURRENT structure.

Return Values

Data Type	Description
IFX_void_t	No return value.

7.1.5.6 FIO_VINETIC_LT_VOLTAGE

Description

Linetesting voltage measurement.

Prototype

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

Parameters

Data Type	Name	Description
IFX_int32_t	fd	Pointer to a file descriptor.
IFX_int32_t	VINETIC_IO_LT_VOLTAGE	I/O control identifier for this operation.
IFX_int32_t	param	Parameter is a pointer to FIO_VINETIC_LT_VOLTAGE a structure.

Return Values

Data Type	Description
IFX_void_t	No return value.

7.1.5.7 FIO_VINETIC_LT_RESISTANCE

Description

Linetesting resistance measurement.

Prototype

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

Parameters

Data Type	Name	Description
IFX_int32_t	fd	Pointer to a file descriptor.
IFX_int32_t	VINETIC_IO_LT_RESISTANCE	I/O control identifier for this operation.
IFX_int32_t	param	Parameter is a pointer to a FIO_VINETIC_LT_RESISTANCE structure.

Return Values

Data Type	Description
IFX_void_t	No return value.

7.1.5.8 FIO_VINETIC_LT_CAPACITANCE

Description

Linetesting capacitance measurement.

Prototype

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

Parameters

Data Type	Name	Description
IFX_int32_t	fd	Pointer to a file descriptor.
IFX_int32_t	VINETIC_IO_LT_CAPACITANCE	I/O control identifier for this operation.
IFX_int32_t	param	Parameter is a pointer to a FIO_VINETIC_LT_CAPACITANCE structure.

Return Values

Data Type	Description
IFX_void_t	No return value.

7.1.5.9 FIO_VINETIC_LT_IMPEDANCE

Description

Linetesting impedance measurement.

Prototype

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

Parameters

Data Type	Name	Description
IFX_int32_t	fd	Pointer to a file descriptor.
IFX_int32_t	VINETIC_IO_LT_IMPEDANCE	I/O control identifier for this operation.
IFX_int32_t	param	Parameter is a pointer to a FIO_VINETIC_LT_IMPEDANCE structure.

Return Values

Data Type	Description
IFX_void_t	No return value.

7.1.5.10 FIO_VINETIC_LT_AC_MEASUREMENTS

Description

Linetesting AC measurements (PCM4-like).

Prototype

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

Parameters

Data Type	Name	Description
IFX_int32_t	fd	Pointer to a file descriptor.
IFX_int32_t	VINETIC_IO_LT_AC_MEASUREMENTS	I/O control identifier for this operation.
IFX_int32_t	param	Parameter is a pointer to a FIO_VINETIC_LT_AC_MEASUREMENTS structure.

Return Values

Data Type	Description
IFX_void_t	No return value.

7.1.5.11 FIO_VINETIC_LT_AC_DIRECT
Description

Linetesting AC measurement (most parameters configurable).

Prototype

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

Parameters

Data Type	Name	Description
IFX_int32_t	fd	Pointer to a file descriptor.
IFX_int32_t	FIO_VINETIC_LT_AC_DIRECT	I/O control identifier for this operation.
IFX_int32_t	param	Parameter is a pointer to a FIO_VINETIC_LT_AC_DIRECT structure.

Return Values

Data Type	Description
IFX_void_t	No return value.

7.1.5.12 FIO_VINETIC_LT_RESISTANCE_RC
Description

Linetesting resistance RC measurement.

Prototype

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

Parameters

Data Type	Name	Description
IFX_int32_t	fd	Pointer to a file descriptor.
IFX_int32_t	FIO_VINETIC_LT_RESISTANCE_RC	I/O control identifier for this operation.
IFX_int32_t	param	Parameter is a pointer to a VINETIC_IO_LT_RESISTANCE_RC structure.

Return Values

Data Type	Description
IFX_void_t	No return value.

7.1.5.13 FIO_VINETIC_LT_CONFIG
Description

Linetesting configuration.

Prototype

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

Parameters

Data Type	Name	Description
IFX_int32_t	fd	Pointer to a file descriptor.
IFX_int32_t	FIO_VINETIC_LT_CONFIG	I/O control identifier for this operation.
IFX_int32_t	param	Parameter is a pointer to a VINETIC_IO_LT_CONFIG structure.

Return Values

Data Type	Description
IFX_void_t	No return value.

7.1.6 Polling Interface

Control the device and channels in polling mode.

Table 72 IO-control Overview of Polling Interface

Name	Description
FIO_VINETIC_POLL_EVT	This interface is to call periodically to read the VINETIC® status registers for polling mode.
FIO_VINETIC_POLL_DEV_ADD	Polling Add.

Table 73 Constant Overview of Polling Interface

Name	Description
VIN_BUF_HDR1_CH	Packet Header Masks for channel number.
VIN_BUF_HDR1_DEV	Packet Header Masks for device number.
VIN_BUF_HDR2_LEN	Packet Header Masks for payload length in words.
VIN_BUF_HDR2_ODD	Packet Header Masks for odd bit (set if the last byte in the last word is padding).

Table 74 Function Overview of Polling Interface

Name	Description
VINETIC_DrvCtrl	Polling control function
VINETIC_Poll_Down	Normal polling downstream interface functions.
VINETIC_Poll_DownIntlv	Interleaved polling downstream interface functions.
VINETIC_Poll_Events	Polling event handling
VINETIC_Poll_Up	Normal polling upstream interface functions.
VINETIC_Poll_UpIntlv	Interleaved polling upstream interface functions.

7.1.6.1 FIO_VINETIC_POLL_EVT

Description

This interface is to call periodically to read the VINETIC® status registers for polling mode.

Prototype

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

Parameters

Data Type	Name	Description
IFX_int32_t	fd	File descriptor.
IFX_int32_t	FIO_VINETIC_POLL_EVT	I/O control identifier for this operation.
IFX_int32_t	param	

Return Values

Data Type	Description
IFX_void_t	No return value.

7.1.6.2 FIO_VINETIC_POLL_DEV_ADD

Description

Driver configuration.

Prototype

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

Parameters

Data Type	Name	Description
IFX_int32_t	fd	File descriptor.
IFX_int32_t	FIO_VINETIC_POLL_DEV_ADD	I/O control identifier for this operation.
IFX_int32_t	param	

Return Values

Data Type	Description
IFX_void_t	No return value.

7.2 Function Reference

This chapter contains the function reference.

Table 75 Function Overview of Non TAPI Interfaces

Name	Description
VINETIC_GpioConfig	Configure a VINETIC® IO or GPIO pin.
VINETIC_GpioGet	Read the value from a VINETIC® IO or GPIO pin.
VINETIC_GpioIntMask	Set the interrupt enable mask.
VINETIC_GpioRelease	Release a VINETIC® IO or GPIO pin resource.
VINETIC_GpioReserve	Reserve a VINETIC® IO or GPIO pin resource.
VINETIC_GpioSet	Set the value of a VINETIC® IO or GPIO pin.
VINETIC_IrqPollConf	Polling control function.
VINETIC_OpenKernel	Open the device from kernel mode.
VINETIC_Poll_Down	Normal polling downstream interface functions.
VINETIC_Poll_DownIntlv	Interleaved polling downstream interface functions.
VINETIC_Poll_Events	Polling event handling.

Table 75 Function Overview of Non TAPI Interfaces (cont'd)

Name	Description
VINETIC_Poll_Up	Normal polling upstream interface functions.
VINETIC_Poll_UpIntlv	Interleaved polling upstream interface functions.
VINETIC_ReleaseKernel	Release a VINETIC® IO or GPIO pin resource.

7.2.1 VINETIC_GpioConfig

Description

Configure a VINETIC® IO or GPIO pin.

Prototype

```
IFX_int32_t VINETIC_GpioConfig ( );
```

Return Values

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

7.2.2 VINETIC_GpioGet

Description

Read the value from a VINETIC® IO or GPIO pin.

Prototype

```
IFX_int32_t VINETIC_GpioGet (
    IFX_int32_t ioHandle,
    IFX_uint16_t* nGet,
    IFX_uint16_t nMask );
```

Parameters

Data Type	Name	Description
IFX_int32_t	ioHandle	Handle returned by VINETIC_GpioReserve .
IFX_uint16_t*	nGet	Pointer where the read value shall be stored.
IFX_uint16_t	nMask	Only bits set to '1' will be stored.

Return Values

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

7.2.3 VINETIC_GpioIntMask

Description

Set the interrupt enable mask.

Prototype

```
IFX_int32_t VINETIC_GpioIntMask (
    IFX_int32_t ioHandle,
    IFX_uint16_t nSet,
    IFX_uint16_t nMask,
    IFX_uint32_t nMode );
```

Parameters

Data Type	Name	Description
IFX_int32_t	ioHandle	Handle returned by VINETIC_GpioReserve .
IFX_uint16_t	nSet	Bitmask for interrupts to mask (0 = unmasked, 1 = masked).
IFX_uint16_t	nMask	Mask to write to interrupt enable register.
IFX_uint32_t	nMode	Mode according to VINETIC_GPIO_MODE .

Return Values

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

7.2.4 VINETIC_GpioRelease

Description

Release a VINETIC® IO or GPIO pin resource.

Prototype

```
IFX_int32_t VINETIC_GpioRelease (
    IFX_int32_t ioHandle );
```

Parameters

Data Type	Name	Description
IFX_int32_t	ioHandle	Handle returned by VINETIC_GpioReserve .

Return Values

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

7.2.5 VINETIC_GpioReserve
Description

Reserve a VINETIC® IO or GPIO pin resource.

Prototype

```
IFX_int32_t VINETIC_GpioReserve (
    IFX_int32_t devHandle,
    IFX_uint16_t nGpio );
```

Parameters

Data Type	Name	Description
IFX_int32_t	devHandle	Handle to either VINETIC® device or channel structure.
IFX_uint16_t	nGpio	Mask for GPIOs to reserve (0 = free, 1 = reserve).

Return Values

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

7.2.6 VINETIC_GpioSet
Description

Set the value of a VINETIC® IO or GPIO pin.

Prototype

```
IFX_int32_t VINETIC_GpioSet (
    IFX_int32_t ioHandle,
    IFX_uint16_t nSet,
    IFX_uint16_t nMask );
```

Parameters

Data Type	Name	Description
IFX_int32_t	ioHandle	Handle returned by VINETIC_GpioReserve .
IFX_uint16_t	nSet	Values to store.
IFX_uint16_t	nMask	Only bits set to '1' will be stored.

Return Values

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

7.2.7 VINETIC_IrqPollConf

Description

Polling control function.

Prototype

```
VINETIC_void_t VINETIC_IrqPollConf (
    VINETIC_IO_DRVCTRL* pCtrl );
```

Parameters

Data Type	Name	Description
VINETIC_IO_DRVCTRL*	pCtrl	

Return Values

Data Type	Description
VINETIC_void_t	No return value.

7.2.8 VINETIC_OpenKernel

Description

Open the device from Kernel mode.

Prototype

```
IFX_int32_t VINETIC_OpenKernel (
    IFX_int32_t nDev,
    IFX_int32_t nCh );
```

Parameters

Data Type	Name	Description
IFX_int32_t	nDev	Index of the VINETIC® device.
IFX_int32_t	nCh	Index of the VINETIC® channel (1 = channel 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 not already done this will
allocate internal memory for each new device
allocate io memory
initialize the device
set up the interrupt

7.2.9 VINETIC_Poll_Down

Description

Normal polling downstream interface functions.

Prototype

```
IFX_void_t VINETIC_Poll_Down (
    const int nElements,
    IFX_void_t ** ppBuffers );
```

Parameters

Data Type	Name	Description
const int	nElements	
IFX_void_t **	ppBuffers	

Return Values

Data Type	Description
IFX_void_t	No return value.

7.2.10 VINETIC_Poll_DownIntlv

Description

Interleaved polling downstream interface functions.

Prototype

```
VINETIC_void_t VINETIC_Poll_DownIntlv (
    const int nElements,
    IFX_void_t ** ppBuffers );
```

Parameters

Data Type	Name	Description
const int	nElements	
IFX_void_t **	ppBuffers	

Return Values

Data Type	Description
VINETIC_void_t	No return value.

7.2.11 VINETIC_Poll_Events

Description

Polling event handling.

Prototype

```
IFX_void_t VINETIC_Poll_Events (
    IFX_void_t );
```

Parameters

Data Type	Name	Description
IFX_void_t		

Return Values

Data Type	Description
IFX_void_t	No return value.

7.2.12 VINETIC_Poll_Up

Description

Normal polling upstream interface functions.

Prototype

```
VINETIC_void_t VINETIC_Poll_Up (
    IFX_int32_t * pElements,
    IFX_void_t ** ppBuffers );
```

Parameters

Data Type	Name	Description
IFX_int32_t *	pElements	
IFX_void_t **	ppBuffers	

Return Values

Data Type	Description
VINETIC_void_t	No return value.

7.2.13 VINETIC_Poll_UpIntlv
Description

Interleaved polling upstream interface functions.

Prototype

```
VINETIC_void_t VINETIC_Poll_UpIntlv (
    IFX_int32_t * pElements,
    IFX_void_t ** ppBuffers );
```

Parameters

Data Type	Name	Description
IFX_int32_t *	pElements	
IFX_void_t **	ppBuffers	

Return Values

Data Type	Description
VINETIC_void_t	No return value.

7.2.14 VINETIC_ReleaseKernel
Description

Release a VINETIC® IO or GPIO pin resource.

Prototype

```
IFX_int32_t VINETIC_ReleaseKernel (
    IFX_int32_t nHandle );
```

Parameters

Data Type	Name	Description
IFX_int32_t	nHandle	Handle returned by VINETIC_GpioReserve .

Return Values

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

7.3 Type Definition Reference

This chapter contains the reference of data types and structures of all modules.

7.3.1 IO-control Reference

This chapter contains the IO-control reference.

Table 76 IO-control Overview of Non TAPI Interfaces

Name	Description
FIO_VINETIC_BASICDEV_INIT	Initialize VINETIC® Device driver information for one chip.
FIO_VINETIC_DEV_GPIO_CFG	Configure device GPIO pins 0.
FIO_VINETIC_DEV_GPIO_SET	Set GPIO pins values.
FIO_VINETIC_DEV_RESET	Reset VINETIC® Device driver internal structure for one chip.
FIO_VINETIC_DOW_PHI	PHI download using structure VINETIC_IO_PHI.
FIO_VINETIC_DOWNFPI	FPI download, only for internal use.
FIO_VINETIC_DOWNLOAD_CRAM	Download cram coefficients to one or all VINETIC channels.
FIO_VINETIC_DRV_CTRL	Driver configuration.
FIO_VINETIC_GPIO_CONFIG	Configure GPIO pins.
FIO_VINETIC_GPIO_GET	Get GPIO pin values.
FIO_VINETIC_GPIO_RELEASE	Set GPIO pin values.
FIO_VINETIC_GPIO_RESERVE	Reserve GPIO pins for use.
FIO_VINETIC_GPIO_SET	Set GPIO pin values.
FIO_VINETIC_GR909_RESULT	Read Results of a GR909 Measurement.
FIO_VINETIC_GR909_START	Start a GR909 Measurement.
FIO_VINETIC_GR909_STOP	Stop a GR909 Measurement.
FIO_VINETIC_INIT	VINETIC® Initialization.
FIO_VINETIC_LASTERR	Get the last occurred error.
FIO_VINETIC_LT_AC_DIRECT	Linetesting AC measurement (most parameters configurable).
FIO_VINETIC_LT_AC_MEASUREMENTS	Linetesting AC measurements (PCM4-like).
FIO_VINETIC_LT_CAPACITANCE	Linetesting capacitance measurement.
FIO_VINETIC_LT_CONFIG	Linetesting configuration.
FIO_VINETIC_LT_CURRENT	Linetesting current measurement.
FIO_VINETIC_LT_IMPEDANCE	Linetesting impedance measurement.
FIO_VINETIC_LT_RESISTANCE	Linetesting resistance measurement.
FIO_VINETIC_LT_RESISTANCE_RC	Linetesting resistance RC measurement.
FIO_VINETIC_LT_VOLTAGE	Linetesting voltage measurement.
FIO_VINETIC_OFFSET_CALIBRATION	Offset calibration.
FIO_VINETIC_POLL_DEV_ADD	Driver configuration.
FIO_VINETIC_POLL_EVT	This interface is to call periodically to read the VINETIC® status registers for polling mode.
FIO_VINETIC_RCMD	Read command.
FIO_VINETIC_REPORT_SET	Set the driver report levels if the driver is compiled with ENABLE_TRACE.
FIO_VINETIC_RSC	Read short command.

Table 76 IO-control Overview of Non TAPI Interfaces (cont'd)

Name	Description
FIO_VINETIC_RUNTIME_TRACE_SET	Driver runtime tracing of register accesses.
FIO_VINETIC_VERS	Read relevant version information.
FIO_VINETIC_WCMD	Write a VINETIC® command.
FIO_VINETIC_WSC	Write short command.

7.3.2 Constant Reference

This chapter contains the constant reference.

Table 77 Constant Overview of Non TAPI Interfaces

Name	Description
DC_DWLD	Flag for VINETIC_IO_INIT to perform a DCCTL download in case of V1.4.
FW_AUTODWLD	Flag for VINETIC_IO_INIT to send the auto download command in case of V1.4.
FW_AUTODWLD	Maximal Command/Data Words.
MAX_GR909_VAL	Maximal GR909 test values in a state machine.
MAX_PACKET_WORD	Maximal Packet Words.
NO_AC_DWLD	Flag for VINETIC_IO_INIT to avoid AC download in case of V1.4.
NO_BCONF	Flag for VINETIC_IO_INIT to avoid no board configuration.
NO_CRAM_DWLD	Flag for VINETIC_IO_INIT to avoid CRAM download in case of V1.4.
NO_EDSP_START	Flag for VINETIC_IO_INIT to avoid EDSP start.
NO_FW_DWLD	Flag for VINETIC_IO_INIT to avoid firmware download.
NO_PHI_DWLD	Flag for VINETIC_IO_INIT to avoid PHI download in case of V1.4.
VIN_BUF_HDR1_CH	Packet Header Masks for channel number.
VIN_BUF_HDR1_DEV	Packet Header Masks for device number.
VIN_BUF_HDR2_LEN	Packet Header Masks for payload length in words.
VIN_BUF_HDR2_ODD	Packet Header Masks for odd bit (set if the last byte in the last word is padding).
VINETIC_ANA_CH_NR	VINETIC® maximum analog channel number.
VINETIC_CH_NR	VINETIC® maximum channel number.
VINETIC_IO_CRAM_DATA_SIZE	Maximum size.

Table 78 Constant Reference for Non TAPI Interfaces

Name and Description	Value
DC_DWLD Flag for VINETIC_IO_INIT to perform a DC control download in case of V1.4.	200 _H
FW_AUTODWLD Flag for VINETIC_IO_INIT to send the auto download command in case of V1.4.	20 _H
MAX_CMD_WORD Maximal Command/Data Words.	31 _D
MAX_GR909_VAL VINETIC® maximum analog channel number.	15 _D

Table 78 Constant Reference for Non TAPI Interfaces (cont'd)

Name and Description	Value
MAX_PACKET_WORD Maximal Packet Words.	255 _D
NO_AC_DWLD Flag for VINETIC_IO_INIT to avoid AC download in case of V1.4.	80 _H
NO_BCONF Flag for VINETIC_IO_INIT to avoid no board configuration.	1 _H
NO_CRAM_DWLD Flag for VINETIC_IO_INIT to avoid CRAM download in case of V1.4.	10 _H
NO_EDSP_START Flag for VINETIC_IO_INIT to avoid EDSP start.	2 _H
NO_FW_DWLD Flag for VINETIC_IO_INIT to avoid firmware download.	100 _H
NO_PHI_DWLD Flag for VINETIC_IO_INIT to avoid PHI download in case of V1.4.	8 _H
VIN_BUF_HDR1_CH Packet Header Masks for channel number.	3 _H
VIN_BUF_HDR1_DEV Packet Header Masks for device number.	FC _H
VIN_BUF_HDR2_LEN Packet Header Masks for payload length in words.	FF _H
VIN_BUF_HDR2_ODD Packet Header Masks for odd bit (set if the last byte in the last word is padding).	2000 _H
VINETIC_ANA_CH_NR VINETIC® maximum analog channel number.	4 _D
VINETIC_CH_NR VINETIC® maximum channel number.	8 _D
VINETIC_IO_CRAM_DATA_SIZE Maximum size.	250 _D

7.3.3 Structure Reference

This chapter contains the Structure reference.

Table 79 Structure Overview of Device Driver Interfaces

Name	Description
VINETIC_BasicDeviceInit_t	VINETIC® Basic Device Initialization structure.
VINETIC_GPIO_CONFIG	GPIO Configuration Structure.
VINETIC_IO_CRAM	Structure for CRAM download with the V1.4 and the V2.x format.
VINETIC_IO_CRAM_REG_CFG	CRAM download related setting of register values (used for activation).
VINETIC_IO_DRVCTRL	Driver global configuration structure.
VINETIC_IO_EMBEDCTRL	Embedded Controller Download structure.
VINETIC_IO_GPIO_CONTROL	GPIO Control Structure.
VINETIC_IO_GR909	GR909 IO Structure.
VINETIC_IO_INIT	Structure used for device initialization.

Table 79 Structure Overview of Device Driver Interfaces (cont'd)

Name	Description
VINETIC_IO_LT_AC_DIRECT	Linetesting AC Direct Measurement (all parameters of the measurement can be configured by the application).
VINETIC_IO_LT_AC_LM_FILTER	Linetesting AC Direct Measurement Level Meter Filter Configuration.
VINETIC_IO_LT_AC_MEASUREMENTS	Linetesting AC Measurement.
VINETIC_IO_LT_AC_TG	Linetesting AC Direct Measurement Tone Generator Configuration.
VINETIC_IO_LT_CAPACITANCE	Linetesting Capacitance Measurement.
VINETIC_IO_LT_CONFIG	Linetesting Configuration.
VINETIC_IO_LT_CURRENT	Linetesting Current Measurement.
VINETIC_IO_LT_IMPEDANCE	Linetesting Impedance Measurement.
VINETIC_IO_LT_LL_CONFIG	Part of data structure VINETIC_IO_LT_CONFIG . Used to configure the integration times for current, voltage, and resistance measurements.
VINETIC_IO_LT_RESISTANCE	Linetesting Resistance Measurement.
VINETIC_IO_LT_RESISTANCE_RC	Linetesting Resistance Measurement via RC network.
VINETIC_IO_LT_RESULT	Linetesting Measurement Result.
VINETIC_IO_LT_RESULT_CONFIG	Linetesting Measurement Result Configuration.
VINETIC_IO_LT_VOLTAGE	Linetesting Voltage Measurement.
VINETIC_IO_MB_CMD	IO structure for write and read chip commands for debugging purposes only.
VINETIC_IO_OFFSET_CALIBRATION	Offset Calibration.
VINETIC_IO_OFFSET_VALUE	Offset Calibration Results.
VINETIC_IO_READ_SC	IO structure for read short commands.
VINETIC_IO_TAPI_LT_CONFIG	Part of data structure VINETIC_IO_LT_CONFIG . Provides information to the Linetesting TAPI level whether or not a low-pass filter is used, and also the type of SLIC versions used (since not all SLIC versions can be determined through the BCR1 register).
VINETIC_IO_USR	VINETIC® User Interface structure.
VINETIC_IO_VERSION	Version IO structure.
VINETIC_IO_WRITE_SC	IO structure for writing short commands.

7.3.3.1 VINETIC_BasicDeviceInit_t

Description

VINETIC® Basic Device Initialization structure.

Prototype

```
typedef struct
{
    VIN_ACCESS AccessMode;
    IFX_uint32_t nBaseAddress;
    IFX_int32_t nIrqNum;
} VINETIC_BasicDeviceInit_t;
```

Parameters

Data Type	Name	Description
VIN_ACCESS	AccessMode	Access mode for VINETIC® device.
IFX_uint32_t	nBaseAddress	VINETIC® physical base address.
IFX_int32_t	nIrqNum	VINETIC® device irq number, as defined by the OS.

7.3.3.2 VINETIC_GPIO_CONFIG
Description

GPIO Configuration structure.

Prototype

```
typedef struct
{
    IFX_uint16_t nGpio;
    IFX_uint32_t nMode;
    IFX_void_t (*callback)(int nDev, int nCh, unsigned short nEvt);
} VINETIC_GPIO_CONFIG_t;
```

Parameters

Data Type	Name	Description
IFX_uint16_t	nGpio	Mask for GPIO resources.
IFX_uint32_t	nMode	GPIO mode (input, output, interrupt).
IFX_void_t	(*callback)(int nDev, int nCh, unsigned short nEvt)	Callback for interrupt routine.

7.3.3.3 VINETIC_IO_CRAM
Description

Structure for CRAM download with the V1.4 and the V2.x format. This structure is used for the ioctl.

Prototype

```
typedef struct
{
    IFX_uint16_t bBroadCast;
    VINETIC_IO_CRAM_FORMAT nFormat;
    IFX_uint8_t nStartAddr;
    IFX_uint32_t nLength;
    IFX_uint16_t aData[VINETIC_IO_CRAM_DATA_SIZE];
    VINETIC_IO_CRAM_REG_CFG bcr1;
    VINETIC_IO_CRAM_REG_CFG bcr2;
    VINETIC_IO_CRAM_REG_CFG tstr2;
    IFX_uint16_t nCRC;
} VINETIC_IO_CRAM_t;
```

Parameters

Data Type	Name	Description
IFX_uint16_t	bBroadCast	Set to 1 if the CRAM coefficients should be downloaded to all channels, if set to 0, the CRAM download is only done on one channel (according to the file descriptor).
VINETIC_IO_CRAM_FORMAT	nFormat	Defines the CRAM download format version.
IFX_uint8_t	nStartAddr	Start offset of the CRAM coefficients in this download.
IFX_uint32_t	nLength	Number of coefficients to be downloaded in (16-bit) Words.
IFX_uint16_t	aData[VINETIC_IO_CRAM_D ATA_SIZE]	Coefficients array to be downloaded.
VINETIC_IO_CRAM_REG_CFG	bcr1	CRAM coefficient specific settings for bcr1 register.
VINETIC_IO_CRAM_REG_CFG	bcr2	CRAM coefficient specific settings for bcr2 register.
VINETIC_IO_CRAM_REG_CFG	tstr2	CRAM coefficient specific settings for tstr2 register.
IFX_uint16_t	nCRC	CRAM CRC checksum.

7.3.3.4 VINETIC_IO_CRAM_REG_CFG
Description

CRAM download related setting of register values (used for activation).

Prototype

```
typedef struct
{
    IFX_uint16_t nData;
    IFX_uint16_t nMask;
} VINETIC_IO_CRAM_REG_CFG_t;
```

Parameters

Data Type	Name	Description
IFX_uint16_t	nData	Contains the values of the bits that should be changed (see nMask).
IFX_uint16_t	nMask	Defines which bits should be modified to the corresponding value in nData by setting them to 1.

7.3.3.5 VINETIC_IO_DRVCTRL

Description

Driver global configuration structure.

Prototype

```
typedef struct
{
    IFX_int32_t bPoll;
    IFX_int32_t bDataEvt;
    IFX_int32_t bEvtSel;
    IFX_void_t* pBufferPool;
    IFX_void_t* (*getBuf)(void *pBuf);
    IFX_int32_t* (*putBuf)(void *pBuf);
} VINETIC_IO_DRVCTRL_t;
```

Parameters

Data Type	Name	Description
IFX_int32_t	bPoll	Driver polling/interrupt mode control flag. 0 configure driver for interrupt mode. 1 configure driver for polling mode.
IFX_int32_t	bDataEvt	Packets control. 0 packets read by normal/interleave polling routines. 1 packets read on event polling.
IFX_int32_t	bEvtSel	Report events mode 0 event read by application 1 event read via poll/select
IFX_void_t*	pBufferPool	Pointer to buffer pool control structure. The pointer is used for the retrieve and return functions to identify the buffer handlers control structure. The buffer pool is expected to be pre initialized.
IFX_void_t*	(*getBuf)(void *pBuf)	Function pointer to retrieve a empty buffer from the pool. The buffer is protected and can be exclusively used by the user. The size is predefined by the buffer pool handler.
IFX_int32_t*	(*putBuf)(void *pBuf)	Function pointer to return a used buffer to the pool.

7.3.3.6 VINETIC_IO_EMBDCTRL

Description

Embedded Controller download structure.

Prototype

```
typedef struct
{
    IFX_uint32_t* pData;
    IFX_uint32_t nSize;
} VINETIC_IO_EMBEDCTRL_t;
```

Parameters

Data Type	Name	Description
IFX_uint32_t*	pData	Data buffer.
IFX_uint32_t	nSize	Amount of Data Size in bytes.

7.3.3.7 VINETIC_IO_GPIO_CONTROL

Description

GPIO Control Structure.

Prototype

```
typedef struct
{
    IFX_int32_t ioHandle;
    IFX_uint16_t nGpio;
    IFX_uint16_t nMask;
} VINETIC_IO_GPIO_CONTROL_t;
```

Parameters

Data Type	Name	Description
IFX_int32_t	ioHandle	GPIO handle.
IFX_uint16_t	nGpio	GPIO resource mask. Least significant bit corresponds to pin 0.
IFX_uint16_t	nMask	Mask for current command.

7.3.3.8 VINETIC_IO_GR909

Description

GR909 IO Structure.

This structure is used for the ioctl.

Prototype

```
typedef struct
{
    IFX_uint8_t bAll;
    IFX_uint8_t mode;
    IFX_uint32_t values[MAX_GR909_VAL];
    IFX_uint8_t status;
```

```

    IFX_uint8_t nChipRev;
} VINETIC_IO_GR909_t;

```

Parameters

Data Type	Name	Description
IFX_uint8_t	bAll	GR909 measurement user control flag. 0 Single measurement will be done. 1 All measurements will be done in the user state machine.
IFX_uint8_t	mode	Actual measurement mode.
IFX_uint32_t	values[MAX_GR909_VAL]	Buffer for measurements results. Indexes in GR909_RESULT_PARAM are used for each measurement.
IFX_uint8_t	status	Measurement status. This can have the following values: GR909_TEST_OK - Measurement processed without error GR909_TEST_RUNNING - Measurement on going GR909_TEST_FAILED - Test failed. GR909_MEASUREMENT_FAILED - Measurement failed due to internal error.
IFX_uint8_t	nChipRev	Vinetic chip version for which measurement are done.

7.3.3.9 VINETIC_IO_INIT

Description

Structure used for device initialization.

Prototype

```

typedef struct
{
    IFX_uint8_t* pPRAMfw;
    IFX_uint32_t pram_size;
    IFX_uint8_t* pDRAMfw;
    IFX_uint32_t dram_size;
    IFX_uint8_t* pPHIfw;
    IFX_uint32_t phi_size;
    IFX_uint8_t* pCram;
    IFX_uint32_t cram_size;
    IFX_uint8_t* pBBDbuf;
    IFX_uint32_t bbd_size;
    IFX_uint32_t nFlags;
    IFX_uint16_t nPramCRC;
    IFX_uint16_t nDramCRC;
    IFX_uint16_t nPhiCrc;
    IFX_uint16_t nDcCrc;
}

```



```

    IFX_uint16_t nAcCrc;
    IFX_uint16_t nCramCrc;
} VINETIC_IO_INIT_t;

```

Parameters

Data Type	Name	Description
IFX_uint8_t*	pPRAMfw	Firmware PRAM pointer or NULL if not needed.
IFX_uint32_t	pram_size	Size of PRAM firmware in bytes.
IFX_uint8_t*	pDRAMfw	Firmware DRAM pointer or NULL if not needed.
IFX_uint32_t	dram_size	Size of DRAM firmware in bytes.
IFX_uint8_t*	pPHIfw	Pointer optional PHI program.
IFX_uint32_t	phi_size	Size of PHI program in bytes.
IFX_uint8_t*	pCram	Pointer to CRAM.
IFX_uint32_t	cram_size	Size of CRAM coefficients in bytes.
IFX_uint8_t*	pBBDbuf	Pointer to BBD format data.
IFX_uint32_t	bbd_size	Size of BBD buffer.
IFX_uint32_t	nFlags	Flags for initialization. Most of the flags are only used from experts to modify the default initialization. • NO_BCONF: no board configuration will be done. VINETIC® must be properly got out of reset. • NO_PHI_DWLD: no PHI download will be done, a properly PHI download before is assumed. • NO_EDSP_START: no EDPS start is done. The VINETIC® will not work until that command is given. • NO_CRAM_DWLD: no CRAM coefficients are downloaded. The default ROM coefficients are used. • FW_AUTODWLD: firmware auto download. • NO_AC_DWLD: avoid AC download in case of V1.4, no effect with any other chip version. • DC_DWLD: do a DC download in case of V1.4. • NO_FW_DWLD: no firmware download.
IFX_uint16_t	nPramCRC	Return values of PRAM CRC after firmware download.
IFX_uint16_t	nDramCRC	Return values of DRAM CRC after firmware download.
IFX_uint16_t	nPhiCrc	Return values of PHI checksum after PHI program download. 0 if not done.

Data Type	Name	Description
IFX_uint16_t	nDcCrc	Return values of DCCTL checksum after DCCTL download. 0 if not done.
IFX_uint16_t	nAcCrc	Return values of AC control after ALM-DSP download. 0 if not done.
IFX_uint16_t	nCramCrc	Return values of CRAM checksum after CRAM download. 0 if not done.

7.3.3.10 VINETIC_IO_LT_AC_DIRECT

Description

Linetesting AC Direct Measurement (all parameters of the measurement can be configured by the application).

Prototype

```
typedef struct
{
    VINETIC_IO_LT_RS rsSelect;
    VINETIC_IO_LT_AC_TH_FILTER thFilter;
    VINETIC_IO_LT_AC_LM_FILTER lmFilter;
    VINETIC_IO_LT_AC_LM_SELECT lmSelect;
    VINETIC_IO_LT_AC_TG tg[2];
    VINETIC_IO_LT_RESULT_CONFIG cfg;
    VINETIC_IO_LT_RESULT res00;
} VINETIC_IO_LT_AC_DIRECT_t;
```

Parameters

Data Type	Name	Description
VINETIC_IO_LT_RS	rsSelect	
VINETIC_IO_LT_AC_TH_FILTER	thFilter	Linetesting AC Direct Measurement Transhybrid Filter Configuration.
VINETIC_IO_LT_AC_LM_FILTER	lmFilter	Linetesting AC Direct Measurement Level Meter Filter Structure.
VINETIC_IO_LT_AC_LM_SELECT	lmSelect	Linetesting AC Direct Measurement Input Selection.
VINETIC_IO_LT_AC_TG	tg[2]	Linetesting AC Direct Measurement Tone Generator Selection.
VINETIC_IO_LT_RESULT_CONFIG	cfg	Level Meter Result Configuration.
VINETIC_IO_LT_RESULT	res	Level Meter Result.

7.3.3.11 VINETIC_IO_LT_AC_LM_FILTER

Description

Linetesting AC Direct Measurement Level Meter Filter Configuration.

Prototype

```
typedef struct
{
    VINETIC_IO_LT_AC_LM_FILTER_TYPE type;
    VINETIC_IO_LT_BP_QUAL qual;
    IFX_int32_t freq;
} VINETIC_IO_LT_AC_LM_FILTER_t;
```

Parameters

Data Type	Name	Description
VINETIC_IO_LT_AC_LM_FILTER_TYPE	type	Select level meter filter structure.
VINETIC_IO_LT_BP_QUAL	qual	Select level meter bandpass quality (only Vinetic 1.x).
IFX_int32_t	freq	Select level meter filter frequency.

7.3.3.12 VINETIC_IO_LT_AC_MEASUREMENTS
Description

Linetesting AC Measurement.

Prototype

```
typedef struct
{
    VINETIC_IO_LT_AC_MODE mode;
    IFX_int32_t freq;
    IFX_int32_t level;
    VINETIC_IO_LT_RESULT_CONFIG cfg;
    VINETIC_IO_LT_RESULT res00;
    VINETIC_IO_LT_RESULT res_noise;
} VINETIC_IO_LT_AC_MEASUREMENTS_t;
```

Parameters

Data Type	Name	Description
VINETIC_IO_LT_AC_MODE	mode	Linetesting AC Measurement Mode Selection (select the PCM4 measurement to be executed).
IFX_int32_t	freq	Signal frequency to be used for this measurement.
IFX_int32_t	level	Signal level to be used for this measurement.
VINETIC_IO_LT_RESULT_CONFIG	cfg	Level Meter Result Configuration.
VINETIC_IO_LT_RESULT	res	Level Meter Result.
VINETIC_IO_LT_RESULT	res_noise	Level Meter Result (only used for SNR measurement).

7.3.3.13 VINETIC_IO_LT_AC_TG

Description

Linetesting AC Direct Measurement Tone Generator Configuration.

Prototype

```
typedef struct
{
    VINETIC_IO_LT_AC_TG_STATE state;
    IFX_int32_t freq;
    IFX_int32_t level;
    VINETIC_IO_LT_BP_QUAL qual;
} VINETIC_IO_LT_AC_TG_t;
```

Parameters

Data Type	Name	Description
VINETIC_IO_LT_AC_TG_STATE	state	Select tone generator state (on, off, unchanged).
IFX_int32_t	freq	Select tone generator frequency.
IFX_int32_t	level	Select tone generator level.
VINETIC_IO_LT_BP_QUAL	qual	Select tone generator quality (only used for Vinetic 1.x).

7.3.3.14 VINETIC_IO_LT_CAPACITANCE

Description

Linetesting Capacitance Measurement.

Prototype

```
typedef struct
{
    VINETIC_IO_LT_CAPACITANCE_MODE mode;
    VINETIC_IO_LT_RESULT_CONFIG cfg;
    VINETIC_IO_LT_RESULT res[2];
    IFX_uint32_t slope;
} VINETIC_IO_LT_CAPACITANCE_t;
```

Parameters

Data Type	Name	Description
VINETIC_IO_LT_CAPACITANCE_MODE	mode	Linetesting Capacitance Measurement Mode Selection (select between Tip/Ring, Tip/Ground, Ring/Ground).
VINETIC_IO_LT_RESULT_CONFIG	cfg	Level Meter Result Configuration.

Data Type	Name	Description
VINETIC_IO_LT_RESULT	res[2]	Level Meter Result (rising and falling slope current).
IFX_uint32_t	slope	Returns the ramp slope the driver decided to use for this measurement.

7.3.3.15 VINETIC_IO_LT_CONFIG

Description

Linetesting Configuration.

Prototype

```
typedef struct
{
    VINETIC_IO_LT_CONFIG_MODE access;
    VINETIC_IO_LT_LL_CONFIG ltDrvCfg;
    VINETIC_IO_TAPI_LT_CONFIG ltTapiCfg;
} VINETIC_IO_LT_CONFIG_t;
```

Parameters

Data Type	Name	Description
VINETIC_IO_LT_CONFIG_MODE	access	Command to read/write configuratuion. [in]
VINETIC_IO_LT_LL_CONFIG	ltDrvCfg	Driver level linetesting configuration data. [in/out]
VINETIC_IO_TAPI_LT_CONFIG	ltTapiCfg	TAPI level linetesting configuration data. [in/out]

7.3.3.16 VINETIC_IO_LT_LL_CONFIG

Description

Part of data structure [VINETIC_IO_LT_CONFIG](#). Used to configure the integration times for current, voltage, and resistance measurements. Applies only to Version 2.x VINETIC devices.

Prototype

```
typedef struct
{
    unsigned short CurrentIntegrationTime;
    unsigned short nVoltageIntegrationTime;
    unsigned short nResistanceIntegrationTime;
} VINETIC_IO_LT_LL_CONFIG_t;
```

Parameters

Data Type	Name	Description
unsigned short	CurrentIntegrationTime	Integration time for current measurement (Vinetic 2.x only) in ms. Valid values are 0..1000, whereas 0 configures a non integrated measurement. [in/out]
unsigned short	nVoltageIntegrationTime	Integration time for current measurement (Vinetic 2.x only) in ms. Valid values are 0..1000, whereas 0 configures a non integrated measurement. [in/out]
unsigned short	nResistanceIntegrationTime	Integration time for current measurement (Vinetic 2.x only) in ms. Valid values are 0..1000, whereas 0 configures a non integrated measurement. [in/out]

7.3.3.17 VINETIC_IO_LT_CURRENT
Description

Linetesting Current Measurement.

Prototype

```
typedef struct
{
    VINETIC_IO_LT_CURRENT_MODE mode;
    VINETIC_IO_LT_CURRENT_SUBMODE submode;
    VINETIC_IO_LT_RESULT_CONFIG cfg;
    VINETIC_IO_LT_RESULT res00;
} VINETIC_IO_LT_CURRENT_t;
```

Parameters

Data Type	Name	Description
VINETIC_IO_LT_CURRENT_MODE	mode	Current Measurement Mode Selection (IT, IL).
VINETIC_IO_LT_CURRENT_SUBMODE	submode	Current Measurement Submode Selection (DC, AC).
VINETIC_IO_LT_RESULT_CONFIG	cfg	Level Meter Result Configuration.
VINETIC_IO_LT_RESULT	res	Level Meter Result.

7.3.3.18 VINETIC_IO_LT_IMPEDANCE
Description

Linetesting Impedance Measurement.

Prototype

```
typedef struct
{
    VINETIC_IO_LT_IMPEDANCE_MODE mode;
    VINETIC_IO_LT_IMPEDANCE_SUBMODE submode;
    IFX_uint32_t freq;
```

```

    IFX_uint32_t level;
    VINETIC_IO_LT_RESULT_CONFIG cfg;
    VINETIC_IO_LT_RESULT res00;
} VINETIC_IO_LT_IMPEDANCE_t;

```

Parameters

Data Type	Name	Description
VINETIC_IO_LT_IMPEDANCE_MODE	mode	Mode Selection (select between Ring/Tip, Ring/Ground and Tip/Ground).
VINETIC_IO_LT_IMPEDANCE_SUBMODE	submode	Submode Selection (select signal source for the Impedance measurement).
IFX_uint32_t	freq	Signal frequency to be used for impedance measurement.
IFX_uint32_t	level	Signal level to be used for the impedance measurement.
VINETIC_IO_LT_RESULT_CONFIG	cfg	Level Meter Result Configuration.
VINETIC_IO_LT_RESULT	res	Level Meter Result.

7.3.3.19 VINETIC_IO_LT_RESISTANCE
Description

Linetesting Resistance Measurement.

Prototype

```

typedef struct
{
    VINETIC_IO_LT_RESISTANCE_MODE mode;
    VINETIC_IO_LT_VOLTAGE_MODE submode;
    VINETIC_IO_LT_RESULT_CONFIG cfg;
    VINETIC_IO_LT_RESULT res_U1;
    VINETIC_IO_LT_RESULT res_U2;
    VINETIC_IO_LT_RESULT res_I1;
    VINETIC_IO_LT_RESULT res_I2;
} VINETIC_IO_LT_RESISTANCE_t;

```

Parameters

Data Type	Name	Description
VINETIC_IO_LT_RESISTANCE_MODE	mode	Linetesting Resistance Measurement Mode Selection (select between Tip/Ring, Tip/Ground, Ring/Ground).
VINETIC_IO_LT_VOLTAGE_MODE	submode	Linetesting Resistance Measurement Submode Selection (the application has to tell the driver which IO pins are connected to Tip/Ring).
VINETIC_IO_LT_RESULT_CONFIG	cfg	Level Meter Result Configuration.
VINETIC_IO_LT_RESULT	res_U1	Level Meter Result first voltage.

Data Type	Name	Description
VINETIC_IO_LT_RESULT	res_U2	Level Meter Result second voltage (not used for off hook resistance measurement).
VINETIC_IO_LT_RESULT	res_I1	Level Meter Result first current.
VINETIC_IO_LT_RESULT	res_I2	Level Meter Result second current (not used for off hook resistance measurement).

7.3.3.20 VINETIC_IO_LT_RESISTANCE_RC

Description

Linetesting Resistance Measurement via RC network.

Prototype

```
typedef struct
{
    VINETIC_IO_LT_RESISTANCE_MODE mode;
    VINETIC_IO_LT_VOLTAGE_MODE iopin;
    VINETIC_IO_LT_RESULT_CONFIG cfg;
    VINETIC_IO_LT_RESULT res00;
    IFX_int32_t reciprocal_scale;
} VINETIC_IO_LT_RESISTANCE_RC_t;
```

Parameters

Data Type	Name	Description
VINETIC_IO_LT_RESISTANCE_MODE	mode	Linetesting Resistance Measurement Mode Selection (select between Tip/Ground and Ring/Ground. Ring/Tip is not supported!).
VINETIC_IO_LT_VOLTAGE_MODE	iopin	Linetesting Resistance Measurement Submode Selection (the application has to tell the driver which io pins are connected to Tip/Ring).
VINETIC_IO_LT_RESULT_CONFIG	cfg	Level Meter Result Configuration.
VINETIC_IO_LT_RESULT	res	Level Meter Result first voltage.
IFX_int32_t	reciprocal_scale	Reciprocal value of the threshold scaling factor.

7.3.3.21 VINETIC_IO_LT_RESULT

Description

Linetesting Measurement Result.

Prototype

```
typedef struct
{
    IFX_uint16_t nSamples;
    IFX_int32_t nShift;
```



```

    IFX_int32_t nLMRES;
    IFX_uint16_t nGain;
    IFX_uint8_t bRevPol;
    VINETIC_IO_LT_RS rs;
} VINETIC_IO_LT_RESULT_t;

```

Parameters

Data Type	Name	Description
IFX_uint16_t	nSamples	Number of Samples.
IFX_int32_t	nShift	Shift factor used for the measurement Attention: for Vinetic 2.1 a negative shift factor means 1/n Shift.
IFX_int32_t	nLMRES	Level Meter result.
IFX_uint16_t	nGain	Gain factor (only used for Vinetic 1.x).
IFX_uint8_t	bRevPol	Normal or Reverse Polarity.
VINETIC_IO_LT_RS	rs	Rectifier, Squarer or None of them was set.

7.3.3.22 VINETIC_IO_LT_RESULT_CONFIG

Description

Linetesting Measurement Result Configuration.

Prototype

```

typedef struct
{
    IFX_uint8_t nHwRevision;
    IFX_uint16_t nLineMode;
} VINETIC_IO_LT_RESULT_CONFIG_t;

```

Parameters

Data Type	Name	Description
IFX_uint8_t	nHwRevision	Chip Revision.
IFX_uint16_t	nLineMode	Linemode when the measurement was done.

7.3.3.23 VINETIC_IO_LT_VOLTAGE

Description

Linetesting Voltage Measurement.

Prototype

```

typedef struct
{
    VINETIC_IO_LT_VOLTAGE_MODE mode;
    VINETIC_IO_LT_VOLTAGE_SUBMODE submode;
    IFX_uint8_t bForeignVoltage;
}

```

```
VINETIC_IO_LT_RESULT_CONFIG cfg;
VINETIC_IO_LT_RESULT res00;
} VINETIC_IO_LT_VOLTAGE_t;
```

Parameters

Data Type	Name	Description
VINETIC_IO_LT_VOLTAGE_MODE	mode	Voltage measurement mode selection (LM_SEL).
VINETIC_IO_LT_VOLTAGE_SUBMODE	submode	Voltage measurement submode selection (AC, DC).
IFX_uint8_t	bForeignVoltage	Voltage measurement select foreign voltage (if set to TRUE the slic is set to HIRT mode during the measurement).
VINETIC_IO_LT_RESULT_CONFIG	cfg	Level Meter Result Configuration.
VINETIC_IO_LT_RESULT	res	Level Meter Result.

7.3.3.24 VINETIC_IO_MB_CMD
Description

IO structure for write and read chip commands for debugging purposes only.

Prototype

```
typedef struct
{
    IFX_uint16_t cmd1;
    IFX_uint16_t cmd2;
    IFX_uint16_t pData[256];
} VINETIC_IO_MB_CMD_t;
```

Parameters

Data Type	Name	Description
IFX_uint16_t	cmd1	Command 1 according users manual.
IFX_uint16_t	cmd2	Command 2 according users manual.
IFX_uint16_t	pData[256]	Read or write data.

7.3.3.25 VINETIC_IO_OFFSET_CALIBRATION
Description

Offset Calibration.

Prototype

```
typedef struct
{
    IFX_uint8_t nChannel;
    VINETIC_IO_OFFSET_CALIBRATION_MODE nMode;
```

```

VINETIC_IO_OFFSET_VALUE nSlic;
VINETIC_IO_OFFSET_VALUE nPrefi;
VINETIC_IO_OFFSET_VALUE nSlicPregain;
IFX_uint8_t nHwRevision;
IFX_int32_t error;
} VINETIC_IO_OFFSET_CALIBRATION_t;

```

Parameters

Data Type	Name	Description
IFX_uint8_t	nChannel	Select the channel to be offset calibrated.
VINETIC_IO_OFFSET_CALIBRATION_MODE	nMode	Select the offset calibration mode (DC_SLIC, DC_PREFI).
VINETIC_IO_OFFSET_VALUE	nSlic	Returns the offsets for DC_SLIC calibration.
VINETIC_IO_OFFSET_VALUE	nPrefi	Returns the offsets for DC_PREFI calibration.
VINETIC_IO_OFFSET_VALUE	nSlicPregain	Returns the offsets for DC_SLIC calibration with LM_PREGAIN, VINETIC 2.1 only.
IFX_uint8_t	nHwRevision	Returns the chip revision.
IFX_int32_t	error	Returns success or an error code for each channel the according bit is set in an error case e.g. Bit 0 is set for channel 0, etc.

7.3.3.26 VINETIC_IO_OFFSET_VALUE

Description

Offset Calibration Results.

Prototype

```

typedef struct
{
    Short nOFR[VINETIC_CH_NR];
} VINETIC_IO_OFFSET_VALUE_t;

```

Parameters

Data Type	Name	Description
Short	nOFR[VINETIC_CH_NR]	Returns the OFR values for the calibrated channels.

7.3.3.27 VINETIC_IO_READ_SC

Description

IO structure for read short commands.

Prototype

```

typedef struct

```

```
{
    IFX_uint8_t nBc;
    IFX_uint16_t nCmd;
    IFX_uint8_t nCh;
    IFX_uint8_t count;
    IFX_uint16_t pData[256];
} VINETIC_IO_READ_SC_t;
```

Parameters

Data Type	Name	Description
IFX_uint8_t	nBc	If set to 1 a broadcast on all channel will be done. Not available for every command.
IFX_uint16_t	nCmd	Write command.
IFX_uint8_t	nCh	Channel id.
IFX_uint8_t	count	Read words count.
IFX_uint16_t	pData[256]	Read data.

7.3.3.28 VINETIC_IO_TAPI_LT_CONFIG

Description

Part of data structure [VINETIC_IO_LT_CONFIG](#). Provides information to the Linetesting TAPI level whether or not a low-pass filter is used, and also the type of SLIC versions used (since not all SLIC versions can be determined through the BCR1 register).

Prototype

```
typedef struct
{
    unsigned short nLowPassFilter;
    unsigned short nSlicType;
} VINETIC_IO_TAPI_LT_CONFIG_t;
```

Parameters

Data Type	Name	Description
unsigned short	nLowPassFilter	Information if Low Pass Filter is used between the Codec and SLIC (0 if no LPF is used, 1 if LPF is used. [in/out])
unsigned short	nSlicType	Information of the SLIC version used. The SLIC type identifiers are according the enum TAPI_LT_SLIC. [in/out]

7.3.3.29 VINETIC_IO_USR

Description

VINETIC User Interface Structure.

Used for all firmware messages and if messages for input and output are very different, where a simple structure will not be optimal

Prototype

```
typedef struct
{
    IFX_uint8_t* pWrBuf;
    IFX_uint32_t nWrCnt;
    IFX_uint8_t* pRdBuf;
    IFX_uint32_t nRdCnt;
} VINETIC_IO_USR_t;
```

Parameters

Data Type	Name	Description
IFX_uint8_t*	pWrBuf	
IFX_uint32_t	nWrCnt	
IFX_uint8_t*	pRdBuf	
IFX_uint32_t	nRdCnt	

7.3.3.30 VINETIC_IO_VERSION
Description

Version IO structure.

Prototype

```
typedef struct
{
    IFX_uint8_t nType;
    IFX_uint8_t nChannel;
    IFX_uint16_t nChip;
    IFX_uint32_t nTapiVers;
    IFX_uint32_t nDrvVers;
    IFX_uint16_t nEdspVers;
    IFX_uint16_t nEdspIntern;
} VINETIC_IO_VERSION_t;
```

Parameters

Data Type	Name	Description
IFX_uint8_t	nType	Chip type.
IFX_uint8_t	nChannel	Number of supported analog channels.
IFX_uint16_t	nChip	
IFX_uint32_t	nTapiVers	Included TAPI version.
IFX_uint32_t	nDrvVers	Driver version.
IFX_uint16_t	nEdspVers	EDSP major version..
IFX_uint16_t	nEdsplIntern	EDSP version step.

7.3.3.31 VINETIC_IO_WRITE_SC

Description

IO structure for writing short commands.

Prototype

```
typedef struct
{
    IFX_uint8_t nBc;
    IFX_uint16_t nCmd;
    IFX_uint8_t nCh;
} VINETIC_IO_WRITE_SC_t;
```

Parameters

Data Type	Name	Description
IFX_uint8_t	nBc	If set to 1 a broadcast on all channel will be done. Not available for every command.
IFX_uint16_t	nCmd	Write command.
IFX_uint8_t	nCh	Channel id.

7.3.4 Enumerator Reference

This chapter contains the Enumerator reference.

Table 80 Enumerator Overview of Non TAPI Interfaces

Name	Description
DEV_ERR	Driver error codes.
GR909_RESULT_PARAM	Measurement results parameters alignment in result buffer.
GR909_TEST_RESULT	GR909 Measurements status values.
GR909_TESTS	GR909 possible measurements.
VIN_ACCESS	VINETIC® Access Modes.
VINETIC_GPIO_MODE	GPIO pin configuration modes.
VINETIC_IO_CHIP_MAJOR_REVISION	VINETIC® Chip Major Revision.
VINETIC_IO_CHIP_REVISION	VINETIC® Chip Revision.
VINETIC_IO_CHIP_TYPE	VINETIC® chip types, depending on register Revision and firmware download.
VINETIC_IO_CRAM_FORMAT	CRAM file format version.
VINETIC_IO_LT_AC_LM_FILTER_TYPE	Linetesting AC Direct Measurement Level Meter Filter Structure.
VINETIC_IO_LT_AC_LM_SELECT	Linetesting AC Direct Measurement Input Selection.
VINETIC_IO_LT_AC_MODE	Linetesting AC Measurement Mode Selection.
VINETIC_IO_LT_AC_TG_STATE	Linetesting AC Direct Measurement Tone Generator State.
VINETIC_IO_LT_AC_TH_FILTER	Linetesting AC Direct Measurement Transhybrid Filter Configuration.

Table 80 Enumerator Overview of Non TAPI Interfaces (cont'd)

Name	Description
VINETIC_IO_LT_BP_QUAL	Bandpass quality selection (Vinetic 1.x only).
VINETIC_IO_LT_CAPACITANCE_MODE	Linetesting Capacitance Measurement Mode Selection.
VINETIC_IO_LT_CONFIG_MODE	Linetesting Configuration Mode.
VINETIC_IO_LT_CURRENT_MODE	Linetesting Current Measurement Mode Selection.
VINETIC_IO_LT_CURRENT_SUBMODE	Linetesting Current Measurement Submode Selection.
VINETIC_IO_LT_IMPEDANCE_MODE	Linetesting Impedance Measurement Mode Selection.
VINETIC_IO_LT_IMPEDANCE_SUBMODE	Linetesting Impedance Measurement Submode Selection.
VINETIC_IO_LT_RESISTANCE_MODE	Linetesting Resistance Measurement Mode Selection.
VINETIC_IO_LT_RS	Linetesting Magnitude Processing.
VINETIC_IO_LT_TG	Linetesting AC Direct Measurement Tone Generator Selection.
VINETIC_IO_LT_VOLTAGE_MODE	Linetesting Voltage Measurement Mode Selection.
VINETIC_IO_LT_VOLTAGE_SUBMODE	Linetesting Voltage Measurement Submode Selection.
VINETIC_IO_OFFSET_CALIBRATION_MODE	Offset Calibration Mode Selection.

7.3.4.1 DEV_ERR

Description

Driver error codes

Prototype

```
typedef enum
{
    ERR_OK = 0,
    ERR_CERR = 0x01,
    ERR_CIBX_OF = 0x02,
    ERR_HOST = 0x03,
    ERR_MIPS_OL = 0x04,
    ERR_NO_COBX = 0x05,
    ERR_NO_DATA = 0x06,
    ERR_NO_FIBXMS = 0x07,
    ERR_MORE_DATA = 0x08,
    ERR_NO_MBXEMPTY = 0x09,
    ERR_NO_DLRDY = 0x0A,
    ERR_WRONGDATA = 0x0B,
    ERR_OBXML_ZERO = 0x0C,
    ERR_TEST_FAIL = 0x0D,
    ERR_HW_ERR = 0x0E,
    ERR_PIBX_OF = 0x0F,
    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 reported by VINETIC®, see last command.
ERR_CIBX_OF	2 _H	Command inbox overflow reported by VINETIC®.
ERR_HOST	3 _H	Host error reported by VINETIC®.
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 by VINETIC® 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 downstreaming.
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.

Name	Value	Description
ERR_GR909	1D _H	No Data to copy to user space for GR909 measurement.
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 [2CPE].
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.

Name	Value	Description
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.
ERR_UNKNOWN	40 _H	Some internal state occurred, 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.

7.3.4.2 GR909_RESULT_PARAM

Description

Measurement results parameters alignment in result buffer.

Prototype

```
typedef enum
{
    HFV_AC_SCALFACT = 0,
    HFV_RG_DC = 1,
    HFV_RG_DC_AC = 2,
    HFV_RG_AC = 3,
    HFV_RG_PERIOD = 4,
    HFV_TG_DC = 5,
    HFV_TG_DC_AC = 6,
    HFV_TG_AC = 7,
    HFV_TG_PERIOD = 8,
    HFV_TR_DC = 9,
    HFV_TR_DC_AC = 10,
    HFV_TR_AC = 11,
    HFV_TR_PERIOD = 12,
    RF_IT_TR_NP = 0,
    RF_V_TR_NP = 1,
    RF_IT_TR_RP = 2,
    RF_V_TR_RP = 3,
    RF_IT_TG_NP = 4,
    RF_V_TG_NP = 5,
    RF_IT_TG_RP = 6,
    RF_V_TG_RP = 7,
    RF_IT_RG_NP = 8,
    RF_V_RG_NP = 9,
    RF_IT_RG_RP = 10,
    RF_V_RG_RP = 11,
    RO_15MA_IT = 0,
    RO_15MA_VOLT = 1,
    RO_30MA_IT = 2,
    RO_30MA_VOLT = 3,
    RNG_AC_RMS = 0,
```

```

RNG_DC_VOLT = 1,
RNG_SAMPLNR = 2,
RNG_SHIFT = 3,
RNG_IT_RESULT = 4
} GR909_RESULT_PARAM_t;

```

Parameters

Name	Value	Description
HFV_AC_SCALFACT	0 _D	AC scaling factor used to avoid 32-bit overflows.
HFV_RG_DC	1 _D	Ring/Ground DC.
HFV_RG_DC_AC	2 _D	Ring/Ground DC part of AC.
HFV_RG_AC	3 _D	Ring/Ground AC.
HFV_RG_PERIOD	4 _D	Ring/Ground Period.
HFV_TG_DC	5 _D	Tip/Gnd DC.
HFV_TG_DC_AC	6 _D	Tip/Gnd DC part of AC.
HFV_TG_AC	7 _D	Tip/Gnd AC.
HFV_TG_PERIOD	8 _D	Tip/Gnd Period.
HFV_TR_DC	9 _D	Tip/Ring DC.
HFV_TR_DC_AC	10 _D	Tip/Ring DC part of AC.
HFV_TR_AC	11 _D	Tip/Ring AC.
HFV_TR_PERIOD	12 _D	Tip/Ring Period.
RF_IT_TR_NP	0 _D	Resistive Fault current normal polarity Tip/Ring.
RF_V_TR_NP	1 _D	Resistive Fault voltage normal polarity Tip/Ring.
RF_IT_TR_RP	2 _D	Resistive Fault current reverse polarity Tip/Ring.
RF_V_TR_RP	3 _D	Resistive Fault voltage reverse polarity Tip/Ring.
RF_IT_TG_NP	4 _D	Resistive Fault current normal polarity Tip/Ground.
RF_V_TG_NP	5 _D	Resistive Fault voltage normal polarity Tip/Ground.
RF_IT_TG_RP	6 _D	Resistive Fault current reverse polarity Tip/Ground.
RF_V_TG_RP	7 _D	Resistive Fault voltage reverse polarity Tip/Ground.
RF_IT_RG_NP	8 _D	Resistive Fault current normal polarity Ring/Gnd.
RF_V_RG_NP	9 _D	Resistive Fault voltage normal polarity Ring/Gnd.
RF_IT_RG_RP	10 _D	Resistive Fault current reverse polarity Ring/Gnd.

Name	Value	Description
RF_V_RG_RP	11 _D	Resistive Fault voltage reverse polarity Ring/Gnd.
RO_15MA_IT	0 _D	Receiver Offhook current for 15 mA DC regulation.
RO_15MA_VOLT	1 _D	Receiver Offhook DCN-DCP feed voltage for 15 mA DC regulation.
RO_30MA_IT	2 _D	Receiver Offhook current for 30 mA DC regulation.
RO_30MA_VOLT	3 _D	Receiver Offhook DCN-DCP feed voltage for 30 mA DC regulation.
RNG_AC_RMS	0 _D	Ringer AC rms.
RNG_DC_VOLT	1 _D	Ringer DC.
RNG_SAMPLNR	2 _D	Ringer sample rate.
RNG_SHIFT	3 _D	Ringer shift factor.
RNG_IT_RESULT	4 _D	Ringer current.

7.3.4.3 GR909_TEST_RESULT

Description

GR909 Measurements status values.

Prototype

```
typedef enum
{
    GR909_TEST_OK = 0x0,
    GR909_TEST_RUNNING = 0x1,
    GR909_TEST_FAILED = 0x2,
    GR909_MEASUREMENT_FAILED = 0x07
} GR909_TEST_RESULT_t;
```

Parameters

Name	Value	Description
GR909_TEST_OK	0 _H	GR909 Test OK.
GR909_TEST_RUNNING	1 _H	GR909 Test ongoing.
GR909_TEST_FAILED	2 _H	GR909 test failed.
GR909_MEASUREMENT_FAILED	07 _H	GR909 Measurement failed due to an internal error.

7.3.4.4 GR909_TESTS

Description

GR909 possible measurements.

Prototype

```
typedef enum
{
    GR909_HAZARDOUS_VOLTAGE = 0,
    GR909_FOREIGN_FORCE = 1,
    GR909_RESISTIVE_FAULT = 2,
    GR909_RECEIVER_OFF_HOOK = 3,
    GR909_RINGER = 4,
    MAX_GR909_TESTS = 5
} GR909_TESTS_t;
```

Parameters

Name	Value	Description
GR909_HAZARDOUS_VOLTAGE	0 _D	GR909 Hazardous Voltage measurement.
GR909_FOREIGN_FORCE	1 _D	GR909 Foreign Force measurement.
GR909_RESISTIVE_FAULT	2 _D	GR909 Resistive Fault measurement.
GR909_RECEIVER_OFF_HOOK	3 _D	GR909 Receiver Offhook measurement.
GR909_RINGER	4 _D	GR909 Ringer measurement.
MAX_GR909_TESTS	5 _D	Number of possible GR909 measurements

7.3.4.5 VIN_ACCESS
Description

VINETIC® Access Modes.

Only available in VINETIC® Version 2.1: 8-bit INTEL Mux, 8-bit INTEL Demux, 8-bit Motorola, and SCI.

Prototype

```
typedef enum
{
    VIN_ACCESS_SPI = 0,
    VIN_ACCESS_SCI = 1,
    VIN_ACCESS_PAR_16BIT = 2,
    VIN_ACCESS_PAR_8BIT = 3,
    VIN_ACCESS_PARINTEL_DMUX16 = 4,
    VIN_ACCESS_PARINTEL_MUX16 = 5,
    VIN_ACCESS_PARINTEL_MUX8 = 6,
    VIN_ACCESS_PARINTEL_DMUX8 = 7,
    VIN_ACCESS_PARINTEL_DMUX8_BE = 8,
    VIN_ACCESS_PARINTEL_DMUX8_LE = 9,
    VIN_ACCESS_PAR_8BIT_V2 = 10
} VIN_ACCESS_t;
```

Parameters

Name	Value	Description
VIN_ACCESS_SPI	0 _D	Host interface used, SPI.
VIN_ACCESS_SCI	1 _D	Host interface used, SCI.
VIN_ACCESS_PAR_16BIT	2 _D	Host interface used, parallel Motorola 16 bit.
VIN_ACCESS_PAR_8BIT	3 _D	Host interface used, parallel Motorola 8 bit.
VIN_ACCESS_PARINTEL_DMUX16	4 _D	Host interface used, parallel Intel 16 bit demultiplexed.
VIN_ACCESS_PARINTEL_MUX16	5 _D	Host interface used, parallel Intel 16 bit multiplexed.
VIN_ACCESS_PARINTEL_MUX8	6 _D	Host interface used, parallel Intel 8 bit multiplexed.
VIN_ACCESS_PARINTEL_DMUX8	7 _D	Host interface used, parallel Intel 8 bit demultiplexed.
VIN_ACCESS_PARINTEL_DMUX8_BE	8 _D	Access parallel intel demux 8-bit big endian. Supported by V1 chip version. This mode should not be used anymore.
VIN_ACCESS_PARINTEL_DMUX8_LE	9 _D	Access parallel intel demux 8-bit little endian. Supported by V2 chip version. This mode should not be used anymore.
VIN_ACCESS_PAR_8BIT_V2	10 _D	Access parallel motorola 8-bit big endian with 16-bit processor interface. Supported by V2 chip version. This mode should not be used anymore.

7.3.4.6 VINETIC_GPIO_MODE
Description

GPIO and IO pin configuration modes.

Prototype

```
typedef enum
{
    GPIO_MODE_INPUT = 0x100,
    GPIO_MODE_OUTPUT = 0x200,
    GPIO_MODE_INT = 0x400,
    GPIO_INT_RISING = 0x1000,
    GPIO_INT_FALLING = 0x2000,
    GPIO_INT_DUP_05 = 0x0000,
    GPIO_INT_DUP_45 = 0x0001,
    GPIO_INT_DUP_85 = 0x0002,
    GPIO_INT_DUP_125 = 0x0003,
    GPIO_INT_DUP_165 = 0x0004,
    GPIO_INT_DUP_205 = 0x0005,
    GPIO_INT_DUP_245 = 0x0006,
    GPIO_INT_DUP_285 = 0x0007,
    GPIO_INT_DUP_325 = 0x0008,
```

```

        GPIO_INT_DUP_365 = 0x0009,
        GPIO_INT_DUP_405 = 0x000A,
        GPIO_INT_DUP_445 = 0x000B,
        GPIO_INT_DUP_485 = 0x000C,
        GPIO_INT_DUP_525 = 0x000D,
        GPIO_INT_DUP_565 = 0x000E,
        GPIO_INT_DUP_605 = 0x000F
    } VINETIC_GPIO_MODE_t;

```

Parameters

Name	Value	Description
GPIO_MODE_INPUT	100 _H	
GPIO_MODE_OUTPUT	200 _H	
GPIO_MODE_INT	400 _H	
GPIO_INT_RISING	1000 _H	
GPIO_INT_FALLING	2000 _H	
GPIO_INT_DUP_05	0000 _H	
GPIO_INT_DUP_45	0001 _H	
GPIO_INT_DUP_85	0002 _H	
GPIO_INT_DUP_125	0003 _H	
GPIO_INT_DUP_165	0004 _H	
GPIO_INT_DUP_205	0005 _H	
GPIO_INT_DUP_245	0006 _H	
GPIO_INT_DUP_285	0007 _H	
GPIO_INT_DUP_325	0008 _H	
GPIO_INT_DUP_365	0009 _H	
GPIO_INT_DUP_405	000A _H	
GPIO_INT_DUP_445	000B _H	
GPIO_INT_DUP_485	000C _H	
GPIO_INT_DUP_525	000D _H	
GPIO_INT_DUP_565	000E _H	
GPIO_INT_DUP_605	000F _H	

7.3.4.7 VINETIC_IO_CHIP_MAJOR_REVISION

Description

VINETIC® Chip Major Revision.

Prototype

```

typedef enum
{
    VINETIC_V1x = 1,
    VINETIC_V2x = 2
} VINETIC_IO_CHIP_MAJOR_REVISION_t;

```


Parameters

Name	Value	Description
VINETIC_V1x	1 _D	Version V1.x.
VINETIC_V2x	2 _D	Version V2.x.

7.3.4.8 VINETIC_IO_CHIP_REVISION
Description

VINETIC® Chip Revision.

Prototype

```
typedef enum
{
    VINETIC_V13 = 0x42,
    VINETIC_V14 = 0x84,
    VINETIC_V15 = 0x85,
    VINETIC_V16 = 0x86,
    VINETIC_V21 = 0x90,
    VINETIC_V22 = 0xA0,
    VINETIC_V21_S = 0xB0
} VINETIC_IO_CHIP_REVISION_t;
```

Parameters

Name	Value	Description
VINETIC_V13	42 _H	Version V1.3.
VINETIC_V14	84 _H	Version V1.4.
VINETIC_V15	85 _H	Version V1.5, no longer available.
VINETIC_V16	86 _H	Version V1.6, internal version.
VINETIC_V21	90 _H	Version V2.1, VINETIC®-4M.
VINETIC_V22	A0 _H	Version V2.2, VINETIC®-4M/-4C.
VINETIC_V21_S	B0 _H	Version V2.1 of VINETIC®-4S.

7.3.4.9 VINETIC_IO_CHIP_TYPE
Description

VINETIC® chip types, depending on register REVISION and firmware download.

Prototype

```
typedef enum
{
    VINETIC_TYPE_S = 0x0,
    VINETIC_TYPE_M = 0x1,
    VINETIC_TYPE_VIP = 0x2,
    VINETIC_TYPE_C = 0x4
} VINETIC_IO_CHIP_TYPE_t;
```

Parameters

Name	Value	Description
VINETIC_TYPE_S	0 _H	Chip type S.
VINETIC_TYPE_M	1 _H	Chip type M.
VINETIC_TYPE_VIP	2 _H	Chip type VIP.
VINETIC_TYPE_C	4 _H	Chip type C.

7.3.4.10 VINETIC_IO_CRAM_FORMAT
Description

CRAM file format version.

Prototype

```
typedef enum
{
    VINETIC_IO_CRAM_FORMAT_1 = 0,
    VINETIC_IO_CRAM_FORMAT_2_1 = 1,
    VINETIC_IO_CRAM_FORMAT_2_2 = 2
} VINETIC_IO_CRAM_FORMAT_t;
```

Parameters

Name	Value	Description
VINETIC_IO_CRAM_FORMAT_1	0 _D	CRAM download format for V1.6 and older.
VINETIC_IO_CRAM_FORMAT_2_1	1 _D	CRAM download format 2.1 for VINETIC V2.x.
VINETIC_IO_CRAM_FORMAT_2_2	2 _D	CRAM download format 2.2 for VINETIC V2.x.

7.3.4.11 VINETIC_IO_LT_AC_LM_FILTER_TYPE
Description

Linetesting AC Direct Measurement Level Meter Filter Structure.

Prototype

```
typedef enum
{
    VINETIC_IO_LT_AC_LM_FILTER_OFF = ,
    VINETIC_IO_LT_AC_LM_FILTER_BP = ,
    VINETIC_IO_LT_AC_LM_FILTER_NOTCH = ,
    VINETIC_IO_LT_AC_LM_FILTER_LP = ,
    VINETIC_IO_LT_AC_LM_FILTER_HP = ,
    VINETIC_IO_LT_AC_LM_FILTER_IGNORE = 0xFF
} VINETIC_IO_LT_AC_LM_FILTER_TYPE_t;
```

Parameters

Name	Value	Description
VINETIC_IO_LT_AC_LM_FILTER_OFF		Select to bypass the level meter filter.
VINETIC_IO_LT_AC_LM_FILTER_BP		Select a bandpass filter structure in the level meter filter.
VINETIC_IO_LT_AC_LM_FILTER_NOTCH		Select a notch filter structure in the level meter filter.
VINETIC_IO_LT_AC_LM_FILTER_LP		Select a low pass filter structure in the level meter filter.
VINETIC_IO_LT_AC_LM_FILTER_HP		Select a high pass filter structure in the level meter filter.
VINETIC_IO_LT_AC_LM_FILTER_IGNORE	0xFF	Don't change the filter settings.

7.3.4.12 VINETIC_IO_LT_AC_LM_SELECT
Description

Linetesting AC Direct Measurement Input Selection.

Prototype

```
typedef enum
{
    VINETIC_IO_LT_AC_LM_SELECT_TX = 0,
    VINETIC_IO_LT_AC_LM_SELECT_RX = 1,
    VINETIC_IO_LT_AC_LM_SELECT_RX_TX = 2
} VINETIC_IO_LT_AC_LM_SELECT_t;
```

Parameters

Name	Value	Description
VINETIC_IO_LT_AC_LM_SELECT_TX	0 _D	Select TX path to be measured.
VINETIC_IO_LT_AC_LM_SELECT_RX	1 _D	Select RX path to be measured.
VINETIC_IO_LT_AC_LM_SELECT_RX_TX	2 _D	Select sum of RX and TX path to be measured.

7.3.4.13 VINETIC_IO_LT_AC_MODE
Description

Linetesting AC Measurement Mode Selection.

Prototype

```
typedef enum
{
    VINETIC_IO_LT_AC_MODE_LEVEL = 0,
    VINETIC_IO_LT_AC_MODE_GAINTRACKING = 1,
    VINETIC_IO_LT_AC_MODE_TRANSHYBRID = 2,
    VINETIC_IO_LT_AC_MODE_IDLENOISE = 3,
    VINETIC_IO_LT_AC_MODE_SNR = 4
}
```

```
} VINETIC_IO_LT_AC_MODE_t;
```

Parameters

Name	Value	Description
VINETIC_IO_LT_AC_MODE_LEVEL	0 _D	Select the PCM4 Level Measurement.
VINETIC_IO_LT_AC_MODE_GAINTRACKING	1 _D	Select the PCM4 GainTracking Measurement.
VINETIC_IO_LT_AC_MODE_TRANSHYBRID	2 _D	Select the PCM4 Transhybrid Measurement.
VINETIC_IO_LT_AC_MODE_IDLENOISE	3 _D	Select the PCM4 Idle Channel Noise Measurement.
VINETIC_IO_LT_AC_MODE_SNR	4 _D	Select the PCM4 SNR Measurement.

7.3.4.14 VINETIC_IO_LT_AC_TG_STATE

Description

Linetesting AC Direct Measurement Tone Generator State.

Prototype

```
typedef enum
{
    VINETIC_IO_LT_AC_TG_STATE_OFF = 0,
    VINETIC_IO_LT_AC_TG_STATE_ON = 1,
    VIENTIC_IO_LT_AC_TG_STATE_IGNORE = 2
} VINETIC_IO_LT_AC_TG_STATE_t;
```

Parameters

Name	Value	Description
VINETIC_IO_LT_AC_TG_STATE_OFF	0 _D	Switch tone generator off.
VINETIC_IO_LT_AC_TG_STATE_ON	1 _D	Switch tone generator on.
VIENTIC_IO_LT_AC_TG_STATE_IGNORE	2 _D	Do not change the tone generator configuration.

7.3.4.15 VINETIC_IO_LT_AC_TH_FILTER

Description

Linetesting AC Direct Measurement Transhybrid Filter Configuration.

Prototype

```
typedef enum
{
    VINETIC_IO_LT_AC_TH_FILTER_ON = 0,
    VINETIC_IO_LT_AC_TH_FILTER_OFF = 1
} VINETIC_IO_LT_AC_TH_FILTER_t;
```

Parameters

Name	Value	Description
VINETIC_IO_LT_AC_TH_FILTER_ON	0 _D	Switch transhybrid filter on during the measurement.
VINETIC_IO_LT_AC_TH_FILTER_OFF	1 _D	Switch transhybrid filter off during the measurement.

7.3.4.16 VINETIC_IO_LT_BP_QUAL
Description

Bandpass quality selection (Vinetic 1.x only).

Prototype

```
typedef enum
{
    VINETIC_IO_LT_BP_QUAL_1 = 0,
    VINETIC_IO_LT_BP_QUAL_2 = 1,
    VINETIC_IO_LT_BP_QUAL_4 = 2,
    VINETIC_IO_LT_BP_QUAL_8 = 3,
    VINETIC_IO_LT_BP_QUAL_16 = 4,
    VINETIC_IO_LT_BP_QUAL_32 = 5,
    VINETIC_IO_LT_BP_QUAL_64 = 6,
    VINETIC_IO_LT_BP_QUAL_128 = 7
} VINETIC_IO_LT_BP_QUAL_t;
```

Parameters

Name	Value	Description
VINETIC_IO_LT_BP_QUAL_1	0 _D	Select bandpass quality 1.
VINETIC_IO_LT_BP_QUAL_2	1 _D	Select bandpass quality 2.
VINETIC_IO_LT_BP_QUAL_4	2 _D	Select bandpass quality 4.
VINETIC_IO_LT_BP_QUAL_8	3 _D	Select bandpass quality 8.
VINETIC_IO_LT_BP_QUAL_16	4 _D	Select bandpass quality 16.
VINETIC_IO_LT_BP_QUAL_32	5 _D	Select bandpass quality 32.
VINETIC_IO_LT_BP_QUAL_64	6 _D	Select bandpass quality 64.
VINETIC_IO_LT_BP_QUAL_128	7 _D	Select bandpass quality 128.

7.3.4.17 VINETIC_IO_LT_CAPACITANCE_MODE
Description

Linetesting Capacitance Measurement Mode Selection.

Prototype

```
typedef enum
{
    VINETIC_IO_LT_CAPACITANCE_MODE_RING_TIP = 0,
```

```
VINETIC_IO_LT_CAPACITANCE_MODE_RING_GROUND = 1,
VINETIC_IO_LT_CAPACITANCE_MODE_TIP_GROUND = 2
} VINETIC_IO_LT_CAPACITANCE_MODE_t;
```

Parameters

Name	Value	Description
VINETIC_IO_LT_CAPACITANCE_MODE_RING_TIP	0 _D	Capacitance will be measured between Ring and Tip.
VINETIC_IO_LT_CAPACITANCE_MODE_RING_GROUND	1 _D	Capacitance will be measured between Ring and Ground.
VINETIC_IO_LT_CAPACITANCE_MODE_TIP_GROUND	2 _D	Capacitance will be measured between Tip and Ground.

7.3.4.18 VINETIC_IO_LT_CONFIG_MODE

Description

Linetesting Configuration Mode.

Prototype

```
typedef enum
{
    VINETIC_IO_LT_CONFIG_READ = 0,
    VINETIC_IO_LT_CONFIG_WRITE = 1
} VINETIC_IO_LT_CONFIG_MODE_t;
```

Parameters

Name	Value	Description
VINETIC_IO_LT_CONFIG_READ	0 _D	Read out the current linetesting configuration.
VINETIC_IO_LT_CONFIG_WRITE	1 _D	Configure the driver with new parameters.

7.3.4.19 VINETIC_IO_LT_CURRENT_MODE

Description

Linetesting Current Measurement Mode Selection.

Prototype

```
typedef enum
{
    VINETIC_IO_LT_CURRENT_MODE_IT = 0,
    VINETIC_IO_LT_CURRENT_MODE_IL = 1
} VINETIC_IO_LT_CURRENT_MODE_t;
```

Parameters

Name	Value	Description
VINETIC_IO_LT_CURRENT_MODE_IT	0 _D	Select DC Current IT.
VINETIC_IO_LT_CURRENT_MODE_IL	1 _D	Select DC Current IL.

7.3.4.20 VINETIC_IO_LT_CURRENT_SUBMODE
Description

Linetesting Current Measurement Submode Selection.

Prototype

```
typedef enum
{
    VINETIC_IO_LT_CURRENT_SUBMODE_DC = 0,
    VINETIC_IO_LT_CURRENT_SUBMODE_AC = 1
} VINETIC_IO_LT_CURRENT_SUBMODE_t;
```

Parameters

Name	Value	Description
VINETIC_IO_LT_CURRENT_SUBMODE_DC	0 _D	Measure the DC part of the current.
VINETIC_IO_LT_CURRENT_SUBMODE_AC	1 _D	Measure the AC part of the current.

7.3.4.21 VINETIC_IO_LT_IMPEDANCE_MODE
Description

Linetesting Impedance Measurement Mode Selection.

Prototype

```
typedef enum
{
    VINETIC_IO_LT_IMPEDANCE_MODE_RING_TIP = 0,
    VINETIC_IO_LT_IMPEDANCE_MODE_RING_GROUND = 1,
    VINETIC_IO_LT_IMPEDANCE_MODE_TIP_GROUND = 2
} VINETIC_IO_LT_IMPEDANCE_MODE_t;
```

Parameters

Name	Value	Description
VINETIC_IO_LT_IMPEDANCE_MODE_RING_TIP	0 _D	Impedance will be measured between Ring and Tip.
VINETIC_IO_LT_IMPEDANCE_MODE_RING_GROUND	1 _D	Impedance will be measured between Ring and Ground.
VINETIC_IO_LT_IMPEDANCE_MODE_TIP_GROUND	2 _D	Impedance will be measured between Tip and Ground.

7.3.4.22 VINETIC_IO_LT_IMPEDANCE_SUBMODE

Description

Linetesting Impedance Measurement Submode Selection.

Prototype

```
typedef enum
{
    VINETIC_IO_LT_IMPEDANCE_SUBMODE_RG = 0,
    VINETIC_IO_LT_IMPEDANCE_SUBMODE_TG = 1
} VINETIC_IO_LT_IMPEDANCE_SUBMODE_t;
```

Parameters

Name	Value	Description
VINETIC_IO_LT_IMPEDANCE_SUBMODE_RG	0 _D	Select the Ring Generator as signal source for the Impedance measurement.
VINETIC_IO_LT_IMPEDANCE_SUBMODE_TG	1 _D	Select the Tone Generator as signal source for the Impedance measurement.

7.3.4.23 VINETIC_IO_LT_RESISTANCE_MODE

Description

Linetesting Resistance Measurement Mode Selection.

Prototype

```
typedef enum
{
    VINETIC_IO_LT_RESISTANCE_MODE_RING_TIP = 0,
    VINETIC_IO_LT_RESISTANCE_MODE_RING_GROUND = 1,
    VINETIC_IO_LT_RESISTANCE_MODE_TIP_GROUND = 2
} VINETIC_IO_LT_RESISTANCE_MODE_t;
```

Parameters

Name	Value	Description
VINETIC_IO_LT_RESISTANCE_MODE_RING_TIP	0 _D	Resistance will be measured between Ring and Tip.
VINETIC_IO_LT_RESISTANCE_MODE_RING_GROUND	1 _D	Resistance will be measured between Ring and Ground.
VINETIC_IO_LT_RESISTANCE_MODE_TIP_GROUND	2 _D	Resistance will be measured between Tip and Ground.

7.3.4.24 VINETIC_IO_LT_RS

Description

Linetesting Magnitude Processing.

Prototype

```
typedef enum
{
    VINETIC_IO_LT_RS_SQUARER = 0,
    VINETIC_IO_LT_RS_RECTIFIER = 1,
    VINETIC_IO_LT_RS_NONE = 2
} VINETIC_IO_LT_RS_t;
```

Parameters

Name	Value	Description
VINETIC_IO_LT_RS_SQUARER	0 _D	Activate a squarer before the levelmeter.
VINETIC_IO_LT_RS_RECTIFIER	1 _D	Activate a rectifier before the levelmeter.
VINETIC_IO_LT_RS_NONE	2 _D	Deactivate level meter magnitude processing.

7.3.4.25 VINETIC_IO_LT_TG

Description

Linetesting AC Direct Measurement Tone Generator Selection.

Prototype

```
typedef enum
{
    VINETIC_IO_LT_TG_1 = 0,
    VINETIC_IO_LT_TG_2 = 1
} VINETIC_IO_LT_TG_t;
```

Parameters

Name	Value	Description
VINETIC_IO_LT_TG_1	0 _D	Select tone generator 1.
VINETIC_IO_LT_TG_2	1 _D	Select tone generator 2.

7.3.4.26 VINETIC_IO_LT_VOLTAGE_MODE

Description

Linetesting Voltage Measurement Mode Selection.

Prototype

```
typedef enum
{
    VINETIC_IO_LT_VOLTAGE_MODE_IO4_IO3 = 0,
```

```

VINETIC_IO_LT_VOLTAGE_MODE_IO4_IO2 = 1,
VINETIC_IO_LT_VOLTAGE_MODE_IO3_IO2 = 2,
VINETIC_IO_LT_VOLTAGE_MODE_DCN_DCP = 3,
VINETIC_IO_LT_VOLTAGE_MODE_IO4 = 4,
VINETIC_IO_LT_VOLTAGE_MODE_IO3 = 5,
VINETIC_IO_LT_VOLTAGE_MODE_IO2 = 6,
VINETIC_IO_LT_VOLTAGE_MODE_VDD = 7
} VINETIC_IO_LT_VOLTAGE_MODE_t;

```

Parameters

Name	Value	Description
VINETIC_IO_LT_VOLTAGE_MODE_IO4_IO3	0 _D	Select differential voltage measurement IO4 - IO3.
VINETIC_IO_LT_VOLTAGE_MODE_IO4_IO2	1 _D	Select differential voltage measurement IO4 - IO2.
VINETIC_IO_LT_VOLTAGE_MODE_IO3_IO2	2 _D	Select differential voltage measurement IO3 - IO2.
VINETIC_IO_LT_VOLTAGE_MODE_DCN_DCP	3 _D	Select voltage measurement DCN-DCP.
VINETIC_IO_LT_VOLTAGE_MODE_IO4	4 _D	Select voltage measurement on IO4.
VINETIC_IO_LT_VOLTAGE_MODE_IO3	5 _D	Select voltage measurement on IO3.
VINETIC_IO_LT_VOLTAGE_MODE_IO2	6 _D	Select voltage measurement on IO2.
VINETIC_IO_LT_VOLTAGE_MODE_VDD	7 _D	Select voltage measurement on VDD.

7.3.4.27 VINETIC_IO_LT_VOLTAGE_SUBMODE

Description

Linetesting Voltage Measurement Submode Selection.

Prototype

```

typedef enum
{
    VINETIC_IO_LT_VOLTAGE_SUBMODE_DC = 0,
    VINETIC_IO_LT_VOLTAGE_SUBMODE_AC = 1
} VINETIC_IO_LT_VOLTAGE_SUBMODE_t;

```

Parameters

Name	Value	Description
VINETIC_IO_LT_VOLTAGE_SUBMODE_DC	0 _D	Select DC voltage measurement.
VINETIC_IO_LT_VOLTAGE_SUBMODE_AC	1 _D	Select AC voltage measurement.

7.3.4.28 VINETIC_IO_OFFSET_CALIBRATION_MODE

Description

Offset Calibration Mode Selection.

Prototype

```
typedef enum
{
    VINETIC_IO_OFFSET_CALIBRATION_MODE_DC_SLIC = ,
    VINETIC_IO_OFFSET_CALIBRATION_MODE_DC_PREFI = ,
    VINETIC_IO_OFFSET_CALIBRATION_MODE_DC_SLIC_PREGAIN = ,
    VINETIC_IO_OFFSET_CALIBRATION_MODE_ALL = 0xFF
} VINETIC_IO_OFFSET_CALIBRATION_MODE_t;
```

Parameters

Name	Value	Description
VINETIC_IO_OFFSET_CALIBRATION_MODE_DC_SLIC		Select the DC_SLIC offset calibration to be done.
VINETIC_IO_OFFSET_CALIBRATION_MODE_DC_PREFI		Select the DC_PREFI offset calibration to be done.
VINETIC_IO_OFFSET_CALIBRATION_MODE_DC_SLIC_PREGAIN		Select the DC_SLIC offset with LM_PREGAIN enabled, VINETIC 2.1 only.
VINETIC_IO_OFFSET_CALIBRATION_MODE_ALL	FF _H	Select the all offset calibrations to be done.

8 Line Testing Interfaces

This section describes line testing interfaces.

- [Chapter 8.1](#) provides a reference for the line testing functions.
- [Chapter 8.2](#) provides the reference for all types defined for line testing, including
 - [Chapter 8.2.1](#), struct reference
 - [Chapter 8.2.2](#), enum reference

8.1 Function Reference

This chapter contains the function reference.

Table 81 Function Overview of Line Testing Interfaces

Name	Description
Ifxphone_LT_AC_Gain_Tracking	Executes a gaintracking measurement, similar to a PCM4.
Ifxphone_LT_AC_Idle_Noise	This service executes a idle channel noise measurement.
Ifxphone_LT_AC_Level	Service for AC level measurement.
Ifxphone_LT_AC_Level_Measurement	Executes a AC level measurement similar to a PCM4.
Ifxphone_LT_AC_SNR	This service executes a SNR measurement.
Ifxphone_LT_AC_Transhybrid	This service executes a transhybrid measurement.
Ifxphone_LT_Capacitance	Capacitance measurement.
Ifxphone_LT_Charge	Performs charging of any capacitance between tip/ring to the original voltage found during the preceeding execution of the Ifxphone_LT_Discharge function.
Ifxphone_LT_Current	This service executes a current measurement.
Ifxphone_LT_Discharge	Performs discharging to the specified voltage level for a given channel.
Ifxphone_LT_GR909_Config	Configure system parameters (SLIC) to use for values calculation, e.g Voltage divider resistors.
Ifxphone_LT_GR909_GetResults	Gets GR-909 measurement results.
Ifxphone_LT_GR909_Start	Start a GR-909 test or test sequence according to measurement mask, IFX_LT_GR909_MASK_t .
Ifxphone_LT_Impedance	This service executes an impedance measurement.
Ifxphone_LT_Network	Performs the so-called group test where a set of linetesting functions are called in order to gather a set of capacitive and resistive measurements which are then used to calculate all resistive and capacitive components of the network.
Ifxphone_LT_Network_Ground	Performs the so-called Network to ground measurment procedure measuring the ring/ground and tip/ground capacitances and resistances of the local loop.
Ifxphone_LT_Network_Tip_Ring	Performs Network Tip/Ring measurment with Receiver Off-hook indication, measuring the capacitances and the resistances of the local loop.
Ifxphone_LT_Resistance	This service executes a resistance measurement.
Ifxphone_LT_Resistance_RC	Is intended to measure high ohmic resistances with an alternative method compared to the above mentioned Resistance measurement.

Table 81 Function Overview of Line Testing Interfaces (cont'd)

Name	Description
Ifxphone_LT_Voltage	This service executes a voltage measurement.
TAPI_LT_Configure	Performs Linetesting configuration.

8.1.1 Ifxphone_LT_AC_Gain_Tracking

Description

This service executes a gain tracking measurement, similar to a PCM4.

Usage of the input parameters of [IFXPHONE_LTEST](#) struct

- Field mode: not used.
- Field submode: not used.
- Field values[[LTEST_FREQ](#)] represents the tone frequency in Hz used for gaintracking measurement.
- Field values[[LTEST_LEVEL](#)] represents the tone level for gaintracking measurement in 0.001 dBm0.

Output fields of [IFXPHONE_LTEST](#) struct

- Field result: result of measurement in 0.001 dBm0.
- Field error: error return value.

Prototype

```
IFX_int32_t Ifxphone_LT_AC_Gain_Tracking (
    IFX_int32_t fd,
    IFXPHONE_LTEST *pLineTest );
```

Parameters

Data Type	Name	Description
IFX_int32_t	fd	File Descriptor
IFXPHONE_LTEST	*pLineTest	Pointer to IFXPHONE_LTEST .

Return Values

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

Example

```
typedef struct
{
    int mode;
    int submode;
    float result;
    int values[5];
    int error;
} ltest;
ltest LineTest;
```

```

/* set line testing structure */
memset (&LineTest, 0, sizeof(LineTest));
LineTest.values[LTEST_FREQ] = 500; /* measure with 500 Hz */
LineTest.values[LTEST_LEVEL] = -10000; /* measure with -10.0dB */

/* execute measurement */
Ifxphone_LT_AC_Gain_Tracking (fd, &LineTest);

/* print measurement result */
printf("used measurement freq %d Hz, ", LineTest.freq);
if (LineTest.Error != LTEST_SUCCESSFUL)
    printf("LTEST error happened!");
else
    printf("LTEST successful with result %f dB",
        LineTest.result / 1000);

```

8.1.2 Ifxphone_LT_AC_Idle_Noise

Description

This service executes a idle channel noise measurement.

Usage of the input parameters of **IFXPHONE_LTEST** struct

- Field mode: not used.
- Field submode: not used.
- Field values[]: not used.

Output fields of **IFXPHONE_LTEST** struct

- Field result: result of measurement in 0.001 dBm0.
- Field error: error return value.

Prototype

```

IFX_void_t Ifxphone_LT_AC_Idle_Noise (
    IFX_int32_t fd,
    IFXPHONE_LTEST *pLineTest );

```

Parameters

Data Type	Name	Description
IFX_int32_t	fd	File Descriptor.
IFXPHONE_LTEST	*pLineTest	Pointer to IFXPHONE_LTEST .

Return Values

Data Type	Description
IFX_void_t	The return value not used.

Example

```

typedef struct
{
    int mode;

```

```

    int submode;
    float result;
    int values[5];
    int error;
} ltest;
ltest LineTest;

/* set line testing structure */
memset (&LineTest, 0, sizeof(LineTest));

/* execute measurement */
Ifxphone_LT_AC_Idle_Noise (fd, &LineTest);

/* print measurement result */
if (LineTest.Error != LTEST_SUCCESSFUL)
    printf("LTEST error happened!");
else
    printf("LTEST successful with result %f dBm0",
        LineTest.result / 10000);

```

8.1.3 Ifxphone_LT_AC_Level

Description

Service for AC level measurement.

Usage of the input parameters of **IFXPHONE_LTEST** struct

- Field mode: **LTEST_RX** to perform measurement in receive direction, **LTEST_TX** to perform measurement in transmit direction, **LTEST_RXTX** to perform measurement in receive and transmit direction,
- Field submode: **LTEST_NO_TONE** to perform measurement without sending a tone, **LTEST_NO_TONE** to send a tone.
- Field values[**LTEST_FREQ**] represents the tone frequency in Hz used for the measurement.
- Field values[**LTEST_LEVEL**] represents the tone level for the measurement in 0.001 dBm0.

Output fields of **IFXPHONE_LTEST** struct

- Field result: result of measurement in 0.001 dBm0.
- Field error: error return value.

Prototype

```

IFX_void_t Ifxphone_LT_AC_Level (
    IFX_int32_t fd,
    IFXPHONE_LTEST *pLineTest );

```

Parameters

Data Type	Name	Description
IFX_int32_t	fd	File Descriptor.
IFXPHONE_LTEST	*pLineTest	Pointer to IFXPHONE_LTEST .

Return Values

Data Type	Description
IFX_void_t	The return value not used.

Example

```
typedef struct
{
    int mode;
    int submode;
    float result;
    int values[5];
    int error;
} ltest;
ltest LineTest;

/* set line testing structure */
memset (&LineTest, 0, sizeof(LineTest));
LineTest.mode = LTEST_TX;
LineTest.submode = LTEST_TONE;
LineTest.values[LTEST_FREQ] = 500; /* measure with 500 Hz */
LineTest.values[LTEST_LEVEL] = -10000; /* measure with -10.0dB */

/* execute measurement */
Ifxphone_LT_AC_Level (fd, &LineTest);

/* print measurement result */
if (LineTest.Error != LTEST_SUCCESSFUL)
    printf("LTEST error happened!");
else
    printf("LTEST successful with result %f dBm0",
        LineTest.result / 1000);
```

8.1.4 Ifxphone_LT_AC_Level_Measurement

Description

This service executes a AC level measurement similar to a PCM4. The parameters cannot be configured as flexible as for Ifxphone_LT_AC_Level.

Usage of the input parameters of [IFXPHONE_LTEST](#) struct

- Field mode: not used.
- Field submode: not used.
- Field values[LTEST_FREQ] represents the tone frequency in Hz used for the measurement.
- Field values[LTEST_LEVEL] represents the tone level for the measurement in 0.001 dBm0.

Output fields of [IFXPHONE_LTEST](#) struct

- Field result: result of measurement in 0.001 dBm0.
- Field error: error return value.

Prototype

```
IFX_void_t Ifxphone_LT_AC_Level_Measurement (
    IFX_int32_t fd,
    IFXPHONE_LTEST *pLineTest );
```

Parameters

Data Type	Name	Description
IFX_int32_t	fd	File Descriptor.
IFXPHONE_LTEST	*pLineTest	Pointer to IFXPHONE_LTEST .

Return Values

Data Type	Description
IFX_void_t	The return value not used.

Example

```
typedef struct
{
    int mode;
    int submode;
    float result;
    int values[5];
    int error;
} ltest;
ltest LineTest;

/* set line testing structure */
memset (&LineTest, 0, sizeof(LineTest));
LineTest.values[LTEST_FREQ] = 500; /* measure with 500 Hz */
LineTest.values[LTEST_LEVEL] = -10000; /* measure with -10.0dB */

/* execute measurement */
Ifxphone_LT_AC_Level_Measurement (fd, &LineTest);

/* print measurement result */
if (LineTest.Error != LTEST_SUCCESSFUL)
    printf("LTEST error happened!");
else
    printf("LTEST successful with result %f dBm0",
        LineTest.result / 1000);
```

8.1.5 Ifxphone_LT_AC_SNR

Description

This service executes a SNR measurement.

Usage of the input parameters of **IFXPHONE_LTEST** struct

- Field mode: not used.
- Field submode: not used.
- Field values[LTEST_FREQ] represents the tone frequency in Hz used for the measurement.
- Field values[LTEST_LEVEL] represents the tone level for the measurement in 0.001 dBm0.

Output fields of **IFXPHONE_LTEST** struct

- Field result: result of measurement in 0.001 dBm0.
- Field error: error return value.

Prototype

```
IFX_void_t Ifxphone_LT_AC_SNR (
    IFX_int32_t fd,
    IFXPHONE_LTEST *pLineTest );
```

Parameters

Data Type	Name	Description
IFX_int32_t	fd	File Descriptor.
IFXPHONE_LTEST	*pLineTest	Pointer to IFXPHONE_LTEST .

Return Values

Data Type	Description
IFX_void_t	The return value not used.

Example

```
typedef struct
{
    int mode;
    int submode;
    float result;
    int values[5];
    int error;
} ltest;
ltest LineTest;

/* set line testing structure */
memset (&LineTest, 0, sizeof(LineTest));
LineTest.values[LTEST_FREQ] = 500; /* measure with 500 Hz */
LineTest.values[LTEST_LEVEL] = -10000; /* measure with -10.0dB */

/* execute measurement */
Ifxphone_LT_AC_SNR (fd, &LineTest);
```

```

/* print measurement result */
printf("used measurement freq %d Hz, ", LineTest.freq);
if (LineTest.Error != LTEST_SUCCESSFUL)
    printf("LTEST error happened!");
else
    printf("LTEST successful with result %f dB",
        LineTest.result / 1000);

```

8.1.6 Ifxphone_LT_AC_Transhybrid

Description

This service executes a transhybrid measurement.

Usage of the input parameters of **IFXPHONE_LTEST** struct

- Field mode: not used.
- Field submode: not used.
- Field values[LTEST_FREQ] represents the tone frequency in Hz used for the measurement.
- Field values[LTEST_LEVEL] represents the tone level for the measurement in 0.001 dBm0.

Prototype

```

IFX_void_t Ifxphone_LT_AC_Transhybrid (
    IFX_int32_t fd,
    IFXPHONE_LTEST *pLineTest );

```

Parameters

Data Type	Name	Description
IFX_int32_t	fd	File Descriptor.
IFXPHONE_LTEST	*pLineTest	Pointer to IFXPHONE_LTEST .

Return Values

Data Type	Description
IFX_void_t	The return value not used.

Example

```

typedef struct
{
    int mode;
    int submode;
    float result;
    int values[5];
    int error;
} ltest;
ltest LineTest;

/* set line testing structure */
memset (&LineTest, 0, sizeof(LineTest));
LineTest.values[LTEST_FREQ] = 500; /* measure with 500 Hz */

```

```

LineTest.values[LTEST_LEVEL] = -10000; /* measure with -10.0dB */

/* execute measurement */
Ifxphone_LT_AC_Transhybrid (fd, &LineTest);

/* print measurement result */
printf("used measurement freq %d Hz, ", LineTest.freq);
if (LineTest.Error != LTEST_SUCCESSFUL)
    printf("LTEST error happened!");
else
    printf("LTEST successful with result %f dBm0",
        LineTest.result / 1000);

```

8.1.7 Ifxphone_LT_Capacitance

Description

Capacitance measurement.

Usage of the input parameters of **IFXPHONE_LTEST** struct

- Field mode: **LTEST_PARAMETER** parameter measurement, **LTEST_NETWORK** network measurement.
- Field submode: **LTEST_RING_TIP** to perform measurement between ring and tip, **LTEST_RING_GROUND** to perform measurement between ring and ground, **LTEST_TIP_GROUND** to perform measurement between tip and ground.
- Field values[] not used.

Output fields of **IFXPHONE_LTEST** struct

- Field result: result of measurement in nF.
- Field error: error return value.

Prototype

```

IFX_void_t Ifxphone_LT_Capacitance (
    IFX_int32_t fd,
    IFXPHONE_LTEST *pLineTest );

```

Parameters

Data Type	Name	Description
IFX_int32_t	fd	File Descriptor.
IFXPHONE_LTEST	*pLineTest	Pointer to IFXPHONE_LTEST .

Return Values

Data Type	Description
IFX_void_t	The return value not used.

Example

```

typedef struct
{
    int mode;
    int submode;

```

```

float result;
int values[5];
int error;
} ltest;
ltest LineTest;

/* set line testing structure */
memset (&LineTest, 0, sizeof(LineTest));
LineTest.mode = LTEST_PARAMETER;
LineTest.submode = LTEST_RING_TIP;

/* execute measurement */
Ifxphone_LT_Capacity (fd, &LineTest);

/* print measurement result */
if (LineTest.Error != LTEST_SUCCESSFUL)
    printf("LTEST error happened!");
else
    printf("LTEST successful with result %f nF",
        LineTest.result);

```

8.1.8 Ifxphone_LT_Charge

Description

Performs charging of any capacitance between tip/ring to the original voltage found during the preceeding execution of the [Ifxphone_LT_Discharge](#) function. The corresponding parameters returned by the Discharge function must be copied to the respective fields of the Charge function.

Prototype

```

IFX_void_t Ifxphone_LT_Charge (
    IFX_int32_t fd,
    IFXPHONE_LTEST_CHARGE_t *pLineTest );

```

Parameters

Data Type	Name	Description
IFX_int32_t	fd	File Descriptor.
IFXPHONE_LTEST_CHARGE_t	*pLineTest	Pointer to IFXPHONE_LTEST_CHARGE_t .

Return Values

Data Type	Description
IFX_void_t	The return value not used.

8.1.9 Ifxphone_LT_Current

Description

This service executes a current measurement.

Usage of the input parameters of **IFXPHONE_LTEST** struct

- Field mode: **LTEST_ITRANS** to perform measurement of transversal current, **LTEST_ILONG** to perform measurement of longitudinal current.
- Field submode: **LTEST_AC** to perform an AC measurement, **LTEST_DC** to perform a DC measurement.
- Field values[**LTEST_FREQ**] represents the tone frequency in Hz used for the measurement.
- Field values[**LTEST_LEVEL**] represents the tone level for the measurement in 0.001 dBm0.

Output fields of **IFXPHONE_LTEST** struct

- Field result: result of measurement in A.
- Field error: error return value.

Prototype

```
IFX_void_t Ifxphone_LT_Current (
    IFX_int32_t fd,
    IFXPHONE_LTEST *pLineTest );
```

Parameters

Data Type	Name	Description
IFX_int32_t	fd	File Descriptor.
IFXPHONE_LTEST	*pLineTest	Pointer to IFXPHONE_LTEST .

Return Values

Data Type	Description
IFX_void_t	The return value not used.

Example

```
typedef struct
{
    int mode;
    int submode;
    float result;
    int values[5];
    int error;
} ltest;

ltest LineTest;

/* set line testing structure */
memset (&LineTest, 0, sizeof(LineTest));
LineTest.mode = LTEST_ITRANS;
LineTest.submode = LTEST_AC;

/* execute measurement */
Ifxphone_LT_Current (fd, &LineTest);
```

```
/* print measurement result */
if (LineTest.Error != LTEST_SUCCESSFUL)
    printf("LTEST error happened!");
else
    printf("LTEST successful with result %f A", LineTest.result);
```

8.1.10 Ifxphone_LT_Discharge

Description

Performs discharging to the specified voltage level for a given channel. The current DC characteristics parameters found are saved in their respective elements of structure IFXPHONE_LTEST_DISCHARGE. These parameters must be used for any following calls to the [Ifxphone_LT_Charge](#) function.

Prototype

```
IFX_void_t Ifxphone_LT_Discharge (
    IFX_int32_t fd,
    IFXPHONE_LTEST_DISCHARGE_t *pLineTest );
```

Parameters

Data Type	Name	Description
IFX_int32_t	fd	File Descriptor.
IFXPHONE_LTEST_DISCHARGE_t	*pLineTest	Pointer to IFXPHONE_LTEST_DISCHARGE_t .

Return Values

Data Type	Description
IFX_void_t	The return value not used.

8.1.11 Ifxphone_LT_GR909_Config

Description

Configure system parameters (SLIC) to use for values calculation, e.g Voltage divider resistors.¹⁾

Prototype

```
IFX_int32_t Ifxphone_LT_GR909_Config (
    IFX_LT_GR909_CFG_t *p_cfg );
```

Parameters

Data Type	Name	Description
IFX_LT_GR909_CFG_t	*p_cfg	Handle to IFX_LT_GR909_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

8.1.12 Ifxphone_LT_GR909_GetResults

Description

Gets Gr909 measurement results.¹⁾

Prototype

```
IFX_int32_t Ifxphone_LT_GR909_GetResults (
    IFX_int32_t fd_line,
    IFX_LT_GR909_RESULT_t *p_res );
```

Parameters

Data Type	Name	Description
IFX_int32_t	fd_line	Line file descriptor.
IFX_LT_GR909_RESULT_t	*p_res	Handle to result structure.

Return Values

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

¹⁾ GR-909 only supported with 4 VIP

8.1.13 Ifxphone_LT_GR909_Start

Description

Start a GR909 test or test sequence according to measurement mask [IFX_LT_GR909_MASK_t](#).¹⁾

Prototype

```
IFX_int32_t Ifxphone_LT_GR909_Start (
    IFX_int32_t fd_line,
    IFX_boolean_t b_euLike,
    IFX_int32_t meas_mask );
```

Parameters

Data Type	Name	Description
IFX_int32_t	fd_line	Line file descriptor.
IFX_boolean_t	b_euLike	IFX_TRUE : EU like powerline frequency (50 Hz) IFX_FALSE : US like power line frequency (60 Hz)
IFX_int32_t	meas_mask	Measurement mask set with values out of \ref IFX_LT_GR909_MASK_t .

Return Values

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

8.1.14 Ifxphone_LT_Impedance

Description

This service executes an impedance measurement.

Usage of the input parameters of [IFXPHONE_LTEST](#) struct

- Field mode: [LTEST_IMPEDANCE_WITH_RG](#) to perform an impedance measurement via ring generator (the only supported mode).
- Field submode: [LTEST_RING_TIP](#) to perform measurement between ring and tip, [LTEST_RING_GROUND](#) to perform measurement between ring and ground, [LTEST_TIP_GROUND](#) to perform measurement between tip and ground.
- Field values[[LTEST_FREQ](#)] represents the frequency in Hz used for the measurement. These only frequencies supported are 20 and 25 Hz.
- Field values[[LTEST_LEVEL](#)] represents the level for the measurement in mV. The voltage range supported is between 10000 and 25000 mV.

Output fields of [IFXPHONE_LTEST](#) struct

- Field result: result of measurement in Ohm.
- Field error: error return value.

Prototype

```
IFX_void_t Ifxphone_LT_Impedance (
    IFX_int32_t fd,
    IFXPHONE_LTEST *pLineTest );
```

Parameters

Data Type	Name	Description
IFX_int32_t	fd	File Descriptor.
IFXPHONE_LTEST	*pLineTest	Pointer to IFXPHONE_LTEST .

Return Values

Data Type	Description
IFX_void_t	The return value not used.

Example

```
typedef struct
{
    int mode;
    int submode;
    float result;
    int values[5];
    int error;
} ltest;
ltest LineTest;

/* set line testing structure */
memset (&LineTest, 0, sizeof(LineTest));
LineTest.mode = LTEST_IMPEDANCE_WITH_RG; /* use tone generator */
LineTest.submode = LTEST_RING_TIP;
LineTest.values[LTEST_FREQ] = 20; /* measurement with 20 Hz */
LineTest.values[LTEST_LEVEL] = 10000; /* measurement with 10000 mV */

/* execute measurement */
Ifxphone_LT_Impedance (fd, &LineTest);

/* print measurement result */
if (LineTest.Error != LTEST_SUCCESSFUL)
    printf("LTEST error happened!");
else
    printf("LTEST successful with result %f Ohm",
        LineTest.result);
```

Array indices such as LTEST_FREQ are defined in lib_TAPI.h.

8.1.15 Ifxphone_LT_Network

Description

This service performs the so-called group test where a set of linetesting functions are called in order to gather a set of capacitive and resistive measurements which are then used to calculate all resistive and capacitive components of the network. The components calculated are the capacitances found between tip/gnd, ring/gnd and tip/ring respectively, and resistances found between tip/gnd, ring/gnd and tip/ring respectively. As input parameter the measuring channel's device file descriptor is provided. A pointer of type IFXPHONE_LTEST_NET is passed as return by reference parameter. The measuring sequence is as follows:

- DC offset calibration
- AC Foreign voltage measurement, tip/ground
- AC Foreign voltage measurement, tip/ring
- AC Foreign voltage measurement, ring/ground
- DC Foreign voltage measurement, tip/ground
- DC Foreign voltage measurement, tip/ring
- DC Foreign voltage measurement, ring/ground
- Discharge (ifxphone_LT_Discharge)
- Network to ground measurement (ifxphone_LT_Network_Ground)
- Charge (ifxphone_LT_Charge)
- Network tip/ring measurement (ifxphone_LT_Network_Tip_Ring)
- Components calculation

Prototype

```
IFX_void_t Ifxphone_LT_Network (
    IFX_int32_t fd,
    IFXPHONE_LTEST_NET *pLineTest );
```

Parameters

Data Type	Name	Description
IFX_int32_t	fd	File Descriptor.
IFXPHONE_LTEST_NET	*pLineTest	Pointer to line testing structure IFXPHONE_LTEST_NET .

Return Values

Data Type	Description
IFX_void_t	The return value not used.

Example

```
...
#include <lib_tapi_lt.h>

IFXPHONE_LTEST_NET ltNet;

/* setup network linetesting data structure */
memset (&ltNet, 0, sizeof(IFXPHONE_LT_NET));

/* execute network (grou test) measurement for channel identified by fd */
```

```

Ifxphone_LT_Network (fd, &ltNet);

/* print measurement result for measuring channel */
if (ltNET.nErr != 0)
{
    printf("Network measurement failed:\n");
    switch(ltNET.nErr)
    {
        case 1:
            printf("Off-hook detected!\n");
            break;
        case 2:
            printf("Invalid operating mode!\n");
            break;
        case 3:
            printf("Other general error occurred!\n");
            break;
        case 4:
            printf("Unsupported VINETIC device\n");
            break;
        case 5:
            printf("Unsupported SLIC type\n");
            break;
    }
}
else
    printf("Network measurement successful:\n"
        "\n Rtg=%f Ohm, \n Rtr=%f Ohm, \n Rrg=%f Ohm, "
        "\n R_AB=%f Ohm, \n R_BA=%f Ohm, "
        "\n Ctg=%f nF, \n Ctr=%f nF, \n Crg=%f nF, "
        "\n Vtg_ac=%f V, \n Vtr_ac=%f V, \n Vrg_ac=%f V"
        "\n Vtg_dc=%f V, \n Vtr_dc=%f V, \n Vrg_dc=%f V\n",
        ltNet.nRtg, ltNet.nRtr, ltNet.nRrg, ltNet.nR_AB, ltNet.nR_BA,
        ltNet.nCtg, ltNet.nCtr, ltNet.nCrg,
        ltNet.nVtg_ac, ltNet.nVtr_ac, ltNet.nVrg_ac,
        ltNet.nVtg_dc, ltNet.nVtr_dc, ltNet.nVrg_dc);

```

8.1.16 Ifxphone_LT_Network_Ground

Description

This service performs the so-called Network to ground measurement procedure measuring the ring/ground and tip/ground capacitances and resistances of the local loop. As input parameter the measuring channel's device file descriptor is provided. A pointer of type **IFXPHONE_LTEST_NET_GND** is passed as return by reference parameter.

Prototype

```

IFX_void_t Ifxphone_LT_Network_Ground (
    IFX_int32_t fd,
    IFXPHONE_LTEST_NET_GND *pLineTest );

```

Parameters

Data Type	Name	Description
IFX_int32_t	fd	File Descriptor.
IFXPHONE_LTEST_NET_GND	*pLineTest	Pointer to line testing structure IFXPHONE_LTEST_NET_GND .

Return Values

Data Type	Description
IFX_void_t	The return value not used.

Example

```

...
#include <lib_tapi_lt.h>

IFXPHONE_LTEST_NET_GND ltNet;

/* setup network linetesting data structure */
memset (&ltNet, 0, sizeof(IFXPHONE_LT_NET_GND));

/* execute network to ground measurement for channel identified by fd */
Ifxphone_LT_Network_Ground (fd, &ltNet);

/* print measurement result for measuring channel */
if (ltNET.nErr != 0)
{
    printf("Network to ground measurement failed:\n");
    switch(ltNET.nErr)
    {
        case 2:
            printf("Invalid operating mode!\n");
            break;
        case 3:
            printf("Other general error occurred!\n");
            break;
        case 4:
            printf("Unsupported VINETIC device\n");
            break;
        case 5:
            printf("Unsupported SLIC type\n");
            break;
    }
}
else
    printf("Network to ground measurement successful:\n"
        "\n Ctg_total=%f nF, \n Crg_total=%f nF, \n Rtg=%f Ohm, \n Rrg=%f Ohm, \n"
        "\n Ktg=%f, \n Krg=%f",
        ltNet.nCtg_total, ltNet.nCrg_total, ltNet.nRtg_total,
        ltNet.nRrg_total, ltNet.nKtg, ltNet.nKrg);

```

8.1.17 Ifxphone_LT_Network_Tip_Ring

Description

This service performs Network Tip/Ring measurement with Receiver Off-hook indication and optionally Signature Detection, measuring the capacitances and the resistances of the local loop. The measurement uses an equivalent circuit to describe the local loop. This equivalent circuit consists of capacitance Cring in series with resistance Rring, together in parallel with resistance Rtipring. The measurement determines the values of these three components using the following described interface. As input parameter the measuring channel's device file descriptor is provided. A pointer of type IFXPHONE_LTEST_NET_TIP_RING (previously IFXPHONE_LTEST_NET) is passed as return by reference parameter. As prerequisite of using this function correctly is to call the linetesting function TAPI_LT_Configure to select the appropriate SLIC type used, and provide information whether the low-pass filter between the Codec and SLIC is used. This information is necessary for correct calculation of the ring capacitance, Cring.

Prototype

```
IFX_void_t Ifxphone_LT_Network_Tip_Ring (
    IFX_int32_t fd,
    IFXPHONE_LTEST_NET_TIP_RING *pLineTest );
```

Parameters

Data Type	Name	Description
IFX_int32_t	fd	File Descriptor.
IFXPHONE_LTEST_NET_TIP_RING	*pLineTest	Pointer to IFXPHONE_LTEST_NET_TIP_RING .

Return Values

Data Type	Description
IFX_void_t	The return value not used.

Example

```
...
#include <lib_tapi_lt.h>

IFXPHONE_LTEST_NET_TIP_RING ltNET;

/* setup network linetesting data structure */
memset (&ltNET, 0, sizeof(IFXPHONE_LT_NET_TIP_RING));

ltNet.bSign = 1; /* execute Signature Detection*/
/* execute network tip/ring measurement for channel identified by fd */
Ifxphone_LT_Network_Tip_Ring (fd, &ltNET);

/* print measurement result for measuring channel */
if (ltNET.nErr != 0)
{
    printf("Network Tip/Ring measurement failed:\n");
    switch(ltNET.nErr)
```

```

{
    case 1:
        printf("Off-hook detected!\n");
        break;
    case 2:
        printf("Invalid operating mode!\n");
        break;
    case 3:
        printf("Other general error occurred!\n");
        break;
    case 4:
        printf("Unsupported VINETIC device\n");
        break;
    case 5:
        printf("Unsupported SLIC type\n");
        break;
}
else
    printf("Network Tip/Ring measurement successful:\n"
        "\n Cring=%f nF, \n Rring=%f Ohm, \n Rtipring=%f Ohm, "
        "\n Vzener=%f V, \n Rzener=%f Ohm, \n Rbalance=%f %, \n ROH=%s, "
        "\n Csign=%f nF, \n R_AB=%f Ohm, \n R_BA=%f Ohm\n",
        ltNET.nCring, ltNET.nRring, ltNET.nRtipring,
        ltNET.nVzener, ltNET.nRzener, ltNET.nRbalance,
        ltNET.bROH ? "yes":"no",
        ltNet.nCsign, ltNet.nR_AB, ltNet.nR_BA);

```

8.1.18 Ifxphone_LT_Resistance

Description

This service executes a resistance measurement.

Usage of the input parameters of **IFXPHONE_LTEST** struct

- Field mode: **LTEST_PARAMETER** to perform a parameter measurement, **LTEST_NETWORK** to perform a network measurement.
- Field submode: **LTEST_RING_TIP** to perform measurement between ring and tip, **LTEST_RING_GROUND** to perform measurement between ring and ground, **LTEST_TIP_GROUND** to perform measurement between tip and ground, **LTEST_RING_TIP_OFFHOOK** to perform measurement of resistance between Tip and Ring in Active Mode without line disturbance, **LTEST_TIP_BATTERY** to perform measurement of resistance between Tip and Battery, **LTEST_RING_BATTERY** to perform measurement of resistance between Ring and Battery.
- Field values[] not used.

Output fields of **IFXPHONE_LTEST** struct

- Field result: result of measurement in Ohm.
- Field error: error return value.

Prototype

```

IFX_void_t Ifxphone_LT_Resistance (
    IFX_int32_t fd,
    IFXPHONE_LTEST *pLineTest );

```

Parameters

Data Type	Name	Description
IFX_int32_t	fd	File Descriptor.
IFXPHONE_LTEST	*pLineTest	Pointer to IFXPHONE_LTEST .

Return Values

Data Type	Description
IFX_void_t	The return value not used.

Example

```
typedef struct
{
    int mode;
    int submode;
    float result;
    int values[5];
    int error;
} ltest;
ltest LineTest;

/* set line testing structure */
memset (&LineTest, 0, sizeof(LineTest));
LineTest.mode = LTEST_PARAMETER;
LineTest.submode = LTEST_RING_TIP;

/* execute measurement */
Ifxphone_LT_Resistance (fd, &LineTest);

/* print measurement result */
if (LineTest.Error != LTEST_SUCCESSFUL)
    printf("LTEST error happened!");
else
    printf("LTEST successful with result %f Ohm",
        LineTest.result);
```


8.1.19 Ifxphone_LT_Resistance_RC

Description

The Resistance RC measurement is intended to measure high ohmic resistances with an alternative method compared to the above mentioned Resistance measurement. Please note that this measurement is not available for VINETIC 1.x.

Usage of the input parameters of **IFXPHONE_LTEST** struct

- Field mode: **LTEST_RING_TIP** to perform measurement between ring and tip, **LTEST_RING_GROUND** to perform measurement between ring and ground, **LTEST_TIP_GROUND** to perform measurement between tip and ground.
- Field submode: **LTEST_RING_TIP_HIGH_OHMIC** measurement expects a high ohmic measurement between ring and tip, **LTEST_RING_TIP_CAPACITIVE** measurement expects a capacitive load between tip and ring.
- Field values[] not used.

Output fields of **IFXPHONE_LTEST** struct

- Field result: result of measurement in Ohm.
- Field error: error return value.

Prototype

```
IFX_void_t Ifxphone_LT_Resistance_RC (
    IFX_int32_t fd,
    IFXPHONE_LTEST *plfxLineTest );
```

Parameters

Data Type	Name	Description
IFX_int32_t	fd	File Descriptor.
IFXPHONE_LTEST	*plfxLineTest	Pointer to IFXPHONE_LTEST .

Return Values

Data Type	Description
IFX_void_t	The return value not used.

Example

```
typedef struct
{
    int mode;
    int submode;
    float result;
    int values[5];
    int error;
} ltest;

ltest LineTest;

/* set line testing structure */
memset (&LineTest, 0, sizeof(LineTest));
LineTest.mode = LTEST_RING_TIP;
```

```

LineTest.submode = LTEST_RING_TIP_CAPACITIVE;

/* execute measurement */
Ifxphone_LT_Resistance_RC (fd, &LineTest);

/* print measurement result */
if (LineTest.Error != LTEST_SUCCESSFUL)
    printf("LTEST error happened!");
else
    printf("LTEST successful with result %f Ohm",
        LineTest.result);

```

8.1.20 Ifxphone_LT_Voltage

Description

This service executes a voltage measurement.

Usage of the input parameters of **IFXPHONE_LTEST** struct

- Field mode: **LTEST_FOREIGN_VOLTAGE** to perform a measurement of foreign voltage, **LTEST_FEEDED_VOLTAGE** to perform a measurement of feeded voltage..
- Field values[] not used.

Possible submodes

- **LTEST_RING_TIP_AC.**
- **LTEST_RING_TIP_DC.**
- **LTEST_RING_GROUND_AC.**
- **LTEST_RING_GROUND_DC.**
- **LTEST_TIP_GROUND_AC.**
- **LTEST_TIP_GROUND_DC.**
- **LTEST_VBATH.**
- **LTEST_VBATL.**
- **LTEST_VBATR.**
- **LTEST_VDD.**
- **LTEST_VSLD1.**
- **LTEST_VSLD2.**
- **LTEST_RING_TIP_DCN_DCP**

Output fields of **IFXPHONE_LTEST** struct

- Field result: result of measurement in V.
- Field error: error return value.

Prototype

```

IFX_void_t Ifxphone_LT_Voltage (
    IFX_int32_t fd,
    IFXPHONE_LTEST *pLineTest );

```

Parameters

Data Type	Name	Description
IFX_int32_t	fd	File Descriptor.
IFXPHONE_LTEST	*pLineTest	Pointer to IFXPHONE_LTEST .

Return Values

Data Type	Description
IFX_void_t	The return value not used.

Example

```
typedef struct
{
    int mode;
    int submode;
    float result;
    int values[5];
    int error;
} ltest;

ltest LineTest;

/* set line testing structure */
memset (&LineTest, 0, sizeof(LineTest));
LineTest.mode = LTEST_FOREIGN_VOLTAGE;
LineTest.submode = LTEST_RING_TIP_AC;

/* execute measurement */
Ifxphone_LT_Voltage(fd, &LineTest);

/* print measurement result */
if (LineTest.Error != LTEST_SUCCESSFUL)
    printf("LTEST error happened!");
else
    printf("LTEST successful with result %f V",
        LineTest.result);
```

8.1.21 TAPI_LT_Configure

Description

The following function performs Linetesting configuration. The configuration parameters affect both the driver level as well TAPI level processing. For more detailed information on usage of this function please see [VINETIC_IO_LT_CONFIG](#).

Prototype

```
IFX_void_t TAPI_LT_Configure (
    IFX_int32_t fd,
    VINETIC_IO_LT_CONFIG *pCfg );
```

Parameters

Data Type	Name	Description
IFX_int32_t	fd	Measuring channel's device file descriptor. In order for the information passed to reach the targeted channel one must use the channel's device file descriptor.
VINETIC_IO_LT_CONFIG	*pCfg	Pointer to VINETIC_IO_LT_CONFIG .

Return Values

Data Type	Description
IFX_void_t	The return value not used.

8.2 Type Definition Reference

This chapter contains the reference of data types and structures of all modules.

8.2.1 Structure Reference

This chapter contains the Structure reference.

Table 82 Structure Overview of Line Testing Interfaces

Name	Description
IFX_LT_GR909_CFG_t	GR909 parameter configuration structure e.g to set parameters suitable to the slic used.
IFX_LT_GR909_FEMF_t	Foreign electromotive forces test results.
IFX_LT_GR909_HPT_t	Hazardous potential test results.
IFX_LT_GR909_RESULT_t	GR909 results structure.
IFX_LT_GR909_RFT_t	Resistive faults test results.
IFX_LT_GR909_RIT_t	Ringer impedance test results.
IFX_LT_GR909_ROH_t	Receiver off-hook test results.
IFXPHONE_LTEST_CHAR GE_t	Data structure used for the Charge linetesting function.
IFXPHONE_LTEST_DISCHARGE_t	Structure used for Discharge linetesting function.
IFXPHONE_LTEST_NET_GND	Network to ground measurement data structure.
IFXPHONE_LTEST_NET	Network (group test) data structure.
IFXPHONE_LTEST_NET_TIP_RING	Network to ground measurement data structure.

8.2.1.1 IFX_LT_GR909_CFG_t

Description

GR909 parameter configuration structure e.g to set parameters suitable to the slic used.

Prototype

```
typedef struct
{
    IFX_float_t f_R1;
    IFX_float_t f_R2;
    IFX_float_t f_R3;
} IFX_LT_GR909_CFG_t;
```

Parameters

Data Type	Name	Description
IFX_float_t	f_R1	High resistor of the voltage divider connected to the line.

Data Type	Name	Description
IFX_float_t	f_R2	Low resistor of the voltage divider in parallel to the internal resistor of 1 MOhm.
IFX_float_t	f_R3	Low resistor of the voltage divider in parallel to the internal resistor of 750 Ohm.

8.2.1.2 IFX_LT_GR909_FEMF_t

Description

Foreign electromotive forces test results.

Prototype

```
typedef struct
{
    IFX_boolean_t b_result;
    IFX_float_t f_femf_ac_r2g;
    IFX_float_t f_femf_ac_t2g;
    IFX_float_t f_femf_ac_t2r;
    IFX_float_t f_femf_dc_r2g;
    IFX_float_t f_femf_dc_t2g;
    IFX_float_t f_femf_dc_t2r;
} IFX_LT_GR909_FEMF_t;
```

Parameters

Data Type	Name	Description
IFX_boolean_t	b_result	Femf result, passed or failed.
IFX_float_t	f_femf_ac_r2g	Femf ac ring wire to gnd value, [Vrms].
IFX_float_t	f_femf_ac_t2g	Femf ac tip wire to gnd value, [Vrms].
IFX_float_t	f_femf_ac_t2r	Femf ac tip wire to ring value, [Vrms].
IFX_float_t	f_femf_dc_r2g	Femf dc ring wire to gnd value, [V].
IFX_float_t	f_femf_dc_t2g	Femf dc tip wire to gnd value, [V].
IFX_float_t	f_femf_dc_t2r	Femf dc tip wire to ring value, [V].

8.2.1.3 IFX_LT_GR909_HPT_t

Description

Hazardous potential test results.

Prototype

```
typedef struct
{
    IFX_boolean_t b_result;
    IFX_float_t f_hpt_ac_r2g;
    IFX_float_t f_hpt_ac_t2g;
    IFX_float_t f_hpt_ac_t2r;
    IFX_float_t f_hpt_dc_r2g;
```

```

    IFX_float_t f_hpt_dc_t2g;
    IFX_float_t f_hpt_dc_t2r;
} IFX_LT_GR909_HPT_t;

```

Parameters

Data Type	Name	Description
IFX_boolean_t	b_result	Hpt result, passed or failed.
IFX_float_t	f_hpt_ac_r2g	Hpt ac ring wire to gnd value, [Vrms].
IFX_float_t	f_hpt_ac_t2g	Hpt ac tip wire to gnd value, [Vrms].
IFX_float_t	f_hpt_ac_t2r	Hpt ac tip wire to ring value, [Vrms].
IFX_float_t	f_hpt_dc_r2g	Hpt dc ring wire to gnd value, [V].
IFX_float_t	f_hpt_dc_t2g	Hpt dc tip wire to gnd value, [V].
IFX_float_t	f_hpt_dc_t2r	Hpt dc tip wire to ring value, [V].

8.2.1.4 IFX_LT_GR909_RESULT_t

Description

GR909 results structure.

Prototype

```

typedef struct
{
    IFX_uint32_t valid_mask;
    IFX_LT_GR909_HPT_t hpt;
    IFX_LT_GR909_FEMF_t femf;
    IFX_LT_GR909_RFT_t rft;
    IFX_LT_GR909_ROH_t roh;
    IFX_LT_GR909_RIT_t rit;
} IFX_LT_GR909_RESULT_t;

```

Parameters

Data Type	Name	Description
IFX_uint32_t	valid_mask	Valid results mask, set with IFX_LT_GR909_MASK_t .
IFX_LT_GR909_HPT_t	hpt	Hazardous potential test results.
IFX_LT_GR909_FEMF_t	femf	Foreign electromotive forces test results.
IFX_LT_GR909_RFT_t	rft	Resistive faults test results.
IFX_LT_GR909_ROH_t	roh	Receiver offhook test results.
IFX_LT_GR909_RIT_t	rit	Ringer impedance test results.

8.2.1.5 IFX_LT_GR909_RFT_t

Description

Resistive faults test results.

Prototype

```
typedef struct
{
    IFX_boolean_t b_result;
    IFX_float_t f_rft_ac_r2g;
    IFX_float_t f_rft_ac_t2g;
    IFX_float_t f_rft_ac_t2r;
} IFX_LT_GR909_RFT_t;
```

Parameters

Data Type	Name	Description
IFX_boolean_t	b_result	Rft result, passed or failed.
IFX_float_t	f_rft_ac_r2g	Rft ac ring wire to gnd value, [Ohm].
IFX_float_t	f_rft_ac_t2g	Rft ac tip wire to gnd value, [Ohm].
IFX_float_t	f_rft_ac_t2r	Rft ac tip wire to ring value, [Ohm].

8.2.1.6 IFX_LT_GR909_RIT_t

Description

Ringer impedance test results.

Prototype

```
typedef struct
{
    IFX_boolean_t b_result;
    IFX_float_t f_rit_value;
} IFX_LT_GR909_RIT_t;
```

Parameters

Data Type	Name	Description
IFX_boolean_t	b_result	Rit result, passed or failed.
IFX_float_t	f_rit_value	Rit value, [Ohm].

8.2.1.7 IFX_LT_GR909_ROH_t

Description

Receiver off-hook test results.

Prototype

```
typedef struct
{
    IFX_boolean_t b_result;
    IFX_float_t f_rft_t2r_l;
    IFX_float_t f_rft_t2r_h;
} IFX_LT_GR909_ROH_t;
```


Parameters

Data Type	Name	Description
IFX_boolean_t	b_result	Roh result, passed or failed.
IFX_float_t	f_rft_t2r_l	Roh tip wire to ring wire value for low voltage, [Ohm].
IFX_float_t	f_rft_t2r_h	Roh tip wire to ring wire value for high voltage, [Ohm].

8.2.1.8 IFXPHONE_LTEST
Description

Generic line testing structure, used by several tests.

Prototype

```
typedef struct
{
    IFX_int32_t mode;
    IFX_int32_t submode;
    IFX_int32_t values[5];
    IFX_int32_t result;
    IFX_int32_t error;
} IFXPHONE_LTEST_t;
```

Parameters

Data Type	Name	Description
IFX_int32_t	mode	Test mode, defined in LTEST_MODE . Input field.
IFX_int32_t	submode	Test mode, defined in LTEST_SUBMODE . Input field.
IFX_int32_t	values[5]	Value depends on the array index, given by LTEST_VALUES . Input field.
IFX_int32_t	result	Result of measurement. Output field.
IFX_int32_t	error	Error code, one value defined in LTEST_ERROR . Output field.

8.2.1.9 IFXPHONE_LTEST_CHARGE_t
Description

Data structure used for the Charge linetesting function.

Prototype

```
typedef struct
{
    IFX_int32_t bNet;
    IFX_int32_t U_PDR1_dischg;
    IFX_int32_t U_ACT1_dischg;
    IFX_uint16_t orig_VLIM;
    IFX_uint16_t orig_VK1;
```

```

    IFX_uint16_t orig_IK1;
    IFX_int32_t nErr;
} IFXPHONE_LTEST_CHARGE_t;

```

Parameters

Data Type	Name	Description
IFX_int32_t	bNet	Set if function called within TAPI_LL_LT_Network function [in].
IFX_int32_t	U_PDR1_dischg	Initial tip/ring voltage that is discharged from (PDR mode) [in].
IFX_int32_t	U_ACT1_dischg	Initial tip/ring voltage that is discharged from (ACT mode) [in].
IFX_uint16_t	orig_VLIM	Original Vlim setting [in].
IFX_uint16_t	orig_VK1	Original Vk1 setting [in].
IFX_uint16_t	orig_IK1	Original Ik1 setting [in].
IFX_int32_t	nErr	Function status [out].

8.2.1.10 IFXPHONE_LTEST_DISCHARGE_t

Description

Structure used for Discharge linetesting function.

Prototype

```

typedef struct
{
    IFX_int32_t bNet;
    IFX_int32_t Udischarge;
    IFX_int32_t U_PDR1_dischg;
    IFX_int32_t U_ACT1_dischg;
    IFX_uint16_t orig_VLIM;
    IFX_uint16_t orig_VK1;
    IFX_uint16_t orig_IK1;
    IFX_int32_t nErr;
} IFXPHONE_LTEST_DISCHARGE_t;

```

Parameters

Data Type	Name	Description
IFX_int32_t	bNet	Set if function called within TAPI_LL_LT_Network function [in].
IFX_int32_t	Udischarge	Voltage level to discharge to (Volts) [in].
IFX_int32_t	U_PDR1_dischg	Initial tip/ring voltage that is discharged from (PDR mode) [out].
IFX_int32_t	U_ACT1_dischg	Initial tip/ring voltage that is discharged from (ACT mode) [out].
IFX_uint16_t	orig_VLIM	Original Vlim setting [out].
IFX_uint16_t	orig_VK1	Original Vk1 setting [out].
IFX_uint16_t	orig_IK1	Original Ik1 setting [out].
IFX_int32_t	nErr	Function status [out].

8.2.1.11 IFXPHONE_LTEST_NET_GND

Description

Network to ground measurement data structure.

Prototype

```
typedef struct
{
    IFX_int32_t nErr;
    IFX_int32_t bNet;
    IFX_float_t nCtg_total;
    IFX_float_t nCrg_total;
    IFX_float_t nRtg_total;
    IFX_float_t nRrg_total;
    IFX_float_t nKtg;
    IFX_float_t nKrg;
} IFXPHONE_LTEST_NET_GND_t;
```

Parameters

Data Type	Name	Description
IFX_int32_t	nErr	Measurement status, out. 0 _H VINETIC_IO_LT_NET_SUCCESS, measurement was successful, no errors occurred. 2 _H VINETIC_IO_LT_NET_ERR_INVALID_OPMODE, measurement was attempted in from neither PDR nor ACT modes. 3 _H VINETIC_IO_LT_NET_ERR_OTHER, other general failure occurred during the measurement. 4 _H VINETIC_IO_LT_NET_ERR_VIN_UNSUPP, the measurement is not supported for the underlying version of VINETIC® device. 5 _H VINETIC_IO_LT_NET_ERR_SLIC_UNSUPP, the measurement is not supported for the underlying SLIC type.
IFX_int32_t	bNet	Set if function called within TAPI_LL_LT_Network function [in].
IFX_float_t	nCtg_total	Calculated total Ctip/gnd capacitance [out].
IFX_float_t	nCrg_total	Calculated total Cring/gnd capacitance [out].
IFX_float_t	nRtg_total	Calculated total Rtip/gnd resistance [out].
IFX_float_t	nRrg_total	Calculated total Rring/gnd resistance [out].
IFX_float_t	nKtg	Calculated mean ratio between Ctip/gnd and Ctip/ring [out].
IFX_float_t	nKrg	Calculated mean ratio between Cring/gnd and Ctip/ring [out].

8.2.1.12 IFXPHONE_LTEST_NET

Description

Network (group test) data structure.

Prototype

```
typedef struct
{
    IFX_int32_t nErr;
    IFX_float_t nRtg;
    IFX_float_t nRtr;
    IFX_float_t nRrg;
    IFX_float_t nR_AB;
    IFX_float_t nR_BA;
    IFX_float_t nCtg;
    IFX_float_t nCtr;
    IFX_float_t nCrg;
    IFX_float_t nVtg_ac;
    IFX_float_t nVtr_ac;
    IFX_float_t nVrg_ac;
    IFX_float_t nVtg_dc;
    IFX_float_t nVtr_dc;
    IFX_float_t nVrg_dc;
} IFXPHONE_LTEST_NET_t;
```

Parameters

Data Type	Name	Description
IFX_int32_t	nErr	Measurement status, out. 0 _H VINETIC_IO_LT_NET_SUCCESS, measurement was successful, no errors occurred. 1 _H VINETIC_IO_LT_NET_ERR_OFFHOOK_DETECTED, an off-hook was detected during the measurement. 2 _H VINETIC_IO_LT_NET_ERR_INVALID_OPMODE, measurement was attempted in from neither PDR nor ACT modes. 3 _H VINETIC_IO_LT_NET_ERR_OTHER, other general failure occurred during the measurement. 4 _H VINETIC_IO_LT_NET_ERR_VIN_UNSUPP, the measurement is not supported for the underlying version of VINETIC® device. 5 _H VINETIC_IO_LT_NET_ERR_SLIC_UNSUPP, the measurement is not supported for the underlying SLIC type.
IFX_float_t	nRtg	Calculated R tip/gnd resistance in Ohm, [out].
IFX_float_t	nRtr	Ccalculated Rtip/ring resistance in Ohm, [out].
IFX_float_t	nRrg	Calculated Rring/gnd resistance in Ohm, [out].
IFX_float_t	nR_AB	Calculated R_AB resistance in Ohm, [out].
IFX_float_t	nR_BA	Calculated R_BA resistance in Ohm, [out].
IFX_float_t	nCtg	Calculated C tip/gnd capacitance in nF, [out].
IFX_float_t	nCtr	Calculated C tip/ring capacitance in nF, [out].
IFX_float_t	nCrg	Calculated C ring/gnd capacitance in nF, [out].
IFX_float_t	nVtg_ac	Calculated foreign AC Vtip/gnd voltage, [out].
IFX_float_t	nVtr_ac	Calculated foreign AC Vtip/ring voltage [out].

Data Type	Name	Description
IFX_float_t	nVrg_ac	Calculated foreign AC Vring/gnd voltage [out].
IFX_float_t	nVtg_dc	Calculated foreign DC Vtip/gnd voltage [out].
IFX_float_t	nVtr_dc	Calculated foreign DC Vtip/ring voltage [out].
IFX_float_t	nVrg_dc	Calculated foreign DC Vring/gnd voltage [out].

8.2.1.13 IFXPHONE_LTEST_NET_TIP_RING

Description

Network to ground measurement data structure.

Prototype

```
typedef struct
{
    IFX_int32_t nErr;
    IFX_int32_t bNet;
    IFX_int32_t bSign;
    IFX_float_t nCring;
    IFX_float_t nRring;
    IFX_float_t nRtipring;
    IFX_float_t nVzener;
    IFX_float_t nRzener;
    IFX_float_t nRbalance;
    IFX_boolean_t bROH;
    IFX_float_t nCsign;
    IFX_float_t nR_AB;
    IFX_float_t nR_BA;
} IFXPHONE_LTEST_NET_TIP_RING_t;
```

Parameters

Data Type	Name	Description
IFX_int32_t	nErr	Measurement status, out. 0 _H VINETIC_IO_LT_NET_SUCCESS, measurement was successful, no errors occurred. 1 _H VINETIC_IO_LT_NET_ERR_OFFHOOK_DETECTED, an off-hook was detected during the measurement. 2 _H VINETIC_IO_LT_NET_ERR_INVALID_OPMODE, measurement was attempted in from neither PDR nor ACT modes. 3 _H VINETIC_IO_LT_NET_ERR_OTHER, other general failure occurred during the measurement. 4 _H VINETIC_IO_LT_NET_ERR_VIN_UNSUPP, the measurement is not supported for the underlying version of VINETIC® device. 5 _H VINETIC_IO_LT_NET_ERR_SLIC_UNSUPP, the measurement is not supported for the underlying SLIC type.
IFX_int32_t	bNet	Set if function called within TAPI_LL_LT_Network function [in].
IFX_int32_t	bSign	With or without executing signature detection [in]
IFX_float_t	nCring	Calculated Cring capacitance in nF, out.

Data Type	Name	Description
IFX_float_t	nRring	Calculated Rring resistance in Ohm, out.
IFX_float_t	nRtipring	Calculated Rtipring resistance in Ohm, out.
IFX_float_t	nVzener	Zener voltage in Volts, signature component.
IFX_float_t	nRzener	Resistance in series to the Zener diode, signature component.
IFX_float_t	nRbalance	Resistive balance in percent. A result of 50%, indicates that the resistive component of the network is fully symmetrical. A result of 0%, indicates that the resistive component is between tip/ground. A result of 100%, indicates that the resistive component is between ring/ground. This result should be ignored for high ohmic load (e.g. 1MOhm).
IFX_boolean_t	bROH	Receiver Off-hook flag is set if the Zener voltage exceeds 3 V.
IFX_float_t	nCsign	Calculated signature capacitance in nF [out].
IFX_float_t	nR_AB	Calculated R_AB resistance in Ohm [out].
IFX_float_t	nR_BA	Calculated R_BA resistance in Ohm [out].

8.2.2 Enumerator Reference

This chapter contains the Enumerator reference.

Table 83 Enumerator Overview of Line Testing Interfaces

Name	Description
IFX_LT_GR909_MASK_t	GR909 tests masks.
LTEST_ERROR	Error codes.
LTEST_MODE	Test mode.
LTEST_SIGNAL_TYPE	Signal type.
LTEST_SUBMODE	Test submode.
LTEST_VALUES	Values.

8.2.2.1 IFX_LT_GR909_MASK_t

Description

GR909 tests masks.

Prototype

```
typedef enum
{
    IFX_LT_GR909_HPT_MASK = (1 << 0),
    IFX_LT_GR909_FEMF_MASK = (1 << 1),
    IFX_LT_GR909_RTF_MASK = (1 << 2),
    IFX_LT_GR909_ROH_MASK = (1 << 3),
    IFX_LT_GR909_RIT_MASK = (1 << 4)
} IFX_LT_GR909_MASK_t;
```

Parameters

Name	Value	Description
IFX_LT_GR909_HPT_MASK	(1 << 0)	Mask to select HPT.
IFX_LT_GR909_FEMF_MASK	(1 << 1)	Mask to select FEMF.
IFX_LT_GR909_RTF_MASK	(1 << 2)	Mask to select RFT.
IFX_LT_GR909_ROH_MASK	(1 << 3)	Mask to select ROH.
IFX_LT_GR909_RIT_MASK	(1 << 4)	Mask to select RIT.

8.2.2.2 LTEST_ERROR
Description

Line testing submode.

Prototype

```
typedef enum
{
    LTEST_SUCCESSFUL = 0,
    LTEST_ERROR_MEASUREMENT_FAILED = 1,
    LTEST_ERROR_HOOK_DETECTED = 2,
    LTEST_ERROR_ALREADY_RUNNING = 3,
    LTEST_ERROR_NOT_IMPLEMENTED = 4,
    LTEST_GR909_PASSED = 0,
    LTEST_GR909_ERROR_ALREADY_RUNNING = 1,
    LTEST_GR909_HAZARDOUS_VOLTAGE_FAILED = 2,
    LTEST_GR909_FOREIGN_FORCE_FAILED = 3,
    LTEST_GR909_RESISTIVE_FAULT_FAILED = 4,
    LTEST_GR909_RECEIVER_OFF_HOOK_FAILED = 5,
    LTEST_GR909_RINGER_FAILED = 6,
    LTEST_GR909_ERROR_MEASUREMENT_FAILED = 7,
    LTEST_GR909_ERROR_NOT_IMPLEMENTED = 8
} LTEST_ERROR_t;
```

Parameters

Name	Value	Description
LTEST_SUCCESSFUL	0	Successful.
LTEST_ERROR_MEASUREMENT_FAILED	1	Error: Measurement Failed.
LTEST_ERROR_HOOK_DETECTED	2	Error: Hook Detected.
LTEST_ERROR_ALREADY_RUNNING	3	Error: Already Running.
LTEST_ERROR_NOT_IMPLEMENTED	4	Error: Not Implemented.
LTEST_GR909_PASSED	0	GR909 Passed.
LTEST_GR909_ERROR_ALREADY_RUNNING	1	GR909 Error: Already Running.
LTEST_GR909_HAZARDOUS_VOLTAGE_FAILED	2	GR909 Error: HAZARDOUS VOLTAGE Failed.

Name	Value	Description
LTEST_GR909_FOREIGN_FORCE_FAILED	3	GR909 Error: FOREIGN FORCE Failed.
LTEST_GR909_RESISTIVE_FAULT_FAILED	4	GR909 Error: RESISTIVE FAULT Failed.
LTEST_GR909_RECEIVER_OFF_HOOK_FAILED	5	GR909 Error: RECEIVER OFF HOOK Failed.
LTEST_GR909_RINGER_FAILED	6	GR909 Error: RINGER Failed.
LTEST_GR909_ERROR_MEASUREMENT_FAILED	7	GR909 Error: Measurement Failed.
LTEST_GR909_ERROR_NOT_IMPLEMENTED	8	GR909 Error: Not implemented.

8.2.2.3 LTEST_MODE

Description

Line testing submode.

Prototype

```
typedef enum
{
    LTEST_FOREIGN_VOLTAGE = 0,
    LTEST_FEEDED_VOLTAGE = 1,
    LTEST_BATTERY_VOLTAGE = 2,
    LTEST_RING_VOLTAGE = 3,
    LTEST_INTERNAL_VOLTAGE = 4,
    LTEST_PARAMETER = 0,
    LTEST_NETWORK = 1,
    LTEST_IMPEDANCE_WITH_TG = 2,
    LTEST_IMPEDANCE_WITH_RG = 3,
    LTEST_ITRANS = 0,
    LTEST_ILONG = 1,
    LTEST_RX = 0,
    LTEST_TX = 1,
    LTEST_RXTX = 2,
    LTEST_GR909_HAZARDOUS_VOLTAGE = 0,
    LTEST_GR909_FOREIGN_FORCE = 1,
    LTEST_GR909_RESISTIVE_FAULT = 2,
    LTEST_GR909_RECEIVER_OFF_HOOK = 3,
    LTEST_GR909_RINGER = 4,
    LTEST_GR909_ALL = 5
} LTEST_MODE_t;
```

Parameters

Name	Value	Description
LTEST_FOREIGN_VOLTAGE	0	Foreign Voltage.
LTEST_FEEDED_VOLTAGE	1	Feeded Voltage.

Name	Value	Description
LTEST_BATTERY_VOLTAGE	2	Battery Voltage.
LTEST_RING_VOLTAGE	3	Ring Voltage.
LTEST_INTERNAL_VOLTAGE	4	Internal Voltage.
LTEST_PARAMETER	0	Parameter.
LTEST_NETWORK	1	Network.
LTEST_IMPEDANCE_WITH_TG	2	Impedance with TG.
LTEST_IMPEDANCE_WITH_RG	3	Impedance with RG.
LTEST_ITRANS	0	ITRAN.
LTEST_ILONG	1	ILONG.
LTEST_RX	0	RX.
LTEST_TX	1	TX.
LTEST_RXTX	2	RXTX.
LTEST_GR909_HAZARDOUS_VOLTAGE	0	GR909 HAZARDOUS VOLTAGE.
LTEST_GR909_FOREIGN_FORCE	1	GR909 FOREIGN FORCE.
LTEST_GR909_RESISTIVE_FAULT	2	GR909 RESISTIVE FAULT.
LTEST_GR909_RECEIVER_OFF_HOOK	3	GR909 RECEIVER OFF HOOK.
LTEST_GR909_RINGER	4	GR909 RINGER.
LTEST_GR909_ALL	5	GR909 All.

8.2.2.4 LTEST_SIGNAL_TYPE

Description

Line testing signal type.

Prototype

```
typedef enum
{
    LTEST_SINUS = 0,
    LTEST_NOISE = 1
} LTEST_SIGNAL_TYPE_t;
```

Parameters

Name	Value	Description
LTEST_SINUS	0	Sinus.
LTEST_NOISE	1	Noise.

8.2.2.5 LTEST_SUBMODE

Description

Line testing submode.

Prototype

```
typedef enum
```

```
{
    LTEST_RING_TIP_AC = 0,
    LTEST_RING_TIP_DC = 1,
    LTEST_RING_GROUND_AC = 2,
    LTEST_RING_GROUND_DC = 3,
    LTEST_TIP_GROUND_AC = 4,
    LTEST_TIP_GROUND_DC = 5,
    LTEST_VBATH = 6,
    LTEST_VBATL = 7,
    LTEST_VBATR = 8,
    LTEST_VDD = 9,
    LTEST_VSLD1 = 10,
    LTEST_VSLD2 = 11,
    LTEST_OBSOLETE = 12,
    LTEST_VHR = 13,
    LTEST_RING_TIP_DCN_DCP = 14,
    LTEST_RING_TIP = 0,
    LTEST_RING_GROUND = 1,
    LTEST_TIP_GROUND = 2,
    LTEST_RING_TIP_OFFHOOK = 3,
    LTEST_TIP_BATTERY = 4,
    LTEST_RING_BATTERY = 5,
    LTEST_RING_TIP_HIGH_OHMIC = 0,
    LTEST_RING_TIP_CAPACITIVE = 1,
    LTEST_AC = 0,
    LTEST_DC = 1,
    LTEST_NO_TONE = 0,
    LTEST_TONE = 1,
    LTEST_KEEP_TONE = 2,
    LTEST_NO_TH = 0,
    LTEST_TH = 1
} LTEST_SUBMODE_t;
```

Parameters

Name	Value	Description
LTEST_RING_TIP_AC	0	Ring Tip AC.
LTEST_RING_TIP_DC	1	Ring Tip DC.
LTEST_RING_GROUND_AC	2	Ring Ground AC.
LTEST_RING_GROUND_DC	3	Ring Ground DC.
LTEST_TIP_GROUND_AC	4	Tip Ground AC.
LTEST_TIP_GROUND_DC	5	Tip Ground DC.
LTEST_VBATH	6	VBATH.
LTEST_VBATL	7	VBATL.
LTEST_VBATR	8	VBATR.
LTEST_VDD	9	VDD.
LTEST_VSLD1	10	VSLD1.
LTEST_VSLD2	11	VSLD2.

Name	Value	Description
LTEST_OBSOLETE	12	Obsolete.
LTEST_VHR	13	VHR.
LTEST_RING_TIP_DCN_DCP	14	Ring Tip DCN DCP.
LTEST_RING_TIP	0	Ring Tip.
LTEST_RING_GROUND	1	Ring Ground.
LTEST_TIP_GROUND	2	Tip Ground.
LTEST_RING_TIP_OFFHOOK	3	Ring Tip offhook.
LTEST_TIP_BATTERY	4	Tip Battery.
LTEST_RING_BATTERY	5	Ring Battery.
LTEST_RING_TIP_HIGH_OHMIC	0	Ring Tip High Ohmic.
LTEST_RING_TIP_CAPACITIVE	1	Ring Tip Capacitive.
LTEST_AC	0	AC.
LTEST_DC	1	DC.
LTEST_NO_TONE	0	No tone.
LTEST_TONE	1	Tone.
LTEST_KEEP_TONE	2	Keep tone.
LTEST_NO_TH	0	No TH.
LTEST_TH	1	TH.

8.2.2.6 LTEST_VALUES

Description

Line testing signal type.

Prototype

```
typedef enum
{
    LTEST_FREQ = 0,
    LTEST_LEVEL = 1,
    LTEST_SIGNAL = 2
} LTEST_VALUES_t;
```

Parameters

Name	Value	Description
LTEST_FREQ	0	Frequency.
LTEST_LEVEL	1	Level.
LTEST_SIGNAL	2	Signal.

References

- [1] VINETIC® Prel. Data Sheets
- [2] VINETIC® Version 1.4 Prel. User's Manual Software Description Rev. 2.0, 2004-10-19
- [3] VINETIC®-4M/-4C (PEF 3314/-3394) Version 2.1/ Version 2.2 Prel. User's Manual Software Description Rev. 2.0, 2005-01-20
- [4] Prel. Addendum Rev. 1.0 to VINETIC®-4M/-4C (PEF 3314/-3394) Version 2.1/ Version 2.2 Prel. User's Manual Software Description Rev. 2.0, 2005-07-29
- [5] VINETIC®-4S (PEF 3304) Version 2.1 Prel. User's Manual Software Description Rev. 1.0, 2005-03-22
- [6] Prel. Addendum Rev. 1.0 to VINETIC®-4S (PEF 3304) Version 2.1 Prel. User's Manual Software Description Rev. 1.0, 2005-07-29
- [7] VINETIC® Version 1.4 Prel. User's Manual System Reference DS1, 2003-10-15
- [8] Prel. Addendum Rev. 2.0 to VINETIC® Version 1.4 Prel. User's Manual System Reference DS1, 2005-05-18
- [9] VINETIC® Version 2.1/Version 2.2 Prel. User's Manual System Description Rev. 2.0, 2005-05-18
- [10] VINETIC® Version 1.4/Version 2.2 Prel. User's Manual EDSP Firmware Description Rev. 2.0, 2005-05-18
- [11] VINETIC®-4VIP Version 1.4 Prel. EDSP Firmware Overview Rev. 4.0, 2005-05-17
- [12] VINETIC®-4M/-4C (PEF 3314/-3394) Version 2.2 EDSP Prel. Firmware Overview Rev. 2.0, 2005-06-14
- [13] VINETIC® Driver Porting and Integration Guide Rev. 1.0, 2005-07-27
- [14] T.38 Fax Agent Release 1.1 User's Manual Programmer's Reference Rev. 1.0, 2005-09-22
- [15] T.38 Protocol Stack Release 1.16 User's Manual Programmer's Reference Rev. 1.0, 2005-09-22
- [16] T.38 Test Application Release 1.0 User's Manual Programmer's Reference Rev. 1.0, 2005-09-22
- [17] VINETIC® Embedded Controller Firmware Release Notes
- [18] VINETIC® EDSP Firmware Release Notes
- [19] VINETIC® Driver Software Release Notes
- [20] VINETIC® System Packages Release Notes
- [21] VINETIC® T.38 Fax Relay Package Release Notes

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

G

GPIO	General Purpose Input Output
------	------------------------------

I

IETF	Internet Engineering Task Force
IFX	Infineon
IO	Input/Output

J

JB	Jitter Buffer
----	---------------

L

LEC	Line Echo Cancellor
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

SINBT Supplier's Information Note

SSRC Synchronization source

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