VINETIC®

Voice and Internet Enhanced Telephony Interface Circuit

VINETIC®-CPE Version 2.1 Device Driver
Driver and API Description
for
VINETIC®-2CPE (PEB 3332), Version 2.1
VINETIC®-1CPE (PEB 3331), Version 2.1

Communication Solutions

Infineon technologies

Never stop thinking.

Edition 2006-01-31

Published by Infineon Technologies AG,
St.-Martin-Strasse 53,
81669 München, Germany

Attention please!

The information herein is given to describe certain components and shall not be considered as a guarantee of characteristics.

Terms of delivery and rights to technical change reserved.

We hereby disclaim any and all warranties, including but not limited to warranties of non-infringement, regarding circuits, descriptions and charts stated herein.

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.

Previous Version: none

Page	Subjects (major changes since last revision)

Trademarks

Table of Contents

List of Figures

List of Tables

Preface

This document describes the VINETIC®-CPE Version 2.1 device driver structure and usage. If not otherwise specified, the description in this document applies to both VINETIC®-2CPE and VINETIC®-1CPE devices.

To simplify matters, the following synonyms are used:

VINETIC®-CPE: Synonym used for the system consisting of codec versions VINETIC®-2CPE/-1CPE Version 2.1 and SLIC versions SLIC-DC Version 1.2 or SLIC-E Version 2.1.

Organization of this Document

This document is divided into 5 chapters. It is organized as follows:

Chapter 1 provides an overview of the VINETIC®-CPE device driver package. It gives all information needed to compile the device driver, and it explains the configuration options.

Chapter 2 describes the first steps necessary for software setup and the TAPI usage for some basic operations.

Chapter 3 provides a description of the remaining TAPI functionality.

Chapter 4 is intended to serve as reference for the VINETIC®-CPE TAPI interfaces.

Chapter 5 is intended to serve as reference for the VINETIC®-CPE Device Driver 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®-CPE 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®-CPE 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 [2].

1.2 Files

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

Table 1 VINETIC®-CPE 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®-CPE 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®-CPE 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®-2CPE:

```
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®-CPE, Version 2.1 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

Not yet supported.

1.4 Compilation

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

To retrieve the device driver sources 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
```

1.4.1 Linux

If you have your toolchain in place, you should have included the path to your cross-compiler in the PATH and know the path to your Linux kernel header files. Building the device driver is done in two simple 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.

Table 2 Linux configure options

Option	Description	Required
--build=<value>	Value shows the name of your development platform and depends on your compiler, e.g. i686-pc-linux	No
--host=<value>	Value shows the name of your target platform which is required to choose the correct cross-compiler, e.g. powerpc-linux *Note: Ensure that your compiler tools <value>-gcc, etc. can be found in the PATH, e.g. powerpc-linux-gcc. This way you can have more than one cross-compiler installed on your system.*	Yes
--enable-kernelincl=<value>	Value is the path to your kernel header files, e.g. /home/${USER}/linux/include	Yes
--with-access-mode=<value>	Value is the desired µC access mode for your system, which should be one of the following constants: 8-bit Intel Mux, 8-bit Intel Demux, 8-bit Motorola, and SCI.	Yes
--with-access-width=<value>	Value is the desired µC access width the device driver does. It should be either 16 (default) or 8. From the device driver's view, all accesses to the VINETIC® are 16-bit (register size). The VINETIC®-CPE, version 2.x provides only a 8-bit bus interface, which still leaves you two choices despite the access mode: • The VINETIC®-CPE device driver still does 16-bit accesses and the bus controller split each access in two 8-bit accesses (assuring the correct timing) or • If the bus controller cannot be configured to split 16-bit accesses as described above, you can force the VINETIC®-CPE device driver to do only 8-bit accesses. In either case, the device driver will do the accesses in accordance to controller's endianess. Refer to option --enable-byte-swap if your controller's endianess does not match the bus endianess. The SCI interface is specified for 8-bit accesses only.	No

Table 2 Linux configure options (cont'd)

Option	Description	Required
--enable-byte-swap	This is a special option for the VINETIC®-CPE, version 2.x device driver to enable byte swapping inside the VINETIC®-CPE. This option is intended to support little-endian bus accesses (such as Intel) in combination with big-endian controllers and vice versa. Example: MIPS controller in big-endian mode and the bus controller performs little-endian bus accesses. In this case, byte swapping is required and can be done very efficiently by the VINETIC®-CPE device. This eliminates the need for modifying the device driver code.	No
--enable-2cpe	Creates Makefiles to generate a VINETIC®-CPE, version 2.x device driver.	Yes
--enable-lt	Enable GR-909 measurement support.	No
--enable-trace	Enable IFX traces, trace level can be configured at runtime	No
--enable-warnings	Enable compiler warnings (GCC only), recommended	No
--prefix=<value>	Value is the path to your install directory	No

1.4.2 VxWorks

Not supported yet.

1.5 Interface Names and Backwards Compatibility

The device driver TAPI interfaces have been renamed[1)] 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 4.3**.

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

1.6 GR-909 Support

The GR-909 functionality is delivered in a separate library (lib_tapi_lt_vincpe.c, lib_tapi_lt_gr909.h) to allow also floating point calculations of the results which is not possible on driver level for some operating systems. To enable GR-909 support in the device driver, configure option --enable-lt or the equivalent define shall be used.

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

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 2.4**
- Establishment of connections and conferences, **Chapter 2.5**

2.1 File Descriptors

Communication between application software and the driver is achieved through system calls performed on specific file descriptors. For the VINETIC®-CPE 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®-CPE 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®-2CPE 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®-2CPE, /dev/vin10 addresses VINETIC®-2CPE number 1 and /dev/vin20 addresses VINETIC®-2CPE 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®-2CPE, /dev/vin11 addresses VINETIC®-2CPE number 1/channel 1 and /dev/vin23 addresses VINETIC®-2CPE 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®-2CPE devices[1)]

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

Table 3 Device Nodes for a System including two VINETIC®-2CPE devices[1)]

Device Node	VINETIC® Number	Addressed Resource Set	Included Resources
/dev/vin20	Device 2	Entire device 2	VINETIC® chip
/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®-2CPE 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®-2CPE 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 4**, in the reference description of each TAPI command, states whether a command must be applied to a data or phone channel.

Examples in **Figure 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®-2CPE 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).

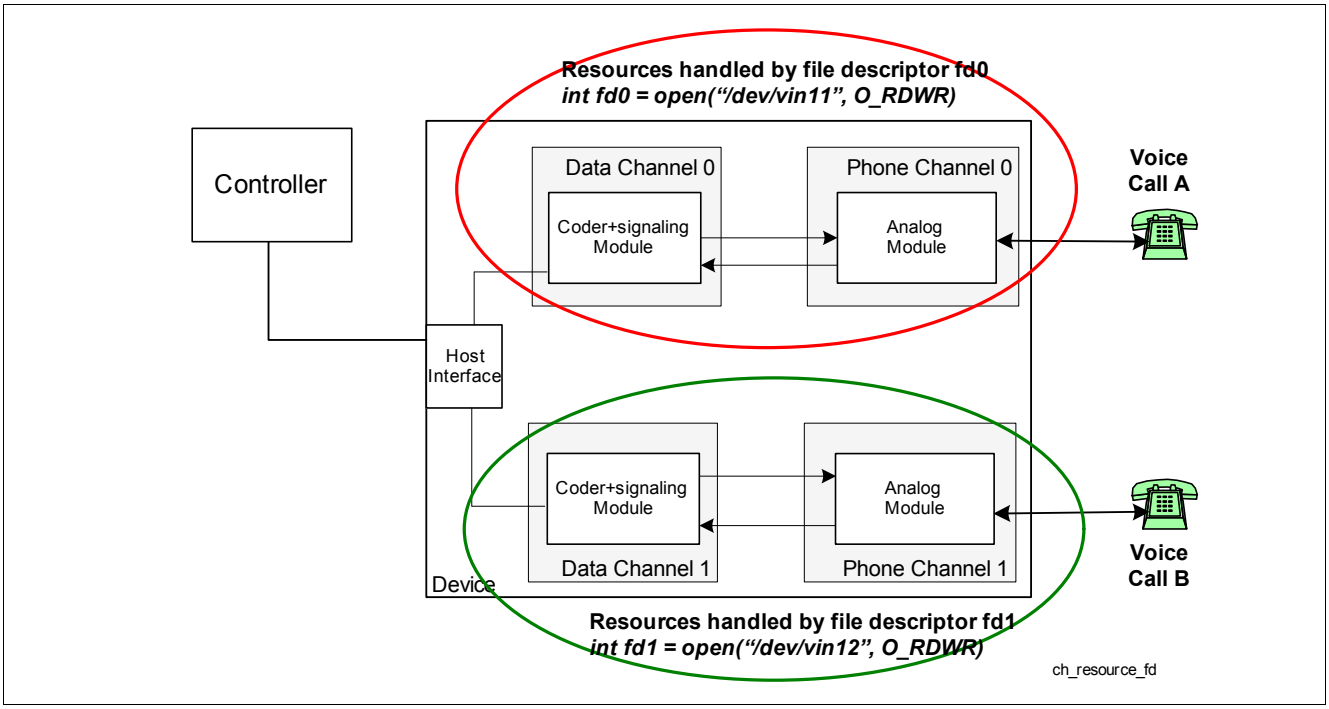

Figure 1 Example of File Descriptors and Resources (1)

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 2.5**.

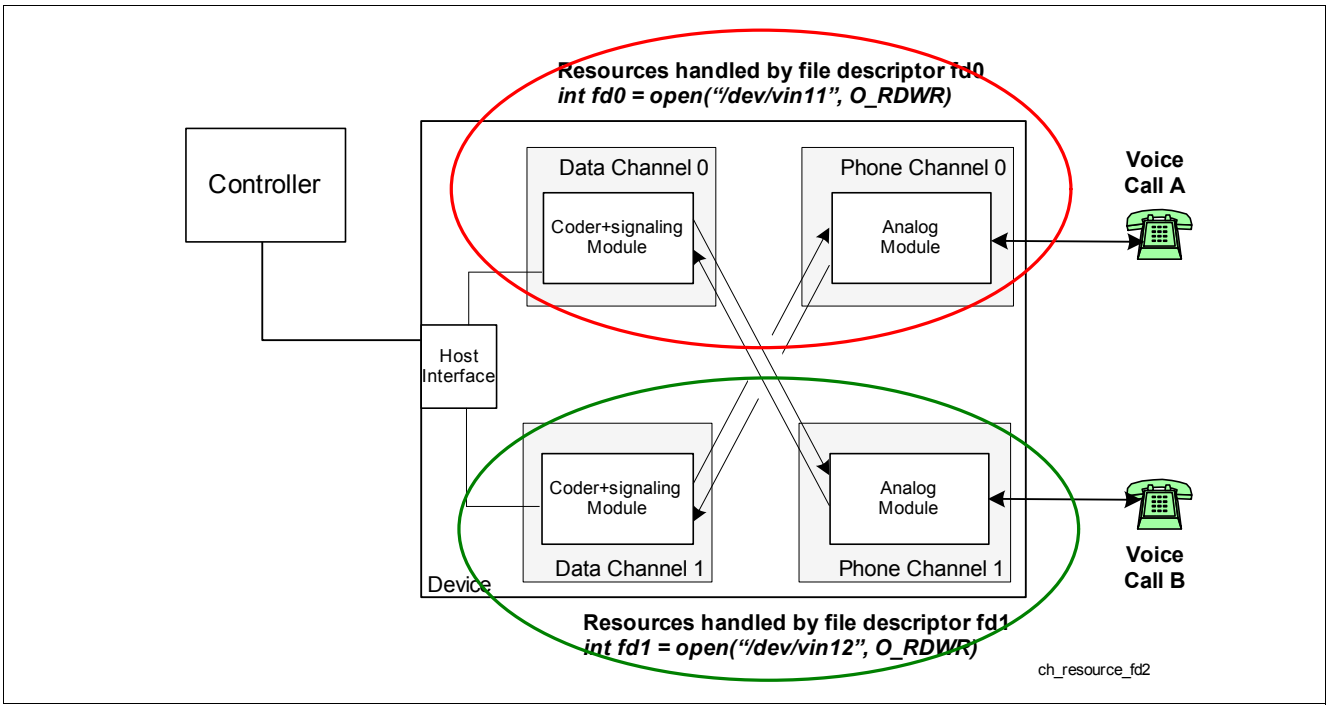

Figure 2 Example of File Descriptors and Resources (2)

2.2 Initialization

The foundation for all subsequent operations is the successful initialization of the device driver and the VINETIC®-CPE. 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 initialization is done with command **IFX_TAPI_CH_INIT**, the argument is a pointer to **IFX_TAPI_CH_INIT_t**. It is necessary to configure:

- nMode: TAPI channel type, valid value for VINETIC®-CPE 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.

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.byt"

/* 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 3 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.

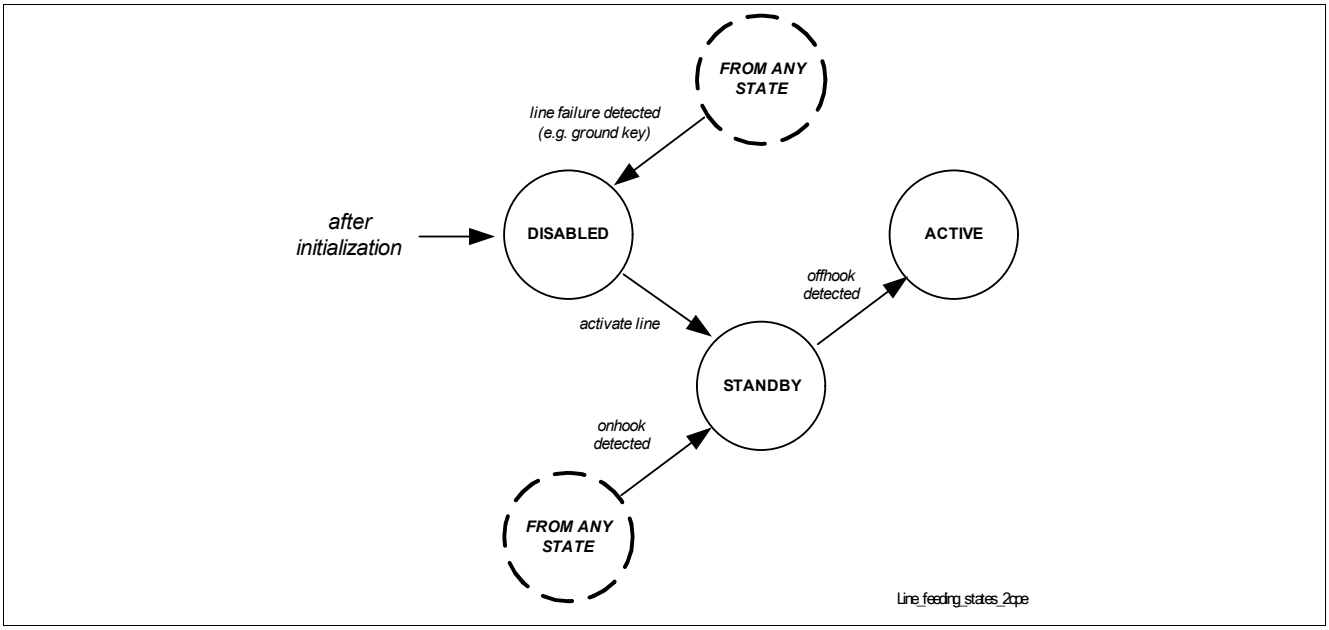

Figure 3 Sequence of Line Feeding Modes

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 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 2.4.1**, hook-state detection.
- **Chapter 2.4.2**, DTMF/pulse digit detection.
- **Chapter 2.4.3**, 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], &rfds);
if (width < fd[0])
{
   width = fd[0];
}
fd[1] = open("/dev/vin12", O_RDWR, 0x644);
/* Open channel file descriptor for channel 1 */
FD_SET(fd[1], &rfds);
if (width < fd[1])
{
   width = fd[1];
}

/* Now wait for events and data */
ret = select(width + 1, &rfds, IFX_NULL, IFX_NULL, IFX_NULL);
/* Select woke up from exception on fd_dev or data on fd[x] */
if (FD_ISSET(fd_dev, &rfds))
{
   /* an exception occurred, get status of two channels */
   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], &rfds))
   {
      printf("Handle data\n");
   }
}
```

2.4.1 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 2.4**).

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;
}
```

2.4.2 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 */
   }
}
```

2.4.3 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);
```

2.4.4 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.nMinTime = 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);
```

2.5 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®-CPE 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 two analog module ports. These interfaces are relevant for scenarios where a call must be established between two phones directly connected[1] to the VINETIC®-CPE.
- **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 add/remove a link between a PCM port to an ALM port. It is relevant for scenarios where a call must be established between one phone directly connected to the VINETIC®-CPE and another phone connected via PCM.

1) Connected via SLIC or DAA device.

2.5.1 Internal Call

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

Figure 4 depicts a typical example. It is very important to note the role of the data channel: it is required for signal detection/generation (such as DTMF, Fax/Modem tones, call progress tones, caller ID, ...) while the coder part is idle (jitter buffer and RTP are inactive).

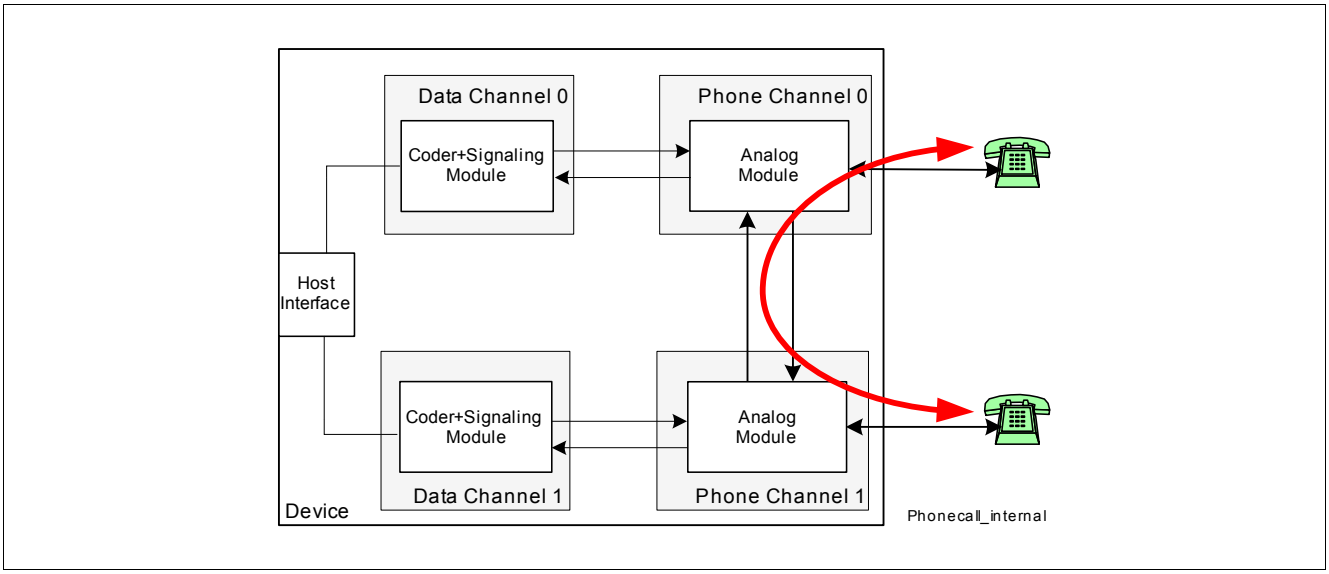

Figure 4 Simple Internal Call

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);
```

2.5.2 Voice Call with External Party

Figure 5 depicts the usage of the resources for two VoIP calls in parallel.

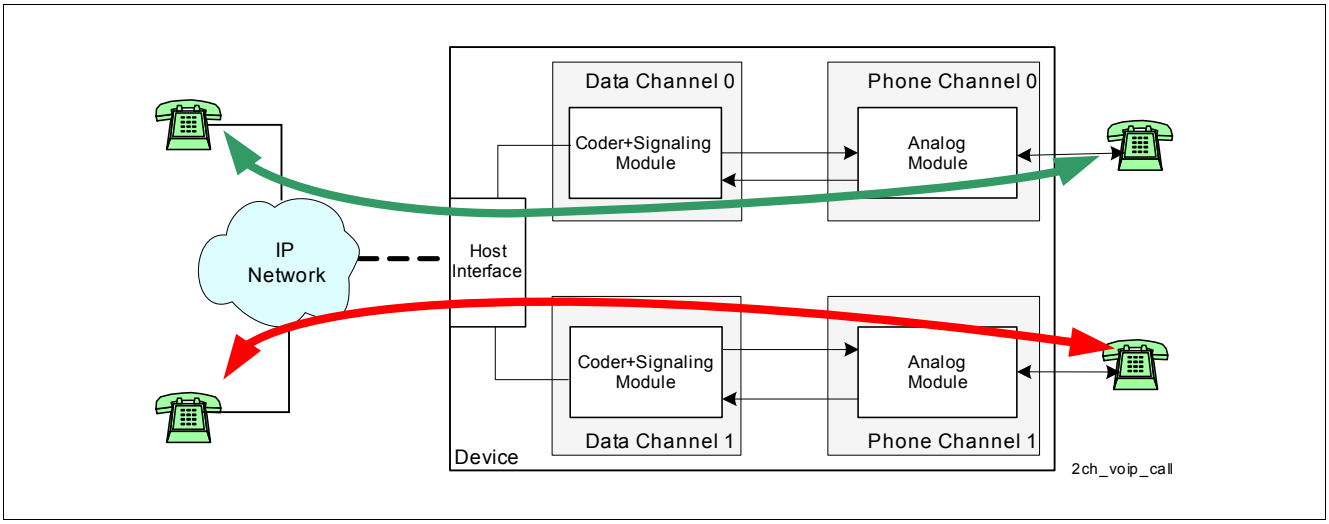

Figure 5 Two Independent VoIP Calls

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 5 */
/* 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 5 */
```

2.5.3 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.

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

2.5.3.2 Conference with an Internal and an External Party

Continuing the setup from **Chapter 2.5.1**, an external party can be added (see **Figure 6**). 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.

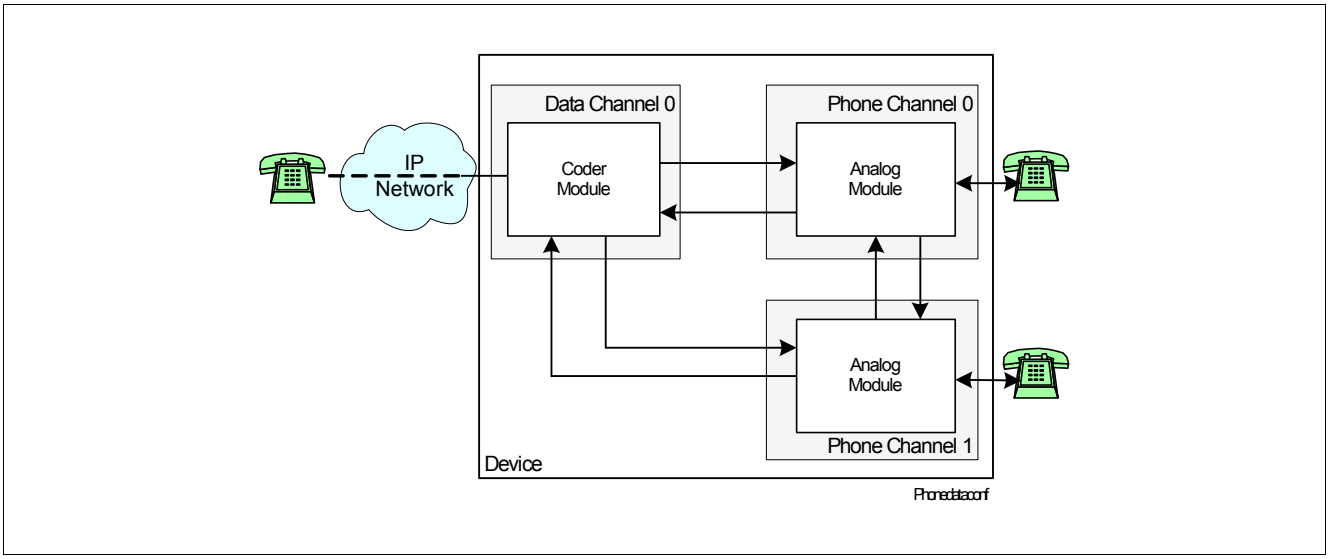

Figure 6 External and Internal Conference

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 2.5.3.1**, 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.

2.5.3.3 Conference with Two External Parties

Starting from the setup from **Chapter 2.5.3.1**, two external parties can be added to the local phone channel 0 (see **Figure 7**). 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);
```

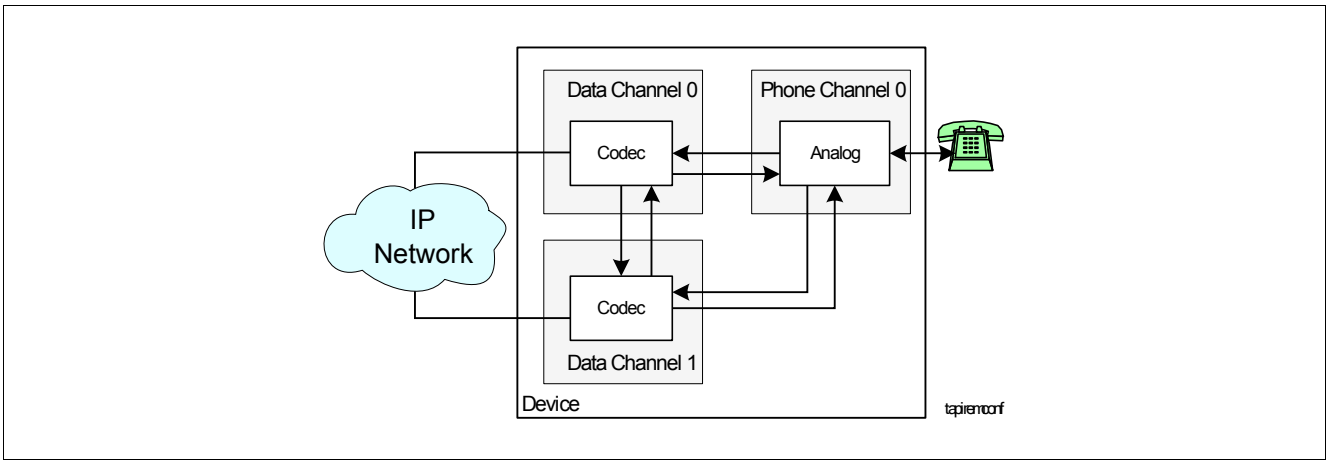

Figure 7 External Conference

Removing the Connection

To get back the setup as after the initialization described in **Chapter 2.5.3.1**, 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**
- Usage of programmable I/O pins (GPIOs), **Chapter 3.7**
- Usage of GR-909 tests, **Chapter 3.8**

3.1 Channel Configuration

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

- Input/output gains (volume), **Chapter 3.1.1**
- Line Echo Canceller (LEC), **Chapter 3.1.3**
- PCM interface, **Chapter 3.1.2**
- Jitter Buffer parameters, **Chapter 3.1.4**
- RTP parameters, **Chapter 3.1.5**

3.1.1 Phone Volume

It is possible to tune input (RX) and output (TX) gains independently for the analog phone port[1)] using the interface **IFX_TAPI_PHONE_VOLUME_SET** for the 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®-CPE 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 2.5**.

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 8** shows some possible flows for a CPE system including a VINETIC® and a DuSLIC-S chipset (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.

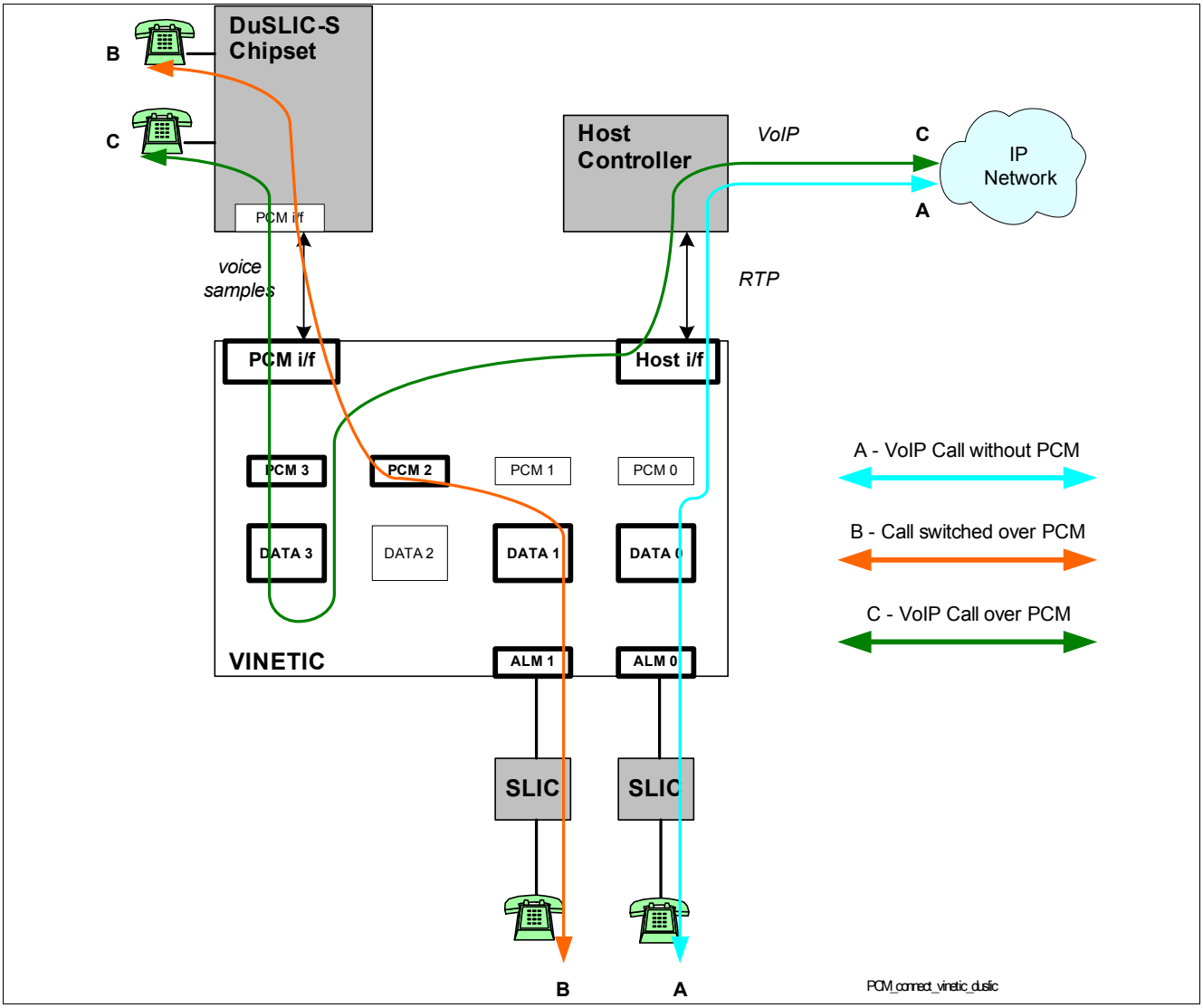

Figure 8 Connection of a DuSLIC via PCM Interface

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 8**), 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 8**), the LEC must[1)] 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 8**.

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);
```

3.2 Encoder/Decoder Control

Configuration of the encoder is done using interfaces:

- **IFX_TAPI_ENC_TYPE_SET**, to select the encoder (alias vocoder) type
- **IFX_TAPI_ENC_FRAME_LEN_SET**, to set the packetization time

- **IFX_TAPI_ENC_VAD_CFG_SET**, to enable/disable Voice Activity Detection (silence compression and comfort noise generation)
- **IFX_TAPI_ENC_VOLUME_SET**, to configure the encoding level

After configuration of the encoder parameters, start/stop encoding can be controlled via **IFX_TAPI_ENC_START** and **IFX_TAPI_ENC_STOP**. This will also automatically start/stop generation of RTP frames. Before starting encoding, the RTP session parameters have to be set (see also **Chapter 3.1.5**); otherwise, random RTP parameters will apply!

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

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

Example - Encoder/Decoder Control

```
IFX_int32_t encFrameLen, encType, encVAD;

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

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

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

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

/* Do something ... */

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

3.3 RTCP and Jitter Buffer Statistics

RTCP and jitter buffer statistics 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 detectection of tones specified and stored in an internal table called “Tone Table.” Besides typical DTMF and call progress tones[1], it is possible to define tones with more complex cadences and combinations of frequencies.

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.

3.4.1 Tone Table

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

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

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

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

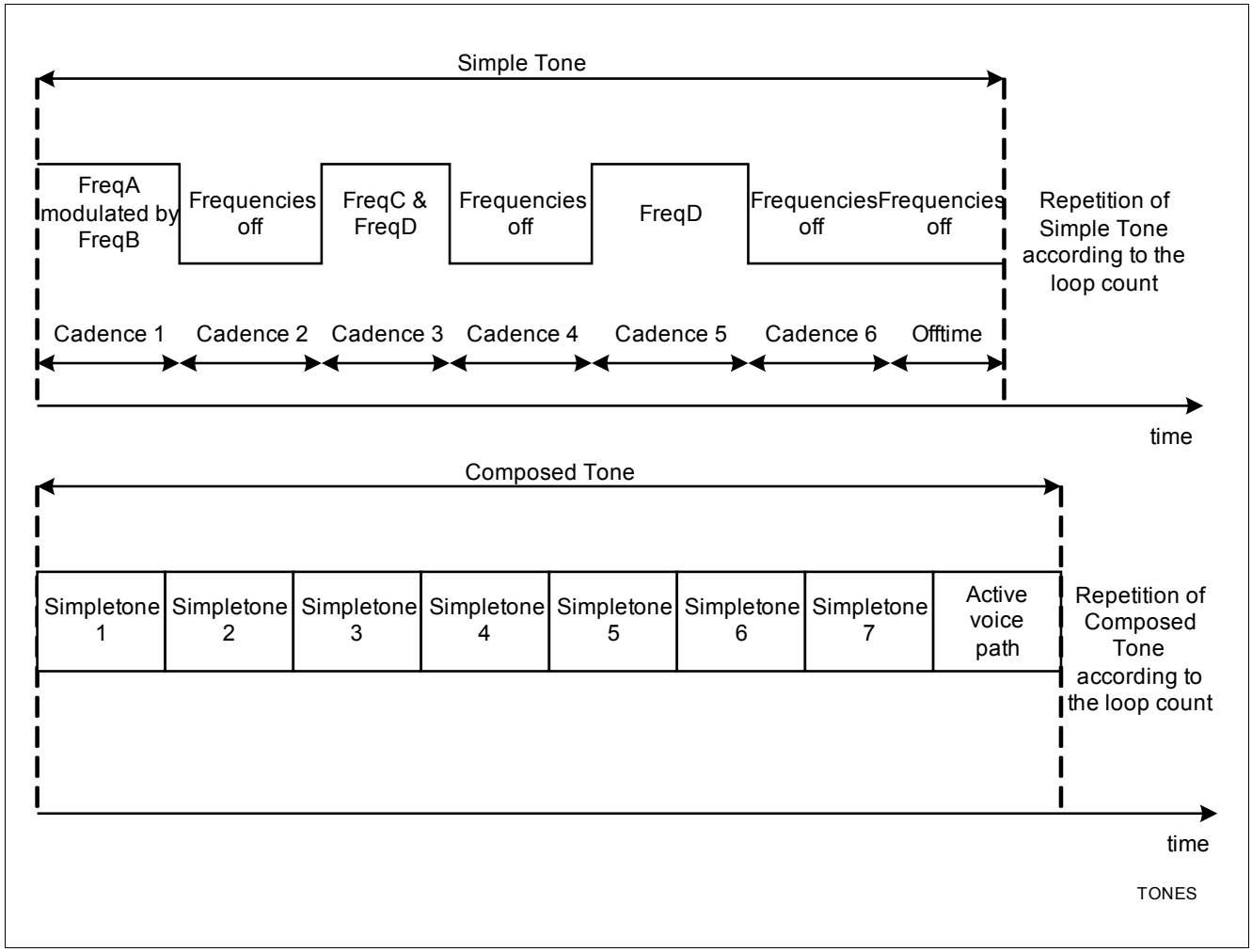

Figure 9 Simple and composed tones

As depicted in **Figure 9**, 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 loopcount of a composed tone is greater than zero, it is also possible to activate the voice path between the loops.

3.4.2 Playing Tones

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

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

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

Examples

```
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 requencies 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 4 reports the predefined tones entries in the tone table.

Table 4 Tone Table - Predefined Tones

Index	Frequency 1	Frequency 2	Note
1	697	1209	DTMF digit 1
2	697	1336	DTMF digit 2
3	697	1477	DTMF digit 3
4	770	1209	DTMF digit 4
5	770	1336	DTMF digit 5
6	770	1477	DTMF digit 6
7	852	1209	DTMF digit 7
8	852	1336	DTMF digit 8
9	852	1477	DTMF digit 9
10	941	1209	DTMF digit * (star)
11	941	1336	DTMF digit 0 (zero)
12	941	1477	DTMF digit # (pound)
13	800	-	-
14	1000	-	-
15	1250	-	-
16	950	-	-
17	1100	-	-
18	1400	-	-
19	1500	-	-
20	1600	-	-
21	1800	-	-
22	2100	-	-

Table 4 Tone Table - Predefined Tones (cont'd)

Index	Frequency 1	Frequency 2	Note
23	2300	-	-
24	2450	-	-
25	350	440	Dial tone example
26	440	480	Ringing tone example
27	480	620	Busy tone example
28	697	1633	DTMF digit A
29	770	1633	DTMF digit B
30	852	1633	DTMF digit C
31	941	1633	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 36**).

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 encoding[1)] of the special tone.

3.4.8 Call Progress Tone Detection

The VINETIC®-CPE 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.

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

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®-CPE.

- 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 5**; the various types are supported[1)] according to Telcordia, ETSI, BT and NTT standards.

Table 5 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®-CPE 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).

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

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 6 (on-hook transmission associated with ringing), Table 7 (on-hook transmission not associated with ringing), Table 8 (off-hook transmission) and Table 9 (abbreviations used).

Table 6 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 7 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 8 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 9 Caller ID Generation - Abbreviations

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

3.5.2 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 10** 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 54**.

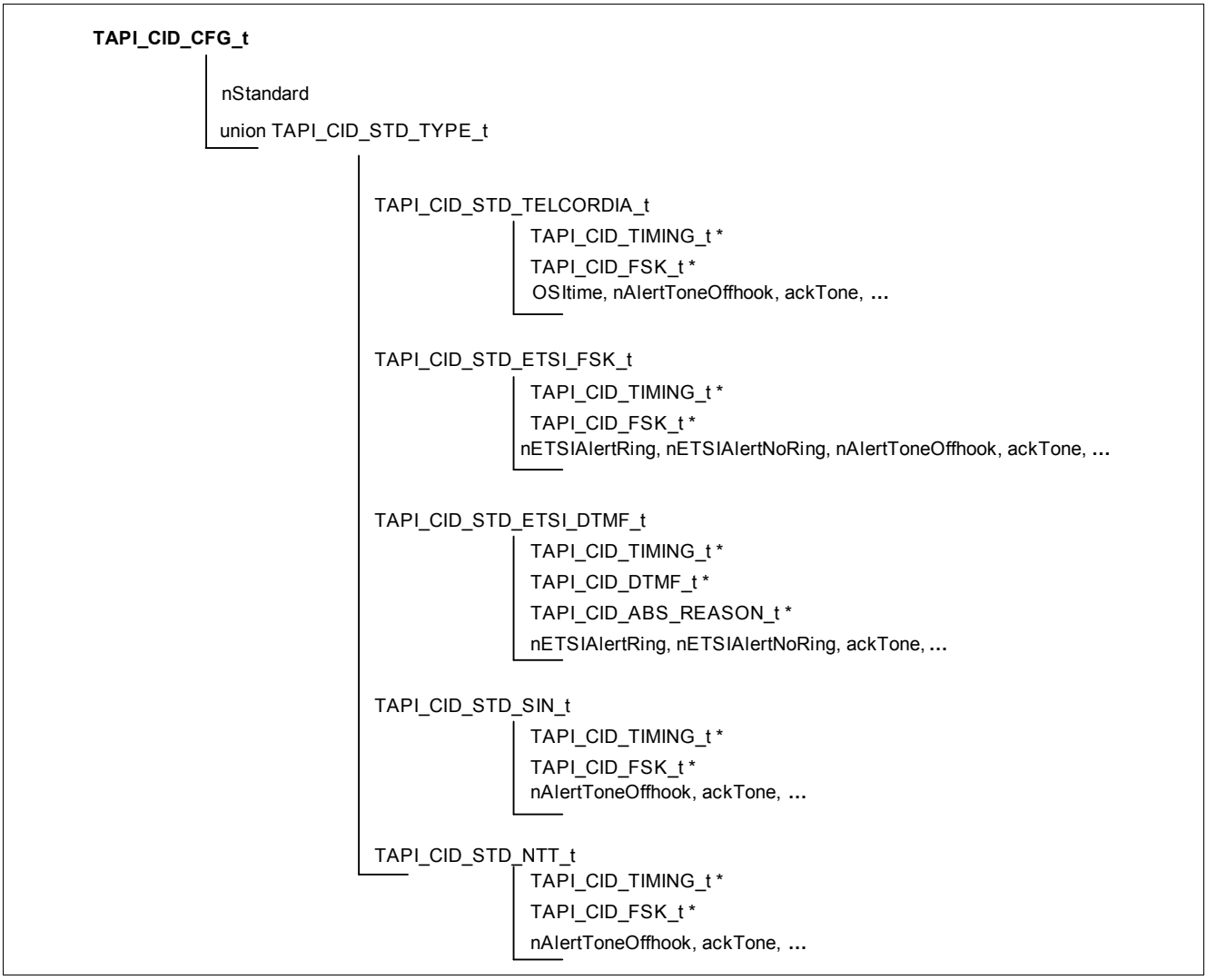

Figure 10 Overview of the Configuration structure

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 10 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 10 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 10 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, then start periodical ringing (using the preprogrammed cadence pattern).

3.5.3.1 Configure Ring Cadence

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

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

Some examples of cadence patterns:

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

3.5.3.2 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);

/* .... 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.

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

- **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 11** 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 11 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_char_t* PHONE_NUMBER = "123456789";
const IFX_int32_t MAX_MSG_LENGTH = 3;

/* Application decides to issue a VMWI, to light the CPE LED */
/* Implemented only for on-hook */
IFX_TAPI_CID_MSG_t cidTypeVMWI;
IFX_TAPI_CID_MSG_ELEMENT_t message[MAX_MSG_LENGTH];
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.msgType = IFX_TAPI_CID_MT_MWI;
/* One message element to be transmitted (enable VMWI) */
cidTypeVMWI.nMsgElements = 1;
/* Prepare message to be displayed: Visual Indication */
message[0].value.elementType = IFX_TAPI_CID_ST_VISINDIC;
/* VMWI Enable (to light the LED) */
```

```
message[0].value.element = IFX_TAPI_CID_VMWI_EN;

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

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

Example 5 - Transparent Transmission

```
/* Application decides to send a CID type 1, already coded in the right */
/* FSK format */
IFX_TAPI_CID_MSG_t cid_info;
IFX_TAPI_CID_MSG_ELEMENT_t message;
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.msgType = IFX_TAPI_CID_MT_CSUP;
cid_info.nMsgElements = 1;
cid_info.msg = &message;
/* Send sequence */
ioctl(fd, IFX_TAPI_CID_TX_SEQ_START, &cid_info);
```

3.5.5 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**
- 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 2.4.2**).

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));
/* Start CID RX Engine */
ret = ioctl(fd, IFX_TAPI_CID_RX_START, 0);
/* 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 pump[1]; 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.

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

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) &startSig );
/* Event will be reported through IFX_TAPI_TONE_STATUS_GET, signal field */
/* After detetion 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.
- **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 µs */
jbCfgVoice.nInitialSize = 0x02D0;
/* Minimum JB size 10 ms = 0x50 * 125 µs */
jbCfgVoice.nMinSize = 0x50;
/* Maximum JB size 180 ms = 0x5A0 * 125 µs */
jbCfgVoice.nMaxSize = 0x5A0;

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

/* Disable LEC */
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®-CPE 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 frames[1)] with the same interfaces defined for packetized voice: the read interface to get modulated data-pump frames from VINETIC®-CPE, 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 fransmission.

3.7 General-Purpose IOs

The VINETIC®-CPE device driver's GPIO module allows access to the VINETIC®-CPE GPIO pins[1)] 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 set[2)] (**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(!) */
```

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.

1) Not available in all packages.

2) Only if the GPIO is configured as output.

```
/* 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);

/* 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);
```

3.8 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 (50Hz) or US like (60Hz). 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);
         }
      }
   }
```

4 TAPI Interfaces

This section describes all TAPI interfaces.

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

4.1 ioctl commands of TAPI Interfaces

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

Table 12 TAPI Interface Overview

Name	Description
Initialization Service	Initialization Service
Operation Control Service	Operation Control Service
Metering Service	Metering Service
Tone Control Service	Tone Control Service
Dial Service	Dial Service
Signal Detection Service	Signal Detection Service
CID Features Service	CID Features
Connection Control Service	Connection Control Service
Miscellaneous Services	Miscellaneous Services
Ringing Service	Ringing Service
PCM Service	PCM Service
Fax T.38 Service	Fax T.38 Service.

4.1.1 Initialization Service

This service sets the default initialization of device and hardware.

Table 13 IO-control Overview of Initialization Service

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

Table 14 Structure Overview of Initialization Service

Name	Description
IFX_TAPI_CH_INIT_t	TAPI initialization structure used for IFX_TAPI_CH_INIT.

Table 15 Enumerator Overview of Initialization Service

Name	Description
IFX_TAPI_CH_INIT_MODE_t	TAPI initialization modes, selection for target system.
IFX_TAPI_CH_INIT_COUNTRY_t	Country selection for IFX_TAPI_CH_INIT_t.

4.1.1.1 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);
```

4.1.2 Operation Control Service

Table 16 IO-control Overview of Operation Control Service

Name	Description
IFX_TAPI_ENC_AGC_CFG_GET	Get the current AGC coefficients of a coder module.
IFX_TAPI_ENC_AGC_CFG_SET	Set the AGC coefficient for a coder module This implementation assumes that an index of a AGC resource is fix assigned to the related index of the coder module.
IFX_TAPI_ENC_AGC_SET	Enable/Disable the AGC resource. This implementation assumes that the AGC resources inside a VINETIC® are fixed assigned to the related Coder-Module.
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 17 Structure Overview of Operation Control Service

Name	Description
IFX_TAPI_LEC_CFG_t	Line echo canceller (LEC) configuration.
IFX_TAPI_LINE_VOLUME_t	Phone speaker phone and microphone volume settings used for ioctl IFX_TAPI_PHONE_VOLUME_SET.
IFX_TAPI_LINE_HOOK_VT_t	Structure used for validation times of hook, hook flash and pulse dialing.

4.1.2.1 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

4.1.2.2 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);
```

4.1.2.3 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.
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);
```

4.1.2.4 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

4.1.2.5 IFX_TAPI_LINE_FEED_SET

Description

This service sets the line feeding mode.

Prototype

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

Parameters

Data Type	Name	Description
IFX_int32_t	fd	File descriptor. It is applicable to phone channel file descriptors containing an analog interface.
IFX_int32_t	param	The parameter represents a mode value of IFX_TAPI_LINE_FEED_t.

Return Values

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

Remarks

For all battery switching modes the hardware must be able to support it for example by programming coefficients.

Example

```
/* Set line mode to power down */
ioctl(fd, IFX_TAPI_LINE_FEED_SET, IFX_TAPI_LINE_FEED_DISABLED);
```

4.1.2.6 IFX_TAPI_LINE_HOOK_STATUS_GET

Description

This service reads the hook status from the driver.

Prototype

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

Parameters

Data Type	Name	Description
IFX_int32_t	fd	File descriptor. It is applicable to phone channel file descriptors containing an analog interface.
IFX_int32_t	IFX_TAPI_LINE_HOOK_STATUS_GET	I/O control identifier for this operation
IFX_int32_t*	param	The parameter is a pointer to the status. 0: no hook detected 1: hook detected

Return Values

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

Example

```
IFX_TAPI_LEC_CFG_t param;
IFX_int32_t fd;
IFX_int32_t nHook;
```

```
ioctl(fd, IFX_TAPI_LINE_HOOK_STATUS_GET, &nHook);
switch (nHook)
{
   case 0:
         /* On hook */
      break;
   case 1:
         /* Off hook */
      break;
   default :
         /* Unknown state */
      break;
}
```

4.1.2.7 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);
```

4.1.2.8 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);
```

4.1.2.9 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);
```

4.1.2.10 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);
```

4.1.3 Tone Control Service

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

Table 18 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_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 19 Constant Overview of Tone Control Service

Name	Description
IFX_TAPI_TONE_STEPS_MAX	Maximum tone generation steps, also called cadences.
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.

Table 20 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 21 Union Overview of Tone Control Service

Name	Description
IFX_TAPI_TONE_t	Tone descriptor.

Table 22 Enumerator Overview of Tone Control Service

Name	Description
IFX_TAPI_TONE_TG_t	Defines the tone generator usage.
IFX_TAPI_TONE_GROUP_t	Tone grouping.
IFX_TAPI_TONE_FREQ_t	Frequency setting for a cadence step.
IFX_TAPI_TONE_MODULATION_t	Modulation setting for a cadence step.
IFX_TAPI_TONE_TYPE_t	Tone types.
IFX_TAPI_TONE_t	This chapter define a union for the tone descriptor.
IFX_TAPI_TONE_SIMPLE_t	
IFX_TAPI_TONE_COMPOSED_t	This chapter define the structure of the tone descriptor.
IFX_TAPI_TONE_FREQ_t	

4.1.3.1 IFX_TAPI_TONE_LOCAL_PLAY

Description

Start/stop generation of a tone towards local port, the parameter (if greater than 0) gives the tone table index of the tone to be played.

If the parameter is equal to zero, stop current tone generation.

Prototype

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

Parameters

Data Type	Name	Description
IFX_int32_t	fd	File descriptor. It is applicable to channel file descriptors.
IFX_int32_t	IFX_TAPI_TONE_LOCAL_PLAY	I/O control identifier for this operation
IFX_int32_t	param	The parameter is the index of the tone to the predefined tone table (range 1 - 31) or custom tones added previously (index 32 - 255). Index 0 means tone stop. Using the upper bits modify the default tone playing source:

Return Values

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

Remarks

This can be a predefined, simple or a composed tone. The tone codes are assigned previously on system start. Index 1 - 31 is predefined by the driver and covers the original TAPI.

Example

```
IFX_TAPI_LINE_VOLUME_t param;
IFX_int32_t fd;

/* Play tone index 34 */
ioctl(fd, IFX_TAPI_TONE_LOCAL_PLAY, 34);
```

4.1.3.2 IFX_TAPI_TONE_NET_PLAY

Description

Start/stop generation of a tone towards network, the parameter (if greater than 0) gives the tone table index of the tone to be played.

If the parameter is equal to zero, stop current tone generation.

Prototype

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

Parameters

Data Type	Name	Description
IFX_int32_t	fd	File descriptor. It is applicable to data channel file descriptors.
IFX_int32_t	IFX_TAPI_TONE_NET_PLAY	I/O control identifier for this operation
IFX_int32_t	param	The parameter is the index of the tone to the predefined tone table (range 1 - 31) or custom tones added previously (index 32 - 255). Index 0 means tone stop. Using the upper bits modify the default tone playing source.

Return Values

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

Remarks

This can be a predefined, simple or a composed tone. The tone codes are assigned previously on system start. Index 1 - 31 is predefined by the driver and covers the original TAPI.

Example

```
/* Play tone index 34 */
ioctl(fd, IFX_TAPI_TONE_NET_PLAY, 34);
```

4.1.3.3 IFX_TAPI_TONE_STOP

Description

Stop tone generation.

Prototype

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

Parameters

Data Type	Name	Description
IFX_int32_t	fd	File descriptor. It is applicable to channel file descriptors.
IFX_int32_t	IFX_TAPI_TONE_TONE_STOP	I/O control identifier for this operation
IFX_int32_t	param	Tone table index of the tone to be stopped.

Return Values

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

Example

```
/* Stop playing tone index 34 */
ioctl(fd, IFX_TAPI_TONE_STOP, 34);
```

4.1.3.4 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);
```

4.1.4 Dial Service

Contains services for dialing.

Table 23 IO-control Overview of Dial Service

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

4.1.4.1 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);
```

4.1.4.2 IFX_TAPI_PULSE_ASCII_GET

Description

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

Prototype

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

Parameters

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

Return Values

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

Example

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

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

4.1.4.3 IFX_TAPI_PULSE_GET

Description

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

Prototype

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

Parameters

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

Return Values

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

Remarks

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

Example

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

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

4.1.4.4 IFX_TAPI_PULSE_READY

Description

This service checks if a pulse digit was received.

Prototype

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

Parameters

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

Return Values

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

Remarks

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

Example

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

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

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

4.1.4.5 IFX_TAPI_TONE_DTMF_ASCII_GET

Description

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

Prototype

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

Parameters

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

Return Values

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

Example

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

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

4.1.4.6 IFX_TAPI_TONE_DTMF_GET

Description

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

Prototype

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

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

Parameters

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

Return Values

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

Remarks

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

Example

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

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

4.1.4.7 IFX_TAPI_TONE_DTMF_READY_GET

Description

This service checks if a DTMF digit was received.

Prototype

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

Parameters

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

Return Values

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

Remarks

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

Example

```
IFX_int32_t fd;
IFX_int32_t ready;

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

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

4.1.5 Signal Detection Service

.

Table 24 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 25 Structure Overview of Signal Detection Service

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

Table 26 Enumerator Overview of Signal Detection Service

Name	Description
IFX_TAPI_SIG_t	List the tone detection options.
IFX_TAPI_TONE_CPTD_DIRECTION_t	Specifies the CPT signal for CPT detection.

4.1.5.1 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

4.1.5.2 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 switches to modem coefficients after the ending of CED Not every combination is possible. It depends on the under lying device and firmware.

Example

```
IFX_TAPI_SIG_DETECTION_t detect_sig;
IFX_TAPI_CH_STATUS_t param;
IFX_int32_t fd;

memset(&param, 0, sizeof(IFX_TAPI_CH_STATUS_t));
```

```
memset(&detect_sig, 0, sizeof(IFX_TAPI_SIG_DETECTION_t));

detect_sig.sig = IFX_TAPI_SIG_CEDTX | IFX_TAPI_SIG_PHASEREVRX;
ioctl(fd, IFX_TAPI_SIG_DETECT_ENABLE, &detect_sig);
ioctl(fd, IFX_TAPI_CH_STATUS_GET, &param);
if (param.signal & IFX_TAPI_SIG_CED)
{
   printf("Got CED from receive.\n");
}
```

4.1.5.3 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);
```

4.1.5.4 IFX_TAPI_TONE_CPTD_STOP

Description

Stops the call progress tone detection.

Prototype

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

Parameters

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

Return Values

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

4.1.6 CID Features Service

Caller ID support.

Table 27 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 28 Structure Overview of CID Features

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

Table 29 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 30 Enumerator Overview of CID Features

Name	Description
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_SERVICE_TYPE_t	Caller ID Services (defined in ETSI EN 300 659-3).
IFX_TAPI_CID_VMWI_t	List of VMWI settings.
IFX_TAPI_CID_ABSREASON_t	List of ABSCLI/ABSNAME settings.
IFX_TAPI_CID_RX_STATE_t	CID Receiver Status.
IFX_TAPI_CID_RX_ERROR_t	CID Receiver Errors.
IFX_TAPI_CID_HOOK_MODE_t	Caller ID transmission modes.
IFX_TAPI_CID_MSG_TYPE_t	Caller ID Message Type 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_ALERT_ETSI_t	List of ETSI Alerts.
IFX_TAPI_CID_VMWI_t	List of VMWI settings.

4.1.6.1 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);
```

4.1.6.2 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);
```

4.1.6.3 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);
```

4.1.6.4 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" );
}
```

4.1.6.5 IFX_TAPI_CID_RX_STOP

Description

Stop the CID Receiver.

Prototype

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

Parameters

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

Return Values

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

Example

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

4.1.6.6 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);
```

4.1.6.7 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.

4.1.7 Connection Control Service

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

Table 31 IO-control Overview of Connection Control Service

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

Table 32 Structure Overview of Connection Control Service

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_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_PHONE_t	Phone channel mapping structure used for IFX_TAPI_MAP_PHONE_ADD and IFX_TAPI_MAP_PHONE_REMOVE.
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 33 Enumerator Overview of Connection Control Service

Name	Description
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_PKT_EV_OOB_t	Enumeration for IFX_TAPI_PKT_RTP_CFG_t event setting.
IFX_TAPI_MAP_DATA_TYPE_t	Data channel destination types (conferencing).
IFX_TAPI_DATA_MAP_START_STOP_t	Start/Stop information for data channel mapping.

4.1.7.1 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) */
```

4.1.7.2 IFX_TAPI_DEC_START

Description

Starts the decoding of data.

Prototype

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

Parameters

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

Return Values

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

Example

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

4.1.7.3 IFX_TAPI_DEC_STOP

Description

Stops the decoding of data.

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

Prototype

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

Parameters

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

Return Values

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

Example

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

4.1.7.4 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

4.1.7.5 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);
```

4.1.7.6 IFX_TAPI_ENC_FRAME_LEN_GET

Description

This service reads the configured encoding length.

Prototype

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

Parameters

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

Return Values

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

Example

```
IFX_int32_t fd;
IFX_int32_t nFrameLength;

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

4.1.7.7 IFX_TAPI_ENC_FRAME_LEN_SET

Description

This service configures the encoding length.

Prototype

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

Parameters

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

Return Values

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

Example

```
FX_int32_t fd;
IFX_int32_t nFrameLength;

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

4.1.7.8 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) */
```

4.1.7.9 IFX_TAPI_ENC_START

Description

Start encoding and packetization.

Prototype

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

Parameters

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

Return Values

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

Remarks

The voice is serviced with the drivers read and write interface. The data format is dependent on the selected setup (coder, chip setup, and so on). For example, a RTP setup will receive RTP packets. Read and write are always

non-blocking. If the codec has not been set before with **IFX_TAPI_ENC_TYPE_SET** the encoder is started with G711.

Example

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

4.1.7.10 IFX_TAPI_ENC_STOP

Description

Stop encoding and packetization on this channel.

Prototype

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

Parameters

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

Return Values

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

Example

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

4.1.7.11 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);
```

4.1.7.12 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.

Data Type	Name	Description
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);
```

4.1.7.13 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.

Data Type	Name	Description
IFX_int32_t	IFX_TAPI_ENC_VOLUME_SET	I/O control identifier for this operation
IFX_int32_t	param	Recording volume, default 0x100

Return Values

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

Example

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

4.1.7.14 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);
```

4.1.7.15 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);
```

4.1.7.16 IFX_TAPI_JB_STATISTICS_RESET

Description

Resets the jitter buffer statistics.

Prototype

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

Parameters

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

Return Values

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

Example

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

4.1.7.17 IFX_TAPI_MAP_DATA_ADD

Description

This interface connects the data channel to a phone channel.

Prototype

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

4.1.7.18 IFX_TAPI_MAP_DATA_REMOVE

Description

This interface removes a data channel from a phone channel.

Prototype

```
IFX_int32_t ioctl (
        IFX_int32_t fd,
        IFX_TAPI_MAP_DATA_REMOVE,
        IFX_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 */
```

4.1.7.19 IFX_TAPI_MAP_PCM_ADD

Description

This interface connectes the PCM channel to a 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);
```

4.1.7.20 IFX_TAPI_MAP_PCM_REMOVE

Description

This interface removes the PCM channel from the phone channel.

Prototype

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

4.1.7.21 IFX_TAPI_MAP_PHONE_ADD

Description

This interface connects a phone channel to another phone channel.

Prototype

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

4.1.7.22 IFX_TAPI_MAP_PHONE_REMOVE

Description

This interface removes a phone channel from a phone channel.

Prototype

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

4.1.7.23 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);
```

4.1.7.24 IFX_TAPI_PKT_RTCP_STATISTICS_RESET

Description

Resets the RTCP statistics.

Prototype

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

Parameters

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

Return Values

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

Example

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

4.1.7.25 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);
```

4.1.7.26 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[6D] = 0x5;
param.nPTdown[12D] = 0x4;
ioctl(fd, IFX_TAPI_PKT_RTP_PT_CFG_SET, &param);
```

4.1.8 Miscellaneous Services

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

Table 34 IO-control Overview of Miscellaneous Services

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

Table 35 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 36 Union Overview of Miscellaneous Services

Name	Description
IFX_TAPI_EXCEPTION_t	Contains TAPI exception information.

Table 37 Enumerator Overview of Miscellaneous Service

Name	Description
IFX_TAPI_CAP_TYPE_t	Enumeration used for phone capabilities types.
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_LINE_HOOK_STATUS_t	Enumeration for hook status events.
IFX_TAPI_DIALING_STATUS_t	Enumeration for dial status events.
IFX_TAPI_LINE_STATUS_t	Enumeration for phone line status information.

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

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

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

4.1.8.2 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);
```

4.1.8.3 IFX_TAPI_CAP_NR

Description

This service returns the number of capabilities.

Prototype

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

Parameters

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

Data Type	Name	Description
IFX_int32_t	IFX_TAPI_CAP_NR	I/O control identifier for this operation
IFX_int32_t*	param	Pointer to the number of capabilities which are returned

Return Values

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

Example

```
IFX_TAPI_LINE_VOLUME_t param;
IFX_int32_t fd;
IFX_int32_t nCapNr;

/* Get the cap list size */
ioctl(fd, IFX_TAPI_CAP_NR, &nCapNr);
```

4.1.8.4 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, &rfds, NULL, NULL, NULL);
/* Select woke up from exception on fd_dev or data on fd[x] */
if (FD_ISSET(fd_dev, &rfds))
{
   /* Exception occurred */
   status[0].channels = 2;
   /* Get the status */
   ioctl(fd_dev, IFX_TAPI_CH_STATUS_GET, stat);
   for (i = 0; i < 2; i++)
   {
      if (stat[i].hook || stat[i].dialing ||
          stat[i].line || stat[i].digit || stat[i].signal)
      {
         printf("Handle exception\n");
      }
   }
}

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

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

4.1.8.5 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

4.1.8.6 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");
}
```

4.1.8.7 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);
```

4.1.8.8 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");
}
```

4.1.8.9 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);
```

4.1.9 Ringing Service

Table 38 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_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 39 Structure Overview of Ringing Service

Name	Description
IFX_TAPI_RING_CADENCE_t	Structure for ring cadence used in IFX_TAPI_RING_CADENCE_HR_SET.

4.1.9.1 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);
```

4.1.9.2 IFX_TAPI_RING_CADENCE_SET

Description

This service sets the ring cadence for the non-blocking ringing services.

Prototype

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

Parameters

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

Data Type	Name	Description
IFX_int32_t	IFX_TAPI_RING_CADENCE_SET	I/O control identifier for this operation
IFX_int32_t	param	The parameter defines the cadence. This value contains the encoded cadence sequence. One bit represents ring cadence voltage for 0.5 sec.

Return Values

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

Remarks

The number of ring bursts can be counted via exception PSTN line is ringing of service IFX_TAPI_EXCEPTION_GET.

Example

```
/* pattern of 3 sec. ring and 1 sec. pause : 1111 1100 1111 1100 ... */
IFX_uint32_t nCadence = 0xFCFCFCFC;
IFX_int32_t fd;

/* set the cadence sequence */
ioctl(fd, IFX_TAPI_RING_CADENCE_SET, nCadence);
```

4.1.9.3 IFX_TAPI_RING_START

Description

This service starts the non-blocking ringing on the phone line using the preconfigured ring cadence.

Prototype

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

Parameters

Data Type	Name	Description
IFX_int32_t	fd	File descriptor. It is applicable to phone channel file descriptors containing an analog interface.
IFX_int32_t	IFX_TAPI_CID_TX_SEQ_START	I/O control identifier for this operation
IFX_int32_t	param	Parameter is ignored.

Return Values

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

Remarks

This interface does not implement CID.

4.1.9.4 IFX_TAPI_RING_STOP

Description

This service stops non-blocking ringing on the phone line which was started before with service IFX_TAPI_RING_START.

Prototype

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

Parameters

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

Return Values

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

Example

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

4.1.10 PCM Service

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

Table 40 IO-control Overview of PCM Service

Name	Description
IFX_TAPI_PCM_ACTIVATION_GET	This service gets the activation status of the PCM time slots configured for this channel.
IFX_TAPI_PCM_ACTIVATION_SET	This service activate/deactivates the PCM time slots configured for this channel.
IFX_TAPI_PCM_CFG_GET	This service gets the configuration of the PCM interface.
IFX_TAPI_PCM_CFG_SET	This service sets the configuration of the PCM interface.

4.1.10.1 IFX_TAPI_PCM_ACTIVATION_GET

Description

This service gets the activation status of the PCM time slots configured for this channel.

Prototype

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

Parameters

Data Type	Name	Description
IFX_int32_t	fd	File descriptor. It is applicable to phone channel file descriptors containing a PCM interface.
IFX_int32_t	IFX_TAPI_PCM_ACTIVATION_GET	I/O control identifier for this operation
IFX_int32_t	param	The parameter points to an integer which returns the status 0: The time slot is deactive 1: The time slot is active

Return Values

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

Example

```
IFX_TAPI_LINE_VOLUME_t param;
IFX_int32_t fd;
IFX_return_t ret;
IFX_int32_t bAct;
```

```
ret = ioctl(fd, IFX_TAPI_PCM_ACTIVATION_GET, &bAct);
if ((IFX_SUCCESS == ret) && (1 == bAct))
{
   printf("Activated\n");
}
```

4.1.10.2 IFX_TAPI_PCM_ACTIVATION_SET

Description

This service activate / deactivates the PCM time slots configured for this channel.

Prototype

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

Parameters

Data Type	Name	Description
IFX_int32_t	fd	File descriptor. It is applicable to phone channel file descriptors containing a PCM interface.
IFX_int32_t	IFX_TAPI_PCM_ACTIVATION_SET	I/O control identifier for this operation
IFX_int32_t	param	The parameter defines the activation status 0 deactivate the time slot 1 activate the time slot

Return Values

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

Example

```
/* Activate the PCM timeslot */
ioctl(fd, IFX_TAPI_PCM_ACTIVATION_SET, 1);
```

4.1.10.3 IFX_TAPI_PCM_CFG_GET

Description

This service gets the configuration of the PCM interface.

Prototype

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

```
        IFX_TAPI_PCM_CFG_GET,
        IFX_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	The parameter points to a IFX_TAPI_PCM_CFG_t structure.

Return Values

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

Example

```
IFX_TAPI_PCM_CFG_t Pcm;
IFX_int32_t fd;

memset(&Pcm, 0, sizeof(IFX_TAPI_PCM_CFG_t));
ioctl(fd, IFX_TAPI_PCM_CFG_GET, &Pcm);
```

4.1.10.4 IFX_TAPI_PCM_CFG_SET

Description

This service sets the configuration of the PCM interface.

Prototype

```
IFX_int32_t ioctl (
        IFX_int32_t fd,
        IFX_TAPI_PCM_CFG_SET,
        IFX_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);
```

4.1.11 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 packets). All fax services apply to data channels except otherwise stated.

Table 41 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 42 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 43 Enumerator Overview of Fax T.38 Service

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

4.1.11.1 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);
```

4.1.11.2 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);
```

4.1.11.3 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);
```

4.1.11.4 IFX_TAPI_T38_STOP

Description

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

Prototype

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

Parameters

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

Return Values

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

Example

```
IFX_TAPI_LINE_VOLUME_t param;
IFX_int32_t fd;

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

4.2 Type Definition Reference

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

4.2.1 Basic Type Definitions

This section describes the basic type definitions such as:

- **IFX_return_t**
- **IFX_boolean_t**
- **IFX_uint8_t**
- **IFX_int8_t**
- **IFX_uint32_t**
- **IFX_int32_t**
- **IFX_uint16_t**
- **IFX_int16_t**
- **IFX_char_t**
- **IFX_void_t**

4.2.1.1 IFX_return_t

Description

All the APIs return a Success or a Failure based on their execution Status. The return code is set to IFX_ERROR only if an error occurs, otherwise its value is IFX_SUCCESS.

Prototype

```
typedef enum
{
        IFX_ERROR = -1,
        IFX_SUCCESS = 0
} IFX_return_t;
```

Parameters

Name	Value	Description
IFX_ERROR	-1_D	Operation failed.
IFX_SUCCESS	0_D	Operation was successful.

4.2.1.2 IFX_boolean_t

Description

Definition for true and false.

Prototype

```
typedef enum
{
        IFX_FALSE = 0,
        IFX_TRUE = 1
} IFX_boolean_t;
```

Parameters

Name	Value	Description
IFX_FALSE	0_D	False.
IFX_TRUE	1_D	True.

4.2.1.3 IFX_uint8_t

Prototype

```
typedef unsigned char IFX_uint8_t;
```

Parameters

Data Type	Name	Description
unsigned char	IFX_uint8_t	This is the unsigned char 8-bit datatype.

4.2.1.4 IFX_int8_t

Prototype

```
typedef char IFX_int8_t;
```

Parameters

Data Type	Name	Description
char	IFX_int8_t	This is the char 8-bit datatype.

4.2.1.5 IFX_uint32_t

Prototype

```
typedef unsigned int IFX_uint32_t;
```

Parameters

Data Type	Name	Description
unsigned int	IFX_uint32_t	This is the unsigned int 32-bit datatype.

4.2.1.6 IFX_int32_t

Prototype

```
typedef int IFX_int32_t;
```

Parameters

Data Type	Name	Description
int	IFX_int32_t	This is the int 32-bit datatype.

4.2.1.7 IFX_uint16_t

Prototype

```
typedef unsigned short IFX_uint16_t;
```

Parameters

Data Type	Name	Description
unsigned short	IFX_uint16_t	This is the unsigned short int 16-bit datatype.

4.2.1.8 IFX_int16_t

Prototype

```
typedef short IFX_int16_t;
```

Parameters

Data Type	Name	Description
short	IFX_int16_t	This is the short int 16-bit datatype.

4.2.1.9 IFX_char_t

Prototype

```
typedef char IFX_char_t;
```

Parameters

Data Type	Name	Description
char	IFX_char_t	This is the char 8-bit datatype.

4.2.1.10 IFX_void_t

Prototype

```
typedef void IFX_void_t;
```

Parameters

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

4.2.2 IO-control Reference

This chapter contains the IO-control reference.

Table 44 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.
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.

Table 44 IO-control Overview of TAPI Interfaces (cont'd)

Name	Description
IFX_TAPI_LINE_HOOK_VT_SET	Specifies the time for hook, pulse digit and hook flash validation.
IFX_TAPI_LINE_LEVEL_SET	This service enables or disables a high level path of a phone channel.
IFX_TAPI_MAP_DATA_ADD	This interface adds the data channel to an analog phone device.
IFX_TAPI_MAP_DATA_REMOVE	This interface removes a data channel from an analog phone device.
IFX_TAPI_MAP_PCM_ADD	This interface adds the PCM channel to an analog phone device.
IFX_TAPI_MAP_PCM_REMOVE	This interface removes the PCM channel to an analog phone device.
IFX_TAPI_MAP_PHONE_ADD	This interface adds the phone channel to another analog phone channel.
IFX_TAPI_MAP_PHONE_REMOVE	This interface removes a phone channel from an analog phone device.
IFX_TAPI_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_EV_OOB_SET	This service controls the DTMF sending mode.
IFX_TAPI_PKT_RTCP_STATISTICS_GET	Retrieves RTCP statistics.
IFX_TAPI_PKT_RTCP_STATISTICS_RESET	Resets the RTCP statistics.
IFX_TAPI_PKT_RTP_CFG_SET	This interface configures RTP and RTCP fields for a new connection.
IFX_TAPI_PKT_RTP_PT_CFG_SET	Change the payload type table.
IFX_TAPI_PULSE_ASCII_GET	This service reads the ASCII character representing the pulse digit out of the receive buffer.
IFX_TAPI_PULSE_GET	This service reads the dial pulse digit out of the receive buffer.
IFX_TAPI_PULSE_READY	This service checks if a pulse digit was received.
IFX_TAPI_RING_CADENCE_HR_SET	This service sets the high resolution ring cadence for the non-blocking ringing services.
IFX_TAPI_RING_CADENCE_SET	This service sets the ring cadence for the non-blocking ringing services.
IFX_TAPI_RING_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.

Table 44 IO-control Overview of TAPI Interfaces (cont'd)

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.
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.

4.2.3 Constant Reference

This chapter contains the constant reference.

Table 45 Constant Reference for TAPI Interfaces

Name and Description	Value
IFX_TAPI_IOC_MAGIC Magic number for ioctls	0000000_B
IFX_TAPI_TONE_STEPS_MAX Maximum tone generation steps, also called cadences.	6_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_RING_CADENCE_MAX Max number of ring cadences.	40_D
IFX_TAPI_CID_MSG_LEN_MAX Max number of octets for a CID message element.	50_D

Table 45 Constant Reference for TAPI Interfaces (cont'd)

Name and Description	Value
IFX_TAPI_CID_SIZE_MAX Max length for a CID message.	128_D
IFX_TAPI_ENC_TYPE_MAX Max number of supported vocoders for the payload table.	00000000_D
IFX_TAPI_TONE_SIMPLE_MAX Max number of simple tones part of a composed tone.	7_D
IFX_TAPI_CID_RX_FIFO_SIZE CID RX FIFO Size	10_D
IFX_TAPI_PULSE_FIFO_SIZE FIFO size for the detected pulse digits	20_D
IFX_TAPI_DTMF_FIFO_SIZE FIFO size for the detected DTMF digits	40_D
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_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

4.2.4 Structure Reference

This chapter contains the Structure reference.

Table 46 Structure Overview of TAPI Interfaces

Name	Description
IFX_TAPI_EXCEPTION_BITS_t	Exception bits.
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.

Table 46 Structure Overview of TAPI Interfaces (cont'd)

Name	Description
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_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_JB_CFG_t	Structure for jitter buffer configuration used by IFX_TAPI_JB_CFG_SET.
IFX_TAPI_JB_STATISTICS_t	Structure for Jitter Buffer statistics used by ioctl IFX_TAPI_JB_STATISTICS_GET.
IFX_TAPI_LEC_CFG_t	Line echo canceller (LEC) configuration.
IFX_TAPI_LINE_HOOK_VT_t	Structure used for validation times of hook, hook flash and pulse dialing.
IFX_TAPI_LINE_VOLUME_t	Phone speaker phone and microphone volume settings used for ioctl IFX_TAPI_PHONE_VOLUME_SET.
IFX_TAPI_MAP_DATA_t	Phone channel mapping structure used for IFX_TAPI_MAP_DATA_ADD and IFX_TAPI_MAP_DATA_REMOVE.
IFX_TAPI_MAP_PCM_t	Phone channel mapping structure used for IFX_TAPI_MAP_PCM_ADD and IFX_TAPI_MAP_PCM_REMOVE.
IFX_TAPI_MAP_PHONE_t	Phone channel mapping structure used for IFX_TAPI_MAP_PHONE_ADD and IFX_TAPI_MAP_PHONE_REMOVE.
IFX_TAPI_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.

Table 46 Structure Overview of TAPI Interfaces (cont'd)

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

4.2.4.1 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.
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.

4.2.4.2 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.

4.2.4.3 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.

4.2.4.4 IFX_TAPI_CH_STATUS_t

Description

Structure used for IFX_TAPI_CH_STATUS_GET to query the phone status of one or more channels.

Prototype

```
typedef struct
{
        IFX_char_t channels;
        IFX_char_t hook;
        IFX_char_t dialing;
        IFX_char_t digit;
        IFX_int32_t line;
        IFX_uint32_t signal;
         unknown;
```

```
        IFX_uint32_t device;
        IFX_uint32_t event;
        IFX_uint32_t error;
} IFX_TAPI_CH_STATUS_t;
```

Parameters

Data Type	Name	Description
IFX_char_t	channels	Size of the list. If 0 only the status of this channel is queried. If channels not equal to 0 all channels starting with 0 till the value of channels are retrieved. As the return value it defines the channel number
IFX_char_t	hook	Lists the hook status events. 1_H **IFX_TAPI_LINE_HOOK_STATUS_HOOK**, Hook event detected (on- or off-hook) 2_H **IFX_TAPI_CH_STATUS_FLASH**, Flash hook event detected
IFX_char_t	dialing	Lists the dial status events. 1_H **IFX_TAPI_CH_STATUS_DTMF**, DTMF digit detected 2_H **IFX_TAPI_CH_STATUS_PULSE**, Pulse digit detected
IFX_char_t	digit	Contains the digit in case of a dialing event.
IFX_int32_t	line	Signals the line status and fault conditions. 1_H **IFX_TAPI_LINE_STATUS_RINGING**, PSTN line is ringing 2_H **IFX_TAPI_LINE_STATUS_RINGFINISHED**, ringing finished on PSTN line 4_H **IFX_TAPI_LINE_STATUS_FAX**, Fax status change 10_H **IFX_TAPI_LINE_STATUS_GNDKEY**, Ground key detected 20_H **IFX_TAPI_LINE_STATUS_GNDKEYHIGH**, Ground key high detected 40_H **IFX_TAPI_LINE_STATUS_OTEMP**, Overtemperature detected 200_H **IFX_TAPI_LINE_STATUS_GR909RES**, Cid Receiver event occured

Data Type	Name	Description
IFX_uint32_t	signal	Signals the detection of events. The signals/events detected are reported in this field and represented according to the definition of IFX_TAPI_SIG_t. Following description is taken from the definition of IFX_TAPI_SIG_t. Please check it out for further information. 0_H **IFX_TAPI_SIG_NONE**, no signal detected 1_H **IFX_TAPI_SIG_DISRX**, V.21 Preamble Fax Tone, Digital Identification Signal (DIS), receive path 2_H **IFX_TAPI_SIG_DISTX**, V.21 Preamble Fax Tone, Digital Identification Signal (DIS), transmit path 4_H **IFX_TAPI_SIG_DIS**, V.21 Preamble Fax Tone in all path, Digital Identification Signal (DIS) 8_H **IFX_TAPI_SIG_CEDRX**, V.25 2100 Hz (CED) Modem/Fax Tone, receive path 10_H **IFX_TAPI_SIG_CEDTX**, V.25 2100 Hz (CED) Modem/Fax Tone, transmit path 20_H **IFX_TAPI_SIG_CED**, V.25 2100 Hz (CED) Modem/Fax Tone in all paths 40_H **IFX_TAPI_SIG_CNGFAXRX**, CNG Fax Calling Tone (1100 Hz) receive path 80_H **IFX_TAPI_SIG_CNGFAXTX**, CNG Fax Calling Tone (1100 Hz) transmit path 100_H **IFX_TAPI_SIG_CNGFAX**, CNG Fax Calling Tone (1100 Hz) in all paths 200_H **IFX_TAPI_SIG_CNGMODRX**, CNG Modem Calling Tone (1300 Hz) receive path 400_H **IFX_TAPI_SIG_CNGMODTX**, CNG Modem Calling Tone (1300 Hz) transmit path 800_H **IFX_TAPI_SIG_CNGMOD**, CNG Modem Calling Tone (1300 Hz) in all paths 1000_H**IFX_TAPI_SIG_PHASEREVRX**, Phase reversal detection receive path 2000_H**IFX_TAPI_SIG_PHASEREVTX**, Phase reversal detection transmit path 4000_H**IFX_TAPI_SIG_PHASEREV**, Phase reversal detection in all paths 8000_H**IFX_TAPI_SIG_AMRX**, Amplitude modulation receive path 10000_H**IFX_TAPI_SIG_AMTX**, Amplitude modulation transmit path

Data Type	Name	Description
		20000_H**IFX_TAPI_SIG_AM**, Amplitude modulation. 40000_H **IFX_TAPI_SIG_TONEHOLDING_ENDRX**, Modem tone holding signal stopped receive path 80000_H **IFX_TAPI_SIG_TONEHOLDING_ENDTX**, Modem tone holding signal stopped transmit path 100000_H **IFX_TAPI_SIG_TONEHOLDING_END**, Modem tone holding signal stopped all paths 200000_H**IFX_TAPI_SIG_CEDENDRX**, End of signal CED detection receive path 400000_H**IFX_TAPI_SIG_CEDENDTX**, End of signal CED detection transmit path 800000_H**IFX_TAPI_SIG_CEDEND**, End of signal CED detection. 1000000_H**IFX_TAPI_SIG_CPTD**, Call progress tone detected.
IFX_uint32_t	device	Device specific status information
IFX_uint32_t	event	RFC 2833 event payload information. This field contains the last event (**IFX_TAPI_PKT_EV_NUM_t**), that was received and played out by the device.
IFX_uint32_t	error	Runtime error of type **IFX_TAPI_RUNTIME_ERROR_t**

4.2.4.5 IFX_TAPI_CID_ABS_REASON_t

Description

Structure containing CID configuration for ETSI standard using DTMF transmission.

Prototype

```
typedef struct
{
        IFX_char_t nLength;
        IFX_char_t unavailable[TAPI_CID_MAX_MSG_LEN];
        IFX_char_t private[TAPI_CID_MAX_MSG_LEN];
} IFX_TAPI_CID_ABS_REASON_t;
```

Parameters

Data Type	Name	Description
IFX_char_t	nLength	Length of the coded strings.
IFX_char_t	unavailable[TAPI_CID_MAX_MSG_LEN]	String representing code for unavailable/unknown CLI. Default “00”.
IFX_char_t	private[TAPI_CID_MAX_MSG_LEN]	String representing code for private/withheld CLI. Default “01”.

4.2.4.6 IFX_TAPI_CID_CFG_t

Description

Structure containing CID configuration possibilities.

Prototype

```
typedef struct
{
        IFX_int32_t nStandard;
        IFX_TAPI_CID_STD_TYPE_t cfg;
} IFX_TAPI_CID_CFG_t;
```

Parameters

Data Type	Name	Description
IFX_int32_t	nStandard	Standard used (enumerated in IFX_TAPI_CID_STD_t). Default IFX_TAPI_CID_STD_TELCORDIA.
IFX_TAPI_CID_STD_TYPE_t	cfg	Union of the different standards. Default IFX_TAPI_CID_STD_TELCORDIA_t.

4.2.4.7 IFX_TAPI_CID_DTMF_CFG_t

Description

Structure containing the configuration information for DTMF CID.

Prototype

```
typedef struct
{
        IFX_char_t startTone;
        IFX_char_t stopTone;
        IFX_char_t infoSstartTone;
        IFX_char_t startTone;
        IFX_int32_t digitTime;
        IFX_int32_t interDigitTime;
} IFX_TAPI_CID_DTMF_CFG_t;
```

Parameters

Data Type	Name	Description
IFX_char_t	startTone	Tone id for starting tone. Default is DTMF 'A'.
IFX_char_t	stopTone	Tone id for stop tone. Default is DTMF 'C'.
IFX_char_t	infoSstartTone	Tone id for starting information tone. Default is DTMF 'B'.
IFX_char_t	startTone	Tone id for starting redirection tone. Default is DTMF 'D'.
IFX_int32_t	digitTime	Time for DTMF digit duration. Default is 50 ms.
IFX_int32_t	interDigitTime	Time between DTMF digits in ms. Default is 50 ms.

4.2.4.8 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 transmit. Default 300 bits.
IFX_uint32_t	seizureRX	Minimum number of seizure bits for FSK transmit. Default 300 bits.
IFX_uint32_t	markTXOnhook	Number of mark bits for on-hook FSK receive. Default 180 bits.
IFX_uint32_t	markTXOffhook	Number of mark bits for off-hook FSK receive. Default 80 bits.
IFX_uint32_t	markRXOnhook	Minimum number of mark bits for on-hook FSK receive. Default 180 bits.
IFX_uint32_t	markRXOffhook	Minimum number of mark bits for off-hook FSK receive. Default 80 bits.

4.2.4.9 IFX_TAPI_CID_MSG_DATE_t

Description

Structure for element types (IFX_TAPI_CID_SERVICE_TYPE_t) containing date and time information.

Prototype

```
typedef struct
{
        IFX_int32_t elementType;
        IFX_int32_t day;
        IFX_int32_t month;
        IFX_int32_t hour;
        IFX_int32_t mn;
} IFX_TAPI_CID_MSG_DATE_t;
```

Parameters

Data Type	Name	Description
IFX_int32_t	elementType	Element type
IFX_int32_t	day	Day
IFX_int32_t	month	Month
IFX_int32_t	hour	Hour
IFX_int32_t	mn	Minute

4.2.4.10 IFX_TAPI_CID_MSG_STRING_t

Description

Structure for element types (IFX_TAPI_CID_SERVICE_TYPE_t) with dynamic length (line numbers or names).

Prototype

```
typedef struct
{
        IFX_TAPI_CID_SERVICE_TYPE_t elementType;
        IFX_int32_t len;
        IFX_char_t element[TAPI_CID_MAX_MSG_LEN];
} IFX_TAPI_CID_MSG_STRING_t;
```

Parameters

Data Type	Name	Description
IFX_TAPI_CID_SERVICE_TYPE_t	elementType	Element type
IFX_int32_t	len	Length of the message array
IFX_char_t	element[TAPI_CID_MAX_MSG_LEN]	String containing the message element.

4.2.4.11 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.

Note: Not clear need of "info" field for DTMF transmission, to be clarified.

Prototype

```
typedef struct
{
        IFX_int32_t txMode;
        IFX_int32_t messageType;
        IFX_int32_t nMsgElements;
        IFX_TAPI_CID_MSG_ELEMENT_t* message;
} IFX_TAPI_CID_MSG_t;
```

Parameters

Data Type	Name	Description
IFX_int32_t	txMode	Define the Transmission Mode (enumerated in **IFX_TAPI_CID_HOOK_MODE_t**). Default **IFX_TAPI_CID_HM_ONHOOK**.
IFX_int32_t	messageType	Define the Message Type to be displayed (enumerated in **IFX_TAPI_CID_MSG_TYPE_t**). Default **IFX_TAPI_CID_MT_TRANSPARENT**.
IFX_int32_t	nMsgElements	Number of elements of the message array.
IFX_TAPI_CID_MSG_ELEMENT_t*	message	Message array.

4.2.4.12 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

4.2.4.13 IFX_TAPI_CID_MSG_VALUE_t

Description

Structure for element types (IFX_TAPI_CID_SERVICE_TYPE_t) with one value (length 1).

Prototype

```
typedef struct
{
       IFX_TAPI_CID_SERVICE_TYPE_t elementType;
       IFX_uint8_t element;
} IFX_TAPI_CID_MSG_VALUE_t;
```

Parameters

Data Type	Name	Description
IFX_TAPI_CID_SERVICE_TYPE_t	elementType	Element type
IFX_uint8_t	element	Value for the message element

4.2.4.14 IFX_TAPI_CID_RX_DATA_t

Description

Structure for Caller ID receiver data

Prototype

```
typedef struct
{
       IFX_uint8_t data;
       IFX_int32_t nSize;
} IFX_TAPI_CID_RX_DATA_t;
```

Parameters

Data Type	Name	Description
IFX_uint8_t	data	0_D **IFX_TAPI_CID_RX_SIZE_MAX**, Caller Id receiver data.
IFX_int32_t	nSize	Caller Id receiver datasize in bytes.

4.2.4.15 IFX_TAPI_CID_RX_STATUS_t

Description

Structure for Caller ID receiver status.

Prototype

```
typedef struct
{
        IFX_uint8_t nStatus;
        IFX_uint8_t nError;
} IFX_TAPI_CID_RX_STATUS_t;
```

Parameters

Data Type	Name	Description
IFX_uint8_t	nStatus	Caller Id receiver actual status using IFX_TAPI_CID_RX_STATE_t. It can be any of the following values: 0_D **IFX_TAPI_CIDRX_STAT_INACTIVE**, CID Receiver is not active. 1_D **IFX_TAPI_CIDRX_STAT_ACTIVE**, CID Receiver is active. 2_D **IFX_TAPI_CIDRX_STAT_ONGOING**, CID Receiver is just receiving data. 3_D **IFX_TAPI_CIDRX_STAT_DATA_RDY**, CID Receiver is completed.
IFX_uint8_t	nError	Caller Id receiver actual error code using IFX_TAPI_CID_RX_ERROR_t. It can be any of the following values: 0_B **IFX_TAPI_CID_RX_ERROR_NONE**, No Error during CID Receiver operation. 1_D **IFX_TAPI_CID_RX_ERROR_READ**, Reading error during CID Receiver operation.

4.2.4.16 IFX_TAPI_CID_STD_ETSI_DTMF_t

Description

Structure containing CID configuration for ETSI standard using DTMF transmission.

Prototype

```
typedef struct
{
        IFX_TAPI_CID_TIMING_t* pCIDTiming;
        IFX_TAPI_CID_DTMF_CFG_t* pDTMFConf;
        IFX_int32_t nETSIAlertRing;
        IFX_int32_t nETSIAlertNoRing;
        IFX_int32_t nAlertToneOffhook;
        IFX_int32_t nAlertToneOnhook;
```

```
        IFX_int32_t ringPulseTime;
        IFX_char_t ackTone;
        IFX_TAPI_CID_ABS_REASON_t* pABSCLICode;
} IFX_TAPI_CID_STD_ETSI_DTMF_t;
```

Parameters

Data Type	Name	Description
IFX_TAPI_CID_TIMING_t*	pCIDTiming	Pointer to a structure containing timing information. If the parameter is not given, IFX_TAPI_CID_TIMING_t default values will be used.
IFX_TAPI_CID_DTMF_CFG_t*	pDTMFConf	Pointer to a structure containing DTMF configuration parameters. If the parameter is not given, IFX_TAPI_CID_DTMF_CFG_t default values will be used.
IFX_int32_t	nETSIAlertRing	Type of ETSI Alert of on-hook services associated to ringing (enumerated in IFX_TAPI_CID_ALERT_ETSI_t). Default IFX_TAPI_CID_ALERT_ETSI_FR.
IFX_int32_t	nETSIAlertNoRing	Type of ETSI Alert of on-hook services not associated to ringing (enumerated in IFX_TAPI_CID_ALERT_ETSI_t). Default IFX_TAPI_CID_ALERT_ETSI_RP.
IFX_int32_t	nAlertToneOffhook	Tone table index for the alert tone to be used. Required for automatic CID/MWI generation. By default TAPI uses an internal tone definition.
IFX_int32_t	nAlertToneOnhook	Tone table index for the alert tone to be used. Required for automatic CID/MWI generation. By default TAPI uses an internal tone definition.
IFX_int32_t	ringPulseTime	Duration of Ring Pulse, in ms, default 500 ms.
IFX_char_t	ackTone	DTMF ACK after CAS, used for offhook transmission. Default DTMF 'D'.
IFX_TAPI_CID_ABS_REASON_t*	pABSCLICode	Pointer to a structure containing the coding for absence reason of calling number.

4.2.4.17 IFX_TAPI_CID_STD_ETSI_FSK_t

Description

Structure containing CID configuration for ETSI standard using FSK transmission.

Prototype

```
typedef struct
{
        IFX_TAPI_CID_TIMING_t* pCIDTiming;
        IFX_TAPI_CID_FSK_CFG_t* pFSKConf;
        IFX_int32_t nETSIAlertRing;
        IFX_int32_t nETSIAlertNoRing;
```

```
        IFX_int32_t nAlertToneOffhook;
        IFX_int32_t nAlertToneOnhook;
        IFX_int32_t ringPulseTime;
        IFX_char_t ackTone;
} IFX_TAPI_CID_STD_ETSI_FSK_t;
```

Parameters

Data Type	Name	Description
IFX_TAPI_CID_TIMING_t*	pCIDTiming	Pointer to a structure containing timing information. If the parameter is not given, IFX_TAPI_CID_TIMING_t default values will be used.
IFX_TAPI_CID_FSK_CFG_t *	pFSKConf	Pointer to a structure containing FSK configuration parameters. If the parameter is not given, IFX_TAPI_CID_FSK_CFG_t default values will be used.
IFX_int32_t	nETSIAlertRing	Type of ETSI Alert of on-hook services associated to ringing (enumerated in IFX_TAPI_CID_ALERT_ETSI_t). Default IFX_TAPI_CID_ALERT_ETSI_FR.
IFX_int32_t	nETSIAlertNoRing	Type of ETSI Alert of on-hook services not associated to ringing (enumerated in IFX_TAPI_CID_ALERT_ETSI_t). Default IFX_TAPI_CID_ALERT_ETSI_RP.
IFX_int32_t	nAlertToneOffhook	Tone table index for the alert tone to be used. Required for automatic CID/MWI generation. By default TAPI uses an internal tone definition.
IFX_int32_t	nAlertToneOnhook	Tone table index for the alert tone to be used. Required for automatic CID/MWI generation. By default TAPI uses an internal tone definition.
IFX_int32_t	ringPulseTime	Duration of Ring Pulse, in ms, default 500 ms.
IFX_char_t	ackTone	DTMF ACK after CAS, used for off-hook transmission. Default DTMF is ‘D’.

4.2.4.18 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.

4.2.4.19 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'.

4.2.4.20 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'.

4.2.4.21 IFX_TAPI_CID_TIMING_t

Description

Structure containing the timing for CID transmission.

Prototype

```
typedef struct
{
        IFX_int32_t beforeData;
        IFX_int32_t dataOut2restoreTimeOnhook;
        IFX_int32_t dataOut2restoreTimeOffhook;
        IFX_int32_t ack2dataOutTime;
        IFX_int32_t cas2ackTime;
        IFX_int32_t afterAckTimeout;
        IFX_int32_t afterFirstRing;
        IFX_int32_t afterRingPulse;
        IFX_int32_t afterDTASOnhook;
        IFX_int32_t afterLineReversal;
        IFX_int32_t afterOSI;
} IFX_TAPI_CID_TIMING_t;
```

Parameters

Data Type	Name	Description
IFX_int32_t	beforeData	Time to wait after AS, in ms, before data transmission. Default 300 ms. Used only for NTT off-hook trasmission.
IFX_int32_t	dataOut2restoreTimeOnhook	Time to wait after data transmission, in ms, for on-hook services. Default 300 ms.
IFX_int32_t	dataOut2restoreTimeOffhook	Time to wait after data transmission, in ms, for off-hook services. Default 60 ms.
IFX_int32_t	ack2dataOutTime	Time to wait after ack detection, in ms, before data transmission. Default 55 ms.
IFX_int32_t	cas2ackTime	Time-out for ack detection, in ms. Default 160 ms.
IFX_int32_t	afterAckTimeout	Time to wait after ACK time-out, in ms. Default 50 ms.
IFX_int32_t	afterFirstRing	Time to wait after first ring, in ms, before starting data transmission. Default 600 ms.
IFX_int32_t	afterRingPulse	Time to wait after ring pulse, in ms, before starting data transmission. Default 600 ms.
IFX_int32_t	afterDTASOnhook	Time to wait after DTAS, in ms, before starting data transmission. Default 45 ms.
IFX_int32_t	afterLineReversal	Time to wait after line reversal, in ms, before successive operation[1] is performed. Default 100 ms.
IFX_int32_t	afterOSI	Time to wait after OSI signal, in ms, before data transmission. Default 300 ms.

1) Timing between line reversal and DTAS (ETSI standard) or CAR (NTT standard) signal.

4.2.4.22 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.
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.

4.2.4.23 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

Data Type	Name	Description
IFX_uint16_t	nNwJitter	Network jitter value. Not supported anymore
IFX_uint16_t	nPODelay	Play out delay
IFX_uint16_t	nMaxPODelay	Maximum play out delay.
IFX_uint16_t	nMinPODelay	Minimum play out delay.
IFX_uint32_t	nPackets	Received packet number.
IFX_uint16_t	nLost	Lost packets number. Not supported anymore
IFX_uint16_t	nInvalid	Invalid packet number.
IFX_uint16_t	nDuplicate	Duplication packet number. Not supported anymore
IFX_uint16_t	nLate	Late packets number.
IFX_uint16_t	nEarly	Early packets number.
IFX_uint16_t	nResync	Resynchronizations number.
IFX_uint32_t	nIsUnderflow	Total number of injected samples since the beginning of the connection or since the last statistic reset due to jitter buffer underflows.
IFX_uint32_t	nIsNoUnderflow	Total number of injected samples since the beginning of the connection or since the last statistic reset in case of normal jitter buffer operation, which means when there is not a jitter buffer underflow.
IFX_uint32_t	nIsIncrement	Total number of injected samples since the beginning of the connection or since the last statistic reset in case of jitter buffer increments.
IFX_uint32_t	nSkDecrement	Total number of skipped lost samples since the beginning of the connection or since the last statistic reset in case of jitter buffer decrements.
IFX_uint32_t	nDsDecrement	Total number of dropped samples since the beginning of the connection or since the last statistic reset in case of jitter buffer decrements.
IFX_uint32_t	nDsOverflow	Total number of dropped samples since the beginning of the connection or since the last statistic reset in case of jitter buffer overflows.
IFX_uint32_t	nSid	Total number of comfort noise samples since the beginning of the connection or since the last statistic reset.

4.2.4.24 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

4.2.4.25 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

4.2.4.26 IFX_TAPI_LINE_VOLUME_t

Description

Phone speaker phone and microphone volume settings used for ioctl IFX_TAPI_PHONE_VOLUME_SET.

Prototype

```
typedef struct
{
        IFX_int32_t nGainRx;
        IFX_int32_t nGainTx;
} IFX_TAPI_LINE_VOLUME_t;
```

Parameters

Data Type	Name	Description
IFX_int32_t	nGainRx	Volume setting for the receiving path, speakerphone The value is given in dB with the range (-24dB to 24dB)
IFX_int32_t	nGainTx	Volume setting for the transmitting path, microphone The value is given in dB with the range (-24dB to 24dB)

4.2.4.27 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. Phone channels numbers start from 0.
IFX_TAPI_MAP_DATA_TYPE_t	nChType	Type of the destination channel. 0_D **IFX_TAPI_MAP_TYPE_DEFAULT**, Default selected (phone channel) 1_D **IFX_TAPI_MAP_TYPE_CODER**, not supported 2_D **IFX_TAPI_MAP_TYPE_PCM**, type is PCM 3_D **IFX_TAPI_MAP_TYPE_PHONE**, type is phone channel
IFX_TAPI_DATA_MAP_START_STOP_t	nRecStart	Enables or disables the recording service or leaves as it is. 0_D **IFX_TAPI_MAP_DATA_UNCHANGED**, Do not modify the status of the recorder 1_D **IFX_TAPI_MAP_DATA_START**, Recording is started, same as IFX_TAPI_ENC_START 2_D **IFX_TAPI_MAP_DATA_STOP**, Recording is stopped, same as IFX_TAPI_ENC_STOP
IFX_TAPI_DATA_MAP_START_STOP_t	nPlayStart	Enables or disables the play service or leaves as it is. 0_D **IFX_TAPI_MAP_DATA_UNCHANGED**, Do not modify the status of the recorder 1_D **IFX_TAPI_MAP_DATA_START**, Playing is started, same as IFX_TAPI_ENC_START 2_D **IFX_TAPI_MAP_DATA_STOP**, Playing is stopped, same as IFX_TAPI_ENC_STOP

4.2.4.28 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_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.

4.2.4.29 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_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.

4.2.4.30 IFX_TAPI_PCM_CFG_t

Description

Structure for PCM configuration.

Prototype

```
typedef struct
{
        IFX_uint32_t nTimeslotRX;
        IFX_uint32_t nTimeslotTX;
        IFX_uint32_t nHighway;
```

```
        IFX_uint32_t nResolution;
        IFX_uint32_t nRate;
} IFX_TAPI_PCM_CFG_t;
```

Parameters

Data Type	Name	Description
IFX_uint32_t	nTimeslotRX	PCM time slot for the receive direction.
IFX_uint32_t	nTimeslotTX	PCM time slot for the transmit direction.
IFX_uint32_t	nHighway	Defines the PCM highway number which is connected to the channel.
IFX_uint32_t	nResolution	Defines the PCM interface coding. 0_D **IFX_TAPI_PCM_RES_ALAW_8BIT**, 8-bit A-law 1_D **IFX_TAPI_PCM_RES_MLAW_8BIT**, 8bit µ-law 2_D **IFX_TAPI_PCM_RES_LINEAR_16BIT**, 16-bit linear
IFX_uint32_t	nRate	Defines the PCM sample rate in kHz.

4.2.4.31 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.

Data Type	Name	Description
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.
IFX_uint32_t	osent	Sent octets count.
IFX_uint32_t	rssrc	Data source.
IFX_uint32_t	fraction	Receivers fraction loss.
IFX_int32_t	lost	Receivers packet lost.
IFX_uint32_t	last_seq	Extended last seq nr received.
IFX_uint32_t	jitter	Receives interarrival jitter.
IFX_uint32_t	lsr	Last sender report.
IFX_uint32_t	dlsr	Delay since last sender report.

4.2.4.32 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).

Data Type	Name	Description
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.

4.2.4.33 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.

4.2.4.34 IFX_TAPI_RING_CADENCE_t

Description

Structure for ring cadence used in IFX_TAPI_RING_CADENCE_HR_SET.

Prototype

```
typedef struct
{
        IFX_char_t data[TAPI_MAX_CADENCE_BYTES];
        IFX_int32_t nr;
        IFX_char_t initial[TAPI_MAX_CADENCE_BYTES];
        IFX_int32_t initialNr;
} IFX_TAPI_RING_CADENCE_t;
```

Parameters

Data Type	Name	Description
IFX_char_t	data[TAPI_MAX_CADENCE_BYTES]	Pointer to data bytes which contain the encoded cadence sequence. One bit represents ring cadence voltage for 50 ms. A maximum of 40 bytes (320 bits) are allowed.
IFX_int32_t	nr	Number of data bits of cadence sequence. A maximum number of 320 data bits is possible which corresponds to a maximum cadence duration of 16 seconds.
IFX_char_t	initial[TAPI_MAX_CADENCE_BYTES]	Pointer to data bytes which contain the encoded cadence sequence for a first non periodic initial ringing. One bit represents ring cadence voltage for 50ms. A maximum of 40 bytes (320 bits) are allowed.
IFX_int32_t	initialNr	Number of data bits of cadence sequence. A maximum number of 320 data bits is possible which corresponds to a maximum cadence duration of 16 seconds.

4.2.4.35 IFX_TAPI_SIG_DETECTION_t

Description

Structure used for enable and disable signal detection.

Prototype

```
typedef struct
{
        IFX_uint32_t sig;
} 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

4.2.4.36 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

Data Type	Name	Description
IFX_uint8_t	nTraining	Training sequence flag, used by V.17 only, ignored in all other cases. • 0: Short training sequence • 1: Long training sequence
IFX_uint16_t	nDmbsd	Level required before the demodulator sends data.

4.2.4.37 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.

Data Type	Name	Description
IFX_uint8_t	nDbm	Desired output signal power in -dBm.
IFX_uint8_t	nTEP	TEP Generation flag, used by V.27, V.29 and V.17 only, ignored in all other cases. • 0: no TEP generation • 1: TEP generation
IFX_uint8_t	nTraining	Training sequence flag, used by V.17 only, ignored in all other cases. • 0: Short training sequence • 1: Long training sequence
IFX_uint16_t	nMobsm	Level required before the modulation starts.
IFX_uint16_t	nMobrd	Level required before the modulation requests more data.

4.2.4.38 IFX_TAPI_T38_STATUS_t

Description

Structure to read the T.38 fax status and used for IFX_TAPI_T38_STATUS_GET.

Prototype

```
typedef struct
{
        IFX_uint8_t nStatus;
        IFX_uint8_t nError;
} IFX_TAPI_T38_STATUS_t;
```

Parameters

Data Type	Name	Description
IFX_uint8_t	nStatus	T.38 fax status, refer to IFX_TAPI_T38_STATUS_t. 0_D **IFX_TAPI_FAX_T38_DP_OFF**, Data pump is not active 1_D **IFX_TAPI_FAX_T38_DP_ON**, Data pump is active 2_D **IFX_TAPI_FAX_T38_TX_ON**, Transmission is active 3_D **IFX_TAPI_FAX_T38_TX_OFF**, Transmission is finished.
IFX_uint8_t	nError	T.38 fax error, refer to IFX_TAPI_T38_ERROR_t. 0_D **IFX_TAPI_FAX_T38_NO_ERR**, No error occurred 1_D **IFX_TAPI_FAX_T38_ERR**, Fax error occurred, the fax data pump should be deactivated 2_D **IFX_TAPI_FAX_T38_MIPS_OVLD**, MIPS overload 3_D **IFX_TAPI_FAX_T38_READ_ERR**, Error while reading data 4_D **IFX_TAPI_FAX_T38_WRITE_ERR**, Error while writing data 5_D **IFX_TAPI_FAX_T38_DATA_ERR**, Error while setting up the modulator or demodulator

4.2.4.39 IFX_TAPI_TONE_COMPOSED_t

Description

Structure for definition of composed tones.

Prototype

```
typedef struct
{
        IFX_uint8_t format;
        IFX_uint8_t index;
        IFX_uint8_t loop;
        IFX_uint8_t alternateVoice;
        IFX_uint8_t count;
        IFX_uint8_t tones[TAPI_MAX_SIMPLE_TONES];
} IFX_TAPI_TONE_COMPOSED_t;
```

Parameters

Data Type	Name	Description
IFX_uint8_t	format	Indicate the type of the tone descriptor: 1_D **IFX_TAPI_TONE_TYPE_SIMPLE**, the tone descriptor describe a simple tone. 2_D **IFX_TAPI_TONE_TYPE_COMPOSED**, the tone descriptor describes a composed tone.
IFX_uint8_t	index	Tone code ID; 0< ID <255.
IFX_uint8_t	loop	Number of times to play the simple tone sequence, 0 for infinite. Maximum 7.
IFX_uint8_t	alternateVoice	Indicate if the voice path is active between the loops.
IFX_uint8_t	count	0_D **IFX_TAPI_TONE_SIMPLE_MAX**, Number of simple tones used in the composed tone.
IFX_uint8_t	tones[TAPI_MAX_SIMPLE_TONES]	Simple tones to be used. The simple tones are played in the same order as they are stored in the array. *Note: In order to create composed tones only simple tones with a finite loop count can be used.*

4.2.4.40 IFX_TAPI_TONE_CPTD_t

Description

Structure used for IFX_TAPI_TONE_CPTD_START and IFX_TAPI_TONE_CPTD_STOP.

Prototype

```
typedef struct
{
        IFX_int32_t tone;
        IFX_int32_t signal;
} IFX_TAPI_TONE_CPTD_t;
```

Parameters

Data Type	Name	Description
IFX_int32_t	tone	The index of the tone to detect. The tone index must be preconfigured in the tone table, before starting tone detection.
IFX_int32_t	signal	The specification of the signal. 1_H **IFX_TAPI_TONE_CPTD_DIRECTION_RX**, receive direction of the programmed CPT tone 2_H **IFX_TAPI_TONE_CPTD_DIRECTION_TX**, transmit direction of the programmed CPT tone

4.2.4.41 IFX_TAPI_TONE_SIMPLE_t

Description

Struct used to define simple tone characteristics.

Prototype

```
typedef struct
{
        IFX_uint8_t format;
        IFX_uint8_t index;
        IFX_uint8_t loop;
        IFX_int32_t levelA;
        IFX_int32_t levelB;
        IFX_int32_t levelC;
        IFX_int32_t levelD;
        IFX_uint32_t freqA;
        IFX_uint32_t freqB;
        IFX_uint32_t freqC;
        IFX_uint32_t freqD;
        IFX_uint32_t cadence[IFX_TAPI_TONE_STEPS_MAX];
        IFX_uint32_t frequencies[IFX_TAPI_TONE_STEPS_MAX];
        IFX_uint32_t modulation[IFX_TAPI_TONE_STEPS_MAX];
        IFX_uint32_t pause;
} IFX_TAPI_TONE_SIMPLE_t;
```

Parameters

Data Type	Name	Description
IFX_uint8_t	format	Indicate the type of the tone descriptor: 1_D **IFX_TAPI_TONE_TYPE_SIMPLE**, the tone descriptor describe a simple tone. 2_D **IFX_TAPI_TONE_TYPE_COMPOSED**, the tone descriptor describe a composed tone.
IFX_uint8_t	index	Tone code ID; 0< ID <255.
IFX_uint8_t	loop	Number of times to play the simple tone, 0 < loop < 8.
IFX_int32_t	levelA	Power level for frequency A in 0.1 dB steps; -300 < levelA < 0.
IFX_int32_t	levelB	Power level for frequency B in 0.1 dB steps; -300 < levelB < 0.
IFX_int32_t	levelC	Power level for frequency C in 0.1 dB steps; -300 < levelC < 0.
IFX_int32_t	levelD	Power level for frequency D in 0.1 dB steps; -300 < levelD < 0.
IFX_uint32_t	freqA	Tone frequency A in Hz; 0 <= Hz < 4000.
IFX_uint32_t	freqB	Tone frequency B in Hz; 0 <= Hz < 4000.
IFX_uint32_t	freqC	Tone frequency C in Hz; 0 <= Hz < 4000.
IFX_uint32_t	freqD	Tone frequency D in Hz; 0 <= Hz < 4000.

Data Type	Name	Description
IFX_uint32_t	cadence[IFX_TAPI_TONE_STEPS_MAX]	$_D$ **IFX_TAPI_TONE_STEPS_MAX**, Array defining time duration for each cadence steps, with 2 ms granularity. 0 <= cadence <= 32000. The first cadence[X] = 0 (starting from X=1) in the array indicates that X-1 cadences must be played. A tone with cadence[0]=0 can not be processed!
IFX_uint32_t	frequencies[IFX_TAPI_TONE_STEPS_MAX]	Active frequencies for the cadence steps. More than one frequency can be active in the same cadence step. All active frequencies are summed together. 0_H **IFX_TAPI_TONE_FREQNONE**, No frequencies. 1_H **IFX_TAPI_TONE_FREQA**, Frequency A is enabled. 2_H **IFX_TAPI_TONE_FREQB**, Frequency B is enabled. 3_H **IFX_TAPI_TONE_FREQC**, Frequency C is enabled. 4_H **IFX_TAPI_TONE_FREQD**, Frequency D is enabled. F_H **IFX_TAPI_TONE_FREQALL**, All frequencies are enabled.
IFX_uint32_t	modulation[IFX_TAPI_TONE_STEPS_MAX]	Array containing selection for each cadence steps, whether to enable/disable modulation of frequency A with frequency B. Defined values for each array entry: 0_D **IFX_TAPI_TONE_MODULATION_OFF**, Modulation of frequency A with frequency B is disabled. 1_D **IFX_TAPI_TONE_MODULATION_ON**, Modulation of frequency A with frequency B is enabled.
IFX_uint32_t	pause	Some tones require an off time at the end of the tone. The offtime will be added to the last used cadence. Therefore the maximum value for offtime is 32000 - “last used cadence” and have the granularity of 2ms; 0 < 32000 - “last used cadence” < 32000.

4.2.4.42 IFX_TAPI_VERSION_t

Description

Structure used for the TAPI version support check.

Prototype

```
typedef struct
{
        IFX_uint8_t major;
        IFX_uint8_t minor;
} IFX_TAPI_VERSION_t;
```

Parameters

Data Type	Name	Description
IFX_uint8_t	major	Major version number supported
IFX_uint8_t	minor	Minor version number supported

4.2.5 Union Reference

This chapter contains the Union reference.

Table 47 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.

4.2.5.1 IFX_TAPI_CID_MSG_ELEMENT_t

Description

Union of element types

Prototype

```
typedef union
{
        IFX_TAPI_CID_MSG_DATE_t date;
        IFX_TAPI_CID_MSG_STRING_t string;
        IFX_TAPI_CID_MSG_VALUE_t value;
        IFX_TAPI_CID_MSG_TRANSPARENT_t transparent;
} IFX_TAPI_CID_MSG_ELEMENT_t;
```

Parameters

Data Type	Name	Description
IFX_TAPI_CID_MSG_DATE_t	date	Message element including date and time information.
IFX_TAPI_CID_MSG_STRING_t	string	Message element formatted as string.
IFX_TAPI_CID_MSG_VALUE_t	value	Message element formatted as value.
IFX_TAPI_CID_MSG_TRANSPARENT_t	transparent	Message element to be sent with transparent transmission.

4.2.5.2 IFX_TAPI_CID_STD_TYPE_t

Description

Union of the CID configuration structures for different standards.

Prototype

```
typedef union
{
        IFX_TAPI_CID_STD_TELCORDIA_t telcordia;
        IFX_TAPI_CID_STD_ETSI_FSK_t etsiFSK;
        IFX_TAPI_CID_STD_ETSI_DTMF_t etsiDTMF;
        IFX_TAPI_CID_STD_SIN_t sin;
        IFX_TAPI_CID_STD_NTT_t ntt;
} IFX_TAPI_CID_STD_TYPE_t;
```

Parameters

Data Type	Name	Description
IFX_TAPI_CID_STD_TELCORDIA_t	telcordia	Structure defining configuration parameters for Telcordia standard.
IFX_TAPI_CID_STD_ETSI_FSK_t	etsiFSK	Structure defining configuration parameters for ETSI standard, with FSK transmission.
IFX_TAPI_CID_STD_ETSI_DTMF_t	etsiDTMF	Structure defining configuration parameters for ETSI standard, with DTMF transmission.
IFX_TAPI_CID_STD_SIN_t	sin	Structure defining configuration parameters for BT SIN standard.
IFX_TAPI_CID_STD_NTT_t	ntt	Structure defining configuration parameters for NTT standard.

4.2.5.3 IFX_TAPI_EXCEPTION_t

Description

Contains TAPI exception information.

Prototype

```
typedef union
{
        IFX_TAPI_EXCEPTION_BITS_t Bits;
        IFX_uint32_t Status;
} IFX_TAPI_EXCEPTION_t;
```

Parameters

Data Type	Name	Description
IFX_TAPI_EXCEPTION_BITS_t	Bits	Specifies the exception.
IFX_uint32_t	Status	Contains the complete status.

4.2.5.4 IFX_TAPI_TONE_t

Description

This chapter define a union for the tone descriptor.

Prototype

```
typedef union
{
        IFX_TAPI_TONE_SIMPLE_t simple;
        IFX_TAPI_TONE_COMPOSED_t composed;
} IFX_TAPI_TONE_t;
```

Parameters

Data Type	Name	Description
IFX_TAPI_TONE_SIMPLE_t	simple	The parameter points to a IFX_TAPI_TONE_SIMPLE_t structure.
IFX_TAPI_TONE_COMPOSED_t	composed	The parameter points to a IFX_TAPI_TONE_COMPOSED_t structure.

4.2.6 Enumerator Reference

This chapter contains the Enumerator reference.

Table 48 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	ABSCLI/ABSNAME settings.
IFX_TAPI_CID_ALERT_ETSI_t	List of ETSI Alerts.
IFX_TAPI_CID_HOOK_MODE_t	Caller ID transmission modes.
IFX_TAPI_CID_MSG_TYPE_t	Caller ID message types (defined in ETSI EN 300 659-3).
IFX_TAPI_CID_RX_ERROR_t	CID Receiver Errors.
IFX_TAPI_CID_RX_STATE_t	CID Receiver Status.
IFX_TAPI_CID_SERVICE_TYPE_t	Caller ID Services (defined in ETSI EN 300 659-3).
IFX_TAPI_CID_STD_t	List of CID standards.
IFX_TAPI_CID_VMWI_t	List of VMWI settings.
IFX_TAPI_DATA_MAP_START_STOP_t	Start/Stop information for data channel mapping.
IFX_TAPI_DIALING_STATUS_t	Enumeration for dial status events.
IFX_TAPI_ENC_AGC_MODE_t	Specifies the Enable/Disable mode of the AGC resource.
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_LEC_GAIN_t	LEC Gain Levels.
IFX_TAPI_LEC_NLP_t	LEC NLP (Non Linear Processor) Settings.
IFX_TAPI_LEC_TYPE_t	LEC type selection.
IFX_TAPI_LINE_FEED_t	Defines for linefeeding.
IFX_TAPI_LINE_HOOK_STATUS_t	Enumeration for hook status events.
IFX_TAPI_LINE_HOOK_VALIDATION_TYPE_t	Validation types used for structure IFX_TAPI_LINE_HOOK_VT_t.
IFX_TAPI_LINE_LEVEL_t	Specifies the Enable/Disable mode of the high level.
IFX_TAPI_LINE_STATUS_t	Enumeration for phone line status information.

Table 48 Enumerator Overview of TAPI Interfaces (cont'd)

Name	Description
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_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_SIG_t	List the tone detection options.
IFX_TAPI_T38_ERROR_t	T38 Fax errors.
IFX_TAPI_T38_STATUS_t	T38 Fax Datapump states.
IFX_TAPI_T38_STD_t	T38 Fax standards.
IFX_TAPI_TONE_CPTD_DIRECTION_t	Specifies the CPT signal for CPT detection.
IFX_TAPI_TONE_FREQ_t	Frequency setting for a cadence step.
IFX_TAPI_TONE_GROUP_t	Tone grouping.
IFX_TAPI_TONE_MODULATION_t	Modulation setting for a cadence step.
IFX_TAPI_TONE_TG_t	Defines the tone generator usage.
IFX_TAPI_TONE_TYPE_t	Tone types.

4.2.6.1 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

4.2.6.2 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.

4.2.6.3 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

4.2.6.4 IFX_TAPI_CH_INIT_COUNTRY_t

Description

Country selection for **IFX_TAPI_CH_INIT_t**.

For future purposes, not yet used. If different countries are supported by the implementation, this parameter specifies which one.

Prototype

```
typedef enum
{
        IFX_TAPI_CH_INIT_COUNTRY_DEFAULT = 0,
        IFX_TAPI_CH_INIT_COUNTRY_DE = 1,
        IFX_TAPI_CH_INIT_COUNTRY_US = 2,
        IFX_TAPI_CH_INIT_COUNTRY_UK = 3
} IFX_TAPI_CH_INIT_COUNTRY_t;
```

Parameters

Name	Value	Description
IFX_TAPI_CH_INIT_COUNTRY_DEFAULT	0_D	Default country.
IFX_TAPI_CH_INIT_COUNTRY_DE	1_D	Germany.
IFX_TAPI_CH_INIT_COUNTRY_US	2_D	USA.
IFX_TAPI_CH_INIT_COUNTRY_UK	3_D	United Kingdom.

4.2.6.5 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
IFX_TAPI_INIT_MODE_PCM_DSP	2_D	Phone to PCM using DSP features for signal detection.
IFX_TAPI_INIT_MODE_PCM_PHONE	3_D	Phone to PCM not using DSP features for signal detection.
IFX_TAPI_INIT_MODE_NONE	FF_H	Phone to PCM connection without DSP features.

4.2.6.6 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.

4.2.6.7 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. *Note: Defined only CID transmission associated with ringing.*
IFX_TAPI_CID_ALERT_ETSI_DTAS	1_H	DTAS.
IFX_TAPI_CID_ALERT_ETSI_RP	2_H	Ring Pulse.
IFX_TAPI_CID_ALERT_ETSI_LRDTAS	3_H	Line Reversal (alias Polarity Reversal) followed by DTAS.

4.2.6.8 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.

4.2.6.9 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 trasmission.
IFX_TAPI_CID_MT_CSUP	80_H	Call Set-up. Corresponds to Caller ID type 1 and type 2.
IFX_TAPI_CID_MT_MWI	82_H	Message Waiting Indicator
IFX_TAPI_CID_MT_AOC	86_H	Advice of Charge
IFX_TAPI_CID_MT_SMS	89_H	Short Message Service
IFX_TAPI_CID_MT_RES01	$F1_H$	Reserved for Network Operator Use
IFX_TAPI_CID_MT_RES02	$F2_H$	Reserved for Network Operator Use
IFX_TAPI_CID_MT_RES03	$F3_H$	Reserved for Network Operator Use
IFX_TAPI_CID_MT_RES04	$F4_H$	Reserved for Network Operator Use
IFX_TAPI_CID_MT_RES05	$F5_H$	Reserved for Network Operator Use
IFX_TAPI_CID_MT_RES06	$F6_H$	Reserved for Network Operator Use
IFX_TAPI_CID_MT_RES07	$F7_H$	Reserved for Network Operator Use
IFX_TAPI_CID_MT_RES08	$F8_H$	Reserved for Network Operator Use

Name	Value	Description
IFX_TAPI_CID_MT_RES09	$F9_H$	Reserved for Network Operator Use
IFX_TAPI_CID_MT_RES0A	FA_H	Reserved for Network Operator Use
IFX_TAPI_CID_MT_RES0B	FB_H	Reserved for Network Operator Use
IFX_TAPI_CID_MT_RES0C	FC_H	Reserved for Network Operator Use
IFX_TAPI_CID_MT_RES0D	FD_H	Reserved for Network Operator Use
IFX_TAPI_CID_MT_RES0E	FE_H	Reserved for Network Operator Use
IFX_TAPI_CID_MT_RES0F	FF_H	Reserved for Network Operator Use

Remarks

Implemented only **IFX_TAPI_CID_MT_TRANSPARENT** (for Caller ID type 1 and type 2) and **IFX_TAPI_CID_MT_MWI** (for Message Waiting).

4.2.6.10 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.

4.2.6.11 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.

4.2.6.12 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
IFX_TAPI_CID_ST_DURATION	23_H	Duration of the call
IFX_TAPI_CID_ST_NTID	30_H	Network provider id
IFX_TAPI_CID_ST_CARID	31_H	Carrier identity
IFX_TAPI_CID_ST_TERMSEL	40_H	Selection of terminal function
IFX_TAPI_CID_ST_DISP	50_H	Display information, used as INFO for DTMF
IFX_TAPI_CID_ST_SINFO	55_H	Reserved
IFX_TAPI_CID_ST_XOPUSE	$E0_H$	Extension for operator use
IFX_TAPI_CID_ST_TRANSPARENT	FF_H	Transparent mode

4.2.6.13 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.

4.2.6.14 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

4.2.6.15 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.

4.2.6.16 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.

4.2.6.17 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

4.2.6.18 IFX_TAPI_ENC_AGC_MODE_t

Description

Specifies the Enable/Disable mode of the AGC resource.

Prototype

```
typedef enum
{
        IFX_TAPI_ENC_AGC_MODE_DISABLE = 0x0,
        IFX_TAPI_ENC_AGC_MODE_ENABLE = 0x1
} IFX_TAPI_ENC_AGC_MODE_t;
```

Parameters

Name	Value	Description
IFX_TAPI_ENC_AGC_MODE_DISABLE	0_H	Disable AGC
IFX_TAPI_ENC_AGC_MODE_ENABLE	1_H	Enable AGC

4.2.6.19 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.

4.2.6.20 IFX_TAPI_ENC_VAD_t

Description

Enumeration used for ioctl **IFX_TAPI_ENC_VAD_CFG_SET**.

Prototype

```
typedef enum
{
        IFX_TAPI_ENC_VAD_NOVAD = 0,
        IFX_TAPI_ENC_VAD_ON = 1,
        IFX_TAPI_ENC_VAD_G711 = 2
} IFX_TAPI_ENC_VAD_t;
```

Parameters

Name	Value	Description
IFX_TAPI_ENC_VAD_NOVAD	0_D	No voice activity detection.
IFX_TAPI_ENC_VAD_ON	1_D	Voice activity detection on, in this case also comfort noise and spectral information (nicer noise) is switched on.
IFX_TAPI_ENC_VAD_G711	2_D	Voice activity detection on with comfort noise generation without spectral information.

4.2.6.21 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.

4.2.6.22 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.

4.2.6.23 IFX_TAPI_JB_PKT_ADAPT_t

Description

Jitter buffer packet adaptation.

Prototype

```
typedef enum
{
        IFX_TAPI_JB_PKT_ADAPT_RES1 = 0,
        IFX_TAPI_JB_PKT_ADAPT_RES2 = 1,
        IFX_TAPI_JB_PKT_ADAPT_VOICE = 2,
        IFX_TAPI_JB_PKT_ADAPT_DATA = 3
} IFX_TAPI_JB_PKT_ADAPT_t;
```

Parameters

Name	Value	Description
IFX_TAPI_JB_PKT_ADAPT_RES1	0_D	Reserved.
IFX_TAPI_JB_PKT_ADAPT_RES2	1_D	Reserved.
IFX_TAPI_JB_PKT_ADAPT_VOICE	2_D	JB adapted for voice, reduced adjustment speed and packet repetition is off.
IFX_TAPI_JB_PKT_ADAPT_DATA	3_D	JB adapted for data (fax/modem). Reduced adjustment speed and packet repetition is on

4.2.6.24 IFX_TAPI_LEC_GAIN_t

Description

LEC gain levels.

Prototype

```
typedef enum
{
        IFX_TAPI_LEC_GAIN_OFF = 0,
        IFX_TAPI_LEC_GAIN_LOW = 1,
        IFX_TAPI_LEC_GAIN_MEDIUM = 2,
        IFX_TAPI_LEC_GAIN_HIGH = 3
} IFX_TAPI_LEC_GAIN_t;
```

Parameters

Name	Value	Description
IFX_TAPI_LEC_GAIN_OFF	0_D	Turn LEC off.
IFX_TAPI_LEC_GAIN_LOW	1_D	Turn LEC on to low-level.
IFX_TAPI_LEC_GAIN_MEDIUM	2_D	Turn LEC on to normal level.
IFX_TAPI_LEC_GAIN_HIGH	3_D	Turn LEC on to high level.

4.2.6.25 IFX_TAPI_LEC_NLP_t

Description

LEC NLP (Non Linear Processor) settings.

Prototype

```
typedef enum
{
        IFX_TAPI_LEC_NLP_DEFAULT = 0,
        IFX_TAPI_LEC_NLP_ON = 1,
        IFX_TAPI_LEC_NLP_OFF = 2
} IFX_TAPI_LEC_NLP_t;
```

Parameters

Name	Value	Description
IFX_TAPI_LEC_NLP_DEFAULT	0_D	Default NLP on.
IFX_TAPI_LEC_NLP_ON	1_D	NLP on.
IFX_TAPI_LEC_NLP_OFF	2_D	NLP off.

4.2.6.26 IFX_TAPI_LEC_TYPE_t

Description

LEC type selection.

Prototype

```
typedef enum
{
        IFX_TAPI_LEC_TYPE_OFF = 0,
        IFX_TAPI_LEC_NLEC = 1,
        IFX_TAPI_LEC_WLEC = 2
} IFX_TAPI_LEC_TYPE_t;
```

Parameters

Name	Value	Description
IFX_TAPI_LEC_TYPE_OFF	0_D	LEC OFF.
IFX_TAPI_LEC_NLEC	1_D	Near-end LEC.
IFX_TAPI_LEC_WLEC	2_D	Window based LEC (near-end and far-end at the same time).

4.2.6.27 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_RING_BURST = 8,
        IFX_TAPI_LINE_FEED_RING_PAUSE = 9,
        IFX_TAPI_LINE_FEED_METER = 10
} IFX_TAPI_LINE_FEED_t;
```

Parameters

Name	Value	Description
IFX_TAPI_LINE_FEED_ACTIVE	0_D	Feeding mode for phone used for on-hook or off-hook transmission, with normal polarity.
IFX_TAPI_LINE_FEED_ACTIVE_REV	1_D	Feeding mode for phone used for on-hook or off-hook transmission, with reversed polarity.
IFX_TAPI_LINE_FEED_STANDBY	2_D	Feeding mode to be used if the phone is in standby (on-hook without any transmission active). Off-hook detection is possible in this mode.
IFX_TAPI_LINE_FEED_HIGH_IMPEDANCE	3_D	Switch off the line, the device is only able to test the line.
IFX_TAPI_LINE_FEED_DISABLED	4_D	Switch off the line, no operations possible on the line.
IFX_TAPI_LINE_FEED_GROUND_START	5_D	Use ground start, also known as open tip.
IFX_TAPI_LINE_FEED_RING_BURST	8_D	Reserved, needed for ring internal function.
IFX_TAPI_LINE_FEED_RING_PAUSE	9_D	Reserved, needed for ring internal function.
IFX_TAPI_LINE_FEED_METER	10_D	Reserved, needed for internal function.

4.2.6.28 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

4.2.6.29 IFX_TAPI_LINE_HOOK_STATUS_t

Description

Enumeration for hook status events.

Prototype

```
typedef enum
{
        IFX_TAPI_LINE_HOOK_STATUS_HOOK = 0x01,
        IFX_TAPI_LINE_HOOK_STATUS_FLASH = 0x02,
        IFX_TAPI_LINE_HOOK_STATUS_OFFHOOK = 0x04
} IFX_TAPI_LINE_HOOK_STATUS_t;
```

Parameters

Name	Value	Description
IFX_TAPI_LINE_HOOK_STATUS_HOOK	01_H	Hook detected.
IFX_TAPI_LINE_HOOK_STATUS_FLASH	02_H	Hook flash detected.
IFX_TAPI_LINE_HOOK_STATUS_OFFHOOK	04_H	Detected hook event is an off-hook.

4.2.6.30 IFX_TAPI_LINE_LEVEL_t

Description

Specifies the whether the “high level” line output mode should be used. Enabling the “high level” mode, it will be possible to have a extremly high output signal level, above +3 dBm. This mode is required for some special howler tones.

Attention: The “high level” mode allow signal levels potentially dangerous for the human ear!

Prototype

```
typedef enum
{
        IFX_TAPI_LINE_LEVEL_NORMAL = 0x0,
        IFX_TAPI_LINE_LEVEL_HIGH = 0x1
} IFX_TAPI_LINE_LEVEL_t;
```

Parameters

Name	Value	Description
IFX_TAPI_LINE_LEVEL_NORMAL	0_H	Line output level for typical operations.
IFX_TAPI_LINE_LEVEL_HIGH	1_H	High level line, suitable for playing howler tones requiring output levels above 3 dBm. This is not suitable for voice calls or other call progress tone detection.

4.2.6.31 IFX_TAPI_LINE_STATUS_t

Description

Enumeration for phone line status information.

Prototype

```
typedef enum
{
        IFX_TAPI_LINE_STATUS_RINGING = 0x1,
        IFX_TAPI_LINE_STATUS_RINGFINISHED = 0x02,
        IFX_TAPI_LINE_STATUS_FAX = 0x04,
        IFX_TAPI_LINE_STATUS_GNDKEY = 0x10,
        IFX_TAPI_LINE_STATUS_GNDKEYHIGH = 0x20,
        IFX_TAPI_LINE_STATUS_OTEMP = 0x40,
        IFX_TAPI_LINE_STATUS_GNDKEYPOL = 0x80,
        IFX_TAPI_LINE_STATUS_GR909RES = 0x100,
```

```
        IFX_TAPI_LINE_STATUS_CIDRX = 0x200
} IFX_TAPI_LINE_STATUS_t;
```

Parameters

Name	Value	Description
IFX_TAPI_LINE_STATUS_RINGING	1_H	Line is ringing
IFX_TAPI_LINE_STATUS_RINGFINISHED	02_H	Ringing finished
IFX_TAPI_LINE_STATUS_FAX	04_H	Fax detected -> is replaced by signal
IFX_TAPI_LINE_STATUS_GNDKEY	10_H	Ground key detected
IFX_TAPI_LINE_STATUS_GNDKEYHIGH	20_H	Ground key high detected
IFX_TAPI_LINE_STATUS_OTEMP	40_H	Over temperature detected
IFX_TAPI_LINE_STATUS_GNDKEYPOL	80_H	Ground key polarity detected
IFX_TAPI_LINE_STATUS_GR909RES	100_H	
IFX_TAPI_LINE_STATUS_CIDRX	200_H	Caller id received

4.2.6.32 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

4.2.6.33 IFX_TAPI_MAP_DEC_t

Description

Play out enabling and disabling information.

Prototype

```
typedef enum
{
```

```
        IFX_TAPI_MAP_DEC_NONE = 0,
        IFX_TAPI_MAP_DEC_START = 1,
        IFX_TAPI_MAP_DEC_STOP = 2
} IFX_TAPI_MAP_DEC_t;
```

Parameters

Name	Value	Description
IFX_TAPI_MAP_DEC_NONE	0_D	Do not modify playing status.
IFX_TAPI_MAP_DEC_START	1_D	Start playing after mapping.
IFX_TAPI_MAP_DEC_STOP	2_D	Stop playing after mapping.

4.2.6.34 IFX_TAPI_MAP_ENC_t

Description

Recording enabling and disabling information.

Prototype

```
typedef enum
{
        IFX_TAPI_MAP_ENC_NONE = 0,
        IFX_TAPI_MAP_ENC_START = 1,
        IFX_TAPI_MAP_ENC_STOP = 2
} IFX_TAPI_MAP_ENC_t;
```

Parameters

Name	Value	Description
IFX_TAPI_MAP_ENC_NONE	0_D	Do not modify recording status.
IFX_TAPI_MAP_ENC_START	1_D	Start recording after mapping.
IFX_TAPI_MAP_ENC_STOP	2_D	Stop recording after mapping.

4.2.6.35 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_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 packet channel.
IFX_TAPI_MAP_TYPE_PCM	2_D	Type is a PCM channel.

4.2.6.36 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.
IFX_TAPI_PCM_RES_MLAW_8BIT	1_D	µ-law 8-bit
IFX_TAPI_PCM_RES_LINEAR_16BIT	2_D	Linear 16-bit.

4.2.6.37 IFX_TAPI_PKT_EV_NUM_t

Description

Out of band or in band definition.

Prototype

```
typedef enum
{
        IFX_TAPI_PKT_EV_NUM_DTMF_0 = 0,
        IFX_TAPI_PKT_EV_NUM_DTMF_1 = 1,
        IFX_TAPI_PKT_EV_NUM_DTMF_2 = 2,
        IFX_TAPI_PKT_EV_NUM_DTMF_3 = 3,
        IFX_TAPI_PKT_EV_NUM_DTMF_4 = 4,
        IFX_TAPI_PKT_EV_NUM_DTMF_5 = 5,
        IFX_TAPI_PKT_EV_NUM_DTMF_6 = 6,
        IFX_TAPI_PKT_EV_NUM_DTMF_7 = 7,
        IFX_TAPI_PKT_EV_NUM_DTMF_8 = 8,
        IFX_TAPI_PKT_EV_NUM_DTMF_9 = 9,
```

```
        IFX_TAPI_PKT_EV_NUM_DTMF_STAR = 10,
        IFX_TAPI_PKT_EV_NUM_DTMF_HASH = 11,
        IFX_TAPI_PKT_EV_NUM_ANS = 32,
        IFX_TAPI_PKT_EV_NUM_NANS = 33,
        IFX_TAPI_PKT_EV_NUM_ANSAM = 34,
        IFX_TAPI_PKT_EV_NUM_NANSAM = 35,
        IFX_TAPI_PKT_EV_NUM_CNG = 36,
        IFX_TAPI_PKT_EV_NUM_DIS = 54,
        IFX_TAPI_PKT_EV_NUM_NO_EVENT = 0xFFFFFFFF
} IFX_TAPI_PKT_EV_NUM_t;
```

Parameters

Name	Value	Description
IFX_TAPI_PKT_EV_NUM_DTMF_0	0_D	RFC2833 Event number for DTMF tone 0.
IFX_TAPI_PKT_EV_NUM_DTMF_1	1_D	RFC2833 Event number for DTMF tone 1.
IFX_TAPI_PKT_EV_NUM_DTMF_2	2_D	RFC2833 Event number for DTMF tone 2.
IFX_TAPI_PKT_EV_NUM_DTMF_3	3_D	RFC2833 Event number for DTMF tone 3.
IFX_TAPI_PKT_EV_NUM_DTMF_4	4_D	RFC2833 Event number for DTMF tone 4.
IFX_TAPI_PKT_EV_NUM_DTMF_5	5_D	RFC2833 Event number for DTMF tone 5.
IFX_TAPI_PKT_EV_NUM_DTMF_6	6_D	RFC2833 Event number for DTMF tone 6.
IFX_TAPI_PKT_EV_NUM_DTMF_7	7_D	RFC2833 Event number for DTMF tone 7.
IFX_TAPI_PKT_EV_NUM_DTMF_8	8_D	RFC2833 Event number for DTMF tone 8.
IFX_TAPI_PKT_EV_NUM_DTMF_9	9_D	RFC2833 Event number for DTMF tone 9.
IFX_TAPI_PKT_EV_NUM_DTMF_STAR	10_D	RFC2833 Event number for DTMF tone *.
IFX_TAPI_PKT_EV_NUM_DTMF_HASH	11_D	RFC2833 Event number for DTMF tone #.
IFX_TAPI_PKT_EV_NUM_ANS	32_D	RFC2833 Event number for ANS tone.
IFX_TAPI_PKT_EV_NUM_NANS	33_D	RFC2833 Event number for /ANS tone.
IFX_TAPI_PKT_EV_NUM_ANSAM	34_D	RFC2833 Event number for ANSam tone.
IFX_TAPI_PKT_EV_NUM_NANSAM	35_D	RFC2833 Event number for /ANSam tone.
IFX_TAPI_PKT_EV_NUM_CNG	36_D	RFC2833 Event number for CNG tone.
IFX_TAPI_PKT_EV_NUM_DIS	54_D	RFC2833 Event number for DIS signal.
IFX_TAPI_PKT_EV_NUM_NO_EVENT	$FFFFFFFF_H$	No support RFC event received or no event received.

4.2.6.38 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.

4.2.6.39 IFX_TAPI_RUNTIME_ERROR_t

Description

TAPI Runtime Errors.

Prototype

```
typedef enum
{
        IFX_TAPI_RT_ERROR_NONE = 0,
        IFX_TAPI_RT_ERROR_RINGCADENCE_CIDTX = 1,
        IFX_TAPI_RT_ERROR_CIDTX_NOACK = 2
} IFX_TAPI_RUNTIME_ERROR_t;
```

Parameters

Name	Value	Description
IFX_TAPI_RT_ERROR_NONE	0_D	No error.
IFX_TAPI_RT_ERROR_RINGCADENCE_CIDTX	1_D	Ring cadence settings error in cid tx.
IFX_TAPI_RT_ERROR_CIDTX_NOACK	2_D	No acknowledge during CID sequence.

4.2.6.40 IFX_TAPI_SIG_t

Description

List the tone detection options.

Some application maybe not interested whether the signal came from the receive or transmit path. Therefore for each signal a mask exists, that includes receive and transmit path.

Prototype

```
typedef enum
{
        IFX_TAPI_SIG_NONE = 0x0,
        IFX_TAPI_SIG_DISRX = 0x1,
        IFX_TAPI_SIG_DISTX = 0x2,
        IFX_TAPI_SIG_DIS = 0x4,
```

```
        IFX_TAPI_SIG_CEDRX = 0x8,
        IFX_TAPI_SIG_CEDTX = 0x10,
        IFX_TAPI_SIG_CED = 0x20,
        IFX_TAPI_SIG_CNGFAXRX = 0x40,
        IFX_TAPI_SIG_CNGFAXTX = 0x80,
        IFX_TAPI_SIG_CNGFAX = 0x100,
        IFX_TAPI_SIG_CNGMODRX = 0x200,
        IFX_TAPI_SIG_CNGMODTX = 0x400,
        IFX_TAPI_SIG_CNGMOD = 0x800,
        IFX_TAPI_SIG_PHASEREVRX = 0x1000,
        IFX_TAPI_SIG_PHASEREVTX = 0x2000,
        IFX_TAPI_SIG_PHASEREV = 0x4000,
        IFX_TAPI_SIG_AMRX = 0x8000,
        IFX_TAPI_SIG_AMTX = 0x10000,
        IFX_TAPI_SIG_AM = 0x20000,
        IFX_TAPI_SIG_TONEHOLDING_ENDRX = 0x40000,
        IFX_TAPI_SIG_TONEHOLDING_ENDTX = 0x80000,
        IFX_TAPI_SIG_TONEHOLDING_END = 0x100000,
        IFX_TAPI_SIG_CEDENDRX = 0x200000,
        IFX_TAPI_SIG_CEDENDTX = 0x400000,
        IFX_TAPI_SIG_CEDEND = 0x800000,
        IFX_TAPI_SIG_CPTD = 0x1000000,
        IFX_TAPI_SIG_V8BISRX = 0x2000000,
        IFX_TAPI_SIG_V8BISTX = 0x4000000
} IFX_TAPI_SIG_t;
```

Parameters

Name	Value	Description
IFX_TAPI_SIG_NONE	0_H	No signal detected
IFX_TAPI_SIG_DISRX	1_H	V.21 Preamble Fax Tone, Digital identification signal (DIS), receive path.
IFX_TAPI_SIG_DISTX	2_H	V.21 Preamble Fax Tone, Digital identification signal (DIS), transmit path.
IFX_TAPI_SIG_DIS	4_H	V.21 Preamble Fax Tone in all path, Digital identification signal (DIS).
IFX_TAPI_SIG_CEDRX	8_H	V.25 2100 Hz (CED) Modem/Fax Tone, receive path.
IFX_TAPI_SIG_CEDTX	10_H	V.25 2100 Hz (CED) Modem/Fax Tone, transmit path.
IFX_TAPI_SIG_CED	20_H	V.25 2100 Hz (CED) Modem/Fax Tone in all paths.
IFX_TAPI_SIG_CNGFAXRX	40_H	CNG Fax Calling Tone (1100 Hz) receive path.
IFX_TAPI_SIG_CNGFAXTX	80_H	CNG Fax Calling Tone (1100 Hz) transmit path.
IFX_TAPI_SIG_CNGFAX	100_H	CNG Fax Calling Tone (1100 Hz) in all paths.
IFX_TAPI_SIG_CNGMODRX	200_H	CNG Modem Calling Tone (1300 Hz) receive path.

Name	Value	Description
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.

4.2.6.41 IFX_TAPI_T38_ERROR_t

Description

T.38 Fax errors.

Prototype

```
typedef enum
{
        IFX_TAPI_T38_NO_ERR = 0,
        IFX_TAPI_T38_ERR = 1,
        IFX_TAPI_T38_MIPS_OVLD = 2,
        IFX_TAPI_T38_READ_ERR = 3,
        IFX_TAPI_T38_WRITE_ERR = 4,
        IFX_TAPI_T38_DATA_ERR = 5
} IFX_TAPI_T38_ERROR_t;
```

Parameters

Name	Value	Description
IFX_TAPI_T38_NO_ERR	0_D	No Error.
IFX_TAPI_T38_ERR	1_D	Error occurred: Deactivate Datapump.
IFX_TAPI_T38_MIPS_OVLD	2_D	MIPS Overload.
IFX_TAPI_T38_READ_ERR	3_D	Error while reading data.
IFX_TAPI_T38_WRITE_ERR	4_D	Error while writing data.
IFX_TAPI_T38_DATA_ERR	5_D	Error while setting up modulator or demodulator.

4.2.6.42 IFX_TAPI_T38_STATUS_t

Description

T.38 Fax Datapump states.

Prototype

```
typedef enum
{
        IFX_TAPI_T38_DP_OFF = 0x1,
        IFX_TAPI_T38_DP_ON = 0x2,
        IFX_TAPI_T38_TX_ON = 0x4,
        IFX_TAPI_T38_TX_OFF = 0x8
} IFX_TAPI_T38_STATUS_t;
```

Parameters

Name	Value	Description
IFX_TAPI_T38_DP_OFF	1_H	Fax Datapump not active.
IFX_TAPI_T38_DP_ON	2_H	Fax Datapump active.
IFX_TAPI_T38_TX_ON	4_H	Fax transmission is active.
IFX_TAPI_T38_TX_OFF	8_H	Fax transmission is not active.

4.2.6.43 IFX_TAPI_T38_STD_t

Description

T.38 Fax standards.

Prototype

```
typedef enum
{
        IFX_TAPI_T38_STD_V21_STD = 0x1,
        IFX_TAPI_T38_STD_V27_2400_STD = 0x2,
        IFX_TAPI_T38_STD_V27_4800_STD = 0x3,
        IFX_TAPI_T38_STD_V29_7200_STD = 0x4,
        IFX_TAPI_T38_STD_V29_9600_STD = 0x5,
        IFX_TAPI_T38_STD_V17_7200_STD = 0x6,
```

```
        IFX_TAPI_T38_STD_V17_9600_STD = 0x7,
        IFX_TAPI_T38_STD_V17_12000_STD = 0x8,
        IFX_TAPI_T38_STD_V17_14400_STD = 0x9
} IFX_TAPI_T38_STD_t;
```

Parameters

Name	Value	Description
IFX_TAPI_T38_STD_V21_STD	1_H	V.21.
IFX_TAPI_T38_STD_V27_2400_STD	2_H	V.27/2400.
IFX_TAPI_T38_STD_V27_4800_STD	3_H	V.27/4800.
IFX_TAPI_T38_STD_V29_7200_STD	4_H	V.29/7200.
IFX_TAPI_T38_STD_V29_9600_STD	5_H	V.29/9600.
IFX_TAPI_T38_STD_V17_7200_STD	6_H	V.17/7200.
IFX_TAPI_T38_STD_V17_9600_STD	7_H	V.17/9600.
IFX_TAPI_T38_STD_V17_12000_STD	8_H	V.17/12000.
IFX_TAPI_T38_STD_V17_14400_STD	9_H	V.17/14400.

4.2.6.44 IFX_TAPI_TONE_CPTD_DIRECTION_t

Description

Specifies the CPT signal for CPT detection.

Prototype

```
typedef enum
{
        IFX_TAPI_TONE_CPTD_DIRECTION_RX = 0x1,
        IFX_TAPI_TONE_CPTD_DIRECTION_TX = 0x2
} IFX_TAPI_TONE_CPTD_DIRECTION_t;
```

Parameters

Name	Value	Description
IFX_TAPI_TONE_CPTD_DIRECTION_RX	1_H	
IFX_TAPI_TONE_CPTD_DIRECTION_TX	2_H	

4.2.6.45 IFX_TAPI_TONE_FREQ_t

Description

Used for selection of one or more frequencies belonging to a tone cadence.

Prototype

```
typedef enum
{
        IFX_TAPI_TONE_FREQNONE = 0x0,
        IFX_TAPI_TONE_FREQA = 0x1,
        IFX_TAPI_TONE_FREQB = 0x2,
```

```
        IFX_TAPI_TONE_FREQC = 0x4,
        IFX_TAPI_TONE_FREQD = 0x8,
        IFX_TAPI_TONE_FREQALL = 0xF
} IFX_TAPI_TONE_FREQ_t;
```

Parameters

Name	Value	Description
IFX_TAPI_TONE_FREQNONE	0_H	All frequencies are inactive
IFX_TAPI_TONE_FREQA	1_H	Play frequency A
IFX_TAPI_TONE_FREQB	2_H	Play frequency B
IFX_TAPI_TONE_FREQC	4_H	Play frequency C
IFX_TAPI_TONE_FREQD	8_H	Play frequency D
IFX_TAPI_TONE_FREQALL	F_H	Play all frequencies

4.2.6.46 IFX_TAPI_TONE_GROUP_t

Description

Tone grouping.

Prototype

```
typedef enum
{
        IFX_TAPI_TONE_GROUP_NONE = 0,
        IFX_TAPI_TONE_GROUP_AB = 1,
        IFX_TAPI_TONE_GROUP_BC = 2,
        IFX_TAPI_TONE_GROUP_AB_BC = 3
} IFX_TAPI_TONE_GROUP_t;
```

Parameters

Name	Value	Description
IFX_TAPI_TONE_GROUP_NONE	0_D	All tones are sent as single tones
IFX_TAPI_TONE_GROUP_AB	1_D	The frequencies of A and B are sent as dual tone followed by frequency C sent as single tone.
IFX_TAPI_TONE_GROUP_BC	2_D	For group two the frequency A is sent as a single tone followed by frequencies B and C as dual tone.
IFX_TAPI_TONE_GROUP_AB_BC	3_D	The frequencies of A and B are sent as dual tone followed by frequency B and C as dual tone, not yet supported.

4.2.6.47 IFX_TAPI_TONE_MODULATION_t

Description

Modulation setting for a cadence step.

Prototype

```
typedef enum
{
        IFX_TAPI_TONE_MODULATION_OFF = 0,
        IFX_TAPI_TONE_MODULATION_ON = 1
} IFX_TAPI_TONE_MODULATION_t;
```

Parameters

Name	Value	Description
IFX_TAPI_TONE_MODULATION_OFF	0_D	Modulation off for the cadence step
IFX_TAPI_TONE_MODULATION_ON	1_D	Modulation on for the cadence step

4.2.6.48 IFX_TAPI_TONE_TG_t

Description

Defines the tone generator usage.

Prototype

```
typedef enum
{
        IFX_TAPI_TONE_TG1 = 1,
        IFX_TAPI_TONE_TG2 = 2,
        IFX_TAPI_TONE_TGALL = 0xFF
} IFX_TAPI_TONE_TG_t;
```

Parameters

Name	Value	Description
IFX_TAPI_TONE_TG1	1_D	Use tone generator 1.
IFX_TAPI_TONE_TG2	2_D	Use tone generator 2.
IFX_TAPI_TONE_TGALL	FF_H	Use all tone generators.

4.2.6.49 IFX_TAPI_TONE_TYPE_t

Description

Tone types.

Prototype

```
typedef enum
{
        IFX_TAPI_TONE_TYPE_SIMPLE = 1,
        IFX_TAPI_TONE_TYPE_COMPOSED = 2
} IFX_TAPI_TONE_TYPE_t;
```

Parameters

Name	Value	Description
IFX_TAPI_TONE_TYPE_SIMPLE	1_D	Simple tone
IFX_TAPI_TONE_TYPE_COMPOSED	2_D	Composed tone

4.3 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**.

4.3.1 Mapping of ioctl Interfaces

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

Table 49 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		
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.

Table 49 Mapping to old names of TAPI ioctl Interfaces (cont'd)

Old Name	New Name	Remarks
PHONE_SET_TONE_ON_TIME		Not supported anymore.
Dial Service		
PHONE_DTMF_OOB	**IFX_TAPI_PKT_EV_OOB_SET**	
IFXPHONE_GET_PULSE_ASCII	**IFX_TAPI_PULSE_ASCII_GET**	
IFXPHONE_GET_PULSE	**IFX_TAPI_PULSE_GET**	
IFXPHONE_PULSE_READY	**IFX_TAPI_PULSE_READY**	
PHONE_GET_DTMF_ASCII	**IFX_TAPI_TONE_DTMF_ASCII_GET**	
PHONE_GET_DTMF	**IFX_TAPI_TONE_DTMF_GET**	
PHONE_DTMF_READY	**IFX_TAPI_TONE_DTMF_READY_GET**	
Signal Detection Service		
IFXPHONE_DISABLE_SIGDETECT	**IFX_TAPI_SIG_DETECT_DISABLE**	
IFXPHONE_ENABLE_SIGDETECT	**IFX_TAPI_SIG_DETECT_ENABLE**	
IFXPHONE_CPTD_START	**IFX_TAPI_TONE_CPTD_START**	
IFXPHONE_CPTD_STOP	**IFX_TAPI_TONE_CPTD_STOP**	
CID Features		
IFXPHONE_CIDCONF	**IFX_TAPI_CID_CFG_SET**	Important functional change.
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_CID_INFO_TX	**IFX_TAPI_CID_TX_INFO_START**	
IFXPHONE_CID_SEQ_TX	**IFX_TAPI_CID_TX_SEQ_START**	
IFXPHONE_CID2_CONF		Not supported anymore.
IFXPHONE_CID2_GETCONF		Not supported anymore.
IFXPHONE_CID2_SEND		Not supported anymore.
IFXPHONE_DTMFCID_CONF		Not supported anymore.
IFXPHONE_GETCIDCONF		Not supported anymore.
Connection Control Service		
PHONE_PLAY_LEVEL	**IFX_TAPI_DEC_LEVEL_GET**	
PHONE_PLAY_START	**IFX_TAPI_DEC_START**	
PHONE_PLAY_STOP	**IFX_TAPI_DEC_STOP**	
PHONE_PLAY_CODEC	**IFX_TAPI_DEC_TYPE_SET**	
PHONE_PLAY_VOLUME	**IFX_TAPI_DEC_VOLUME_SET**	
IFXPHONE_GET_FRAME	**IFX_TAPI_ENC_FRAME_LEN_GET**	
PHONE_FRAME	**IFX_TAPI_ENC_FRAME_LEN_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_CODEC	**IFX_TAPI_ENC_TYPE_SET**	
PHONE_VAD	**IFX_TAPI_ENC_VAD_CFG_SET**	
PHONE_REC_VOLUME	**IFX_TAPI_ENC_VOLUME_SET**	

Table 49 Mapping to old names of TAPI ioctl Interfaces (cont'd)

Old Name	New Name	Remarks
IFXPHONE_JB_CONF	IFX_TAPI_JB_CFG_SET	
IFXPHONE_JB_STATISTICS	IFX_TAPI_JB_STATISTICS_GET	
IFXPHONE_JB_RESET	IFX_TAPI_JB_STATISTICS_RESET	
IFXPHONE_DATA_ADD	IFX_TAPI_MAP_DATA_ADD	
IFXPHONE_DATA_REMOVE	IFX_TAPI_MAP_DATA_REMOVE	
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_AAL_CONF	IFX_TAPI_PKT_AAL_CFG_SET	
IFXPHONE_AAL_PROFILE	IFX_TAPI_PKT_AAL_PROFILE_SET	
IFXPHONE_RTCP_STATISTICS	IFX_TAPI_PKT_RTCP_STATISTICS_GET	
IFXPHONE_RTCP_RESET	IFX_TAPI_PKT_RTCP_STATISTICS_RESET	
IFXPHONE_RTP_CONF	IFX_TAPI_PKT_RTP_CFG_SET	
IFXPHONE_RTP_PTCONF	IFX_TAPI_PKT_RTP_PT_CFG_SET	
Miscellaneous Service		
PHONE_CAPABILITIES_CHECK	IFX_TAPI_CAP_CHECK	
PHONE_CAPABILITIES_LIST	IFX_TAPI_CAP_LIST	
PHONE_CAPABILITIES	IFX_TAPI_CAP_NR	
IFXPHONE_PHONE_STATUS	IFX_TAPI_CH_STATUS_GET	
IFXPHONE_REPORT_SET	IFX_TAPI_DEBUG_REPORT_SET	
PHONE_EXCEPTION	IFX_TAPI_EXCEPTION_GET	
IFXPHONE_MASK_EXCEPTION	IFX_TAPI_EXCEPTION_MASK	
IFXPHONE_VERSION_CHECK	IFX_TAPI_VERSION_CHECK	
IFXPHONE_GET_VERSION	IFX_TAPI_VERSION_GET	
Ringing Service		
PHONE_RING		Not supported anymore.
IFXPHONE_RING_CADENCE_HIGH_RES	IFX_TAPI_RING_CADENCE_HR_SET	
PHONE_RING_CADENCE	IFX_TAPI_RING_CADENCE_SET	
IFXPHONE_RING_GET_CONFIG		
IFXPHONE_RING_CONFIG		
PHONE_MAXRINGS		Not supported anymore.
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	

Table 49 Mapping to old names of TAPI ioctl Interfaces (cont'd)

Old Name	New Name	Remarks
IFXPHONE_SET_PCM_CONFIG	IFX_TAPI_PCM_CFG_SET	
Fax T.38 Service		
IFXPHONE_FAX_SETDEMOD	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	
Call on Hold Support Service		
IFXPHONE_CALL_HOLD	IFXPHONE_CALL_HOLD	
IFXPHONE_RESUME_CALL	IFXPHONE_CALL_RESUME	

4.3.2 Mapping of Constants

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

Table 50 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	
TAPI_TONE_SRC_DSP	IFX_TAPI_TONE_SRC_DSP	
TAPI_TONE_SRC_TG	IFX_TAPI_TONE_SRC_TG	

4.3.3 Mapping of Structures

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

Table 51 Mapping to old names of TAPI Structures

Old Name	New Name	Remarks
EXCEPTION_BITS	IFX_TAPI_EXCEPTION_BITS_t	
TAPI_PHONE_CAPABILITY	IFX_TAPI_CAP_t	
TAPI_INIT	IFX_TAPI_CH_INIT_t	
TAPI_PHONE_STATUS	IFX_TAPI_CH_STATUS_t	
TAPI_CID_CONF_t	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_MSG_DATE	IFX_TAPI_CID_MSG_DATE_t	
TAPI_CID_MSG_STRING	IFX_TAPI_CID_MSG_STRING_t	
TAPI_CID_INFO	IFX_TAPI_CID_MSG_t	
TAPI_CIDRX_DATA	IFX_TAPI_CID_RX_DATA_t	
TAPI_CIDRX_STATUS	IFX_TAPI_CID_RX_STATUS_t	
TAPI_CID_TIMING	IFX_TAPI_CID_TIMING_t	
TAPI_PHONE_AGC	IFX_TAPI_ENC_AGC_CFG_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_VALIDATION_TIMES	IFX_TAPI_LINE_HOOK_VT_t	
TAPI_PHONE_VOLUME	IFX_TAPI_LINE_VOLUME_t	
TAPI_DATA_MAPPING	IFX_TAPI_MAP_DATA_t	
TAPI_PCM_MAPPING	IFX_TAPI_MAP_PCM_t	
TAPI_PHONE_MAPPING	IFX_TAPI_MAP_PHONE_t	
TAPI_PCM_CONFIG	IFX_TAPI_PCM_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_RING_CADENCE	IFX_TAPI_RING_CADENCE_t	
TAPI_RING_CONFIG	IFX_TAPI_RING_CFG_t	
TAPI_PHONE_SIGDETECTION	IFX_TAPI_SIG_DETECTION_t	
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_TONE_COMPOSED	IFX_TAPI_TONE_COMPOSED_t	
TAPI_PHONE_CPTD	IFX_TAPI_TONE_CPTD_t	
TAPI_TONE_SIMPLE	IFX_TAPI_TONE_SIMPLE_t	
TAPI_PHONE_VERSION	IFX_TAPI_VERSION_t	
TAPI_CID_CONFIG	IFX_TAPI_CID_CFG_t	
TAPI_CID2_CONFIG		Not supported anymore.
TAPI_CID2_DEFAULT_CONFIG		Not supported anymore.
TAPI_CID2_NTT_CONFIG		Not supported anymore.

Table 51 Mapping to old names of TAPI Structures (cont'd)

Old Name	New Name	Remarks
TAPI_COMPOSED_TONE	IFX_TAPI_TONE_COMPOSED_t	
TAPI_DTMFCID		Not supported anymore.
TAPI_PHONE_CID	IFX_TAPI_CID_MSG_t	
TAPI_SIMPLE_TONE	IFX_TAPI_TONE_SIMPLE_t	

4.3.4 Mapping of Unions

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

Table 52 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	

4.3.5 Mapping of Enumerations

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

Table 53 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_AGC_MODE	IFX_TAPI_ENC_AGC_MODE_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	
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	

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

Old Name	New Name	Remarks
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**	
TAPI_VALIDATION_TYPES	**IFX_TAPI_LINE_HOOK_VALIDATION_TYPE_t**	
TONE_GROUP	**IFX_TAPI_TONE_GROUP_t**	

4.3.6 Mapping of Enumerations

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

Table 54 Mapping to old names of TAPI Enumerations

Old Name	New Name	Remarks
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_AGC_MODE	IFX_TAPI_ENC_AGC_MODE_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	
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.

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

Old Name	New Name	Remarks
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**	
TAPI_VALIDATION_TYPES	**IFX_TAPI_LINE_HOOK_VALIDATION_TYPE_t**	
TONE_GROUP	**IFX_TAPI_TONE_GROUP_t**	

5 Device Driver Interfaces

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

5.1 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 55 Non TAPI Interface Overview

Name	Description
Polling Interface	Polling Interface
Basic Interface	Basic Interface
Driver Initialization Interface	Driver Initialization Interface
GPIO Interface	Control the device and channel specific IO pins.
Driver Kernel Interfaces	Kernel Interface for GPIO/IO pin handling.

5.1.1 Polling Interface

Control the device and channels in polling mode.

Table 56 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 57 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 58 Function Overview of Polling Interface

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

5.1.1.1 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

5.1.1.2 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

5.1.2 Basic Interface

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

Table 59 IO-control Overview of Basic Interface

Name	Description
FIO_VINETIC_DRV_CTRL	Driver configuration.
FIO_VINETIC_VERS	Read relevant version information.
FIO_VINETIC_REPORT_SET	Set the driver report levels if the driver is compiled with ENABLE_TRACE.
FIO_VINETIC_WCMD	Write a VINETIC® command.
FIO_VINETIC_RCMD	Read command.
FIO_VINETIC_INIT	VINETIC® initialization.
FIO_VINETIC_RUNTIME_TRACE_SET	Driver runtime tracing of register accesses.
FIO_VINETIC_LASTERR	Get the last occurred error.

Table 60 Constant Overview of Basic Interface

Name	Description
NO_BCONF	Flag for VINETIC_IO_INIT to avoid no board configuration.
NO_EDSP_START	Flag for VINETIC_IO_INIT to avoid EDSP start.
NO_PHI_DWLD	Flag for VINETIC_IO_INIT to avoid PHI download in case of V1.4.
NO_CRAM_DWLD	Flag for VINETIC_IO_INIT to avoid CRAM 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_FW_DWLD	Flag for VINETIC_IO_INIT to avoid firmware download.
DC_DWLD	Flag for VINETIC_IO_INIT to perform a DC control download in case of V1.4.

Table 61 Structure Overview of Basic Interface

Name	Description
VINETIC_IO_DRVCTRL	Driver global configuration structure.

5.1.2.1 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

5.1.2.2 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

5.1.2.3 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

5.1.2.4 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);
```

5.1.2.5 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
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

5.1.2.6 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);
```

5.1.3 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.

5.1.3.1 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

5.1.3.2 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

5.1.4 GPIO Interface

Control the device and channel specific IO pins.

Table 65 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 66 Structure Overview of GPIO Interface

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

5.1.4.1 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

5.1.4.2 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

5.1.4.3 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

5.1.4.4 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

5.1.4.5 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

5.1.4.6 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

5.1.4.7 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

5.1.5 Driver Kernel Interfaces

Kernel Interface for GPIO/IO pin handling.

Table 67 Enumerator Overview of Driver Kernel Interface

Name	Description
VINETIC_GPIO_MODE	GPIO pin configuration modes.

Table 68 Function Overview of Driver Kernel Interface

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

5.2 Type Definition Reference

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

5.2.1 IO-control Reference

This chapter contains the IO-control reference.

Table 69 IO-control Overview of Non TAPI Interfaces

Name	Description
FIO_VINETIC_DRV_CTRL	Driver configuration.
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_VERS	Read relevant version information.
FIO_VINETIC_WCMD	Write a VINETIC® command.
FIO_VINETIC_RCMD	Read command.
FIO_VINETIC_INIT	VINETIC® Initialization.
FIO_VINETIC_LASTERR	Get the last occurred error.
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.
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.

5.2.2 Constant Reference

This chapter contains the constant reference.

Table 70 Constant Overview of Non TAPI Interfaces

Name	Description
MAX_CMD_WORD	Maximal Command/Data Words.
MAX_PACKET_WORD	Maximal Packet Words.
VINETIC_CH_NR	VINETIC® maximum channel number.
VINETIC_ANA_CH_NR	VINETIC® maximum analog channel number.
NO_BCONF	Flag for VINETIC_IO_INIT to avoid no board configuration.
NO_EDSP_START	Flag for VINETIC_IO_INIT to avoid EDSP start.
NO_FW_DWLD	Flag for VINETIC_IO_INIT to avoid firmware download.
VIN_BUF_HDR1_CH	Packet Header Masks for channel number.
VIN_BUF_HDR1_DEV	Packet Header Masks for device number.

Table 70 Constant Overview of Non TAPI Interfaces (cont'd)

Name	Description
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 71 Constant Reference for Non TAPI Interfaces

Name and Description	Value
MAX_CMD_WORD Maximal Command/Data Words.	31_D
MAX_PACKET_WORD Maximal Packet Words.	255_D
VINETIC_CH_NR VINETIC® maximum channel number.	8_D
VINETIC_ANA_CH_NR VINETIC® maximum analog channel number.	4_D
NO_BCONF Flag for VINETIC_IO_INIT to avoid no board configuration.	1_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
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

5.2.3 Structure Reference

This chapter contains the Structure reference.

Table 72 Structure Overview of Non TAPI Interfaces

Name	Description
VINETIC_BasicDeviceInit_t	VINETIC® Basic Device Initialization structure.
VINETIC_GPIO_CONFIG	GPIO Configuration Structure.
VINETIC_IO_DRVCTRL	Driver global configuration structure.
VINETIC_IO_GPIO_CONTROL	GPIO Control Structure.
VINETIC_IO_INIT	Structure used for device initialization
VINETIC_IO_MB_CMD	IO structure for write and read chip commands for debugging purposes only.
VINETIC_IO_USR	VINETIC® User Interface structure.
VINETIC_IO_VERSION	Version IO structure.

5.2.3.1 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.

5.2.3.2 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.

5.2.3.3 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

5.2.3.4 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.

5.2.3.5 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

Data Type	Name	Description
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
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

5.2.3.6 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

5.2.3.7 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	nEdspIntern	EDSP version step.

5.2.4 Enumerator Reference

This chapter contains the Enumerator reference.

Table 73 Enumerator Overview of Non TAPI Interfaces

Name	Description
VINETIC_IO_CHIP_REVISION	VINETIC® Chip Revision.
VINETIC_IO_CHIP_MAJOR_REVISION	VINETIC® Chip Major Revision.
VINETIC_IO_CHIP_TYPE	VINETIC® chip types, depending on register Revision and firmware download.
VIN_ACCESS	VINETIC® Access Modes.
VINETIC_GPIO_MODE	GPIO pin configuration modes.
DEV_ERR	Driver error codes.

5.2.4.1 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_2CPE_V21 = 0x60
} 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.
VINETIC_2CPE_V21	60_H	Version V2.1 of VINETIC®-2CPE/-1CPE.

5.2.4.2 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.

5.2.4.3 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_TYPE_CPE = 0x5
} 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
VINETIC_TYPE_CPE	5_H	Chip type CPE

5.2.4.4 VIN_ACCESS

Description

VINETIC® Access Modes.

Only available in VINETIC®-2CPE/-1CPE 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.

5.2.4.5 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	$000F_H$	

5.2.5 Function Reference

This chapter contains the function reference.

Table 74 Function Overview of Non TAPI Interfaces

Name	Description
VINETIC_OpenKernel	Open the device from kernel mode.
VINETIC_ReleaseKernel	Release a VINETIC® IO or GPIO pin resource.
VINETIC_GpioReserve	Reserve a VINETIC® IO or GPIO pin resource.
VINETIC_GpioRelease	Release a VINETIC® IO or GPIO pin resource.
VINETIC_GpioConfig	Configure a VINETIC® IO or GPIO pin.
VINETIC_GpioSet	Set the value of 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_DrvCtrl	Polling control function
VINETIC_Poll_Events	Polling event handling
VINETIC_Poll_Down	Normal polling downstream interface functions.
VINETIC_Poll_Up	Normal polling upstream interface functions.
VINETIC_Poll_DownIntlv	Interleaved polling downstream interface functions.
VINETIC_Poll_UpIntlv	Interleaved polling upstream interface functions.

5.2.5.1 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

5.2.5.2 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

5.2.5.3 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

5.2.5.4 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

5.2.5.5 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

5.2.5.6 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

5.2.5.7 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

5.2.5.8 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

5.2.5.9 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

5.2.5.10 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

5.2.5.11 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

5.2.5.12 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

5.2.5.13 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

5.2.5.14 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

6 Line Testing Interfaces

This section describes line testing interfaces.

- **Chapter 6.1** provides a reference for the line testing functions.
- **Chapter 6.2** provides the reference for all types defined for line testing, including
  - **Chapter 6.2.1**, struct reference
  - **Chapter 6.2.2**, enum reference

6.1 Function Reference

This chapter contains the function reference.

Table 75 Function Overview of Non TAPI Interfaces

Name	Description
Ifxphone_LT_GR909_Config	Configure system parameters (SLIC) to use for values calculation, e.g Voltage divider resistors.
Ifxphone_LT_GR909_Start	Start a GR-909 test or test sequence according to measument mask, **IFX_LT_GR909_MASK_t**.
Ifxphone_LT_GR909_GetResults	Gets GR-909 measurement results.

6.1.1 Ifxphone_LT_GR909_Config

Description

Configure system parameters (SLIC) to use for values calculation, e.g Voltage divider resistors.

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

6.1.2 Ifxphone_LT_GR909_Start

Description

Start a GR909 test or test sequence according to measument mask **IFX_LT_GR909_MASK_t**.

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

6.1.3 Ifxphone_LT_GR909_GetResults

Description

Gets Gr909 measurement results.

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

6.2 Type Definition Reference

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

6.2.1 Structure Reference

This chapter contains the Structure reference.

Table 76 Structure Overview of Non TAPI Interfaces

Name	Description
IFX_LT_GR909_HPT_t	Hazardous potential test results.
IFX_LT_GR909_FEMF_t	Foreign electromotive forces test results.
IFX_LT_GR909_RFT_t	Resistive faults test results.
IFX_LT_GR909_ROH_t	Receiver off-hook test results.
IFX_LT_GR909_RIT_t	Ringer impedance test results.

6.2.1.1 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].

6.2.1.2 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].

6.2.1.3 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].

6.2.1.4 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].

6.2.1.5 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].

6.2.1.6 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.

6.2.1.7 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.
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.

6.2.2 Enumerator Reference

This chapter contains the Enumerator reference.

Table 77 Enumerator Overview of Non TAPI Interfaces

Name	Description
IFX_LT_GR909_MASK_t	GR909 tests masks.

6.2.2.1 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.

References

[1] VINETIC®-2CPE/-1CPE (PEB 3332/-3331) Version 2.1 Prel. Data Sheet Rev. 1.0, 2005-11-03

[2] VINETIC®-CPE Version 2.1 Device Driver Porting and Integration Guide Rev. 1.0, in preparation

[3] VINETIC®-CPE Prel. User's Manual System Description Rev. 1.0, in preparation

[4] VINETIC®-CPE System 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 Inupt Output
I	
IETF	Internet Engineering Task Force
IFX	Infineon
IO	Input/Output
J	
JB	Jitter Buffer
L	
LEC	Line Echo Canceller
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

Published by Infineon Technologies AG