MAYAH communications

Application Note 10

ISDN CAPI Trace

1 Introduction: General concepts of the application interface Common ISDN API

COMMON ISDN API (CAPI) is an application programming interface standard used to access ISDN equipment connected to basic rate interfaces **(BRI)** and primary rate interfaces **(PRI)**. For application developers this standard provides a well-defined mechanism for communications over ISDN lines without having to adjust to the particularities of specific hardware vendors' implementations. In return, ISDN equipment vendors benefit from a wealth of applications ready to run with their equipment.

Having begun in 1989 with manufacturers defining an application interface that would be accepted in the growing ISDN market, **CAPI** is now a well-established standard. Potential cost savings were the driving force for **COMMON ISDN API** controller and application development. Commercial users are rapidly migrating to ISDN as the principal medium for exchanging data in a wide range of formats.

COMMON ISDN API Version 2.0 reflects more than ten years of ISDN business implementation experience in an exploding market. It incorporates all the benefits of **CAPI Version 1.1** as well as all actual aspects of ISDN (such as Group 3 fax connectivity or video-telephony). It is based on Q.931 / ETS 300 102, but not limited to these protocols. It simplifies the development of ISDN applications through many default values which do not need to be programmed. It keeps applications free of ISDN protocol knowledge, thus making a great variety of applications possible.
COMMON ISDN API provides an abstraction of ISDN services that is independent of the underlying network and of the adapter used to connect to the network. It provides an easy-to-use interface for applications and offers a unified access to different ISDN services such as data, voice, fax, video, telephony, etc.

2 Overview

COMMON-ISDN-API provides a standardized interface which allows any number of application programs to use any number of ISDN drivers and ISDN controllers. Applications can be freely assigned to specific drivers and controllers:

- One application can use one controller

- One application can use more than one controller
- Several applications can share a single controller
- Several applications can share more than one controller

Applications can use different protocols at different protocol levels; **COMMON ISDN API** provides a selection mechanism to support this. **COMMON ISDN API** also performs an abstraction from different protocol variants, creating a standardized network access. All connection-related data such as connection state, display messages etc. is available to applications at any time.

COMMON ISDN API covers the entire signaling protocol as well as protocol layers 1 to 3 (physical and framing layer, data link layer and network layer) of the data channels. The interface of **COMMON ISDN API** is located between Layer 3 and Layer 4, and provides a point of reference for applications and higher level protocols.

COMMON ISDN API offers many commonly used protocols to applications without low-level protocol knowledge. The default protocol is **ISO 7776** (X.75 SLP), i.e. framing protocol **HDLC**, data link protocol **ISO 7776** (X.75 SLP), and a transparent network layer. Other supported framing layer variants are **HDLC inverted**, **PCM** (bit-transparent with byte framing) **64/56 kbit**, and **V.110** sync / async. **COMMON ISDN API** integrates the following data link and network layer protocols: **LAPD** in accordance with **Q.921** for **X.25 D-channel** implementation, **PPP** (Point-to-Point Protocol), **ISO 8208** (X.25 DTE-DTE), **X.25 DCE**, **T.90NL** (with T.70NL compatibility) and **T.30** (Group 3 fax).

Even if not all protocols can be completely fitted into the OSI scheme, **COMMON ISDN API** always supports three layers. Applications can configure each layer. In case of illegal or meaningless protocol stack combinations (e.g. bit-transparent 56 kbit/s and X.25 DCE), **COMMON ISDN API** reports an error.

3 Message Overview

This chapter presents the basic mechanism used by **COMMON ISDN API**, based on message queues for the exchange of commands and data.

The term *message* is a fundamental one in the definition of **COMMON ISDN API**. *Messages* are information defined by **COMMON-ISDN-API,** exchanged between the application and **COMMON ISDN API** by an asynchronous mechanism. This technique achieves operating system independence.

3.1 General Message Protocol

Communication between the application and **COMMON-ISDN-API** always uses the following general protocol:

A message is always followed by an appropriate reply. Messages going from an application to **COMMON ISDN API** are called **Requests**; the corresponding answer from **COMMON ISDN API** is called a **Confirmation**. Messages initiated by **COMMON ISDN API** are called **Indications**; the corresponding answer from an application is called a **Response**. This is also reflected in the naming convention for messages: every message name ends with the appropriate suffix (_REQ, _CONF, _IND and _RESP).

Each message contains a message number. **COMMON ISDN API** always returns the number used in the REQUEST message in the corresponding CONFIRMATION. Applications may choose unique message numbers to identify message correlation before interpreting incoming messages. INDICATIONS from **COMMON ISDN API** are numbered so that an application is guaranteed to get a different message number in every incoming INDICATION.

An application is not allowed to send RESPONSE messages without having received an INDICATION. **COMMON ISDN API** ignores such illegal messages.

3.2 Message Structure

All messages exchanged between applications and **COMMON ISDN API** consist of a fixed-length header and a parameter area of variable length, with one parameter immediately following another. No padding occurs in the message header or parameter area.

Figure 1: Message Layout

Message Header	**Parameter 1**	**Parameter 2**	**...**	**Parameter n**

In order to facilitate future extensions, messages containing more parameters than defined shall be treated as valid messages. **COMMON ISDN API** implementations and applications shall ignore all such additional parameters.

The message header has the following layout:

Figure 2: Message Header Layout

Total Length	**Application ID**	**Command**	**Subcommand**	**Message Nr**

The message header is composed of the following functional elements:

Header element	**Type**	**Contents**
Total Length	Word	Total length of the message including the complete message header.
Application ID	Word	Unique identification number of the application. The application ID is assigned to the application by **COMMON-ISDN-API** in the CAPI_REGISTER operation
Command	Byte	Command
Subcommand	Byte	Command extension
Message number	word	Message number as described in 3.1 above

3.3 Table of Messages

Messages are logically grouped into three kinds:

- Messages concerning the ISDN signaling protocol (D-channel)
- Messages concerning logical connections (B or D-channel)
- Administrative and other messages

The following table gives an overview of the defined messages and their functions.

Table 1: Messages concerning the signaling protocol

Message	Value	Description
CONNECT_REQ	0x02 / 0x80	initiates an outgoing physical connection
CONNECT_CONF	0x02 / 0x81	local confirmation of the request
CONNECT_IND	0x02 / 0x82	indicates an incoming physical connection
CONNECT_RESP	0x02 / 0x83	response to the indication
CONNECT_ACTIVE_IND	0x03 / 0x82	indicates the activation of a physical connection
CONNECT_ACTIVE_RESP	0x03 / 0x83	response to the indication
DISCONNECT_REQ	0x04 / 0x80	initiates clearing down of a physical connection
DISCONNECT_CONF	0x04 / 0x81	local confirmation of the request
DISCONNECT_IND	0x04 / 0x82	indicates the clearing of a physical connection
DISCONNECT_RESP	0x04 / 0x83	response to the indication
ALERT_REQ	0x01 / 0x80	initiates sending of ALERT, i.e. compatibility with call
ALERT_CONF	0x01 / 0x81	local confirmation of the request
INFO_REQ	0x08 / 0x80	initiates sending of signaling information
INFO_CONF	0x08 / 0x81	local confirmation of the request
INFO_IND	0x08 / 0x82	indicates specified signaling information
INFO_RESP	0x08 / 0x83	response to the indication

Table 2: Messages concerning logical connections

Message	Value	Description
CONNECT_B3_REQ	0x82 / 0x80	initiates an outgoing logical connection
CONNECT_B3_CONF	0x82 / 0x81	local confirmation of the request
CONNECT_B3_IND	0x82 / 0x82	indicates an incoming logical connection
CONNECT_B3_RESP	0x82 / 0x83	response to the indication
CONNECT_B3_ACTIVE_IND	0x83 / 0x82	indicates the activation of a logical connection
CONNECT_B3_ACTIVE_RESP	0x83 / 0x83	response to the indication
CONNECT_B3_T90_ACTIVE_IND	0x88 / 0x82	indicates switching from T.70NL to T.90NL
CONNECT_B3_T90_ACTIVE_RESP	0x88 / 0x83	response to the indication
DISCONNECT_B3_REQ	0x84 / 0x80	initiates clearing down of a logical connection
DISCONNECT_B3_CONF	0x84 / 0x81	local confirmation of the request
DISCONNECT_B3_IND	0x84 / 0x82	indicates the clearing down of a logical connection
DISCONNECT_B3_RESP	0x84 / 0x83	response to the indication

DATA_B3_REQ	0x86 / 0x80	initiates sending of data over a logical connection
DATA_B3_CONF	0x86 / 0x81	local confirmation of the request
DATA_B3_IND	0x86 / 0x82	indicates incoming data over a logical connection
DATA_B3_RESP	0x86 / 0x83	response to the indication
RESET_B3_REQ	0x87 / 0x80	initiates the resetting of a logical connection
RESET_B3_CONF	0x87 / 0x81	local confirmation of the request
RESET_B3_IND	0x87 / 0x82	indicates the resetting of a logical connection
RESET_B3_RESP	0x87 / 0x83	response to the indication

Table 3: Administrative and other messages

Message	**Value**	**Description**
LISTEN_REQ	0x05 / 0x80	activates call and info indications
LISTEN_CONF	0x05 / 0x81	local confirmation of the request
FACILITY_REQ	0x80 / 0x80	requests additional facilities (e.g. ext. equipment)
FACILITY_CONF	0x80 / 0x81	local confirmation of the request
FACILITY_IND	0x80 / 0x82	indicates additional facilities (e.g. ext. equipment)
FACILITY_RESP	0x80 / 0x83	response to the indication
SELECT_B_PROTOCOL_REQ	0x41 / 0x80	selects protocol stack used for a logical connection
SELECT_B_PROTOCOL_CONF	0x41 / 0x81	local confirmation of the request
MANUFACTURER_REQ	0xFF / 0x80	manufacturer-specific operation
MANUFACTURER_CONF	0xFF / 0x81	manufacturer-specific operation
MANUFACTURER_IND	0xFF / 0x82	manufacturer-specific operation
MANUFACTURER_RESP	0xFF / 0x83	manufacturer-specific operation

This has been a short general overview about **COMMON-ISDN-API.** If you want more detailed information about this theme click http://www.capi.org. The following chapters deal with the Centauri CAPI trace.

4 MAYAH codec and ISDN CAPI Trace

The MAYAH codec uses the **COMMON ISDN API** for tracing the process of an ISDN connection. The CAPI trace of the MAYAH codec provides a possibility to journalize all messages exchanged between MAYAH codec Software and **COMMON ISDN API**. It will be enabled if the file **isdncapi.run** on the **root** partition of the MAYAH codec flash card is found at MAYAH codec system start. The tracing is addressed to the files **\isdncapi.act** and **\isdncapi.prv** which can be examined afterwards.
NOTE: This option is only for troubleshooting and analyzing. Normal MAYAH codec function is restricted in ISDN trace mode.

5 Executing ISDN CAPI Trace on MAYAH codec

Carry out the following steps to perform the MAYAH codec ISDN CAPI trace:

1) Create an ASCII file called **isdncapi.run** on your PC
2) FTP copy the file **isdncapi.run** to the main directory of the **root** partition of the MAYAH codec flash card (Default FTP user: Admin; default FTP password: Power)
3) Reboot the Centauri so that the **isdncapi.run** can be recovered to enable the trace mode
4) Execute the Centauri ISDN connection(s) you wish to trace
5) FTP download the files **isdncapi.act** and/or **isdncapi.prv** from the **root** partition of the MAYAH codec flash card
6) FTP delete the file **isdncapi.run** from the **root** partition of the MAYAH codec flash card
7) Reboot the MAYAH codec to disable the trace mode
8) Send the files **isdncapi.act** and/or **isdncapi.prv** to Mayah Communications for analysis