NAME

isdnio – ISDN interfaces

SYNOPSIS

#include <sun/audioio.h>
#include <sun/isdnio.h>

int ioctl (int *fd*, **int** command, **/*** *arg* ***/ ...);**

DESCRIPTION

ISDN ioctl commands are a subset of **ioctl**(2) commands that perform a variety of control functions on Integrated Services Digital Network (ISDN) STREAMS devices. The arguments *command* and *arg* are passed to the file designated by *fd* and are interpreted by the ISDN device driver.

fd is an open file descriptor that refers to a stream. *command* determines the control function to be performed as described in the IOCTLS section of this document. *arg* represents additional information that is needed by *command*. The type of *arg* depends upon the command, but generally it is an integer or a pointer to a *command*-specific data structure.

Since these ISDN commands are a subset of *ioctl* and **streamio**(7I), they are subject to errors as described in those interface descriptions.

This set of generic ISDN **ioctl** commands is meant to control various types of ISDN STREAMS device drivers. The following paragraphs give some background on various types of ISDN hardware interfaces and data formats, and other device characteristics.

Controllers, Interfaces, and Channels

This manual page discusses operations on, and facilities provided by ISDN controllers, interfaces and channels. A controller is usually a hardware peripheral device that provides one or more ISDN interfaces and zero or more auxiliary interfaces. In this context, the term interface is synonymous with the term “port”. Each interface can provide one or more channels.

Time Division Multiplexed Serial Interfaces

ISDN BRI-TE, BRI-NT, and PRI interfaces are all examples of Time Division Multiplexed Serial Interfaces. As an example, a Basic Rate ISDN (BRI) Terminal Equipment (TE) interface provides one D-channel and two B-channels on the same set of signal wires. The BRI interface, at the S reference point, operates at a bit rate of 192,000 bits per second. The bits are encoded using a pseudoternary coding system that encodes a logic one as zero volts, and a logic zero as a positive or negative voltage. Encoding rules state that adjacent logic zeros must be encoded with opposite voltages. Violations of this rule are used to indicate framing information such that there are 4000 frames per second, each containing 48 bits. These 48 bits are divided into channels. Not including framing and synchronization bits, the frame is divided into 8 bits for the B1-channel, 1 bit for the D-channel, 8 bits for B2, 1 bit for D, 8 bits for B1, 1 bit for D, and 8 bits for B2. This results in a 64,000 bps B1-channel, a 64,000 bps B2-channel, and a 16,000 bps D-channel, all on the same serial interface.

Basic Rate ISDN

A Basic Rate ISDN (BRI) interface consists of a 16000 bit per second Delta Channel (D-channel) for signaling and X.25 packet transmission, and two 64000 bit per second Bearer Channels (B-channels) for transmission of voice or data.

The CCITT recommendations on ISDN Basic Rate interfaces, I.430, identify several “reference points” for standardization. From (Stallings89);

> “Reference point T (terminal) corresponds to a minimal ISDN network termination at the customer’s premises. It separates the network provider’s equipment from the user’s equipment. Reference point S (system) corresponds to the interface of individual ISDN terminals. It separates user terminal equipment from network-related communications functions. Reference point R (rate) provides a non-ISDN interface between user equipment that is not ISDN-compatible and adaptor equipment. . . . The final reference point . . . is reference point U (user). This interface describes the full-duplex data signal on the subscriber line.”

Some older technology components of some ISDN networks occasionally steal the low order bit of an ISDN B-channel octet in order to transmit in-band signaling information between switches or other components of the network. Even when out-of-band signaling has been implemented in these networks, and the in-band signaling is no longer needed, the bit-robbing mechanism may still be present. This bit robbing behavior does not appreciably affect a voice call, but it will limit the usable bandwidth of a data call to 56000 bits per second instead of 64000 bits per second. These older network components only seem to exist in the United States of America, Canada and Japan. ISDN B-channel data calls that have one end point in the United States, Canada or Japan may be limited to 56000 bps usable bandwidth instead of the normal 64000 bps. Sometimes the ISDN service provider may be able to supply 56kbps for some calls and 64kbps for other calls. On an international call, the local ISDN service provider may advertise the call as 64kbps even though only 56kbps are reliably delivered because of bit-robbing in the foreign ISDN that is not reported to the local switch.

A Basic Rate Interface implements either a Terminal Equipment (TE) interface or a Network Termination (NT) interface. TE’s can be ISDN telephones, a Group 4 fax, or other ISDN terminal equipment. A TE connects to an NT in order to gain access to a public or private ISDN network. A private ISDN network, such as provided by a Private Branch Exchange (PBX), usually provides access to the public network.

If multi-point configurations are allowed by an NT, it may be possible to connect up to eight TE’s to a single NT interface. All of the TE’s in a multipoint configuration share the same D and B-channels. Contention for B-Channels by multiple TEs is resolved by the ISDN switch (NT) through signaling protocols on the D-channel.

Contention for access to the D-channel is managed by a collision detection and priority mechanism. D-channel call control messages have higher priority than other packets. This media access function is managed at the physical layer.

A BRI-TE interface may implement a “Q-channel”, the Q-channel is a slow speed, 800 bps, data path from a TE to an NT. Although the structure of the Q-channel is defined in the I.430 specification, the use of the Q-channel is for further study.

A BRI-NT interface may implement an “S-channel”, the S-channel is a slow speed, 4000 bps, data path from a NT to an TE. The use of the S-channel is for further study.

Primary Rate ISDN

Primary Rate ISDN (PRI) interfaces are either 1.544Mbps (T1 rate) or 2.048Mbps (E1 rate) and are typically organized as 23 B-channels and one D-Channel (23B+D) for T1 rates, and 30 B-Channels and one D-Channel (30B+D) for E1 rates. The D-channels on a PRI interface operate at 64000 bits per second. T1 rate PRI interface is the standard in the United States, Canada and Japan while E1 rate PRI interface is the standard in European countries. Some E1 rate PRI interface implementations allow access to channel zero which is used for framing.

Channel Types

ISDN channels fall into several categories; D-channels, bearer channels, and management pseudo channels. Each channel has a corresponding device name somewhere under the directory **/dev/isdn/** as documented in the appropriate hardware specific manual page.

D-channels There is at most one D-channel per ISDN interface. The D-channel carries signaling information for the management of ISDN calls and can also carry X.25 packet data. In the case of a PRI interface, there may actually be no D-channel if Non-Facility Associated Signaling is used. D-channels carry data packets that are framed and checked for transmission errors according to the LAP-D protocol. LAP-D uses framing and error checking identical to the High Speed Data Link (HDLC) protocol.

B-channels BRI interfaces have two B-channels, B1 and B2. On a BRI interface, the only other type of channel is an H-channel which is a concatenation of the B1 and B2 channels. An H-channel is accessed by opening the “base” channel, B1 in this case, and using the **ISDN_SET_FORMAT** ioctl to change the configuration of the B-channel from 8-bit, 8 kHz to 16-bit, 8kHz.

On a primary rate interface, B channels are numbered from 0 to 31 in Europe and 1 to 23 in the United States, Canada and Japan.

H-Channels A BRI or PRI interface can offer multiple B-channels concatenated into a single, higher bandwidth channel. These concatenated B-channels are referred to as an “H-channels” on a BRI interface. The PRI interface version of an H-channel is referred to as an H*n*-channels where *n* is a number indicating how the B-channels have been aggregated into a single channel.

- A PRI interface H0 channel is 384 kbps allowing 3H0+D on a T1 rate PRI interface and 4H0+D channels on an E1 rate PRI interface.
- A T1 PRI interface H11 channel is 1536 kbps (24×64000bps). This will consume the channel normally reserved for the D-channel, so signaling must be done with Non-Facility Associated Signaling (NFAS) from another PRI interface.
- An E1 PRI interface H12 channel is 1920 kbps (30×64000bps). An H12-channel leaves room for the framing-channel as well as the D-channel.

Auxiliary channels

Auxiliary channels are non-ISDN hardware interfaces that are closely tied to the ISDN interfaces. An example would be a video or audio coder/decoder (codec). The existence of an auxiliary channel usually implies that one or more B-channels can be “connected” to an auxiliary interface in hardware.

Management pseudo-channels

A management pseudo-channel is used for the management of a controller, interface, or hardware channel. Management channels allow for out-of-band control of hardware interfaces and for out-of-band notification of status changes. There is at least one management device per hardware interface.

There are three different types of management channels implemented by ISDN hardware drivers:

- A controller management device handles all ioctls that simultaneously affect hardware channels on different interfaces. Examples include resetting a controller, μ-code downloading of a controller, or the connection of an ISDN B-channel to an auxiliary channel that represents an audio coder/decoder (codec). The latter case would be accomplished using the **ISDN_SET_CHANNEL** ioctl.
- An interface management device handles all ioctls that affect multiple channels on the same interface. Messages associated with the activation and deactivation of an interface arrive on the management device associated with the D channel of an ISDN interface.
- Auxiliary interfaces may also have management devices. See the hardware specific man pages for operations on auxiliary devices.

Trace pseudo-channels

A device driver may choose to implement a trace device for a data or management channel. Trace channels receive a special **M_PROTO** header with the original channel’s original **M_PROTO** or **M_DATA** message appended to the special header. The header is described by:

```
typedef struct {
        uint_t   seq;            /* Sequence number */
        int      type;           /* device dependent */
        struct timeval timestamp;
        char     _f[8];          /* filler */
} audtrace_hdr_t;
```

ISDN Channel types

The **isdn_chan_t** type enumerates the channels available on ISDN interfaces. If a particular controller implements any auxiliary channels then those auxiliary channels will be described in a controller specific manual page. The defined channels are described by the **isdn_chan_t** type as shown below:

```
/* ISDN channels */
typedef enum {
        ISDN_CHAN_NONE = 0x0,       /* No channel given */
        ISDN_CHAN_SELF,             /* The channel performing the ioctl */
```

```
        ISDN_CHAN_HOST,              /* Unix STREAM */

        ISDN_CHAN_CTRL_MGT,          /* Controller management */

        /* TE channel defines */
        ISDN_CHAN_TE_MGT,            /* Receives activation/deactivation */
        ISDN_CHAN_TE_D_TRACE,        /* Trace device for protocol analysis apps */
        ISDN_CHAN_TE_D,
        ISDN_CHAN_TE_B1,
        ISDN_CHAN_TE_B2,

        /* NT channel defines */
        ISDN_CHAN_NT_MGT,            /* Receives activation/deactivation */
        ISDN_CHAN_NT_D_TRACE,        /* Trace device for protocol analysis apps */
        ISDN_CHAN_NT_D,
        ISDN_CHAN_NT_B1,
        ISDN_CHAN_NT_B2,

        /* Primary rate ISDN */
        ISDN_CHAN_PRI_MGT,
        ISDN_CHAN_PRI_D,
        ISDN_CHAN_PRI_B0, ISDN_CHAN_PRI_B1,
        ISDN_CHAN_PRI_B2, ISDN_CHAN_PRI_B3,
        ISDN_CHAN_PRI_B4, ISDN_CHAN_PRI_B5,
        ISDN_CHAN_PRI_B6, ISDN_CHAN_PRI_B7,
        ISDN_CHAN_PRI_B8, ISDN_CHAN_PRI_B9,
        ISDN_CHAN_PRI_B10, ISDN_CHAN_PRI_B11,
        ISDN_CHAN_PRI_B12, ISDN_CHAN_PRI_B13,
        ISDN_CHAN_PRI_B14, ISDN_CHAN_PRI_B15,
        ISDN_CHAN_PRI_B16, ISDN_CHAN_PRI_B17,
        ISDN_CHAN_PRI_B18, ISDN_CHAN_PRI_B19,
        ISDN_CHAN_PRI_B20, ISDN_CHAN_PRI_B21,
        ISDN_CHAN_PRI_B22, ISDN_CHAN_PRI_B23,
        ISDN_CHAN_PRI_B24, ISDN_CHAN_PRI_B25,
        ISDN_CHAN_PRI_B26, ISDN_CHAN_PRI_B27,
        ISDN_CHAN_PRI_B28, ISDN_CHAN_PRI_B29,
        ISDN_CHAN_PRI_B30, ISDN_CHAN_PRI_B31,

        /* Auxiliary channel defines */
        ISDN_CHAN_AUX0, ISDN_CHAN_AUX1, ISDN_CHAN_AUX2, ISDN_CHAN_AUX3,
        ISDN_CHAN_AUX4, ISDN_CHAN_AUX5, ISDN_CHAN_AUX6, ISDN_CHAN_AUX7
} isdn_chan_t;
```

ISDN Interface types

The **isdn_interface_t** type enumerates the interfaces available on ISDN controllers. The defined interfaces are described by the **isdn_interface_t** type as shown below:

```
/* ISDN interfaces */
typedef enum {
        ISDN_TYPE_UNKNOWN = -1,  /* Not known or applicable */
        ISDN_TYPE_SELF = 0,      /*
                                  * For queries, application may
                                  * put this value into "type" to
                                  * query the state of the file
                                  * descriptor used in an ioctl.
```

```
                                    */
        ISDN_TYPE_OTHER,            /* Not an ISDN interface */
        ISDN_TYPE_TE,
        ISDN_TYPE_NT,
        ISDN_TYPE_PRI,
} isdn_interface_t;
```

Activation and Deactivation of ISDN Interfaces

The management device associated with an ISDN D-channel is used to request activation, deactivation and receive information about the activation state of the interface. See the descriptions of the **ISDN_PH_ACTIVATE_REQ** and **ISDN_MPH_DEACTIVATE_REQ** ioctls. Changes in the activation state of an interface are communicated to the D-channel application through **M_PROTO** messages sent up-stream on the management device associated with the D-channel. If the D-channel protocol stack is implemented as a user process, the user process can retrieve the **M_PROTO** messages using the **getmsg**(2) system call.

These **M_PROTO** messages have the following format:

```
typedef struct isdn_message {
        unsigned int            magic;        /* set to ISDN_PROTO_MAGIC */
        isdn_interface_t        type;         /* Interface type */
        isdn_message_type_t     message;      /* CCITT or vendor Primitive */
        unsigned int            vendor[5];    /* Vendor specific content */
} isdn_message_t;

typedef enum isdn_message_type {
        ISDN_VPH_VENDOR = 0,                  /* Vendor specific messages */

        ISDN_PH_AI,                           /* Physical: Activation Ind */
        ISDN_PH_DI,                           /* Physical: Deactivation Ind */

        ISDN_MPH_AI,                          /* Management: Activation Ind */
        ISDN_MPH_DI,                          /* Management: Deactivation Ind */
        ISDN_MPH_EI1,                         /* Management: Error 1 Indication */
        ISDN_MPH_EI2,                         /* Management: Error 2 Indication */
        ISDN_MPH_II_C,                        /* Management: Info Ind, connection */
        ISDN_MPH_II_D                         /* Management: Info Ind, disconn. */
} isdn_message_type_t;
```

IOCTLS

STREAMS IOCTLS

All of the **streamio**(7I) **ioctl** commands may be issued for a device conforming to the the **isdnio** interface.

ISDN interfaces that allow access to audio data should implement a reasonable subset of the **audio**(7I) interface.

ISDN ioctls

ISDN_PH_ACTIVATE_REQ

Request ISDN physical layer activation. This *command* is valid for both TE and NT interfaces. *fd* must be a D-channel file descriptor. *arg* is ignored.

TE activation will occur without use of the **ISDN_PH_ACTIVATE_REQ** ioctl if the device corresponding to the TE D-channel is open, “on”, and the ISDN switch is requesting activation.

ISDN_MPH_DEACTIVATE_REQ

fd must be a NT D-channel file descriptor. *arg* is ignored.

This *command* requests ISDN physical layer de-activation. This is not valid for TE interfaces. A TE interace may be turned off by use of the **ISDN_PARAM_POWER** command or by **close**(2) on the associated *fd*.

ISDN_ACTIVATION_STATUS

fd is the file descriptor for a D-channel, the management device associated with an ISDN interface, or the management device associated with the controller. *arg* is a pointer to an **isdn_activation_status_t** structure. Although it is possible for applications to determine the current activation state with this ioctl, a D-channel protocol stack should instead process messages from the management pseudo channel associated with the D-channel.

```
typedef struct isdn_activation_status {
        isdn_interface_t              type;
        enum isdn_activation_state    activation;
} isdn_activation_status_t;

typedef enum isdn_activation_state {
        ISDN_OFF = 0,                 /* Interface is powered down */
        ISDN_UNPLUGGED,               /* Power but no-physical connection */
        ISDN_DEACTIVATED_REQ          /* Pending Deactivation, NT Only */
        ISDN_DEACTIVATED,             /* Activation is permitted */
        ISDN_ACTIVATE_REQ,            /* Attempting to activate */
        ISDN_ACTIVATED,               /* Interface is activated */
} isdn_activation_state_t;
```

The **type** field should be set to **ISDN_TYPE_SELF**. The device specific interface type will be returned in the type field.

The **isdn_activation_status_t** structure contains the interface type and the current activation state. **type** is the interface type and should be set by the caller to **ISDN_TYPE_SELF**.

ISDN_INTERFACE_STATUS

The **ISDN_INTERFACE_STATUS** ioctl retrieves the status and statistics of an ISDN interface. The requesting channel must own the interface whose status is being requested or the ioctl will fail. *fd* is the file descriptor for an ISDN interface management device. *arg* is a pointer to a **struct isdn_interface_info**. If the **interface** field is set to **ISDN_TYPE_SELF**, it will be changed in the returned structure to reflect the proper device-specific interface of the requesting *fd* .

```
typedef struct isdn_interface_info {
        isdn_interface_t                          interface;

        enum isdn_activation_state                activation;

        unsigned int          ph_ai;              /* Physical: Activation Ind */
        unsigned int          ph_di;              /* Physical: Deactivation Ind */
        unsigned int          mph_ai;             /* Management: Activation Ind */
        unsigned int          mph_di;             /* Management: Deactivation Ind */
        unsigned int          mph_ei1;            /* Management: Error 1 Indication */
```

```
        unsigned int        mph_ei2;        /* Management: Error 2 Indication */
        unsigned int        mph_ii_c;       /* Management: Info Ind, connection */
        unsigned int        mph_ii_d;       /* Management: Info Ind, disconn. */
} isdn_interface_info_t;
```

ISDN_CHANNEL_STATUS

The **ISDN_CHANNEL_STATUS** ioctl retrieves the status and statistics of an ISDN channel. The requesting channel must own the channel whose status is being requested or the ioctl will fail. *fd* is any file descriptor. *arg* is a pointer to a **struct isdn_channel_info**. If the **interface** field is set to **ISDN_CHAN_SELF**, it will be changed in the returned structure to reflect the proper device-specific channel of the requesting *fd* .

```
typedef struct isdn_channel_info {
        isdn_chan_t         channel;

        enum isdn_iostate   iostate;

        struct isdn_io_stats {
        ulong_t             packets;        /* packets transmitted or received */
        ulong_t             octets;         /* octets transmitted or received */
        ulong_t             errors;         /* errors packets transmitted or received */
        } transmit, receive;
} isdn_channel_info_t;
```

ISDN_SET_PARAM

fd is the file descriptor for a management device. *arg* is a pointer to a **struct isdn_param**. This *command* allows the setting of various ISDN physical layer parameters such as timers. This *command* uses the same arguments as the **ISDN_GET_PARAM** *command.*

ISDN_GET_PARAM

fd is the file descriptor for a management device. *arg* is a pointer to a **struct isdn_param** This *command* provides for querying the value of a particular ISDN physical layer parameter.

```
typedef enum {
        ISDN_PARAM_NONE = 0,
        ISDN_PARAM_NT_T101,             /* NT Timer, 5-30 s, in milliseconds */
        ISDN_PARAM_NT_T102,             /* NT Timer, 25-100 ms, in milliseconds */
        ISDN_PARAM_TE_T103,             /* TE Timer, 5-30 s, in milliseconds */
        ISDN_PARAM_TE_T104,             /* TE Timer, 500-1000 ms, in milliseconds */
        ISDN_PARAM_MAINT,               /* Manage the TE Maintenance Channel */
        ISDN_PARAM_ASMB,                /* Modify Activation State Machine */
                                        /* Behavior */
        ISDN_PARAM_POWER,               /* Take the interface online or offline */
        ISDN_PARAM_PAUSE,               /* Paused if == 1, else not paused == 0 */
} isdn_param_tag_t;

enum isdn_param_asmb {
        ISDN_PARAM_TE_ASMB_CCITT88,             /* 1988 bluebook */
        ISDN_PARAM_TE_ASMB_CTS2,                /* Conformance Test Suite 2 */
};
```

```
typedef struct isdn_param {
        isdn_param_tag_t      tag;
        union {
                unsigned int            us;             /* micro seconds */
                unsigned int            ms;             /* Timer value in ms */
                unsigned int            flag;           /* Boolean */
                enum isdn_param_asmb    asmb;
                enum isdn_param_maint   maint;
                struct {
                        isdn_chan_t     channel;        /* Channel to Pause */
                        int             paused;         /* TRUE or FALSE */
                } pause;
                unsigned int            reserved[2];    /* reserved, set to zero */
        } value;
} isdn_param_t;
```

ISDN_PARAM_POWER

If an implementation provides power on and off functions, then power should be on by default. If **flag** is **ISDN_PARAM_POWER_OFF** then a TE interface is forced into state F0, NT interfaces are forced into state G0. If **flag** is **ISDN_PARAM_POWER_ON** then a TE interface will immediately transition to state F3 when the TE D-channel is opened. If **flag** is one, an NT interface will transition to state G1 when the NT D-channel is opened.

Implementations that do not provide **ISDN_POWER** return failure with errno set to **ENXIO**.

ISDN_POWER is different from **ISDN_PH_ACTIVATE_REQ** since CCITT specification requires that if a BRI-TE interface device has power, then it permits activation.

ISDN_PARAM_NT_T101

This parameter accesses the NT timer value T1. The CCITT recommendations specify that timer T1 has a value from 5 to 30 seconds. Other standards may differ.

ISDN_PARAM_NT_T102

This parameter accesses the NT timer value T2. The CCITT recommendations specify that timer T2 has a value from 25 to 100 milliseconds. Other standards may differ.

ISDN_PARAM_TE_T103

This parameter accesses the TE timer value T3. The CCITT recommendations specify that timer T3 has a value from 5 to 30 seconds. Other standards may differ.

ISDN_PARAM_TE_T104

This parameter accesses the TE timer value T4. The CTS2 specifies that timer T4 is either not used or has a value from 500 to 1000 milliseconds. Other standards may differ. CTS2 requires that timer T309 be implemented if T4 is not available.

ISDN_PARAM_MAINT

This parameter sets the multi-framing mode of a BRI-TE interface. For normal operation this parameter should be set to **ISDN_PARAM_MAINT_ECHO**. Other uses of this parameter are dependent on the definition and use of the BRI interface S and Q channels.

ISDN_PARAM_ASMB
There are a few differences in the BRI-TE interface activation state machine standards. This parameter allows the selection of the appropriate standard. At this time, only **ISDN_PARAM_TE_ASMB_CCITT88** and **ISDN_PARAM_TE_ASMB_CTS2** are available.

ISDN_PARAM_PAUSE
This parameter allows a management device to pause the IO on a B-channel. **pause.channel** is set to indicate which channel is to be paused or un-paused. **pause.paused** is set to zero to un-pause and one to pause. *fd* is associated with an ISDN interface management device. *arg* is a pointer to a **struct isdn_param**.

ISDN_SET_LOOPBACK
fd is the file descriptor for an ISDN interface's management device. *arg* is a pointer to an **isdn_loopback_request_t** structure.

```
typedef enum {
        ISDN_LOOPBACK_LOCAL,
        ISDN_LOOPBACK_REMOTE,
} isdn_loopback_type_t;

typedef enum {
        ISDN_LOOPBACK_B1 =          0x1,
        ISDN_LOOPBACK_B2 =          0x2,
        ISDN_LOOPBACK_D =           0x4,
        ISDN_LOOPBACK_E_ZERO =      0x8,
        ISDN_LOOPBACK_S =           0x10,
        ISDN_LOOPBACK_Q =           0x20,
} isdn_loopback_chan_t;

typedef struct isdn_loopback_request {
        isdn_loopback_type_t        type;
        int                         channels;
} isdn_loopback_request_t;
```

An application can receive D-channel data during D-Channel loopback but cannot transmit data. The field type is the bitwise OR of at least one of the following values:

```
ISDN_LOOPBACK_B1          (0x1)     /* loopback on B1-channel */
ISDN_LOOPBACK_B2          (0x2)     /* loopback on B2-channel */
ISDN_LOOPBACK_D           (0x4)     /* loopback on D-channel */
ISDN_LOOPBACK_E_ZERO      (0x8)     /* force E-channel to Zero if */
        /* fd is for NT interface */
ISDN_LOOPBACK_S           (0x10)    /* loopback on S-channel */
ISDN_LOOPBACK_Q           (0x20)    /* loopback on Q-channel */
```

ISDN_RESET_LOOPBACK
arg is a pointer to an **isdn_loopback_request_t** structure. **ISDN_RESET_LOOPBACK** turns off the selected loopback modes.

ISDN data format

The **isdn_format_t** type is meant to be a complete description of the various data modes and rates available on an ISDN interface. Several macros are available for setting the format fields. The **isdn_format_t** structure is shown below:

```
/* ISDN channel data format */

typedef enum {
        ISDN_MODE_NOTSPEC,                  /* Not specified */
        ISDN_MODE_HDLC,                     /* HDLC framing and error */
                                            /* checking */
        ISDN_MODE_TRANSPARENT               /* Transparent mode */
} isdn_mode_t;

/* Audio encoding types (from audioio.h) */
#define AUDIO_ENCODING_NONE         (0)     /* no encoding*/
#define AUDIO_ENCODING_ULAW         (1)     /* mu-law */
#define AUDIO_ENCODING_ALAW         (2)     /* A-law */
#define AUDIO_ENCODING_LINEAR       (3)     /* Linear PCM */

typedef struct isdn_format {
        isdn_mode_t     mode;
        unsigned int    sample_rate;        /* sample frames/sec*/
        unsigned int    channels;           /* # interleaved chans */
        unsigned int    precision;          /* bits per sample */
        unsigned int    encoding;           /* data encoding */
} isdn_format_t;

/*
 * These macros set the fields pointed
 * to by the macro argument (isdn_format_t*)fp in preparation
 * for the ISDN_SET_FORMAT ioctl.
 */
ISDN_SET_FORMAT_BRI_D(fp)                   /* BRI D-channel */
ISDN_SET_FORMAT_PRI_D(fp)                   /* PRI D-channel */
ISDN_SET_FORMAT_HDLC_B64(fp)                /* BRI B-ch @ 56kbps */
ISDN_SET_FORMAT_HDLC_B56(fp)                /* BRI B-ch @ 64kbps */
ISDN_SET_FORMAT_VOICE_ULAW(fp)              /* BRI B-ch voice */
ISDN_SET_FORMAT_VOICE_ALAW(fp)              /* BRI B-ch voice */
ISDN_SET_FORMAT_BRI_H(fp)                   /* BRI H-channel */
```

ISDN Datapath Types

Every STREAMS stream that carries data to or from the ISDN serial interfaces is classified as a channel-stream datapath. A possible ISDN channel-stream datapath device name for a TE could be **/dev/isdn/0/te/b1**.

On some hardware implementations, it is possible to route the data from hardware channel to hardware channel completely within the chip or controller. This is classified as a channel-channel datapath. There does not need to be any open file descriptor for either channel in this configuration. Only when data enters the host and utilizes a STREAMS stream is this classified as an ISDN channel-stream datapath.

ISDN Management Stream

A management stream is a STREAMS stream that exists solely for control purposes and is not intended to carry data to or from the ISDN serial interfaces. A possible management device name for a TE could be **/dev/isdn/0/te/mgt**.

Channel Management IOCTLS

The following ioctls describe operations on individual channels and the connection of multiple channels.

ISDN_SET_FORMAT

fd is a data channel, the management pseudo-channel associated with the data channel, or the management channel associated with the data channel's interface or controller. *arg* is a pointer to a **struct isdn_format_req**. The **ISDN_SET_FORMAT** ioctl sets the format of an ISDN channel-stream datapath. It may be issued on both an open ISDN channel-stream datapath Stream or an ISDN Management Stream. Note that an **open**(2) call for a channel-stream datapath will fail if an **ISDN_SET_FORMAT** has never been issued after a reset, as the mode for all channel-stream datapaths is initially biased to **ISDN_MODE_NOTSPEC**. *arg* is a pointer to an ISDN format type (**isdn_format_req_t**∗).

```
typedef struct isdn_format_req {
        isdn_chan_t      channel;
        isdn_format_t    format;         /* data format */
        int              reserved[4];    /* future use - must be 0 */
} isdn_format_req_t;
```

If there is not an open channel-stream datapath for a requested channel, the default format of that channel will be set for a subsequent **open**(2).

To modify the format of an open STREAM, the driver will disconnect the hardware channel, flush the internal hardware queues, set the new default configuration, and finally reconnect the data path using the newly specified format. Upon taking effect, all state information will be reset to initial conditions, as if a channel was just opened. It is suggested that the user flush the interface as well as consult the hardware specific documentation to insure data integrity.

If a user desires to connect more than one B channel, such as an H-channel, the B-channel with the smallest offset should be specified, then the precision should be specified multiples of 8.

For an H-channel the precision value would be 16. The user should subsequently open the base B-channel. If any of the sequential B-channels are busy the open will fail, otherwise all of the B-channels that are to be used in conjunction will be marked as busy.

The returned failure codes and their descriptions are listed below:

```
EPERM       /* No permission for intented operation */
EINVAL      /* Invalid format request */
EIO         /* Set format attempt failed. */
```

ISDN_SET_CHANNEL

The **ISDN_SET_CHANNEL** ioctl sets up a data connection within an ISDN controller. The **ISDN_SET_CHANNEL ioctl** can only be issued from an ISDN management stream to establish or modify channel-channel datapaths. The ioctl parameter *arg* is a pointer to an ISDN connection request (**isdn_conn_req_t∗**). Once a data path is established, data flow is started as soon as the path endpoints become active. Upon taking effect, all state information is reset to initial conditions, as if a channel was just opened.

The **isdn_conn_req_t** structure is shown below. The five fields include the receive and transmit ISDN channels, the number of directions of the data path, as well as the data format. The reserved field must always be set to zero.

```
/* Number of directions for data flow */
typedef enum {
        ISDN_PATH_NOCHANGE = 0,/* Invalid value */
        ISDN_PATH_DISCONNECT,  /* Disconnect data path */
        ISDN_PATH_ONEWAY,      /* One way data path */
        ISDN_PATH_TWOWAY,      /* Bi-directional data path */
} isdn_path_t;


typedef struct isdn_conn_req {
        isdn_chan_t     from;
        isdn_chan_t     to;
        isdn_path_t     dir;           /* uni/bi-directional or disconnect */
        isdn_format_t   format;        /* data format */
        int             reserved[4];   /* future use - must be 0 */
} isdn_conn_req_t;
```

To specify a read-only, write-only, or read-write path, or to disconnect a path, the **dir** field should be set to **ISDN_PATH_ONEWAY**, **ISDN_PATH_TWOWAY**, and **ISDN_PATH_DISCONNECT** respectively. To modify the format of a channel-channel datapath, a user must disconnect the channel and then reconnect with the desired format.

The returned failure codes and their descriptions are listed below:

```
EPERM           /* No permission for intented operation */
EBUSY           /* Connection in use */
EINVAL          /* Invalid connection request */
EIO             /* Connection attempt failed. */
```

ISDN_GET_FORMAT

The **ISDN_GET_FORMAT** ioctl gets the ISDN data format of the channel-stream datapath described by *fd*. *arg* is a pointer to an ISDN data format request type (**isdn_format_req_t∗**). **ISDN_GET_FORMAT** can be issued on any channel to retrieve the format of any channel it owns. For example, if issued on the TE management channel, the format of any other te channel can be retrieved.

ISDN_GETCONFIG

The **ISDN_GETCONFIG** ioctl is used to get the current connection status of all ISDN channels associated with a particular management STREAM. **ISDN_GETCONFIG** also retrieves a hardware identifier and the generic interface type. *arg* is an ISDN connection table pointer (**isdn_conn_tab_t∗**). The **isdn_conn_tab_t** structure is shown below:

```
typedef struct isdn_conn_tab {
    char              name[ISDN_ID_SIZE];  /* identification string */
    isdn_interface_t  type;
    int               maxpaths;            /* size in entries of app's
                                              array */
    int               npaths;              /*
                                            * number of valid entries
                                            * returned by driver
                                            */
    isdn_conn_req_t   *paths;              /* connection table in app's
                                              memory */
} isdn_conn_tab_t;
```

The table contains a string which is the interface's unique identification string. The second element of this table contains the ISDN transmit and receive connections and configuration for all possible data paths for each type of ISDN controller hardware. Entries that are not connected will have a value of **ISDN_NO_CHAN** in the **from** and **to** fields. The number of entries will always be **ISDN_MAX_CHANS**, and can be referenced in the hardware specific implementation documentation. An **isdn_conn_tab_t** structure is allocated on a per controller basis.

SEE ALSO

ioctl(2), **poll**(2), **read**(2), **write**(2), **audio**(7I), **dbri**(7D), **streamio**(7I)

"ISDN, An Introduction", by William Stallings, Macmillian Publishing Company, ISBN 0-02-415471-7