PBX4Linux

Software based ISDN Private Branch Exchange for Linux

By Andreas Eversberg (Jolly)
(http://isdn.jolly.de)

Documentation for Version 2.5

This page is left blank.

Index

1

Introduction

1.1 What is PBX4Linux?

PBX4Linux is pure software based ISDN PBX, that connects external lines, internal telephones, and optionally voice over IP. It is designed to run with Linux only because it uses the kernel's mISDN passive driver by Karsten Keil. It can work together with OpenH323, that is a voice over IP implementation complied to the H.323 ITU-T standard.

1.2 Philosophy behind

At the end of 2001, I found out, that my ISDN card has a chipset capable of connecting telephones to it. This is called the NT-Mode. So my card can be used to transfer information between telephones and my Linux box. Normally ISDN cards transfer information between a computer and the public telephone system. But at this time there was no protocol for Linux kernel that could talk to ISDN telephones.

My first idea was to use a telephone to make voice over IP calls. Instead of messing with headsets, I wanted to just pick up the phone, get a dial tone, and dial the IP number. On the other side, I wanted to have another phone, that rings and shows me the IP number of the caller. I know, that there are IP phone already but they are more expensive than ISDN phones. So I started to expand the Linux kernel to be able to handle telephones connected to my ISDN card. The patch was written for the old 'HiSax' driver. I hated that patch because it was another dirty addition. I was glad that Karsten Keil wrote the new modular ISDN driver that gave me a well defined API and real multipoint NT-mode. I added some features to the new driver, so real time cross connections, conferences, DTMF decoding and tone generation is possible.

While writing the expansion of the kernel driver, I designed the PBX4Linux, that is now also capable of connecting calls between connected telephones and external ISDN lines connected to the public telephone system, as well as voice over IP using OpenH323 library.

I wanted to have a PBX, that provides features I am missing in standard products, like letting a call ring on my telephone at home and at the same time on my mobile via external call. The called phone, that picks up first, gets connected, the other gets released. Another idea was callback, which helps me to reduce the costs of mobile calls.

Many features followed: Deletion of digits while dialing after pick-up, an answering machine that records during the announcement to trick the caller, and conferences with no member limit but the resources of available channels.

During this book, you will learn about the features it has, how it works and, how to set up your own software based ISDN PBX.

1.3 Who wrote it?

My name is Andreas Eversberg and I live in the northern part of Germany, called 'Schleswig-Holstein'. I currently work for a telephone company, called **Kom***Tel*, which is a full service provider for telephone lines, internet and leased lines in Schleswig-Holstein. I do administration of the network elements like subscriber line cards and ADSL routers. I started writing the PBX and the kernel driver, six months before I got the job at **Kom***Tel*. I see this job as a start of managing life and getting used to "work" and of course to learn and earn money.

My email address is 'jolly@jolly.de'. My homepage is 'http://www.jolly.de'. The PBX4Linux page is 'http://isdn.jolly.de'.

1.4 How much does it cost?

It costs nothing at all. As long as you use it for non commercial purpose, I will not charge you. Using this PBX in your company to handle the daily business is not a commercial purpose in this context. See the appendix for copyright information.

1.5 Some definitions of terms in this document

- **PBX**

Private Branch Exchange (PBX), also called PABX, is a small private telephone system. It features internal calls without getting changed, because calls are routed internally. It has connectivity to the telephone network in most cases. It provides more features than the telephone network provides.

- **H.323**

H.323 is a standard for making calls via packet switched IP networks. It specifies, how voice (and call control) is transferred over IP. H.323 is defined by the International Telecommunication Union (ITU). To be able to interconnect two persons using H.323, both must have an IP connection. If one party is reachable via public telephone network, PBX4Linux can be used as H.323 gateway to transfer telephone calls from and to IP networks.

- **Extension**

Extensions are telephones connected to a PBX. PBX4Linux supports individual configuration of extension. Each extension is defined by a number. Many telephones can have the same extension's number. The extension is identified by the outgoing caller ID of the telephone. On incoming calls, the internal port(s) is/are defined by the extension's configuration, where to route the call to.

- **Internal port**

A telephone is connected directly via ISDN card to the PBX. The card runs in NT-mode, that is simulating an NT1 with hard and software layers.

- **External line**

An external telephone line (from a local telecommunication provider) is connected to the PBX. Multiple external lines may be used for more channels.

- **a-law / mu-law**

Audio samples are not directly transmitted via B-channel. First a 12 bit sample is sampled from the audio input and then converted into 8 bits. This is done by reducing the dynamics for high amplitude samples, to increase the number of samples available for low amplitude. There are two different 'laws', describing the amplitude ranges. The conversion can easily be done by a table of 256 entries (to decode) and 65536 samples (to encode from a 16 bit sample). In Europe, we use "a-law". In America they use "mu-law". The mu-law also allows to drop the lowest significant bit. This is used in America for telephone exchanges that provides only 56 bits. To read more about the coding, refer to any "how-ISDN-works" manual.

- **PRI / BRI / G.703 / s2m / S/T / S0 / PMX / up0 / u2m / uk0 / HDSL / SDSL**

There are two different types of interfaces:

- Primary Rate Interfaces
- Basic Rate Interfaces

The Primary Rate Interface has one 64 bit D-channel and 30 B-channel (channel 0 is used for synchronization). The electric interface is also called "s2m" or "PRI" and is defined in the ITU-Standard "G.703". The bit rate is 2048Kbits/s. The electrical version of a "G.703" interface is limited to several meters of cable length, so a modem must be used. "HDSL" and "SDSL" provide transmission via none insulated twisted pair copper wires over some kilometers. An older standard is called "u2m". PRI interfaces are also called PMX (Primary Multiplex).

The Basic Rate Interface has one 16 bit D-channel and two B-channel. The electrical interface is also called “S/T” or “S0”. It supports point to multipoint connectivity. Also this interface has only a limited range of cable length, so “uk0” or “up0” is used to transfer the signal over several kilometers.

- **B-channel / D-channel**

An ISDN interface has several types of channels. The D-channel is used to transfer signaling information (data channel). The audio data in transferred vi B-channel (binary channel). Also any data transmission like fax or internet is transferred via B-channel. Some interfaces have overhead, monitor and echo channels. These are only relevant between interfaces and do not transfer information between subscribers.

2

Hardware Requirements

2.1 ISDN cards

The first thing needed for the PBX, is of cause at least one ISDN card. At least two cards are required to interconnect internal telephones with external telephone lines. Any card can be used that is supported by the kernel's **mISDN driver** (the next generation passive ISDN driver by Karsten Keil):

- HFC-S PCI based cards (tested)
- Fritzcard PCI (tested)
- Sedlbaur FAX
- Winbond
- HFC-4s / HFC-8s based cards (tested)
- HFC-E1 based cards (tested)

Please mail your experience with untested cards, when using the kernel's mISDN driver.

In order to connect telephones directly to an ISDN card, it must support **NT-Mode**. Then it is possible to use it as internal ISDN port. There currently there are these type of chip sets, that support NT-Mode hardware layer:

- HFC-S PCI based cards
- HFC-4s / HFC-8s chip sets
- HFC-E1

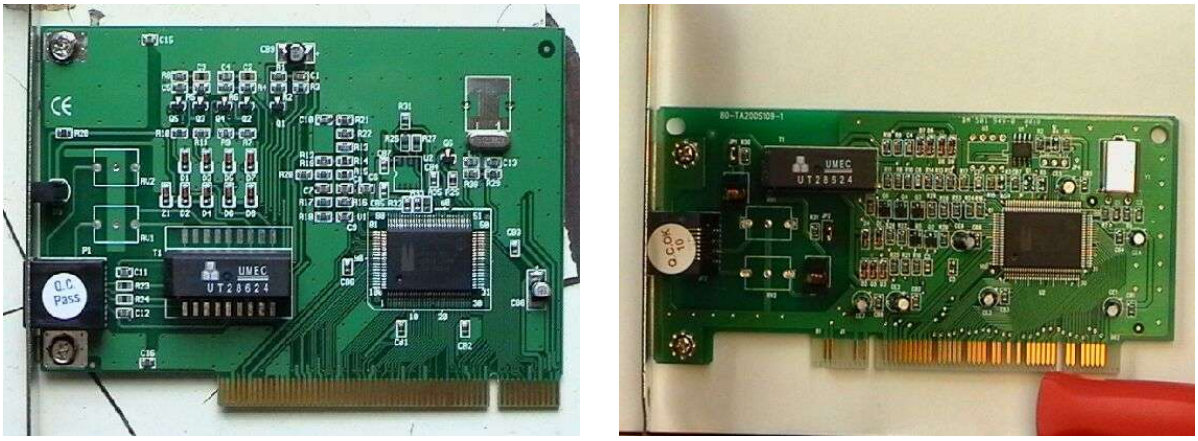
All of these chips have a small ‘Dom’ of Cologne on the top:

Cologne Chip

The following cards **do** have the required chip set capable of NT-Mode:

- Creatix ISDN-S0/PCI
- Trust PCI-Modem
- Acer ISDN 128 Surf PCI
- D-Link DMI-128I+
- Billion/Asuscom (Asuscom/Askey)
- HFC cards from “Conrad Elektronik” (not available anymore)
- Neolec FREEWAY ISDN^{PCI}
- ISDN 128Kbs TA Card (TAS106H)
- Junghanns Asterisk boards (HFC-4S / 8S / E1) www.junghanns.net
- PBX4Linux boards (HFC-4S / 8S / E1)
- Scitel cards

Please mail me other cards you know, that have an NT-Mode capable chip sets. The card will look like:



(Figure: Cards with HFC-PCI, thanx to Ulrich for the pics)

2.2 Connect ISDN telephones to your ISDN card.

Normally cards run in **TE-Mode**, which means ‘**Terminal Equipment’**. Terminal equipment is e.g. an ISDN telephone, an ISDN card, a fax machine, or any other terminal to places calls, dial into the internet, or send and receive faxes. The **NT** is the small box where all ISDN

terminals are connected, which means '**Network Termination'**. It is the small little box also known as 'NTBA' (Germany) or 'NT1' (everywhere else:).

What must be done, to connect telephones to an ISDN card? Don't be shocked when you read the list. It is much easier, as you will find out later in the text. Here is a list of things to do:

- Of course, the card must support NT-Mode hardware layer.
- A driver capable of NT-mode must be installed. (part of mISDN)
- The ISDN telephone must be connected cross-over to the card.
- Power must be supplied to the ISDN bus, in case the telephone(s) has/have no own power supply.
- The ISDN bus must be terminated with resistors of 100 Ohm. (better 50 Ohm)

In order to bring a HFC card into NT-mode, the following module parameter is used:

```
$ modprobe hfcpci protocol=0x12 layermask=3
```

But this will be explained later…

It is **not** possible to use just a cross-over cable for Ethernet. ISDN is differently connected than Ethernet. Additionally terminals and cards have no termination. ISDN cross connection is done by connecting the inner wires (pin 4 and 5) with the wires around them (pin 3 and 6). Pins 1, 2, 7 and 8 are not used. Some special PBX telephones use these pins for extra power supply. Termination is essential, even if the ISDN cable would have almost zero length. All this can be done by using an old (even broken) NT1 (NTBA). Telephone companies throw them away when they are broken. Most times the power supply still works. If you measure about 40 volts between pin 3 and 4, as well as ping 5 and 6, you know that your NT1 is good enough for using it as a power supply. Be sure that the terminator switches inside are turned on.

An NT1 (NTBA) has three types of connectors. The first connects to the underground wire that is connected to the telephone company. The second connects to the ISDN telephones. The third is connected to the power line. Follow the steps to get an NT1 connected to an ISDN card instead of the telephone network:

The VERY SIMPLE way:

Take an ISDN or Ethernet cable and cut it in the middle. Reconnect the inner pins (4 and 5) of one end with the pins around them (3 and 6) of the other end. Do it vice versa. (Connect 3 with 4, 4 with 3, 5 with 6, 6 with 5) Now you have an ISDN cross over cable, that is different to an Ethernet cable. Just connect the ISDN card with the cross over to one plug of your NT1, and a telephone with a normal cable (not crossed) to the other plug. Connect the power line of your NT1 or telephone and **you are done**. You will be able to connect only one telephone, because both plugs are used. Don't use the cross-over cable to directly connect your card to your telephone, unless your card or your cable has termination and your telephone has own power supply.

The COMFORTABLE way:

Step 1: Open the small door to get access to the wire clips. If it has no door, just take a screw driver and open the case. Be aware of high voltage at the switched power supply, even if it is

not connected to the power line. Capacitors may hold high voltage for a long time. Inside are 4 clips to connect an ISDN cable directly to it, rather than connecting it to one of the RJ45 plugs. Each NT on the picture has six yellow clips: One pair (to the left) are used to connect the U-Bus (underground wire), the two pairs (to the right) connect to the S-Bus (telephone). The S-Bus is directly connected in parallel to the two rj45 plugs.

(Figure: NT1 with door open)

Step 2: Find out which clip inside the NT1 is connected to which pin on the rj45 plug. Therefore connect an ISDN cable to one of the two ISDN jacks. Take an Ohm-Meter and find out which clip is connected to which pin of the ISDN cable. Use a small peace of wire, to help the meter's probe get into the hole of the clip. Pin 4 and 5 as well as 3 and 6 of the ISDN cable, is connected through the coil inside the NT1. The coil has low resistance, so your Ohm-Meter will show that they are connected. So you cannot determine between pin 4 and 5 as well as between pin 3 and 6. There are two clips connected to pin 4 and 5 as well as another two to clips connected to 3 and 6. It doesn't matter which clip is connected to pin 4 and which is connected to pin 5, since the **polarity doesn't matter**. (Pin 3 and 6 respectively)

Most NT1 in Europe name the 4 pins: A1 and B1, A2 and B2. Do the following connections:

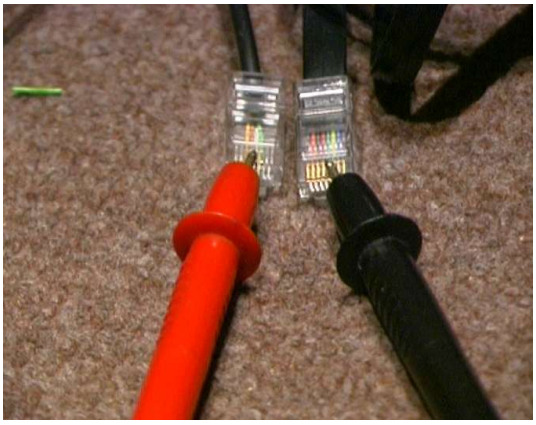
- A1 -> pin 3
- B1 -> pin 6
- A2 -> pin 4
- B2 -> pin 5


(Figure: Find clips)

Step 3: Take another ISDN cable or Ethernet cable and cut off one end. Take the inner pair of the cable (pin 4 and 5) and connect them to the clips, that are connected to the outer pins of the ISDN jacks (pin 3 and 6). Do this with the inner pins of the cable respectively. Again, polarity of a cable pair doesn't matter for the ISDN bus.


(Figure: cable connected to the clips)

Step 4: Use the Ohm-Meter again and verify the connection between the cable, that is connected to the clips, and the cable connected to one of the jacks. Pin 4 and 5 should now connect to pin 3 and 6 and vice versa.

(Figure: Verify cross connection)

Step 5: Connect the cable, which is connected to the clips, to the ISDN card, that should run in NT-Mode. Be sure that termination is switched on. Inside the NT1 are two switches, that must be ON. This is the default.

(Figure: cable connected to the ISDN card)

Step 6: Connect an ISDN telephone to the ISDN cable, that is connected to the ISDN jack. Also connect the power cable.

(Figure: complete setup)

Since there is nothing connected to the U-Bus (underground wire), the internal electronics is disabled, because it gets its power from the U-Bus (not from the power line). Only the power supply is connected to the coils (transformer) inside the NT1 providing power for telephones, as well as termination. This is why the NT1 can be broken. Only power supply must work.

NOTE: If you like to connect more than one telephone, you might experience clicks and sound problems. Use 50 ohm instead of 100 ohm termination. This can be done by adding another 100 ohm resistors in parallel with the ones in the NT1. Two 100 ohm resistors become 50 ohm! One resistor must be connected between pin 3 and 6, and the other between 4 and 5. You may figure out yourself how to connect the cable plus the resistors. Maybe you use a soldering iron.

2.3 Interrupt problems

One serious problem of installing HFC cards are interrupts. Some PCI slots share the same IRQ. Be sure that every HFC card has its individual IRQ. When booting the PC, the IRQ is shown of each PCI card installed. If two cards share the same IRQ the HFC-PCI driver may fail. I experienced this as I installed three cards in one PC.

Another problem might occur with older PC main boards using Fritz cards and IDE hard drives. The interrupts of the IDE controller causes loss of frames coming from the ISDN cards. This results in gaps of the audio during hard drive operation. I use a Fritz-Card, which causes losses of interrupts during IDE hard driver access. The HFC-PCI card works fine. This problem doesn't occur with my PCI Adaptec SCSI controller in the same machine.

If anyone has experience with certain ISDN cards in combination with frame drops, let me know.

2.4 Hard disk

The size of a hard drive doesn't matter, as long as no audio recording is done. Since the PBX has a call recording feature (and answering machine), smaller hard drives get filled quickly during long calls. A recoding with 8 bit mono causes a recorded file of 8 kilobytes per second or 8000 bytes. The best quality of recoding is 16 bit stereo. This causes 32 kilobytes to be written down during one second. If the call lasts one hour, PBX4Linux would record a file of 115.2 megabytes or 115200000 bytes. Even with 8 bits mono (or law codec) is used, the recording would have still a size of 28.8 megabytes or 28800000 bytes. If the PBX would support the LPC-10 codec to record audio, the size would be 1.44 megabytes for recording one hour. A year of audio data could be compressed down to about 12.6 gigabytes. In this case, don't worry about size of your hard drive anymore. Any volunteers are welcome to implement the LPC-10 codec… Note that the call recording is just a feature that is not turned on by default.

There are no requirements for hard drive's speed, since the load will be 800-1600 kilobytes per seconds for 100 calls at a time. Hard drives are much faster these days.

2.5 Memory

Almost every Linux box uses swap memory. PBX4Linux disables swapping of it's code and data memory. Other programs still use swap. If the PBX allocates new memory, that is not available, the memory of other processes must be written to disk first. This causes a high delay of call processing, which also causes audio jitter. Also if other programs on the Linux box are swapped (like the telnet daemon), the PBX will stop for a short period of time, caused

by hard disk traffic. To reduce the risk, you may turn off swap memory when running the PBX. Telephone calls are a real-time task, that don't like delays of any undefined time. Therefore increase physical memory, so the Linux box would run without swapping memory.

It depends on what's running on the Linux box. Just boot your Linux, and after starting PBX look how much of memory is left. Turn off swap, and check what's left:

```
$ swapoff -a

$ free
             total       used       free     shared    buffers     cached
Mem:         52376      50364       2012          0        868       6064
-/+ buffers/cache:      43432       8944
Swap:            0          0          0
```

In this example from my Linux box, there is about 2 megabytes of free memory. Note that the cached memory of about 6 megabytes will be available any time. The cached memory holds recent data from hard drive, that will be accessed faster when accessed again. In this case there are 8944 kilobytes (almost 9 megabytes) of memory still available without swapping. After a while, the machine without swap may run out of memory. I experience this sometimes, because I have many other processes running, like web server, PPPoE and proxy.

Note that a PBX without using OpenH323 would have much less memory consumption, as with OpenH323. You better have more than 64 megabytes of memory. But today, memory is no matter of price anymore. It is much better to leave swap turned on, have much memory, and no other programs running (except for some small tools, like ftp, telnet and ssh daemons).

2.6 CPU

I have no experience with CPU usage. Making an external call, the CPU usage (AMD K6-200) is between 0.5% and 3.5%. On one H.323 call to an internal telephone, it is about 10%. If anyone has more experiences, please let me know. I recommend at least 200 megahertz for the PBX.

2.6 PCM

Almost all ISDN controllers support PCM bus to interconnect them. The HFC-4S / HFC-8S / HFC-E1 support a PCM bus of 128 timeslots with two banks. It is possible to connect up to 128 callers on one bus. This makes large PBXs possible, where 128 calls can be made without CPU load for audio transfer. The driver automatically detects these cards and does hardware crossconnections and conferencing. There are boards available for PBX4Linux. Mail to isdn@jolly.de for more infos.

3

Download & Installation

3.1. Install mISDN driver

Why **mISDN** driver and not HiSax driver:

- The ISDN driver of the kernel (HiSax) is not able to handle the ISDN protocol for connecting telephones to it (**NT-mode**). mISDN provides a user space library which supports NT-mode stacks. The library "i4lnet" is part of the "mISDNuser" package, but not included in the kernel source.
- The ISDN4Linux is not able to link certain ISDN layers with user space programs. mISDN provides direct access to any layer. Each protocol layer is implemented in modules and can be loaded as required.
- Also b-channels have individual stacks with layers, that can be accessed from user space.
- Linux is not a real-time operating system. A PBX application in user space will have a delay and interruptions, when cross-connecting audio streams. If, for example, a call is made from an internal port to an external ISDN line, the b-channels must be cross-connected. To keep the delay as low as possible, a **dsp-module** is included in the mISDN kernel source. It was programmed for the PBX4Linux but can be used in other applications. It will not only cross-connect, it is also able to make conferences with an unlimited number of b-channels (if available), volume control for incoming and outgoing audio data, real time tone generation, DTMF decoding, and it will do it in hardware if supported.

You may choose a **binary version** or install the source as described here. In case of a binary release, you need only one tar archive to download. It will have all modules, libraries and files needed for PBX4Linux. It is compiled for Intel 586 IBM compatible PC with Linux 2.6.0 or higher. To install the binary archiv, download it, untar and install:

```
$ tar xvzf pbx4linux-binary-xxxx.tar.gz

$ cd pbx-binary

$ make install
```

There executables will be installed at "/usr/local/bin/". All support files will be installed at "/usr/local/pbx". Also the kernel modules will be installed at "/usr/local/pbx/modules". To load these modules, you need to specify the path when using "genrc". "genrc" is described later and is used to load and unload kernel modules.

Alternatively the **source version** can be installed. In order to run the PBX, the kernel with **mISDN** is needed, as well as the "**mISDNuser**" package. Download the latest sources from 'http://isdn.jolly.de/', the following packages are needed:

- mISDN_xxxx.tar.gz
- mISDNuser_xxxx.tar.gz

You may use an existing kernel and patch it with mISDN driver or install a complete new kernel. For a new kernel remove the old link "/usr/src/linux". If there is no link, rename or move the "linux" directory, that may be located under "/usr/src". Otherwise you might overwrite your current kernel. Unpack the source:

```
$ cd /usr/src/

$ tar -xvzf linux-x.x.xx_mISDN.tar.gz

$ ln -s linux-xx.xx.xx_mISDN linux

$ cd linux
```

Instead of downloading the complete patched kernel, you may also download the patch of the mISDN driver. Install as shown:

```
$ cd mISDN

$ ./std2kern [-h]
```

This tool will copy the mISDN source from the current directory to the kernel source. Give "h" option to get a list of options. By default the driver will be installed in "/usr/src/linux/" .

Then configure your Kernel:

```
$ cd /usr/src/linux

$ make mrproper

$ make menuconfig
```

You may copy the ".config" file from your previous kernel source directory, to keep your setting. Copy the file **after** "`make mrproper`" and **before** "`make menuconfig`":

```
$ cp ../linux-your.old.versrion/.config .
```

WARNING: Don't just copy the config file, if the kernel version changes too much (e.g. 2.4 to 2.6). Always check the kernel features. Features may have different names in the config file, so they will be disabled in newer versions.

Enable the following kernel features, using "`make menuconfig`":

- ➔ **Enter ISDN subsystem --->**
- **ISDN Support (say 'm')**
- **CAPI2.0 support (say 'y' or say 'm')**
- ➔ **Modular ISDN driver --->**
- **Enable memory leak debug for mISDN (say 'n')**
- **Support modular ISDN driver (MUST say 'm')**
- **Support for AVM Fritz!Cards (say 'y' if installed)**
- **Support for HFC PCI cards (say 'y' if installed)**
- **<other cards…>**
- **Support for digital audio processor (say 'y')**

A card, that supports NT-Mode, you may want to use. Otherwise you will not be able to connect telephones. If your card is an HFC-S PCI, say yes to the following:

- **HFC PCI cards**
- **HFC-4S cards (4 S/T interfaces)**
- **HFC-8S cards (8 S/T interfaces)**
- **HFC-E1 cards (1 E1 interfaces)**

Also say yes to any card you may want to use for external line.

Save and exit kernel configuration, and continue to build the kernel source:

```
$ make dep (not needed for kernel 2.6)

$ make bzlilo

$ make modules

$ make modules_install
```

If everything compiles without errors, **Reboot**! Warnings about unused variables and wrong prototypes can be ignored of course. After reboot, run "**depmod**", in order to create dependencies, so the ISDN modules will be found. In newer kernels the "depmod" is done when installing the modules.

To be able to use the "/dev/mISDN" device, it must be created:

```
$ mknod /dev/mISDN c 46 0
```

It is a character device with major number 46 and minor 0. But before the device exists, the modules must be loaded. Read on, on how to create an rc-script to load and unload the modules.

3.2. mISDNuser

In order to access mISDN device driver and use the NT-mode, install **mISDNuser** package.

```
$ tar -xvzf mISDNuser-x.xx.tar.gz
```

```
$ cd mISDNuser

$ make
```

Two libraries will be created:

- **mISDNuser/lib/libmISDN.a**
- **mISDNuser/i4lnet/libisdnnet.a**

Also some other utils will be part of the package. Just ignore them. Be sure to uncompress mISDNuser the same source directory where you will uncompress the PBX. The paths to the libraries are relative in the Makefile of the PBX. If you choose different locations, you must edit the Makefile.

The **libmISDN.a** provides direct access to the mISDN device. It will be used to communicated directly with the protocol stacks of mISDN.

The **libisdnnet.a** provides NT-mode protocol. The TE-mode is already part of the kernel.

3.3. PBX4Linux

Download the PBX4Linux at http://isdn.jolly.de/download. Uncompress it:

```
$ tar -xvzf pbx4linux_xxxx.tar.gz

$ cd pbx4linux
```

Edit the “Makefile”. At the line "WITH-H323", write “#” in front of it to comment it out. You will need it later when you like to add H.323 support. But first get the PBX running without H.323.

Compile and install the PBX:

```
$ make

$ make install
```

Configuration files, documents and data files will be installed by default at: “/usr/local/pbx”. Edit the “Makefile” in order to choose a different location. Don't worry about your old configuration files, you might already have from older version. They will only be copied, if they don't exist already. If they don’t exist, default configuration files will be installed. The binaries are installed at “/usr/local/bin” by default.

To see a list of start options of PBX4Linux, just enter:

```
$ pbx
```

To run PBX4Linux as background task, enter:

```
$ pbx fork
```

A daemon fork will be done, so closing the shell will not interrupt the PBX. Also use this for startup, if the PBX should be started at boot time.

Note: In order to run PBX4Linux, the mISDN driver must be loaded. Follow the next chapter on how to configure the driver, and creating a start and stop script.

3.4 OpenH323

First visit the OpenH323 project homepage at 'http://www.openh323.org'. OpenH323 comes with two packages:

- pwlib
- openh323

OpenH323 consists of object orientated classes to build H.323 connections. It must have "pwlib" installed in order to compile and run. Be sure to get the latest version from the download area, even if you have them already installed with your distribution. Follow the installation instruction for each package. Here is a short instruction on what to do. If you have trouble, get help from the OpenH323 page. In this instruction we use "/usr/src/" to install all packages to. Be sure that the source tree of PBX4Linux is also located in "/usr/src/". The PBX uses relative links to the source tree of OpenH323.

NOTE: You may skip this part if you do **not** want to use the PBX with H.323 support. Try the PBX without H.323 support first, and do not experience all problems at once.

If you have H.323 already installed with your distribution, you **must** change the following variables in the "Makefile":

```
H323_INCLUDE = -I../openh323/include
H323_LIB = -L../openh323/lib
PWLIB_INCLUDE = -I../pwlib/include/ptlib/unix -I../pwlib/include
PWLIB_LIB = -L../pwlib/lib
```

They must point to the correct location, where "pwlib" and OpenH323 is installed. Otherwise you might get compile errors.

Uncompress "pwlib". Use the same directory where you uncompressed the PBX source:

```
$ cd /usr/src

$ tar -xvzf pwlib.tar.gz
```

Compile "pwlib":

```
$ cd pwlib

$ make both
```

This will take a while, so drink a cup of coffee or visit your children again ☺.

If no error occurred, proceed with installation:

```
$ make install

$ ldconfig
```

```
$ cd ..
```

Uncompress “openh323”. Use the same directory where you uncompressed “pwlib” and the PBX source:

```
$ cd /usr/src

$ tar -xvzf openh323*.tar.gz

$ cd openh323
```

Compile:

```
$ make opt
```

Take another cup of coffee, since this will take really longer. Be sure to have allot of memory, in order to handle all the include files during compilation. Use a minimum of 128k swap space, or you might run out of memory. It helps to have lots of system memory, to increase compile speed. If you have less than 64k, it may take a whole day, caused by massive swapping. If you have trouble, read “ReadMe.txt” and follow the installation instruction there.

If no error occurred, proceed with installation:

```
$ make install

$ ldconfig

$ cd ..
```

In order to compile the PBX with H.323 support, edit the file "main.h", and uncomment the "WITH-H323" define. Remove the ‘#’ in front, and the PBX will compile using OpenH323:

```
$ make

$ make install
```

The following error may occur during compilation, if the paths to the include files of “pwlib” are not correctly set:

```
…
                 from /usr/local/include/ptlib/timer.h:355,
                 from /usr/local/include/ptlib/timer.h:355,
                 from /usr/local/include/ptlib/timer.h:355,
                 from /usr/local/include/ptlib/timer.h:355,
                 from /usr/local/include/ptlib/timer.h:355,
                 from /usr/local/include/ptlib/timer.h:355,
                 from /usr/local/include/ptlib.h:154,
                 from main.h:52,
                 from main.c:26:
/usr/local/include/ptlib/timer.h:147: macro or `#include' recursion too deep
/usr/local/include/ptlib/timer.h:355: macro or `#include' recursion too deep
make: *** [main.o] Unterbrechung
```

In this case, the include variables in the “Makefile” do not point to the correct location. The file “timer.h”, that should be found in the UNIX specific include directory (`pwlib/include/ptlib/unix`), is included from the file “timer.h”, that is located in the main “ptlib” directory (`pwlib/include`). In this example above, the compiler finds the wrong file,

becaue the UNIX specific include directory is not given or doesn't exist. This causes a recursion in this case, because the compiler doesn't know about the UNIX specific directory. It is necessary to have the correct order specified in the "Makefile":

```
PWLIB_INCLUDE = -I/path/to/pwlib/include/ptlib/unix -I/path/to/pwlib/include
```

4

Configuration

4.1. Start and stop mISDN

The following example will assume that your have a HFC-PCI card and a Fritz-Card installed. The Fritz-Card is connected to a normal multipoint ISDN line. Multipoint lines are used in most places. It is possible to connect up to eight telephones to it; the PBX may be connected parallel with other telephones. The HFC-PCI card is connected to a telephone, as described above. With this setup, your PBX will be able to forward calls between external lines and internal telephones.

Many things must be done to setup ISDN4Linux. Therefore I wrote a small tool, that creates a shell scrip to start and stop kernel's mISDN driver. It comes with the PBX4linux package and will be installed with it. Run it with:

```
$ genrc

This program generates a script, which is used to start/stop/restart mISDN
driver. All configuration of cards is done for using with the PBX.

Enter mode of your ISDN card #1. Should it run in NT-mode (for internal
phones), or in TE-mode (for external lines)? If you do not like to add more
cards, say 'done'.
[nt | te | done]: te
```

The 'te' keyword tells 'genrc', that the first card (Fritz-Card) should run in TE-Mode. It will be connected to an ISDN line. Therefore the card must be configured as a "terminal endpoint".

```
Is your card #1 connected to point-to-multipoint line, which supports
multiple telephones on one line (Mehrgeräteanschluss) OR is it a point-to-
point line which is used for PBX and supports extension dialing
(Anlagenanschluss)?
[ptp | ptm]: ptm
```

The type of line we will use, is multipoint. This is the standard type of ISDN lines all over the world.

```
Select driver of ISDN card #1.
```

```
 (1) HFC PCI (Cologne Chip)
 (2) AVM Fritz PCI
 (3) Seldbaur FAX
 (4) Winbond 6692 PCI

Select card number[1-n]: 2
```

Enter ‘2’ to select Fritz-Card. If your card is not in that list, there is no driver for it or ‘genrc’ may be out of date.

```
Summary: Card #1 of type AVM Fritz PCI will run in TE-mode and point-to-
multipoint-mode.

Enter mode of your ISDN card #2. Should it run in NT-mode (for internal
phones), or in TE-mode (for external lines)? If you do not like to add more
cards, say 'done'.
[nt | te | done]: nt
```

The next card we use is the HFC-PCI running in NT-Mode. We want to connect a telephone to it. NT means “network terminator”. In this case our Linux box is the network where the telephone is connected to.

```
Select driver of ISDN card #2.

 (1) HFC PCI (Cologne Chip)

Your card will run in NT-mode. The shown cards are capable of providing
hardware layer for NT-mode.
Select card number[1-n]: 1
```

We are not asked for multipoint, because only multipoint mode is supported for NT-mode. The selected card is HFC-PCI.

```
Summary: Card #2 of type HFC PCI (Cologne Chip) will run in NT-mode and
point-to-multipoint-mode.

Enter mode of your ISDN card #3. Should it run in NT-mode (for internal
phones), or in TE-mode (for external lines)? If you do not like to add more
cards, say 'done'.
[nt | te | done]: done
```

There is no third card, so we use the keyword ‘done’. Now we are asked to enter dsp-module’s options and debugging values. Just enter ‘0’ for every value.

```
Enter options of mISDN_dsp module. For a-LAW, just enter 0.
For u-LAW enter 1.
[0..n | 0xn]: 0

Enter debugging flags mISDN core. For no debug, just enter 0.
[0..n | 0xn]: 0

Enter debugging flags mISDN core. For no debug, just enter 0.
[0..n | 0xn]: 0

Enter debugging flags of cards. For no debug, just enter 0.
[0..n | 0xn]: 0

Enter l1 debugging flags of driver. For no debug, just enter 0.
[0..n | 0xn]: 0

Enter l2 debugging flags of driver. For no debug, just enter 0.
[0..n | 0xn]: 0
```

```
Enter l3 debugging flags of driver. For no debug, just enter 0.
[0..n | 0xn]: 0

Enter dsp debugging flags of driver. For no debug, just enter 0.
[0..n | 0xn]: 0
```

Debugging is explained later in this book. Just enter ‘0’ for the first time. If you experience crashes, read on, on how to use these values.

```
Enter location of the mISDN modules. Enter '0' for kernel's default
location. Enter '1' for binary distribution's location
'/usr/local/pbx/modules' or enter full path to the modules dir.
[0 | 1 | <path>]: 0
```

Depending on the location of the modules, give '0' or '1' here. '0' for default location (source distribution) or '1' for location of the binary distribution.

```
File 'mISDN' is written to the current directory.
$
```

Now the following shell script is written in the current directory with the name “mISDN”:

```
# rc script for mISDN driver

case "$1" in
        start|--start)
                modprobe mISDN_core debug=0x0
                modprobe mISDN_dsp debug=0x0
                modprobe mISDN_l1 debug=0x0
                modprobe mISDN_l2 debug=0x0
                modprobe l3udss1 debug=0x0
                modprobe hfcpci protocol=0x12 layermask=0x3 debug=0x0
                modprobe fritzpci protocol=0x2 layermask=0xf debug=0x0

                sleep 1
                ;;

        stop|--stop)
                rmmod hfcpci
                rmmod fritzpci
                rmmod mISDN_isar
                rmmod l3udss1
                rmmod mISDN_l2
                rmmod mISDN_l1
                rmmod mISDN_core
                ;;

        restart|--restart)
                sh $0 stop
                sleep 2 # some phones will release tei when layer 1 is down
                sh $0 start
                ;;

        help|--help)
                echo "Usage: $0 {start|stop|restart|help}"
                exit 0
                ;;

        *)
                echo "Usage: $0 {start|stop|restart|help}"
                exit 2
                ;;
```

```
esac
```

To load ISDN4Linux, just enter:

```
$ sh mISDN start
```

To unload the modules just enter:

```
$ sh mISDN stop
```

I will now explain what the script does, if the start option is given:

"`modprobe mISDN_core debug=0x0`" will load the kernel mISDN driver core.

"`modprobe mISDN_dsp debug=0x0`" will load the real time audio processor.

"`modprobe mISDN_l1 debug=0x0`" will load the layer-1 protocol for TE-mode.

"`modprobe mISDN_l2 debug=0x0`" will load the layer-2 protocol for TE-mode.

"`modprobe l3udss1 debug=0x0`" will load the layer-3 protocol for TE-mode.

"`modprobe hfcpci protocol=0x12 layermask=0x3 debug=0x0`" will load the card driver for all HFC cards. Protocol type of 0x12 means NT-mode (0x10) and DSS1 (0x02). Layermask of 0x3 means layer-0 (bit 0, card) and layer-1 (bit 1). Layer-2 and layer-3 are realized via libisdnnet library.

"`modprobe fritzpci protocol=0x2 layermask=0xf debug=0x0`" will load the card driver for all Fritz cards. Protocol type of 0x2 means TE-mode (0x00) and DSS1 (0x02). Layermask of 0x7 means layer-0 (bit 0, card), layer-1 (bit 1), layer-2 (bit 2) and layer-3 (bit 3).

"`sleep 1`" will wait some time until all cards are registered.

Look into the system log. You will see some information while loading the modules. Also errors will show up there. In order to see the current configuration, start the PBX with the query option:

```
$ pbx query

** PBX4Linux **  Version X.x (XXX 200x)

Port  1: NT-mode BRI S/T interface port (for phones)

Port  2: TE-mode BRI S/T interface line (for phone lines)

$
```

Connect an ISDN telephone as described above, and pick up the phone. You will not get a dial tone from the kernel. In order to get a dial tone, the application PBX4Linux must be running. But before running the PBX, you must configure interfaces and "extensions".

4.2 PBX lines

A PBX line, also known as point-to-point line, is only capable to connect one PBX (or a special telephone) to it. But it provides dialing of extensions. The PBX, connected to this type of ISDN line, is able receive incoming calls, after the first digit behind the main number is dialed. It is controlled by the PBX, when the number is complete. It depends on the provider, how much digits can be dialed after the main number. The international convention of the maximum length of a phone number is 13 digits. If the county code has two digits (49 in Germany) and the area code has three digits (212 for the city Solingen), and the main number has six digits, then there are two more digits that will be routed to the PBX line. Of course the PBX can define by itself when the number is complete. Inside Germany, more digits are allowed. The 'German Telekom' for example forwards up to 17 digits. The company, I work for, forwards up to 24 digits after the area code. The PBX will signal, when the number is complete. This is supported by the PBX4Linux.

Another advantage of a PBX line is the ability to have more than one ISDN line for the same main number. One ISDN line can handle two calls only at a time. The number of simultaneous calls can be increased by heaving more than one line. PBX4Linux also supports this. There is no know limit, except for the number of cards, that fit into your Linux box.

Note that PBX lines might get monitored by the telephone company, because they are more expensive and more important than normal ISDN lines. In this case, talk to your telephone company to ignore alarms because every reboot, crash or ever restarting of the mISDN driver will cause alarms.

To tell mISDN to use a PBX line, you must set the card into point-to-point mode. This will automatically done when using "genrc". The protocol for the card driver would be 0x22. (0x20 for PTP-mode, 0x02 for DSS1)

```
$ modproble fritzpci protocol=0x22 layermask=0xf debug=0x0
```

This will be included in the "mISDN" shell script, if the option "ptp" is selected when running the "genrc" tool.

PBX lines may also be multipoint. You will be able to connect normal telephones to it. In cast of PBX4Linux it makes no difference. If telephones would be connected parallel to the PBX line, dialing of extension digits will not work when one phone on the bus is ringing.

Another type of PBX line is the PMX line or "primary rate multiplex" line. This type of line provides 30 B-channels in Europe and 22 B-channels in the USA. It is used for really big PBXs. It is also possible to cascade multiple PMX lines to one PBX. I already implemented a driver, to support the European PMX chipset "HFC-E1".

4.3 CLIP No Screening

"CLIP No Screening" or "No Screening CLIP" is a special agreement with your telephone company. Normally caller IDs sent on outgoing calls, are checked against the numbers that are assigned to your ISDN line. If you send a caller ID, for which you have a number assigned, the telephone company will forward it to the called user. If the caller ID you specify doesn't match with any of your assigned numbers, the main number assigned to you, is used. So it is

not possible to fake caller IDs, and pretend that you are your ex-girlfriends new boyfriend, or the boss of your neighbor. But sometimes you may want to pretend that you are the boss of your neighbor, especially when she calls your PBX, and it forwards the call to your mobile telephone using a second line. On the second line to your mobile, the caller ID of the caller of the first line (the boss) is used. You will see when your mobile telephone rings, that the boss calls and not just your stupid main number of your telephone line.

Another way to use this, is when you connect two PBXs. Lets assume you make an H.323 call to another PBX, that transfers the call for you to the public telephone system. In this case you want to send the caller ID of the PBX, you are calling from, and not of the other PBX, that transfers the call.

There are lots of things to use it for. Some of them are not very nice, but most of them are quite funny. I have this feature on my PBX line, for the reasons described above. Ask your local telephone company for the agreement. Note, that this feature is part of the basic ISDN standard ITU-T Q.931, so it is conform, to make this feature available to you. If your telephone company tells you, that this feature is not legal, tell them about the standard.

The caller ID, sent on a normal ISDN line, is normally the local telephone number of type “unknown”. The telephone company will add the national and international prefix in front. When using “CLIP No Screening”, the caller ID must be sent by the PBX including all prefixes. There are three common types of caller IDs that can be sent: Type “subscriber” will just send the caller ID without any prefix. The called party that receives this caller ID will just see that number, as you specified, everywhere in the world. Type “national” must be sent with area code and number. The called party’s Telephone will automatically add the national long distance prefix in front, so it must be omitted. Type “international” must be sent with country code, area code and number. The called party’s Telephone will automatically add the international access code in front, so it must be omitted. Because PBX doesn’t know what access code the called party must dial, it just sends the type “international”, and the prefix is added by the called party’s telephone.

It the connected ID is transmitted without checking, it is called “COLP No Screening”.

4.4 CLIR Ignore

The most wonderful feature that an ISDN line can ever have is “CLIR Ignore”. CLIR stands for “Caller Line Identification Restriction”. The caller is able to hide the caller ID. If a call with an restricted ID is received, the telecommunication system will delete the number. Only the information, that the ID is restricted, is transferred. In Germany most telephone lines restrict caller IDs. It is regulated by the “Regulierungsbehörde RegTP”, that a call which is made anonymously, should not presented to the called party. The law says, that the called party has a right to know who is calling. This is quite strange.

If “CLIR Ignore” is provided to your ISDN line, the caller ID is transmitted even if the caller doesn’t want it. You could see every caller’s ID in your log file. You could always reply a call if you were absent. This feature is used by the police and emergency phones in order to call back, if the caller hangs up. The number will be transmitted with the information, that it should not be displayed. Normally the telephone company deletes the number if it should not be displayed.

Once I enabled the “CLIR Ignore” feature on my telephone line for one minute, to see how the caller ID is transmitted. In fact, a restricted caller ID is transmitted with the information, that it should not be presented. The PBX4Linux also supports suppression of the restricted caller IDs, if your have the “CLIR Ignore”-feature on your ISDN line. Of course the PBX4Linux also has the “CLIR Ignore” feature, that is called “anon-ignore”. It displays anonymous external calls (if the “CLIR Ignore” feature is provided on your ISDN line) as well as anonymous internal calls.

4.5 Previous through-connect

Whenever you make a call, you get tones and patterns from your external line: You hear the dial tone of the local exchange, the ringing of the remote exchange and sometimes announcements. All this happens before and after the called party answers. Also you don’t get charged for it.

It is possible to send own tones during ringing or any stage of call setup, instead of the standard tones of your local exchange. A caller will not get a plain ringing tone, he will hear your ‘music’ or whatever you send him. There are three things required to send own tones before you answer a call:

1. You must have a PBX line, because during call setup there cannot be multiple telephones sending their tones. A PBX line only talks to one PBX and so an audio channel can be assigned before answering of a call.
2. You must have the special feature enabled on your local exchange (telephone company). On Siemens switches (EWSD), the feature is called “PTRUCON” and is available at Version 15 of the APS (firmware). You may ask your telephone company to enable it for your, but don’t expect them to do it in Germany. Tell me if you are successful.
3. You need to enable the ‘inbandpattern” option in ‘options.conf’. See below for configuration. This options will indicate that tones are available at any state of the call.

The procedure of sending own tones is supported by the PBX, and is specified in the ISDN standard Q.931 Annex K. Note that only the audio path to the caller may be connected through, not the path from the caller to the PBX, so this is no ticket to make free calls.

The feature is mainly used for call forwarding through the PBX. If a forwarded gets answered, it will take a while until the connect message is received and forwarded by the PBX. This can cause the first second of the answered call to be lost. In case of previous through-connect, the audio path is connected to the forwarded party prior answer, and so the caller will hear as soon as the party answers.

What do you think about an answering machine that makes you charge only after the beep? ☺

4.6 Leased lines

There exist three types of ISDN basic access leased lines:

- 64Kb one B-channel
- 128Kb two B-channels

- 144Kb two B-channels + D-channel

Both ends of a leased line are terminated by an NT1 (NTBA). On each side, one ISDN card may be connected, both running in TE-Mode. Since leased lines are statically connected, no call setup must be done. The data can be just transferred between both ISDN cards.

It is possible to connect a card, running in TE-Mode (hardware layer), to one end of the leased line, where the NT-Mode protocol (software layer) is used. On the other end, telephones can be connected. A 144Kb leased line is required, because the D-channel must also be transmitted. It is then possible to use the leased line to connect telephones far, far away from the PBX. I have never tested this setup, but it should work. Any card can be used of course, since the hardware layer of this card is running in TE-Mode. For this case, contact me, so I can add this feature to the PBX4Linux.

4.7 General options

The first thing that is done, when the PBX4Linux is started, it loads the file "**options.conf**". The default file is located at "/usr/local/pbx/options.conf", if the default installation path is used. The file contains allot of keywords and comments. Comments start with "#". Here is a description for every keyword used in this file:

➢ **debug [<flags>]**

Use this keyword to specify the debug flags. This can be:

- 0 = debugging off
- >=1 = debugging on

The flags can be given decimal or hexadecimal using the "0x" prefix. Possible flags are defined in "main.h" header file. The definitions begin with "DEBUG_*".

Default: Debugging is tuned off by default.

Example: "`debug 0x8000`" turns graphical call state debugging on by default.

➢ **schedule [<priority>]**

A PBX transfers signaling information as well as audio data. For the audio data it is essential that the PBX responds as fast as possible. Therefore the Linux OS provides a scheduling algorithm to provide real time processing. "SCHED_RR" (round robin) is used which allows one task with the highest priority to be processed until it waits for an action. Be sure to have the highest priority, so no other program will use CPU if PBX4Linux needs to. The priority must be between 0 and 99. If no priority is given, the scheduler is not changed, real time processing is disabled. This is useful for debugging, because software faults, causing endless loops would cause locking of all processes, the shells, the telnet daemon, even the console.

Default: By default the scheduler has priority 0.

Example: "`schedule 99`" set PBX4Linux process in SCHED_RR with priority 99.

- **alaw | ulaw**

Specify the coding of samples, which are used in your country. It affects the internal processing of the PBX. Note that all tones and announcements are encoded "alaw".

Default: By default "alaw" is used.

- **tones_dir <directory>**

Tones and announcements are used to tell the telephone user when to dial, when the call is ringing, why did this call disconnect… All tones are a-law encoded samples. There are two default sets of tones, located at "/usr/local/pbx", if the PBX data is installed in the default location:

- tones_american
- tones_german

Both are located at "/usr/local/pbx/," if the default installation directory was used. This directory can be overridden for each PBX extension's settings file.

Default: American tones are used: "tones_american"

Example: "`tones_dir tones_german`" uses German tones.

- **fetch_tones <directory>[,<directory>[, …]]**

Tones and announcements can be loaded into memory, rather than streamed when they are played. This will prevent jitter, caused by hard disk access. Only the given directories will be loaded. Other tones will still be streamed, such as recordings of the answering machine. Give all tone sets here, separated by a comma. **Don't** put spaces between the directory names. Note that this needs about as much memory as all tones together. If tones are wave files, which are not 16 bit mono, they are converted, so total memory usage may change. The total memory usage will be printed when starting the PBX.

Default: No tones are loaded, so they are directly streamed from hard disk.

Example: "`fetch_tones tones_german,tones_english`" will cause the German and English tone sets to be fetched at startup time. Other tone sets like answering machine announcements will still be streamed from hard disk.

- **extensions_dir <directory>**

Each PBX extension has its own directory. Settings, for example, are stored inside the individual extension's directory. All extensions are stored inside the default installation directory "/usr/local/pbx". Note that each extension is a sub directory that may have files and directories inside.

Default: “extensions” The full path would be (in case of the default installation directory): “/usr/local/pbx/extensions/”.

- **national [<prefix>]**

Depending on your local telephone system, long distance calls within your country, may require a prefix. Germany and most countries in the world require “0” to call outside local area. Denmark, for example, doesn’t have a national prefix. In this case this prefix must be omitted. This is essential to convert from national numbers without prefix to numbers with prefix and vice versa. Caller IDs, for example, are always given without prefixes, but with a type indicator, indicating the type of number. To convert a Caller ID to a human dial able number, PBX4Linux must know, what prefixes national type numbers have.

Default: “0”

Examples: “`national 1`” specifies the prefix to dial long distance within the USA. “`national`” uses no prefix that must be used in Denmark.

- **international [<prefix>]**

Depending on your local telephone system, international calls require a prefix, the “international access code”. Germany and many countries in the world require “00” to call international. This is essential to convert from international numbers without access code to numbers with access code and vice versa. Caller IDs, for example, are always given without access code, but with a type indicator, indicating the type of number. To convert a Caller ID to a human dial able number, PBX4Linux must know, what access code international type numbers have.

Default: “00”

Examples: “`international 011`” specifies the access code to dial international from the USA.

- **te_if [interface1,interface2,…]**

Each port has a number. Port 1 is the first port found, when loading mISDN card drivers. When running “pbx query” all port will be listed. Enter the ports here. In case of our example for “genrc” we need to enter ‘2’ here, because our Fritz card is the second port found. Never connect two ISDN cards to the same ISDN line. This makes every call to be processed twice. Also don’t connect it to an internal port of the same PBX4Linux. This would confuse. Of course, the card may be connected to another PBX.

Default: There is no default value

Example: “`te_if 1,2`” if you connect two ISDN lines to your PBX (or if you have a special PBX line with 4 b-channels). If you use the PBX4Linux, just to interconnect internal telephones, or internal telephones with H.323 applications, enter “`te_if`” without any parameter or comment it out, to specify no external line.

- **nt_if [interface1,interface2,…]**

Here you specify all internal ports, as you did with external ports with "te_if". Depending on the number of internal ISDN ports, the PBX should have, an equal number of cards must be installed using NT-Mode. It is also possible to connect other PBXs to an internal port.

Default: There is no default value.

Example: "`nt_if 3`" if you would like to have one internal port for your PBX. If you use the PBX4Linux, just to interconnect between external lines and H.323, enter "`nt_if`" to specify no internal port or comment it out, to specify no internal ports. In this case, PBX4Linux can be used as an H.323 gateway, or as a callback / call-through machine.

- **h323_name [<name>]**

Each H.323 endpoint has a name, that identifies it. By default, the hostname is used. It can be overridden by specifying a different name.

Default: The hostname is used, if this keyword is omitted, or if no name is given.

Example: "h323_name MyPBX"

- **h323_ringconnect (yes | no)**

When an incoming call via H.323 is ringing, most application will wait for answer and might timeout before the call will be answered. Also if more digits are required due to gateway access, no digits can be dialed. Also some applications will not notify the user about the ringing. In this case it may be useful to connect the H.323 call at ringing state. The H.323 caller will get a connect and then hear the dial or ringing tone and knows what's going on. Overlap dialing is possible via digit information or DTMF. In this case the connected ID (COLP = number of the party that answers) cannot be transmitted to the H.323 caller, when the call gets really connected, because the H.323 connection is already connected.

Default: No connect is sent to the caller.

- **h323_gsm <frame range>**
- **h323_g726 <bits/sample>**
- **h323_lpc10**
- **h323_speex <mode>**
- **h323_xspeex <mode>**
- **h323_law <block size>**

Use one of these keywords, to specify the audio codecs to use for H.323 calls. The first one in the configuration file has the highest priority. This coded is checked first, when negotiating a call with another H.323 endpoint. If both parties support the first codec, the call is completed

with it. If not, the next given codec is tried, and so on. If no codec matches, the call will be disconnected. If a keyword is not given, the codec is not used at all.

The **law** codecs (a-law and mu-law) are used by ISDN, and have the best quality. They will need more than 8000 bytes per second, that cannot be transmitted via a 64k internet connection, because it has always IP overhead. Regarding to the H.323 standard, both law-codecs **must** be supported by all H.323 endpoints. This will ensure, that every H.323 endpoint will be able to communicate with each other. Almost no CPU time is required for these codecs to encode and decode. The given block size defines the frame chunks that are sent. If the size is large, the delay is high, because it takes longer to process large frames. If the size is low the overhead is greater, because the packets have headers and processing is done more often. Use 64 samples as a good value to compromise delay and traffic. The size range is 10 - 240.

The **GSM** codec has a better compression than the 'law' codec. It is used for GSM mobile telephones and needs less than 2000 bytes per second, that can be transmitted even via a modem internet link. The CPU usage is higher. The frame range specified the number of frames per packet. Same notes for "law" codec apply. The frame range is 1 - 7.

The **LPC-10** codec is the craziest thing I've ever seen. It compresses down to 300-400 bytes per seconds, that will use almost no bandwidth of any internet connection. The quality is surprisingly good, as long as only voice is transferred. This also uses more CPU than the "law" codecs. To fill a DVD (9.5 Gig) with LPC-10 compressed audio would take almost a year!

The **G.726** codec is also known as ADPCM, which is used in DECT telephones. It needs not much CPU time and offers quite a good quality for speech. The bits must be 2, 3, 4, or 5. (16, 24, 32, 40 kbits/s) The given value will always include all modes with lower value. Limiting will be useful to control the maximum bandwidth usage.

Speex and **XpiphSpeex** (h323_xspeex) are non-standard codecs. The parameter defines the mode. The mode range is 2 - 6. The given value will always include all modes with lower value. Please refer to http://www.openh323.org for more information.

Default: No codec is used, so it must be specified at least one. Remember, the "law" codec is even required to be supported by any H.323 application.

- **h323_icall [<number>]**

If there is an incoming H.323 call with no destination number specified, the given number here is used. This number must be defined in the configuration file for external numbering "numbering_ext.conf", because H.323 calls are handled as an external call by default. There you **must** specify what to do for the given number. If no number is specified here, any H.323 call without a dialed extension number gets rejected.

Default: No number is specified, H.323 calls without a dialed number get rejected.

Example: "`h323_icall 0`" will accept incoming H.323 calls as they would dial the number '0'.

- **h323_port <port>**

The default TCP port for controlling H.323 connections is 1720. This port will be used to listen to incoming H.323 calls. You may change this port, if you have more than one H.323 application on one machine, that accepts H.323 calls. Note that port 1720 is the default port for H.323 calls on every application.

Default: 1720

Example: "`h323_port 1722`" specifies an alternate port for listening to incoming calls.

- **h323_gatekeeper [<host>]**

To register PBX4Linux with an H.323 gatekeeper, use this keyword. If no host or IP is given, the gatekeeper is searched by broadcasting a message to locate the gatekeeper. The gatekeeper discover procedure is part of the H.323 standard. If the gatekeeper is outside your network, you must specify the host or IP.

Example: "`H323_gatekeeper gatekeeper.mydomain.foo`" will register the PBX at the given gatekeeper.

Default: No gatekeeper is searched.

Note: I have never tested this feature, so feel free to mail me your experience with it. Maybe there should be more features in combination with a gatekeeper.

- **nodtmf**

The mISDN driver features DTMF detection, in conjunction with the mISDN_dsp.o module. DTMF tones are used to control PBX during a connected call. Each ISDN call will use DTMF detection, if this keyword is not given. DTMF detection needs CPU cycles to decode DTMF tones. The 'Goertzel' algorithm is used, which is fast. By default, DTMF detection is enabled.

Example: "`nodtmf`" will disable the DTMF feature.

Default: DTMF detection is enabled.

- **dummyid [<id>]**

If external calls are forwarded via the PBX to external lines, the caller ID may not be available due to restriction of the caller. In this case, no ID is used, causing not to present any ID to the forwarded party. If the party would have a special telephone line, that ignores restricted caller ids, it will see the default telephone number of your telephone line. If your telephone line has the "CLIP No Screening" feature, you may specify a number that will override your default telephone number of your ISDN line.

Default: No caller ID is sent if not available. Your telephone company will use your default number instead. (In Deutschland wird die Kopfnummer verwendet.)

Example: "0" will send a '0' on forwarded calls, whenever the caller ID is not available. The police or any phone that ignores anonymous caller IDs would see this "0" instead of your default telephone number. Note: The police may also see the default telephone number instead of the caller ID sent by the PBX. But please "don't forward calls to the police".

- **inbandpattern [yes|no]**

External callers receive patterns like ringing or an announcement while calling from extern. The tones and announcements are provided by the caller's telephone system. It can be possible to provide them by PBX4Linux. In this case your provider must allow to connect b-channels prior and after connect. A special "progress indicator" (8) is used to signal the presence of tones. I have never tested it. If anyone has experience with it, let me know.

Default: No inband patterns are provided, no indication on external calls are made.

- **dsptones [none|amrican|german|oldgerman]**

Tones/announcements are streamed from user space. It is possible to use the module "dsp.o" instead. This module is required for PBX4Linux anyway. It provides simple tones with much less CPU usage. Tones are much simpler and announcements are not supported. This make PBX4Linux much faster when only used for ISDN telephony. If supported by special hardware, tones are loops that require no bus/CPU load at all, except when the tone changes. For example: HFC-4s/8s chip sets provide sample loops, tones can be stored within the FIFO once, until the tone stops or changes. This works only for ISDN ports. It can be overridden by extension's tone set.

Default: Tones are provided from user space. The sample sets are defined via keyword "tones_dir".

Note: The configuration is loaded and parsed during startup. Whenever this file changes, the PBX must be restarted.

- **email [email]**

Source email address of the PBX. E.g. it is used when sending a mail from the voice box. It is not the address the mails are sent to. Most mail servers require an existing domain in order to accept mails.

4.8 Internal numbering

Whenever an internal telephone is picked up, a dial tone is indicated, inviting to dial a number. The configuration file "**/usr/local/pbx/numbering_int.conf**" will specify, what can

be dialed and what actions will be performed. The digits, that must be dialed to dial a function, are called "**code**". View "numbering_int.conf" for actions, which are already specified there, and change them as required. There is no default value. If a dialing action is not given, it cannot be dialed. Every dialing action can be used multiple times with different codes. Here is a list of actions that can be performed:

- **<code> outdial [<prefix or complete number>]**

Specify, what code must be dialed, to make an external call. Almost every PBX uses "0" to dial out. In the USA, the code may be "9".

Example: "`0 outdial`" will specify a "0" to make an external call. "`1 outdial 5551212`" will specify "1" to call external, using the prefix "5551212". If the given number is not complete, the caller must dial more digits until the number is complete.

- **<code> data [<prefix or complete number>]**

This is exactly the same as the 'outdial' code, but it uses a data call. This can be useful to avoid audio compression and audio conversion by the telephone company. It can be used for making encrypted calls. Normally a telephone makes and accepts only a speech or audio call. Note that the destination must also support data calls. It can be configured for every extension, that incoming data calls will be answered as they would be audio calls, so this option makes sense.

- **<code> internal [<extension>]**

Specify what code must be dialed to make an internal call. If the extension is not given, the given code is used for the extension. For every extension you must specify this action, or it cannot be reached.

Example: "`200 internal`" will specify "200" to make a call to extension "200". "`205 internal 123`" will specify "205" to make a call to extension "123".

- **<code> h323 [<prefix or complete address>]**

Specify what code must be dialed to make a call via H.323. The dialing of H.323 address is explained in the section "Calling H.323". The prefix may be used to dial a complete H.323 address, or to give an incomplete number, that has to be completed by dialing.

- **<code> redial [connect]**

Specify what code must be dialed, to perform redialing the last call to external, internal or H.323. After the dialing code the last dialed number is dialed again. If the caller is internal and if she has the "display_menu" option enabled (see extension's configuration), the last number is shown rather than dialed. Dial '0' to actually dial the shown number. Use '*' and '#' to scroll through the history of earlier dialed numbers. When using the 'connect' keyword, the

call gets connected after dialing the redial code. Keypad facility or DTMF tones can be used to scroll through the number.

- **<code> reply [connect]**

Specify what code must be dialed, to perform reply of the last incoming call. After the dialing code the last incoming call is dialed. If the caller is internal and if she has the "display_menu" option enabled (see extension's configuration), the last number is shown rather than dialed. Dial '0' to actually dial the shown number. Use '*' and '#' to scroll through the history of earlier received numbers. Also the keys '1' and '3' may be used to scroll through the history and '2' to dial.

- **<code> powerdial**

Power dialing will redial the last call to external, internal, or via H.323. Whenever the call is busy or will not be completed, the call is repeated automatically. When the call gets connected, the call will not be redialed after disconnect. This is very useful, if the called party is busy or if the call cannot be completed due to telephone network failures. To stop power dialing, the caller must hang up. The caller must dial the number of seconds to wait between the tries just after the power dial code. After dialing the number of seconds a "#" must be dialed. If only "#" is dialed after the power dial code, one second is used. Whenever the call is redialed, a beep tone will indicate the failure. After the given delay, the call is retried.

NOTE: Power dialing can cause trouble with your phone company, because it may cause high signaling load, especially when power dialing for hours. Don't blame me if your telephone line gets blocked due to too many call setups within a short period of time.

Example: "`67 powerdial`" will specify "67" to redial the last number using power dial feature. The caller must dial "67#", in order to start power dialing with a retry delay of one second. If the caller dials "6760#", the call will be redialed every minute until it connects successfully.

- **<code> abbrev**

Every extension may dial abbreviations, which are specified in the extension's phonebook: "/usr/local/pbx/extensions/<extension>/phonebook". A phonebook entry is a single line, which looks like this:

<abbreviation> <dialing> [<name>]

The abbreviation may be any string of digits. If the abbreviation is dialed, the "dialing" is performed. "dialing" may be any string of digits that may be dialed from the extension, except the code for abbreviation. This will prevent recursions.

Abbreviations can be useful, especially to dial digits, that cannot be dialed from a telephone.

Example: "`7 abbrev`" will specify the code "7" for dialing an abbreviation. If this abbreviation in the phonebook should be dialed: "123 05551212 Directory Assistance", the

caller must dial “7123”. This will cause the dialing of “05551212” to be performed. In this example the “0” in front of “5551212” is used to make an external call.

Assume, that the number to make an H.323 call is “4”. A call to “remote.h323.foo” would be specified in the phonebook: “456 4@remote.h323.foo”. The caller must then dial “7456” in order to dial “4@remote.h323.foo”. The “4” in front of the “@” is the number for H.323 calls. The “@remote.h323.foo” is the H.323 address to be dialed.

- **<code> [callerid]**

This will specify the number to be dialed, in order to change the calling extension’s caller ID. After the code, the caller must dial the new caller ID. After the caller ID, the caller must finish with “#”. The caller IDs type is defined by the digits entered. If the national prefix is dialed in front of the caller ID, the ID is of type ‘national’, and sent without the prefix of course. The same goes for the ‘international’ caller ID type. To make current caller ID anonymous, the caller must dial “#” right after the code. The make allow the presentation again, a new caller ID must be specified. If the caller ID is of type ‘unknown’, the new caller ID will always be of type ‘unknown’. This is the normal case, if not “Clear Ignore” feature is available to the external line.

Example: “`91 callerid`” will specify the prefix “91” to be dialed to change the caller ID. If the caller wants to change her caller ID to “1234”, she must dial “911234#”.

- **<code> [calleridnext]**

This is similar to “callerid” except for specifying a caller ID, which is only used for the next call. Again, the dialing must be completed with “#” at the end of the caller ID. To make the caller ID for the next call anonymous, the caller must dial “#” without a caller ID.

- **<code> set-cfu [destination]**
- **<code> set-cfb [destination]**
- **<code> set-cfnr [destination]**
- **<code> set-cfp [destination]**

Change the current call forwarding for this extension. After the code, the caller must dial the new destination and finish with “#”. If no destination is given, but finished with “#”, the call forwarding is deactivated.

“CFU” means: “Call Forwarding Unconditional”. Every call to the given extension is directly forwarded; the internal phones on the specified ports will not ring.

“CFB” means: “Call Forwarding when Busy”. If the extension is currently calling, every call to the given extension is directly forwarded; the internal phones on the specified ports will not ring. This will be performed if at least one caller uses the extension.

“CFNR” means: “Call Forwarding when No Response”. Every call to the given extension is forwarded after some time of no response; the internal phones on the specified ports will stop ringing.

“CFP” means: “Call Forwarding Parallel”. Every call to the given extension will be forwarded to the given number. The internal phone and the forwarded destination will ring, until someone answered on an internal phone, or on the forwarded destination.

- **<code> cfu-vbox**
- **<code> cfb-vbox**
- **<code> cfnr-vbox**

The current call forwarding for this extension is changed to the answering machine. Just dial the code and it will be done, if the extension allows change of forwarding.

- **<code> test [test code]**

This specifies the code for the test mode. Read the “Test Mode” section for suffixes and their functions. It is possible to specify the ‘test code’. If it is not specified, the caller must dial the ‘test code’. On external lines, it requites the extension dialing feature, which is only available on PBX lines. If a complete ‘test code’ is given, no extension dialing is needed.

Example: “`99 test`” will specify the prefix “99” to be dialed to dial test mode. Example: If the caller would dial “995”, she would hear the hold music. “`99 test 5`” would directly play the hold music, if “99” is dialed.

- **<code> callback <extension>**

This function shall only be used in the configuration file “numbering_ext.conf”. The specified extension will perform a callback if the code is dialed. The number to call back is specified by the caller ID of the caller. Callback is only done to external telephones. Read “Callback” section for detailed information before using this and authentication.

Example: “`8765 callback 200`” will perform a callback from extension, if the external number (MSN) “8765” is dialed.

- **<code> execute <command>**

Whenever the code is dialed, the given command is executed after hang up. If digits are dialed after the number, the digits may be included in the command.

Note: Use this only for commands that exit quickly, and do not cause high CPU load, because this will slow down PBX and cause delays. I have not tested it well. It will do a daemon fork and keep all current file descriptions open. Don’t use it to start programs. Use it to run a tool, which immediately exits, like “echo”, “reboot” or any shell scripts, which may turn on your coffee machine.

Example: “`92 execute echo >> /tmp/log %s`” will execute the “echo” command, which will, in this case, append the dialed digits after the prefix “92” to the file “/tmp/log” after the caller hangs up.

Please don't use this, unless you know what you are doing.

- **<code> play <path>**

Dialing this code, the PBX will play the given audio file. It may be encoded as wave (16bit mono, stereo or 8bit mono) or "law". The type of "law" codec is specified in options.conf. The file may be specified with relative path or with absolute path. If the path is relative, the specified "tones_dir" is used. The suffix ".isdn" or ".wav" will be appended to the file, according to what exist. If the file doesn't exist or if the playing has finished, the suffix "_loop.isdn" or "_loop.wav" will be played afterwards. If this also doesn't exist, nothing is played.

Example: "`93 play /root/alle_meine_entchen`" will play the file "/root/alle_meine_entchen.isdn" or "/root/alle_meine_entchen.wav", if exist. If the file has been played or if it doesn't exist, the file "/root/alle_meine_entchen_loop.isdn" or "/root/alle_meine_entchen_loop.wav" is played until the caller hangs up.

- **<code> pick**

Assume that any extension has an incoming call. In this case, an incoming call cal be answered from a different extension.

Example: "`3 pick`" will connect to knocking call, when the caller dials "3".

- **<code> login [<extension>]**

The login feature is used to log into a different extension than the one, configured at the calling phone. Especially in the file "numbering_ext.conf", it may be used, to log into extensions from external phones. If a caller dials the code for login, she will be asked to enter a password, if the extension is already specified. If the extension is not specified, the caller must dial the extension's number, and then dial the password. The password is specified within the extension's setting file. By default, no password is used, so no password will work. The number of digits, to be dialed for the password, is defined by the length of the extension's password. If the length has been reached, the call is disconnected, if the password is wrong. If the password is correct, the dial tone invites to dial a number. All dialing is now done using DTMF or keypad information, if available.

Example: "`98 login 200`". If the caller dials 98, she will get connected and asked to dial the password of extension 200. After entering the correct password, a dial tone indicates the correctness. Otherwise the caller gets disconnected.

- **<code> dtmf**

If the caller dials the code for the DTMF action, she will be connected, and receive a dial tone. Dialing is now done via DTMF or keypad information. This action may also be used for external call ("numbering_ext.conf") to provide extension dialing, if your phone line is not a PBX line, and in this case not capable of dialing extensions after the main number. In this case the caller will hear a dial tone and is invited to dial. All dialing is now done using DTMF or

keypad information, if available. If keypad dialing is used, the DTMF detection for this call is turned off, because some telephones send both DTMF and keypad information. Otherwise this may cause double dialing of the digits.

If the caller calls from external, she is invited to dial external numbers. If the caller calls from internal, she is invited to dial internal number. Dialing internal number via DTMF makes no sense unless the telephone is analog, and connected to an analog-to-ISDN converter, which doesn't support "*" and "#". Using this function to connect, will allow dialing of digits, "*" and "#" using DTMF.

Example: "`97 dtmf`" will enable DTMF dialing, whenever the caller dials "97".

- **<code> vbox-play**

Dialing this code will play back all recorded calls in your answering machine. A spoken and displayed menu will be available. Read "Using PBX" for more information on answering machine.

- **<code> vbox-record <extension>**

Calls can be directly transferred to the answering machine of the given extension. Instead of forwarding the calls with an extension's forward option, a different code may be dialed, so the caller can directly call the answering machine.

Note: The configuration is loaded and parsed during startup. Whenever this file changes, the PBX must be restarted.

- **<code> calculator [connect]**

Calls to this code will be processed by the calculator. This only works for internal ISDN phones, since they only get the required display information. The calculator is processed by dialing the formula. The optional keyword "connect" is used to send a connect message to the phone, since ISDN phones may stop dialing after a certain number of digits. In this case keypad or DTMF dialing is used.

The formula is only a simple term. It must have two values and one operand. The values may only be entered as positive float values. After dialing the code, the first value is dialed, then the operand and then the second value. To dial the operand, the '*' key is used. If it is pressed once, the multiplication is used. If it is pressed again, the division is used. If it is pressed again, the addition is used. If it is pressed again, the subtraction is used. Depending on the first value, if it is pressed again, the decimal point "." is added. If the second value is entered, the '*' key will directly add the decimal point. After entering the formula, the '#' is used to get the result. If the result is displayed, it may be used for another formula. If '#' is pressed again, the formula is cleared. Here are examples of formulas and how to enter them:

Term:	**Result:**	**Dial:**
5*8	40	5*8#
5.5*8	44	5*****5*8#
5.5*8.8	48.4	5*****5*8*8#

4/8	0.5	4**8#
4+8	12	4***8#
4-8	-4	4****8#

The term is always displayed during dialing, not the keys that are actually entered.

4.9 External numbering

Whenever a call is received from an external line, the number that has been dialed by the caller will be received by the PBX. The configuration file "**/usr/local/pbx/numbering_ext.conf**" will specify, what codes can be dialed, and what functions will be performed. The same functions, as specified in "numbering_int.conf," may be used. Some functions don't make sense, because they require to be made from an extension, rather than from external. View "numbering_ext.conf" for functions, which are already specified there, and alter them as required. There is no default value. If an action is not given, it cannot be dialed. Every action can be used multiple times with different codes.

Example: If a call is received from a normal **multipoint** ISDN line, the dialed numbers can be assigned to internal extensions:

```
1234 internal 200
1235 internal 201
1236 internal 202
```

Incoming calls to the number 1234 will be routed to the extension 200 and so on.

Example: If the external ISDN line is a **PBX** ISDN line, the numbers that are dialed behind the main number can be assigned. Assume that the main number of the PBX line is "1234":

```
0 internal 200
11 internal 201
12 internal 202
```

Incoming calls to the number "1234-0" would be routed to the extension 200, "1234-11" to extension 201 and so on. The caller must dial until the number is complete. It depends on how the PBX line is configured. Some PBX lines will also send the main number on incoming calls. In this case the main number + suffix must be specified:

```
12340 internal 200
123411 internal 201
123412 internal 202
```

Ask your telephone company, not to send the main number with an incoming call, this makes configuration easier.

Be **careful** when using the following example:

```
2 external 5551212
3 external 555
4 external
```

Incoming calls to suffix “2” will be routed to the external number “5551212”. In case, the number “5551212” is a complete number, the call will proceed after receiving the incoming “2”. If the call to suffix “3” is received, it will cause dialing of “555”. This number is not complete, so the caller may dial more digits after “3”. For example: “31212” will cause “5551212” to be dialed. If a call to suffix 4 is received, the call may be forwarded to any number. The PBX4Linux will forward any dialed number after the suffix 4 to an external call. Everyone would be able to call any number external number, as soon as enough digits are routed from the incoming line. **Never define incomplete external numbers**, unless you have a good reason.

Note: The configuration is loaded and parsed during startup. Whenever this file changes, the PBX must be restarted.

4.10 Extensions

An extension is defined by a number. It is less confusing, if this number equals to the number that must be dialed internally or externally. Extensions have all individual files and settings:

- **/usr/local/pbx/extensions/<extension>/settings**

Settings are used to describe features of the extension.

- **/usr/local/pbx/extensions/<extension>/logs**

All calls from and to the extension will be logged with date, time, duration, source and destination.

- **/usr/local/pbx/extensions/<extension>/phonebook**

The phonebook is used to store abbreviated numbers and their names. It is also possible to store any dialing function of the PBX. (The only thing that cannot be stored, is the recall of an abbreviated number. This avoids recursions.)

- **/usr/local/pbx/extensions/<extension>/callbackauth**

Everyone in the world can use the callback feature, as long as she sends her caller ID. After callback, the extension’s password must be entered. If the password is correct, the caller ID is stored in the file “callbackauth”. This is useful, to prevent unauthorized callers from using callback. Next time a callback is done with the same caller ID, the callback is done without authentification.

- **/usr/local/pbx/extensions/<extension>/recordings**

Because we have a computer, and **not** just a dirty box hanging on the wall with CPU speed of Commodore 64 computers, we are able to record calls also. Hard drives became large these days. If call recording is enabled for an extension, all incoming and outgoing calls are stored as wave or “law” files. The filename consists of the date, time, source and destination of this call.

- **/usr/local/pbx/extensions/<extension>/vbox**

The integrated answering machine stores all incoming calls in the "vbox" directory. Also the "announcement" file is store here. Additionally, there will be an "index" file of all currently stored messages.

Extensions can be easily created by using the "genextension" command:

```
$ genextension <extension> <ports> <caller ID>
```

The first parameter is the extension number, for example: "200". The second parameter specifies the internal ports to be used for sending calls to the extension. "1" would send a call to the first internal port as defined in "options.conf". If calls should ring on more then one port, they must be separated by commas without spaces. "1,3,4" would send a call to the internal port 1, 3 and 4. In this case, at least four internal ports must be defined in "options.conf". The third parameter defines the caller ID to be used, whenever the extension makes a call (to external line). The type of caller ID is "undefined", that is the standard type for normal external ISDN lines. If the external ISDN line has the "CLIP No Screening" feature, it must be given with type "subscriber", "national" or "international". The "settings" file must then be altered with an editor. After executing the command the extension will be created at "/usr/local/pbx/extensions/200/".

Also you must make an entry in the "numbering_int.conf" and "numbering_ext.conf" or it cannot be dialed:

```
200 internal
```

or

```
200 internal 200
```

For external numbering, the extension may also be specified as the 'main extension':

```
0 internal 200
```

If you use a normal (multipoint) telephone LINE with an msn number "1234", specify this in "numbering_ext.conf":

```
1234 internal 200
```

Here is a description of all keywords defined in the "/usr/local/extensions/200/**settings**" file:

- **name [<string>]**

Each extension can have a name. The name will be processed internally with the caller/called ID. To actually see the name on a telephone, the 'display_name' feature must be enabled. In this case the display shows the name with the number. Also the 'cnip' feature can be used. In this case it must be enabled for the other party's extension and it must be supported by her telephone.

- **prefix [<prefix>]**

This prefix will be dialed, whenever the extension picks up the phone. Instead of hearing a dial tone, the caller will have the given prefix already dialed. This can be useful, if for example, the caller should be able to directly dial an external number, without dialing the code for external dialing first. If the code is "0", the prefix of "0" would cause directly call external after pickup. If the prefix would be "0<number>", the number would be dialed just after pick up. This can be useful for emergency calls of handicapped persons or little children, if they are unable to dial a number.

Note: The given prefix will not override the dialing rights.

- **cfu [<number> | vbox]**

Whenever this number is given, any incoming call to the extension will cause the call to be forwarded to the given number. The call forward is done through the PBX, since there is no implementation of deflecting a call. The benefit of it is, that the party forwarded to, may control the PBX, answer knocking calls, and even record. To forward to the answering machine, the keyword "vbox" must be used.

- **cfb [<number> | vbox]**

This call forwarding is performed, only if the called extension is busy. That is, if a phone of the extension is picked up, is connected during a call, receives announcements during disconnect, or even makes multiple calls. The person, using the extension, is currently doing something. That's why she is busy. To forward to the answering machine, the keyword "vbox" must be used.

- **cfnr [<number> | vbox]**
- **cfnr_delay [seconds]**

This call forwarding is made if the called person does not answer within a given time. If, by default, the call is not answered after 20 seconds, it is forwarded to the given number. The number of seconds can be changed. To forward to the answering machine, the keyword "vbox" must be used.

- **cfp [<number>]**

This call forwarding will not actually forward the call. Calls will not only ring on internal ports, they will also ring on an external line at the same time. It is possible to pick up the call at any ringing phone. If the called party picks up the phone, all other phones will stop ringing. The benefits are: No more running to the ringing phone, before the call gets forwarded. And no more callers, who will hang up before: the call is forwarded, the mobile rings, the mobile take out of the bag, because the caller doesn't wait that long, because she knows that you have a one-room-30m²-apartment.

- **change_forward (yes | no)**

Allows or disallows the user, to change the forwarding for her extension. This is useful, if the extension's user should not be able to change the forwarding.

- **callerid (none | [s | n | i]<caller ID> [anonymous])**

This will specify the caller ID for this extension. Whenever the extension makes a call or receives a call, the caller ID is used. The caller ID is of type "unknown". This is useful for any ISDN line without the "CLIP No Screening" feature. If the caller ID starts with a small "s", it will be of type "subscriber". If the caller ID starts with "n", it will be of type "national". If the caller ID starts with "i", it will be of type "international". Because the types are used, it is not allowed to put national or international prefix in front of the caller ID's number. For example: A caller ID for the number "1-212-555-1212", which is a national number, would be "n2125551212". The international number would be "i12125551212", because the "i" will indicate international type, the "1" will be the country code, and the rest will indicate area code and number.

Caller ID's may also be sent with the information that it should not be displayed to the called party. The called party will not receive the caller ID, unless she has the "CLIR ignore" feature on her line, that is used by the police, for example. In case of an emergency, the caller ID can be important, even if the call is anonymous.

If no caller ID is given, and if the caller ID should be sent anonymously (not to be displayed), just the keyword "none" is used. In this case, no caller ID is transmitted. The telephone company may use the default number instead.

- **id_next_call [[s | n | i]<caller ID> [anonymous]]**

If this caller ID is given, it will be used for the call made by the extension. After the call, it will be erased from the configuration file.

- **change_callerid (yes | no)**

Allows or disallows the extension's caller to change the caller ID. This is useful, if the extension should not be able to change her caller ID, not even for the next call.

- **clip (asis | hide)**

Calls, that are forwarded, use the originating caller ID. An internal telephone will always show the real caller ID. If a caller makes a call to an extension that is forwarded to a mobile, the mobile will receive the caller ID that is given here:

- **asis** = The original ID of the caller, that is forwarded, will be used. (Only works with "CLIP No Screening" feature)
- **hide** = The extension's caller ID "callerid" will be used.

➢ **colp (asis | hide | force)**

If an extension is called; and if the called party answers, the caller ID of the extension is presented to the caller, showing her, who she is connected with. On **forwarded** calls, the given "connected to ID" may be used:

- **asis** = The ID of the party "forwarded to", will be presented to the calling party, if available, showing here, that she is connected to a different number. (Only works with "COLP No Screening" feature)
- **Hide** = Always the extension's caller ID "callerid" will be presented to the calling party, even for forwarded calls. No one would notice, if the call has been forwarded.
- **Force** = The ID of the party "forwarded to" is used. If the party "forwarded to" will not present any "connected to ID", the dialed number of the party "forwarded to" will be used instead. (Only works with "COLP No Screening" feature)

➢ **clip_prefix [<prefix>]**

When a caller ID is shown on an internal phone, it is show without the outdial prefix. Most telephones have a feature to add the outdial prefix in front, so unanswered calls can be dialed out of the list of caller IDs. But some phones don't have this feature. In this case the outdial prefix can be specified here. This/These is/are the digit(s) that have to be dialed to get an external line.

If no prefix is given, the caller ID is forwarded as is. The type of caller ID is processed by the telephone. If a prefix is given, the caller ID is converted by the PBX and sent as „unknown" type to the telephone. In this case the prefix + access codes are prefixed, as it will be done by the telephone.

Example: A caller ID **+1-809-563-0000** of „international" type „1809563000" and a given prefix of „9" and an international prefix of „00" (see options.conf) will result in an „unknown" caller ID: „90018095630000"

➢ **centrex (yes | no)**

Note: Centex might not work in the current version.

"Called Name Identification Presentation" (CNIP) can be used if supported by the telephone or even by the external line. This feature is user -> network only, which means that it can not be transmitted to the external line, only received from it. This special feature must be supported by the telephone. Some telephones get confused by the CNIP feature, so their operation fail. It is also possible to use display information. CNIP might also store the name in the caller's list of the telephone. Display information may disappear after a few seconds. CNIP can be presented with the 'centrex' feature when

- an internal call is made, and the caller has specified a name for the extension.

- a call is received/answered from external, and the external line also features CNIP / CONP.
- the name is found in the phonebook.

The ‘centrex’ option also features “COnnected Name identification Presentation” (CONP). The name of the party that answered the call can be presented.

➢ **ports [<port 1>[,<port 2>[, …]]]**

This is one of the most important setting. If nothing is specified here, the extension cannot be called. Of course the call forwarding is still possible. The numbers of ports that should be used must be specified here. Multiple ports must be separated by comma and **without spaces**. See the following examples:

- “ports”
  This causes no internal phone to ring, when the extension is called.
- “ports 2”
  The second internal port is used. At least two internal ports must exist in this example.
- “ports 1,3”
  The first and the third port will ring on incoming calls. Remember not to use spaces around the comma.

➢ **rights (none | internal | local | national | international)**

Each extension has dialing rights. **Use only one of the keywords.** By default, **international** calls are allowed. If the rights are set **national**, any international call attempt will cause the call to be disconnected. If the rights are **local**, any national or international call attempt will be forbidden. If the rights are **internal**, no external call is allowed at all. If the rights are none, not even internal calls are allowed. The reception of any call will not be restricted by this option.

Example: If you have children, they may not be allowed to make calls outside your local area. You would give local rights to your kids.

Example: If you have a customer phone in the supermarket, you would give only internal rights to your customers.

Example: If the phone should only be callable, try “none”.

➢ **delete_ext (yes | no)**

Delete function when dialing external numbers. '*****' will delete the last digit, '**#**' will delete the complete external number. After pressing ‘#’ the dial tone will be heard again. Also enable 'display_dialing' to see on the display what is actually dialed.

- **noknocking (yes | no)**

Knocking may be used to receive a call, even if the phone is busy. Knocking must be provided by the ISDN telephone. This is the normal case. If you say “yes”, any call to this extension is rejected as being busy.

- **txvol <volume –8…0…8>**

This will adjust the volume of the extension by the power of 2. It only affects internal phones. If the volume is 0, it means that it is not changed. A value of ‘-1’ would cause the audio to be transmitted from the extension to the remote party at half volume. A value of ‘1’ would cause the audio to be transmitted at double volume. Values of ‘2’ would cause the quadruple volume, and a value of ‘8’ would cause 256 times the volume (very loud). Please note, that the sound wave may be clipped when the volume is increased due to limited range.

Increase this value, if the mouth peace of your telephone is too weak, or you speak with very low volume. Decrease this if, you have the telephone in a loud environment.

You may also use this feature to amplify the mouth peace, so it becomes a hand-free microphone.

- **rxvol <volume>**

This will do the same as “txvol”, but for the audio received by the extension. Increase this value, if the ear peace of your telephone is too weak, if you have an ear disease, or if you have the telephone in a loud environment.

- **tout_setup (off | <seconds>)**
- **tout_dialing (off | <seconds>)**
- **tout_proceeding (off | <seconds>)**
- **tout_alerting (off | <seconds>)**
- **tout_disconnect (off | <seconds>)**

By default there are timeouts during a call. The timeouts can be turned off. All timeouts are only relevant for internal ISDN telephones. Some phones have own timeouts, that cannot be altered. Here is the explanation of all timeouts:

- tout_setup: after picking up the phone, before dialing anything
- tout_dialing: after dialing the last digit(s)
- tout_proceeding: after received the proceeding message
- tout_altering: during ringing
- tout_disconnect: after the call has been disconnected or failed

Note: The timeouts are equal for outgoing and incoming calls. The ringing tone that can be heard when making a call will timeout, as well as ringing of the phone on incoming calls. If both parties have timeouts, the party with the smaller value will timeout first.

- **own_setup (yes | no)**
- **own_proceeding (yes | no)**
- **own_alerting (yes | no)**
- **own_cause (yes | no)**

There are some announcements in the public network, that tells the caller the reason, why a call cannot be completed, or why it has been disconnected. Also the dial tone or busy tone is coming from the network to inform the current call state. The ISDN network supports up to 127 possible causes, but only some of them are specified or assigned with tones. Most causes 3result in a busy tone. The PBX4Linux can give you more detailed announcements, than the public network would do. Read the section "Understanding Causes" for more details about this. If a call gets disconnected, the original announcement, that is coming from the external line can be presented to by the option "own_cause no". If the PBX4Linux announcement should be used instead, the keyword "own_cause yes" must be used.

Tones during setup, proceeding and alerting can also be replaced. It can be useful to make a dialtone or ringing tone sound different.

- **facility (yes | no)**

Facility information are country specific information during call. Germany for example transfers advice of charge information during a call. The facility is encoded using ASN.1 notation, that is just transparently forwarded to the terminal. To enable the feature, 'yes' must be given here. If supported by the telephone, it will display the charge info and even keep track of it, using a counter. The telephone company does not send any money amount, they send 'Units'. Ask you company about the price of one unit. Also note that the charge for a call depends on the length of a call or even the total calls at the end of a month, so the displayed information may differ from the actual charge on the phone bill. Also some services will not be charged by units, like value added service numbers. This option is required if a payphone should be connected to the PBX.

- **display_cause (none | english | german)**

Whenever a call cannot be established, or if the call is disconnected, a number from 1-127 is transmitted. This number gives information about why the call cannot be established, or has been released. The number can be displayed to the ISDN phone that is connected to an internal port. If "none" is defined, nothing is displayed. If "english" or "german" is defined, the value, causing the 'disconnect', will be shown on the display of the phone as English or German text. In front of the text, the number is shown.

Examle: If an unassigned number has been called, the 'cause' value "1" will be transmitted. The display message in English would be "1 – Unallocated number". The display message in German would be "1 – Nummer nicht vergeben".

Displaying 'causes' will give more detailed information, then the telephone would give.

- **display_ext (yes | no)**

If you say "yes," the caller ID will be indicated to the telephone's display (using display information). Also IDs, that have been sent using "CLIP No Screening", will be discovered. In this case the word "fake" is appended to the caller ID, indicating that this number is made up by the caller and, may not be the real caller ID.

- **display_int (yes | no)**

If you say "yes," the caller ID from internal extensions will be indicated to the telephone's display (using display information), and also the internal extension's number.

- **display_h323 (yes | no)**

If you say "yes", the H.323 source address will be displayed after the caller ID: "alias@ip:port" (using display information). This will show additional information about H.323 calls received.

- **display_menu (yes | no)**

It is hard to remember all dialing codes, which are defined in "numbering_int.conf". If you say "yes" here, the keys "*" and "#" can be used to select all codes that have been defined. If, after pick up of the telephone, the key "*" or "#" is pressed, the first code is displayed on the telephone. If "#" is pressed, the next code is displayed. If "*" is pressed, the previous code is displayed. Additionally the code on the display is followed with the function description for this code.

Example: If the first code, defined in "numbering_int.conf" is "external", the display would show after pressing "#" or "*": "0 External". If the next code, defined in "numbering_int.conf" is "2 internal 200", the display would show after pressing "#": "2 Internal 200". After pressing "#", if the end of the codes is reached, the display will continue to display the first code. After pressing "*", if the start of the codes is reached, the display continue to display the last code.

- **display_dialing (yes | no)**

During dialing, the dialed number may alter. If a H.323 call is made, the "*" represent the dot between IP numbers. If you say "yes" here, the display would show, what's actually dialed, instead of what has been entered on the phone. This works for all codes, like internal, external and other dialing functions.

- **tones_dir [<directory>]**

If this directory is given, the default tones directory, which has been specified in "options.conf", will be overridden for this extension. This option makes it possible, to define an individual set of tones for each extension.

Example: An extension may use German tones (“tones_german”) while others use the default tones (“tones_american”).
9

- **record (<type> | off)**

Each call can be recorded. If recording is turned on, the calls will be recorded as wave encoded files. The files will be store at “/usr/local/extension/<extension>/recordings/<file>”. The file name is specified by the date and time, it has been recorded, and by the duration. The following file types can be generated while recording:

- a-law/mu-law: (8 Kbytes/s) (depending on the global mode defined in options.conf)
- wave 16 bit mono: (16 Kbytes/s)
- wave 16 bit stereo: (32 Kbytes/s)
- wave 8 bit mono: (8 Kbytes/s)

When using stereo recording, one party will be heard on one channel, and the other party on the other.

- **password [<pass phrase>]**

The password is needed, to gain access to an extension. When an internal telephone is picked up, no password is needed to make phone calls. For example, if callback is used, this password must be entered after PBX4Linux calls back, in case the phone (identified with the caller ID), from where the callback is done, is not known yet. If the login function is dialed, the password must be entered in order to have access to the extension. In this case the password is used to authenticate external or H.323 calls. Read the section “Internal numbering” and “Callback” for more information.

The following options control the **answering machine (voice box)**. It has nothing to do with the ISDN4Linux “VBox".

- **vbox_mode (normal | parallel | announcement)**

By default, the **normal** mode is used. If someone gets connected to the answering machine, the announcement is played. After playing the announcement, the call is recorded. When the caller hangs up or the time limit exceeds, the recording is stopped. If the **parallel** mode is used, the call is recorded during play of the announcement. This is useful to get the name of the caller, if you record an announcement, where you pretend that you answer the phone. The caller will tell his name before she recognizes that he talks to an answering machine. Many people I know don’t talk to answering machines. If the caller should not be able to talk to the answering machine, the **announcement** mode is used. After the announcement is played, the call is disconnected.

- **vbox_codec (mono | stereo | 8bit | law)**

See “record” for codec descriptions. This codec is used to record calls. If the stereo parameter is used in conjunction with the parallel mode, the caller and the announcement are recorded on different channels. Using stereo for ‘normal’ mode makes no sense.

- **vbox_time (infinite | <seconds>)**

This specifies the limit of recorded calls in seconds. If “infinite” is given, then there is no limit.

- **vbox_display (brief | detailed | off)**

The display on the ISDN telephone will show the current menu and state, when the playback function of the answering machine is called. The current counter of the message is displayed also. For small displays, use “brief”, for larger displays, use “detailed”. If your telephone gets confused, turn it “off”.

- **vbox_language (english | german)**

When the playback function of the answering machine is called, all menu information are spoken by a voice. The voice is available in English or in German. Specify the desired language for the voice.

- **vbox_email [<email>]**
- **vbox_email_file (yes | no)**

To send an email whenever a message is received, give the email address using the “vbox_email” keyword. It is also possible to attach the recorded audio file to that email by saying “yes” to the file option. The source email address is specified in “options.conf”, it must be a valid address, so most servers will only accept valid source and destination addresses.

- **datacall (yes | no)**

Data calls will not be answered by a telephone. Sometimes it may be useful to answer a data call. In order to answer a data call but tell the telephone it is a voice call, say “yes” here. This is useful to answer calls that should be encrypted. Your telephone company may use compression and conversion for audio and speech calls, so you may need data calls instead or encryption is not possible.

- **last_out <number>**
- **last_in <number>**

These keywords may appear multiple times in the configuration file. “last_out” stores the 50 last external, internal or H.323 numbers that has been dialed, including the dialing code. It is used when the redial or power dial function is dialed. “last_in” stores the 50 last incoming

calls. It is used when the reply dial function is dialed. The actual number of entries are defined in “main.h” using “MAX_REMEMBER” definition.

4.11 Directory

In the default installation directory there is a file:

/usr/local/pbx/directory.list

It contains numbers and their names. For every incoming caller ID (CLIP) and outgoing connected ID (COLP) this directory is parsed. If a number is found, the name will be presented with the number. Of course the ‘display_name’ or ‘centrex’ feature must be enabled to be able to present a name.

Format: <phone number> <name>

The number must be given as received:

- National number
  The national prefix must be included or 'n' must be used.
  E.g.: n9036681733 or 19036681733 (in case '1' is the national prefix)
  E.g.: n21250993 or 021250993 (in case '0' is the national prefix)

- International number
  The international prefix must be included or 'i'.
  E.g.: i4921256483 or 0114921256493 (in case '011' is the intl. prefix)
  E.g.: i19036681733 or 0019036681733 (in case '00' is the intl. prefix)

- Subscriber number
  No prefix must be included or 's' must be used.
  E.g.: s50993 or 1733

Note: National numbers always require the area code. International numbers always require the country code + area code.

4.12 Authenticate H.323 calls

If a call via H.323 is received with a destination number given, the number is handled as specified in "numbering_ext.conf". If no destination number is given, the default number is used, as specified in "options.conf" via "h323_icall". It is also possible, to handle H.323 calls as they would have been made from an extension. In this case, the **IP** address of the caller may be mapped to an extension. All IP addresses and their extensions are defined in “/usr/local/pbx/h323_gateway.conf”. This is ok, as long as the connection from the H.323 caller can be trusted, and if the IP address is static. This is useful for LAN, VPN or other private networks. The caller may receive a connect when the call is ringing. This is useful for some applications that will only connect the audio path after connect message. Use the keyword ‘connect’ behind the extension number. Use the keyword ‘dtmf’ to enable DTMF detection. Refer to the comments in the configuration file.

Instead of using the IP address, a **password** can be used. H.323 callers must dial the code for login, which is defined in “numbering_ext.conf”, also she must dial the extension that should be used, if not specified for the code. The caller will get a dial tone and may enter the password for the extension.

5

Using PBX4Linux

5.1 Running

To run PBX4Linux, just enter:

```
$ pbx
```

This will show a list of available options.

```
$ pbx query
```

Will show all available ISDN ports/cards, and if they are configured right to be used with the PBX.

```
$ pbx start
```

The PBX will start and exit in cause of an error. If no error occurred, PBX4Linux will continue to run, until CTRL+C has been pressed. PBX4Linux will then exit by freeing all resources. Sometime PBX4Linux will not stop. It has to be killed with signal "9" then.

It is also possible to run PBX4Linux as a daemon:

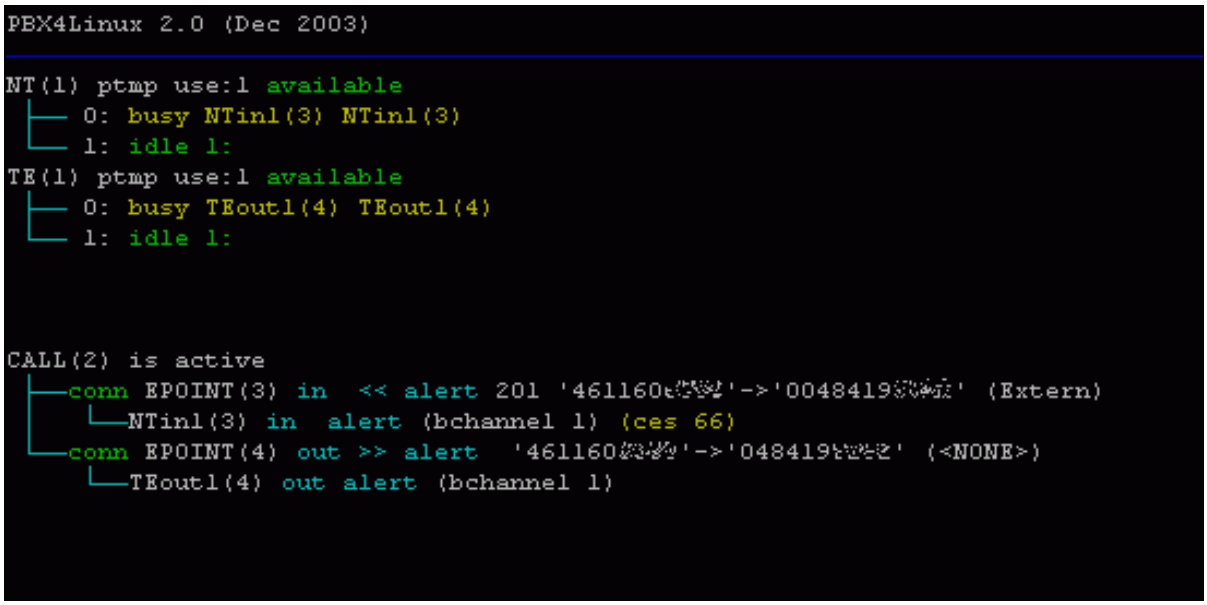
```
$ pbx fork
```

In this case PBX4Linux will still do debug output to the current tty, but will not exit, if the shell is closed. PBX4Linux will do a daemon fork, use "`kill`" or "`killall`" to terminate. Use this to start PBX4Linux during system boot. Be sure that ISDN4Linux is started first.

To get an overview of the current call states, enabled the DEBUG_STATE flag by setting the 'debug' options in "options.conf" to 0x8000. Another way to do this is to start pbx with 'state' option:

```
$ pbx state
```

The screen will look like this:



(Figure: state of a call)

On the this example screen a call is made from an internal phone to an external number. The upper block show all ISDN cards with their b-channels. The lower block show the state of each entity. The ‘call’ connects two ‘endpoints’ the ‘endpoints’ have a port each, that connect them to the telephone and the phone line. The structure of the PBX is explained later in this documentation.

If PBX4Linux is running twice, the second process will recognize the first lock, and exit immediately with a message. The lock is located at “/var/run/pbx.lock”.

5.2 Internal phone

A phone, that is connected to any internal port of the PBX will be identified by its caller ID. In order to use that phone, the caller ID must be programmed at the phone, that is called MSN = Multiple Subscriber Number. Whenever a caller picks up this phone to make a call, the extension, that equals the caller ID, is searched. If the extensions should be “200”, the MSN number must also be 200. Also the extension must exist. Create an extension first, as described in chapter “Configuration”.

If a call is made to the extension, the call is forwarded to all internal ports, that have been configured in the extension’s configuration file. If a phone on the internal port should ring, it must be configured to ring for the extension number, which equals to the MSN number.

Whenever a call is made from the extension, the **caller ID** is “screened” to whatever configured in the extension’s settings. Normally the number, that must be dialed from extern to reach the extension, should be used.

If the internal phone makes an **anonymous** call, the call is forwarded also anonymous by the PBX, regardless what is configured in the extension’s setting.

5.3 Calling intern

If the call is made from **internal** to internal, the caller must dial the code, that is specified in "numbering_int.conf" for the extension to be dialed. If the call is made from **external** to internal, the caller must dial the code, that is specified in "numbering_ext.conf". Also the directory tree of the given extension must exist in "/usr/local/pbx/extansions/". All phones of the extensions will ring, as specified in "/usr/local/pbx/extensions/<extension>/settings".

Example: If "`200 intern`" is specified in "numbering_int.conf" / "numbering_ext.conf", the caller must dial "200" to reach extension 200. If "`0 intern 200`" is specified in "numbering_ext.conf" the caller must dial "0" in order to reach extension 200.

5.4 Calling extern

If a call is made from internal to external, the caller must dial the code, that is specified in "numbering_int.conf". It is standard for most PBXs in the world, to use the "0" for external calls. After dialing the "0", or whatever code is specified, the audio of the external line is cross-connected. The dial tone, and even announcements can be heard now.

During dialing, it is not possible to delete the last number, because it is already sent to the telephone network. PBX4Linux supports deleting, by terminating the call and making a new one. The delete function must be enabled. When disabled the digits are dialed as received. Whenever a "*" is received, the last digit is removed from the dial string, and the call is terminated and reestablished by a new one. By dialing "#", the complete number is removed from the external call, an external dial tone is given. Example: "555124*12" will dial the number "5551212", because the "4" is removed by the "*".

Info: In good old Germany, dialing of the digit '0' was called "Amt", which is the short form for "Postamt" (post office). This term is used, because the old telephone system was controlled by the German "Post", and all the switching systems were located in the buildings of post offices. Earlier the calls were made by operators. The operator was sitting in the post office "Amt".

5.5 Calling H.323

If the call is made from internal to H.323, the caller must dial the code, that is specified in "numbering_int.conf". After that, the caller may dial the number in these formats:

- **Numerical**

The IP number must be dialed, by using 3 digits for all four parts. If the IP number is "192.168.0.1", it must be dialed "192168000001". Always three digits must be dialed for each part, even if parts of the IP number have less than three digits. Add zeros in front, to make each part of the IP number three digits long. After 12 digits, the number is complete, and the call is proceeding. This can be dialed from any phone, even if the phone has a rotary wheel, that has no "*" nor "#".

- **Dot notation**

The IP number must be dialed, by using the "*" key to enter the dot of IP numbers. The "#" key is used to finish dialing. The call is then proceeding. The IP number "192.168.0.1" may be dialed as "192*168*0*1#", or even dialed as the short form of IP numbers "192*168*1#".

- **Dot notation + port**

If a different port than 1720 should be used for the H.323 call, the port must be added right behind the IP number, and separated with a "*". If the IP number "192.168.0.1" and port 1722 should be dialed, the caller must dial "192*168*0*1*1722#". In this case, the short form of IP numbers are not allowed.

- **Dot notation + port + alias**

If the remote H.323 host is a gateway, the caller may also tell the gateway what to dial. The number, the gateway should dial, is specified using the H.323 alias. The alias may be dialed behind the port separated by a "*". If the port should not be dialed, two "*" must be dialed between the IP and the alias. If the IP number is "192.168.0.1", the port is default, and the alias is "1234", the caller must dial "192*168*0*1**1234#". If the port is 1722, the caller must dial "192*168*0*1*1722*1234".

- **Full H.323 address**

A full H.323 address may be: "`[alias]@IP[:port]`". This notation cannot be dialed at all. This address must be specified for an abbreviation or defined at "numbering_inf.conf". The "@" is used to identify this notation. If no alias is given, the "@" will be removed when making the H.323 call, since OpenH323 uses the "@" only in conjunction with a given alias.

It is possible, to define an incomplete prefix or the complete H.323 address in "numbering_int.conf" and "numbering_ext.conf". Here are some **examples** for incomplete and complete prefixes:

- Example: "`5 h323 192168000`"

If this is specified in "numbering_ext.conf", any external caller dialing a "5" must complete the IP number by dialing the last 3 digits of the IP number. This only works on PBX lines of course, since PBX4Linux must receive three more digits in order to complete the number.

- Example: "`5 h323 192.168.0.1:1722/`"

This number is incomplete, because the alias is missing behind the "/". It may also be specified as "192*168*0*1*1722*". The caller must dial the alias, and finish dialing with the "#" key.

- Example: “`5 h323 remote.host.foo/`”

This number is incomplete, because the alias is missing behind the “/”. The caller must dial the alias, and finish dialing with the “#” key.

- Example: “`5 h323 @192.168.0.1`”

This number is complete, because all numbers with the “@” are complete numbers. The call will proceed.

- Example: “`5 h323 1235@h.323-gateway.foo:1722`”

This number is complete, because all numbers with the “@” are complete numbers. The call will proceed. The alias “1235” and the port “1722” is used. The port, as well as the alias may be omitted.

- Example: “`5 h323 remote.host.foo`”

This number will cause an **error**, since it has no valid IP address. If the “@”, the “/” or the “:” would be included, the host would be complete, and not be checked for being a valid IP address. In this case, the call will disconnect with an error.

- Example: “`5 h323 @remote.host.foo`”

This number is complete, because the “@” is used. The call will proceed.

5.6 Caller/Connected ID (CLIP/COLP)

The caller ID is transmitted from the caller to the called party. (CLIP) When the called party answers the call, the connected ID is transmitted from the called party to the caller. (COLP) So why is the connected ID transmitted? Since the called party number is known, because it has been dialed! The answer: The called party may have call forwarding enabled or may answer the call on a different telephone than that one that has been called.

The ID for internal calls may be specified in the extension’s settings. If the caller uses anonymous caller ID, it will be forwarded anonymous. Only the police or any other party with special rights may see even anonymous IDs to call back in an emergency.

5.7 Conference

PBX4Linux also supports conferences with an unlimited number of parties. Actually, PBX4Linux always use conferences for all calls. If a normal call is made, a conference with two parties is set up. The call can be put on hold, to answer an incoming call, or to set up a new one. The ISDN telephone must be used to put the call on hold. The PBX can join a parked call with the current active call. This may be done by using DTMF tones. Read the “Keypad” section for information on how to join calls.

Whenever a conference is put on hold, all parties connected in a conference may continue to talk. If there is only one party in the conference, who did not park her call, she will hear the

hold-music and get a notification, that the call is currently inactive, until someone else retrieves the call.

If any party in a conference hangs up, the party is removed from the conference. If only one party is left, the party will also be disconnected.

5.8 Keypad

Keypad dialing is used to control the call, after it has been connected. The keypad may be dialed using DTMF tones, but also using the ISDN keypad facility. The keypad facility is not supported by all ISDN telephones, so DTMF may be used in most cases. DTMF may be recognized even if no DTMF tone is sent. This may happen during analyzing of voice. Sometimes small parts of the human voice equal to DTMF tones. Therefore a sequence is used, whenever a call is connected. The sequence must be dialed quickly. If pauses of more than 2-3 seconds is made between the digits, the sequence is not detected.

Keypad	DTMF	Function
0	*0#	Disconnect the current call and give a dial tone
3	*3#	Join the current call with the call on hold.
7	*7#	Enable encryption using manual keying. (explained later)
8	*8#	Enable encryption using automatic keying. (explained later)
9	*9#	Disable or abort encryption.

When a call is put on hold, or the call has been disconnected using keypad, the caller may dial using DTMF or ISDN keypad facility. If a call has been disconnected, or if the caller wishes to get back to the dial tone because of wrong dialing, the caller must enter "#" twice, within two seconds.

5.9 Callback

The callback makes only sense, if dialed from external telephones. The caller must dial the code for callback, as specified in "numbering_ext.conf". If the caller doesn't transmit a caller ID, callback is not possible at all. If the feature "CLIR Ignore" is available to the ISDN line, that may only available to police or emergency phones, callback is possible, because the caller ID is available, even though it is restricted. If the ID is not available, the call will be disconnected with an error, indicating that the service is not available. After dialing the callback code, the caller may continue dialing the number, that will be dialed after callback. This feature is only available with PBX lines. The number of digits that may be dialed after the code for callback depends on the maximum number that will be transmitted by the telephone network. It may be enough to dial an abbreviation from the phonebook. If nothing is dialed after the code for callback, the dial tone will be presented after callback.

As soon as the caller hangs up after dialing the callback code, the PBX4Linux will wait 2-3 seconds, and start calling back. The caller ID of the calling party will be used to indicate the callback. This only works if the external line has the "CLIP No Screening" feature. After the caller answered the call, she must enter the extension's password, in order to use the callback. If the password is correct, it will be stored in the extension's password list "/usr/local/pbx/extensions/<extension>/callbackauth". If the password is already specified, the

PBX will directly dial, what has been dialed after the callback coded. If nothing has been dialed, the dial tone is presented.

After callback, the DTMF feature is used to input digits. The caller must dial the same number as she would dial when she picks up an internal phone.

If a call gets disconnected, the caller may dial "#" twice, to get a dial tone again, without doing another callback. During a call, the caller must disconnect using "*0#" sequence. To end the callback, the caller must hang up.

Callback gets rejected during trigger call if

- the caller ID is not available.
- the extension is missing in 'numbering_ext.conf' or not existent.
- the password is not given.

The call gets disconnected after callback if

- the password input timed out.
- any digit is wrong after entering the number of digits equal to the password.
- the caller hangs up.

5.10 Test mode

The test mode is used to make certain tests for internal and external telephones. The code for test mode may be specified in "numbering_int.conf" and "numbering_ext.conf". For external calls using the test mode, the external line must be a PBX line, to support the dialing of more digits after the test mode code. If the code is given in the file "numbering_ext.conf" as prefix, the test mode can be used with multipoint lines also. Here is a list of tests that can be made:

➢ **1**

The call is proceeding. A 'proceeding' signal is sent to the calling ISDN telephone.

➢ **2**

The call is alerting. An 'alerting' signal is sent to the calling ISDN telephone.

➢ **3**

The call will be connected. The caller ID of the caller is sent back as connected ID. The audio is echoed (looped). This is useful to test the audio roundtrip delay.

➢ **4**

The call will be connected with a continuous test tone. This is good for testing interrupt problems, loss of data results in gaps in the tone.

- **5**

The call will be connected with the hold music.

- **6<cause>**

After dialing 6 and three more digits, the call will be connected, and the announcement, that will be specified by the ‘cause’ value. The ‘cause’ value is a number between 1 and 127. If the ‘cause’ value has less than three digits, it must be filled with zeros in front. This is useful to test announcements.

- **7<cause>**

This is the same as the previous function, except that the call gets disconnected. The ‘cause’ value is sent with a disconnect message. This is useful to see how the telephone reacts on different ‘causes’ and how the telephone system interprets the ‘cause’ value. On external calls the call will be disconnected, and the announcement of the telephone system is used.

- **8**

Sends back a ‘disconnect’ message.

- **9**

Connects the call and sends back the text “Welcome to Linux” and “12345678” as connected ID. This is useful to test COLP and display text.

5.11 Understanding Causes

A “cause” value is a number between 1 and 127. It is used to explain why a call cannot be completet. ISDN telephones use them to give text messages on its display. The public telephone system uses them to send announcements, which explain the ‘cause’ value. All cause values are defined in Q.850 of ITU-T standard. In the source file “cause.c” is a list of all causes.

Example: If the PBX4Linux discovered, that a dialed number doesn’t exist, it sends a ‘disconnect’ message with the cause value “1”, that means that the number is not allocated. Someone in the USA would then get the announcement: “The number, you have dialed, is not assigned.” In Germany the announcement would be: “Kein Anschluß unter dieser Nummer.”

5.12 Answering machine (voice box)

The answering machine is divided into two parts. The first part answers a call, plays an announcement and records a message. The second part plays the recorded messages. There are two ways to record messages. The easiest way is to use a dialing code in “numbering_ext.conf” or “numbering_int.conf”:

<code> vbox-record <extension>

When the given code is dialed, the answering machine of the given extension is called. The other way is to forward the call to the answering machine. Instead of entering an external telephone number at **cfu**, **cfb** or **cfnr**, the key word "vbox" will forward the call. To forward to the answering machine after 30 seconds of ringing without answer, use the following parameters in the settings of the extension:

```
cfnr        vbox
cfnr-delay  30
```

When a call is received on the answering machine, it will play the **announcement, which** is stored at "/usr/loca/pbx/extensions/<extension>/vbox/announcement.isdn". After that, it will record until the time expired, as specified with "vbox_timeout", or if the caller hangs up. It is possible to record during the announcement. This is useful to pretend, that the call is not answered by an answering machine. In this case, the caller might begin to talk during the announcement. Then it is useful to have her voice recorded. To use the feature use the following parameter in the settings:

```
vbox_mode   parallel
```

To just play the announcement without recording, the following parameter is used:

```
vbox_mode   announcement
```

After the announcement is played, the call is disconnected.

To **play** the recordings of the answering machine, a special dialing code must be used:

<code> vbox-play

This only works if it is defined in "numbering_int.conf". Because each extension has its own answering machine, it is required to call from that extension. In order to play back messages from extern, the 'login' code or the callback must be used.

After dialing the code for playback, a voice will speak the menu. If no recording is stored in the answering machine, the voice will tell this first. During the speech, the following digits may be dialed using DTMF or keypad dialing:

"1" Play previous message.

"2" Play current message. The date, time and caller ID is spoken and written to the display. Then the display shows the total recording time and the current time playing. Pressing "2" during intro will directly playback. Pressing "2" during playback, will stop. Pressing "2" after stop, will continue the playback.

"3" Play the next message.

"4" Rewind the current playback. Pressing "4" again, will double the rewind speed whenever it is pressed. If the beginning is reached, the playback will continue to play forward. Pressing "2" during rewind, cause continue to play at normal speed.

"5" Stop the current playback. The menu is spoken afterwards.

“6” Wind the current playback in forward direction. Pressing “6” again, will double the wind speed whenever it is pressed. Pressing “2” during rewind, cause continue to play at normal speed.

“7” Use to record the announcement. After pressing 7, the announcement can be recorded by pressing “1”, played by pressing “2”, and aborted by pressing “3”.

“8” Use to store the current message in the recordings directory. After pressing “1”, the message is stored, and aborted by pressing “3”.

“9” Use to delete the current message. After pressing “1”, the message is deleted, and aborted by pressing “3”. When recording, almost the first second is cut off, to avoid recording the sound of the keypad. After recording, a beep tone is added to inform the caller that the recording will start.

“0” Use to call the caller of the current message. This is only possible, if the caller ID is available and not restricted.

It is also possible, to select one of the functions above, by using “*” (for previous item) and “#” (for next item). On the display, the current function is displayed. To execute, press “0”. The display will only be shown on an internal phone, since external lines cannot receive display information.

To send an email whenever a message is received, give the email address using the “vbox_email” keyword within the extension’s config file:

```
vbox_email       someone@somewhere.com
```

It is also possible to attach the recorded audio file:

```
vbox_email_file  yes
```

5.13 Encryption

Encrypted Calls

Just pick up the phone and call somebody who uses also PBX4Linux, press a key and the call will be encrypted. - It's easy isn't it?

There are two types of encryption, that can be selected:

- **pre-shared keying**
- **automatic key exchange**

Both have advantages and disadvantages. Pre-shared keying requires a key to be shared. It is essential that nobody can eavesdrop the key when it is exchanges. The best way to exchange a key is to meet the other party. Always note that if the key gets stolen before and even after the call, the call can easily be decrypted in the future. If you use automatic key exchange, the key will be erased whenever the call has been disconnected. Also you don't need to share the key before you make a call. The key exchange procedure does this for you. To exchange an

encrypted session key, RSA is used, but this may take some seconds up to one minute to calculate the keys.

The man in the middle is one good argument not to use the automatic key exchange, but later you will see that this can easily be avoided.

Preshared keying

Preshared keying requires some security about handling of the keys:

- The key must be good random.
- The key should not be given to other parties (group of people)
- The key must be stored secure.
- The key can be used in the future to decrypt encrypted calls.
- The key must be exchanged in a secure way.

Both parties must be identified by the caller ID or dialed number. In order to select the correct key, the caller must transmit her caller ID or the called party will not know what key to select. The keys are stored in the extension's directory "/usr/local/pbx/extensions/<extension>/secrets". The first keyword is the number of the remote party. The number must be given as dialed (without the external prefix). For incoming calls the number must be given as shown (without the external prefix). If the dialed number doesn't match the caller ID of the same party, make two entries with the same key, but different numbers. The second keyword is the authentication method. Only "manual" keying is supported. The third keyword is the encryption method. Only "blowfish" encryption is supported. The last keyword is a string that represents the key. The string is converted from ASCII into binary. If the string is given with "0x" prefix, it is converted from HEX into binary. For blowfish use the maximum key length of 448 bits (56 bytes or 112 HEX digits). To enable encryption, both parties will have to press digits '7'. If the telephone sends DTMF tones instead of keypad information, both parties must press '*7#'. The display will show if the key is found or if an error occurred. A special inband audio signal is given for successfull encryption (bell) or failure (forn). This method is very fast, because no calcualtion process is required.

To turn of encryption, use keypad digit '9' or DTMF sequence '*9#'. A horn will indicate the deactivation of encryption.

Automatic key exchange

It is very simple to enable this type of key exchange. It works if "crypto" library is enabled during compilation time. Press digit '8' on your keypad. If your telephone sends DTMF tones instead of keypad information press '*8#'. You should now notice the message "Key Exchange" on the display of your telephone. This only works if your telephone supports display messages. Display messages are essential to avoid man in the middle. The called party should now hear some clicks in his telephone. This is data transmission of the identifaction procedure. Messages are exchanges via audio channel, so you will hear click whenever some message is transmitted.

The automatic key exchange procedure:

Both parties have to press digit '8' or DTMF '*8#' to enable automatic key exchange. This must happen within 10 seconds. Both will send a random number and the cpu power to each other. The first party that selects the automatic key exchange, will get the information about the remote party. The cpu speed an a random number is given. The cpu power is used to determine the fastest computer. If both parties have equal speed, the random number is used. (In case of equal random number, the b-channel seems to be looped.) The party with the faster cpu or the bigger random number is master and will receive "Master" on her display. The other party will get "Slave" on the display. The master calculates and RSA key pair. One of the keys, the public key is sent to the slave. The slave generates a random session key, encrypts it with the public key and sends it back to the master. Only the master is able to decrypt the session key using the private key. The session key is just a random number that is calculated from noise of all B-channels. It can be expected to be "real random".

To avoid man-in-the-middle attack, both parites will see some values of their public key on the display if they are equal, ther is no man in the middle. One party can tell the other about the public key. The voice of the party 'signs the public key'.

Both parties will use the session key to encrypt and decrypt the audio data using "Blowfish" algorithm.

Man-in-the-middle attack is shown now. Imagine a device (the man in the middle) is attached to your phone line. It intercepts the call to the other party. The device pretends to be the other party and will use the automatic key exchange procedure. On the other side it preteds beein you and also does the automatic key exchange procedure with the other party. Now the device can listen to the decrypted transmission, because the transmission is only secure between the device and the parties. If the device is master of both connections, both public and private keys may be equal, so you and the other party will receive the same public keys. In this case both parties are slave, so this is an error. If one party would be master, the device must generate a key pair for the other party. It is not possible to generate a key pair with a given public key. Both parties would see different public keys in their display.

Again, to avoid man-in-the-middle attacks:

- During negotiation, one party must see "Slave" and the other "Master" in their display
- Both parties must see the same public key digits in their display. It is not required to see the complete key.

About the key on the display: Each bit of the key will halve the coincidence of not discovering a man-in-the-middle. If you compare the first block (4 hex digits), you will end up with a chance of 1:65536 not discovering a man-in-the-middle. If your compare all four blocks (16 hex digits), the chance will be 1:18446744073709551616.

"My display shows a display message three seconds, then it is gone"

Don't worry about your display, just press keypad '8' or enter DTMF '*8#' again and you will see the last message again.

Now the blowfish works:

The Blowfish encryption and decryption is done by the kernel module "mISDN_dsp.o", that is also used for real time audio processing. A special command is used to enable and disble the audio transmission.

The audio is processed in the file "dsp_blowfish.h". A block of 9 samples is converted into a seven-bit-sample. The resulting 63 bits plus another random bit is used to encode a block of 64 bits using blowfish. The block is then converted into 8 bytes of seven bits plus one byte of 8 bits. The upper bit of the first 8 bytes are used for sync information. Decoding is done by reversing the process.

Encryption in a conference:

It is not possible to encrypt a conference with three or more parties. But it can be done before the parties are joined within a conference. Let's say you like to have a secure conversation with two persons, both have also PBX4Linux. Call the first person and start encryption. Put the person on hold and call the seconds person. Start encryption on the second call. Join the calls as usual (keypad '3' or DTMF '*3#'). Check the public keys with each party before you join, because in a conferrence it is not possible to redisplay them.

5.14 Other tools

`$ gentones`

This tool generates tones. When it is called without parameters, it will show the following help:

```
Usage:
gentones wave2alaw <wav file> <alaw file>
gentones wave2ulaw <wav file> <ulaw file>
gentones tone2alaw <frq1> <frq2> <length> <fade in> <fade out> <alaw file>
gentones tone2ulaw <frq1> <frq2> <length> <fade in> <fade out> <ulaw file>
Length and fade lengths must be given in samples (8000 samples are one
second).
Tones will append to existing files, wav files don't.
```

It can be used, to generate law encoded tones with one or two frequencies. This is mainly used to create patterns, like dial tones, busy tone and other audio patterns. Depending on your telephone system, you need to use “tones2alaw” or “tones2ulaw”. There are two frequencies to specify “frq1” and “frq2”. If only one frequency should be used, enter 0 for “frq2”. The “length” specifies the total length of the tone in samples. 8000 samples are one second. 2000 samples are ¼ of a second. “fade in” and “fade out” is used to make the start and stop of the tone soft. If “fade in” is 800, the sound will fade in within 1/10 of a second. This makes tones very smooth and avoids the ‘click’ sound at the beginning. “fade out” respectively. Whatever specified for “fade in” and “fade out”, the tone will be as long as given at “length”. If you don’t want to use fades, set “frq1” and “frq2” to “0”. I suggest at least 50 sample for fade to silence.

The given file will be appended to “alaw file” or “ulaw file”. If you like to create a new one, but the name still exist, delete the file and then start creating it. This is useful to create a

sample with more than one tone. Example: A busy tone normally is made out of ½ second of a tone, and one ½ second of silence. To add silence, just enter 0 for "frq1" and "frq2".

It is also possible to convert a wave file to a-law or mu-law. The wave file **must** have a sampling rate of 8000. It doesn't matte what bit-resolution or how many channels it has (stereo or mono). The sampling rate will not be converted, so it must be 8000 samples/second. The resolution should be 16 bits for best quality. a-law and mu-law have better resolution than 8 bits. The data of a-law and mu-law is 8 bits sample, but since it is quantised, the quality will be 12 bits. 12 bits sounds almost as good as 16 bits. Wave files are only available with 8 or 16 bits resolution.

The use of a-law or mu-law files, result in a faster processing, since the samples must not be converted into a-law or mu-law for ISDN use. ISDN uses only a-law or mu-law samples.

`$ genwave`

To convert an a-law or mu-law file into a wave file, "genwave" is used. If it is called without parameters, the following help is given:

```
Usage:
genwave ulaw2wave <alaw file> <wav file>
genwave alaw2wave <alaw file> <wav file>
```

Use "ulaw2wave" or "alaw2wave", whatever your law-samples are encoded in. The output file must be given with ".wav" extension. The sample is converted into 16 bits, mono, 8000 samples/s. "alaw" or "ulaw" files are always mono with 8000 samples/s.

Use this tool to convert recordings of calls or answering machine, if they are stored as a-law or mu-law samples.

5.15 Parking a call

Using the HOLD feature

A call can be parked by using the "hold" feature of the ISDN telephone. This should be supported by any telephone. The call will not be released, but only the B-channel. The hold feature is useful to answer or setup a new call while holding the other. If a waiting call without a B-channel should be received, the hold feature must be used in order to answer the call. PBX4Linux supports this feature. There is no limit for the number of calls that can be placed on hold. Most telephones only support one call to be on hold. If a B-channel is available, a call on hold can be retrieved. If no channel is available, the active call must be put on hold first. All this is done by the telephone, so you just need to switch between the calls.

During a call on hold, the remote party will get a notification, that the call is on hold, and receive the hold-music. If the call is a conference, the call gets removed until retrieved.

Using the SUSPEND feature

A call can also be suspended (parked). This feature should be supported by all ISDN telephones, but is not used by many people. An active call can be parked by using a code. This can have up to 8 digits. Some telephones support only two digits, some use fixed digits and some support no digits. The digits are used to identify the call. The call can then be resumed on a different phone. It is even possible to resume a call on any other port of the PBX. This allows to transfer a call or to move a telephone to a different port.

Also during suspend, the remote party will get a notification, as described above for the 'hold' feature.

Please refer to the telephone's manual.

NOTE: These features are only supported internally and not apply to external lines. It is not possible to hold/suspend an external call yet. This is also not possible for most PBX-lines.

6

Tuning

6.1 Load tones into memory

To reduce jitter, caused by hard disk access, edit the file “options.conf” and enable the “**fetch_tones**” option. Specify all tone set that should be loaded into memory. Read the “Configuration” section for option description. This requires more memory, but needs no more hard disk load, when playing tones and announcements.

7

Debugging

7.1 Logging

To be done…

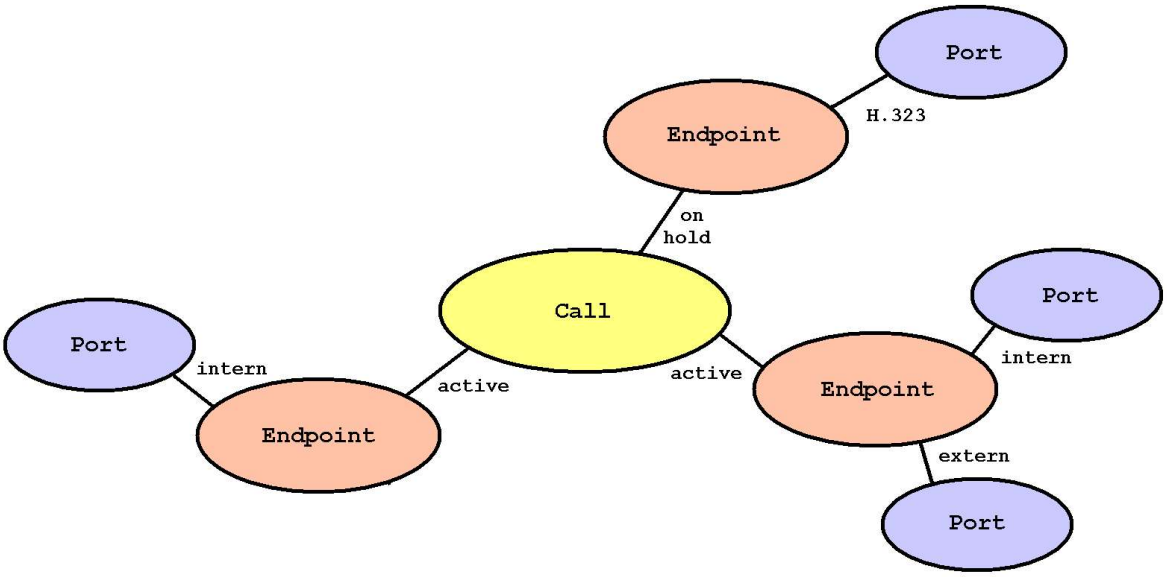
7.2 mISDN debugging

To be done…

8

Architecture

8.1 port, endpoint, call



(Figure: Structure of PBX4Linux)

This picture shows the main structure of the PBX. There are three classes of objects involved:

- **Port**

Whenever someone picks up a phone, or a call is received from extern, a **port** handles the incoming call. Also if someone is called, a port handles the outgoing call. All ports exist, only if a connection is created. An incoming ISDN call creates a port of type ISDN. After a port receives an incoming call, it creates an endpoint, and sends messages to it. An endpoint can have multiple ports, so it is possible to ring a call on multiple ISDN ports. The port class is defined in “port.h”, and implemented in “port.cpp”, “isdn.cpp”, “h323.cpp”, …. The base class “port.cpp” is incomplete and does nothing. It is used to provide functions, which all types of ports use. The is inherited by “isdn.cpp”, “h323.cpp”, ….

- **Endpoint**

An **endpoint** is created, whenever a port receives a call. An endpoint acts as a call manager. It controls the dial tone, does the processing of the dialling, and creates a call, in case of internal, external or H.323 calls. An endpoint can have only one call. The endpoint class is defined in “endpoint.h”, and implemented in “endpoint.cpp”. Because the endpoint controls the PBX, it is large and therefore split into other files: “dialing.cpp”, “dialing_vbox.cpp”.

- **Call**

A **call** is created by an endpoint, whenever an internal, external or H.323 call is made. A call may have an unlimited number of endpoints. If an endpoint disconnects from the call, or sends any messages, no other endpoint will get a note of it. If there are only two endpoints left in a call, the messages from one endpoint are transferred to the other endpoint. If one endpoint releases, the call gets released, and the release is transferred to the other endpoint. If a call is created, the call creates another endpoint and transfers set-up information. The call class is defined in “call.h”, and implemented in “call.cpp”.

- **Messages**

All entities communicated via messages. The messages are buffered, so that they are sent after the entity is processed. The message contains a serial number, where the message was from, and where it is sent to. If an entity is removed, but there is still a pending message, the message is dropped. An entity will be created, and receives a set-up message afterwards. An entity will be removed, when it receives a release message. If an entity is related to others, it will send a release to the other entities before it is removed. Most messages are closely related to ISDN messages with their information and functions, but expanded for more information than ISDN offers. Messages are stored in a forward linked chain of structures. A message is appended to the end of the chain, and removed from the beginning. All messages are defined in “message.h” Functions to send and retrieve a message are implemented in “message.c”.

- **A simple call procedure**

An ISDN telephone, which is connected to an internal ISDN port, is picked up. It sends a set-up message to the ISDN port. A **port** entity receives that set-up message, and creates an **endpoint** entity. This endpoint sends back a message, that it needs more digits, and another message to the port, that the dial tone should be streamed. The caller may now enter digits. These digits are sent to the endpoint entity via the port entity and are processed. Let's assume that the caller dials the code for external calls. The endpoint received the digits for external calls, creates a **call** entity and sends a set-up message to the call entity. The call entity receives the set-up message and creates an **endpoint** entity, and sends the set-up message to it. The endpoint entity receives the set-up messages, and selects a **port**, as specified by the type of number that has to be dialled. In this case it is an external call, so a free port, which connects to an external line is searched. If it is found, the set-up message is sent to that port. From now on, all call-state messages are transferred via this port-endpoint-call-endpoint-port chain. The call entity also interconnects the endpoints, by sending mixer information to the isdn ports, receiving audio data from the endpoints, and sending audio data down to the endpoints.

8.2 Tones and announcements

For the PBX4Linux, all tones must be defined as samples. These samples are **a-law** or **mu-law**, or even **wave** encoded.

The formats are:

- a-law/mu-law: (8 Kbytes/s) (depending on the global mode defined in options.conf)
- wave 16 bit mono: (16 Kbytes/s)
- wave 16 bit stereo: (32 Kbytes/s)
- wave 8 bit mono: (8 Kbytes/s)

Each tone can have two files. The first file has the extension ".isdn" or ".wav" and defines the introduction of the tone pattern. The second file has the extension "_loop.isdn" or "_loop.wav" and defines the endless loop of that tone.

If the first file does not exist, the second loop file is played right as the tone starts. If the second loop file does not exist, the first file is played and silence is played afterwards. If no file exist, silence is played.

Tone names an their occurrence:

➢ **NULL (null pointer) or ""**

Silence

➢ **"busy"**

The called user is busy.

➢ **"dialtone" / "dialpbx"**

An internal dial tone is played and the PBX is waiting for dialing information. "dialpbx" is used for internal dial tone, "dialtone" if the external dial tone is overridden.

➢ **"dialing"**

After receiving 'dialing' information, this tone is played.

➢ **"proceeding"**

The dialed number is complete.

➢ **"ringing" / "ringpbx"**

The called destination is ringing. "ringpbx" is used when an internal phone rings. "ringing" is used if the external ring tone is overridden.

- **“error”**

An unspecified error has occurred.

- **“release”**

The call is disconnected and the PBX waits for release of the call (hang-up of the phone).

- **“ring”**

Sound of a ringing phone. This is used to indicate an incoming call during connected-mode.

- **“test”**

A sine test tone with 1000 Hz.

- **“hold”**

Hold music.

- **“redial”**

This is an indication during power dialing. A short tone (should not be more than one second), indicates, that the call failed after redialing. It is useful to indicate when the call failed and a retry is done.

- **“cause_xx”**

The call cannot be completed as dialed. The given ‘cause’ with the hex number “xx” is played. If the file for the ‘cause’ doesn't exist, the error file is played. ISDN provides ‘cause’ 1-127 which is "cause_01" - "cause_7f". Some ‘causes’ cannot be played; because they occur when audio transfer is not possible.

- **“password”**

When entering the password, this prompt is given.

- **“activated”**

Function activated.

- **“deactivated”**

Function deactivated.

- **“crypt_on”**

Indicates that encryption is now enabled.

- **“crypt_off”**

Indicates that encryption is now disabled or has failed.

- **“cause_80”**

The given extension (msn) is not registered.

- **“cause_81”**

This indicates unauthorized call from extension.

- **“cause_82”**

This indicates unauthorized external call from extension.

- **“cause_83”**

This indicates unauthorized national call from extension.

- **“cause_84”**

This indicates unauthorized international call from extension.

- **“cause_85”**

This indicates unauthorized number or prefix.

- **“cause_86”**

The dialed extension doesn't exist.

- **“cause_87”**

This service is not authorized for the extension.

Additional tone sets can be downloaded from ‘http://isdn.jolly.de/download’.

The **answering machine** (voice box) uses own samples sets. They are installed in “/usr/local/pbx/vbox_english” and “/usr/local/pbx/vbox_german”. I will not describe the tones here. They are self explaining. Note that some tones have short pauses, some have long pauses, and some have no pause at the end.

9

Support

9.1 Mailing list

To be done…

9.2 Buying NT-Mode capable ISDN card

Check out what cards can be used. At http://isdn.jolly.de you will find a list of supported cards and brand names.

9.3 Getting an NT1 / NTBA

Every phone company deals with ISDN NTs. Go to the service men of your local telephone company and ask for the trash bin. Old NTs with expired warranty are thrown away. Service men at telephone companies are very nice people. Almost every one might have an old NT for you if you ask gently. They are broken, but 80-90% of them still have a working power supply. Since the internal electronics is driven by the power from the underground wire, the NT will not disturb the S/T interface. To be sure that the ISDN-coils works, take an volt meter and check if you got about 40 volts between pin 3 and 4, as well as between 5 and 6. ISDN-coils almost never break.

Be carefull: NTs out of the trash might be wet.

Another place is ebay. Many sellers have old (still working) NTs. You may buy one there. Always remember that most phone companies don't sell them, so most offered NTs are still owned by the phone company, even if they are not used anymore. NTs are cheap, so don't feel bad about buying one.

Of course an NT can be bought. Ask your local telephone company to sell one to you.

10

References & Relates Projects

10.1 Asterisk

*Asterisk

Here is a description of the Asterisk project, as it is found at 'http://www.asterisk.org':

Asterisk is a complete PBX in software. It runs on Linux and provides all of the features you would expect from a PBX and more. Asterisk does voice over IP in three protocols, and can interoperate with almost all standards-based telephony equipment using comparatively inexpensive hardware.

Asterisk provides Voicemail services with Directory, Call Conferencing, Interactive Voice Response, Call Queuing. It has support for three-way calling, caller ID services, ADSI, SIP and H.323 (as both client and gateway). Check the 'Features' section for a more complete list.

Asterisk needs no additional hardware for Voice over IP. For interconnection with digital and analog telephony equipment, Asterisk supports a number of hardware devices, most notably all of the hardware manufactured by Asterisk's sponsors, Digium. Digium has single and quad span T1 and E1 interfaces for interconnection to PRI lines and channel banks. In addition, an analog FXO card is available, and more analog interfaces are in the works.

Also supported are the Internet Line Jack and Internet Phone Jack products from Quicknet.

Asterisk supports a wide range of TDM protocols for the handling and transmission of voice over traditional telephony interfaces. Asterisk supports US and European standard signaling types used in standard business phone systems, allowing it to bridge between next generation voice-data integrated networks and existing infrastructure. Asterisk not only supports traditional phone equipment, it enhances them with additional capabilities.

Using the IAX Voice over IP protocol, Asterisk merges voice and data traffic seamlessly across disparate networks. While using Packet Voice, it is possible to send data such as URL information and images in-line with voice traffic, allowing advanced integration of

information.

Asterisk provides a central switching core, with four APIs for modular loading of telephony applications, hardware interfaces, file format handling, and codecs. It allows for transparent switching between all supported interfaces, allowing it to tie together a diverse mixture of telephony systems into a single switching network.

Asterisk is primarily developed on GNU/Linux for x/86. It is known to compile and run on GNU/Linux for PPC. Other platforms and standards based UNIX-like operating systems should be reasonably easy to port for anyone with the time and requisite skill to do so. Asterisk is available in the testing and unstable Debian archives, maintained thanks to Mark Purcell.

A mailing list is available where developers and uses discuss bugs and problem.

10.2 OpenH323

Here is a description of OpenH323, as it is found at ‘http://www.openh323.org’:

„The OpenH323 project aims to create a full featured, interoperable, Open Source implementation of the ITU H.323 teleconferencing protocol that can be used by personal developers and commercial users without charge.

OpenH323 development is coordinated by an Australian company, Equivalence Pty Ltd, but is open to any interested party. Commercial and private use of the OpenH323 code, including use in commercial products and resale, is encouraged through use of the MPL (Mozilla Public license).“

"The aim is to 'commoditize the protocol'. By giving the stack away, maybe we can stop everyone obsessing over how to format the bits on the wire, and get them writing applications instead."
Craig Southeren,
co-founder of OpenH323

A mailing list is available where developers and uses discuss bugs and problem.

10.3 ISDN4Linux

The homepage of ISDN4Linux can be found at ‘http://www.isdn4linux.de’. This is the home of the Kernel driver for ISDN support. Additionally the „isdn4k-utils“ can be downloaded here, in order to configure and use ISDN4Linux. Note that ISDN4Linux is old, and PBX4Linux uses the new mISDN driver. This page has a FAQ and addresses to the mailing lists.

10.4 ITU

To be done…

10.5 Cologne Chip

To be done…

10.6 Werner Cornelius

To be done…

Appendix

Copyright + License

Appendix A. Telephones

To be done…