

Asterisk Tutorial

January 20, 2006

LinuxDays 2006

Patrick Harpes, patrick.harpes@tudor.lu

Johannes Hermen, johannes.hermen@tudor.lu

Contents

1	About Asterisk	4
2	Installing Asterisk	4
2.1	Hardware considerations	4
2.2	Network considerations	4
2.3	Asterisk architecture[1]	4
2.4	Downloading Asterisk	8
2.4.1	Asterisk as pure VoIP server	8
2.4.2	Asterisk as VoIP server in combination with CAPI compatible ISDN cards (Fritz Hardware)	8
2.4.3	Asterisk as VoIP server in combination with hardware phones connected to an HFC-S ISDN card.	9
2.4.4	Asterisk as VoIP server in combination with Zaptel compatible ISDN cards (Digium Hardware)	9
2.4.5	Asterisk as VoIP server in combination with Primary Rate ISDN (PRI) for T1/E1/J1 interfaces.	10
2.4.6	The ztdummy driver	10
2.5	Compiling Asterisk	10
2.5.1	Configure your kernel	10
2.5.2	Install the CAPI drivers for the Fritz card	13
2.5.3	Install the zaphfc drivers for the HFC-S compatible ISDN hardware	13
2.5.4	Loading and testing the zaphfc module	13
2.5.5	Compile and install Asterisk	13
2.5.6	Compile and install zaptel (ztdummy driver)	14
2.5.7	Compile and install chan_capi module	15
2.6	Directory structure	15
2.7	Starting and managing Asterisk server	15
3	Handling SIP Clients	16
3.1	SIP Channel Module [2]	16
3.2	Exercise	18
4	Dial plan[3]	18
4.1	Dial-plan syntax	19
4.1.1	Pattern matching [4]	19
4.2	Introducing Contexts and Extensions [5]	21
4.3	Introduction to Asterisk Dial-plan Priorities [6]	23
4.4	Using Variables in Asterisk Dial-plans [7]	24
4.5	Applications	27
4.6	List of important applications	27
4.6.1	Answer()	27
4.6.2	Wait()	27

4.6.3	Hangup()	28
4.6.4	Playback(filename,options...)	28
4.6.5	Dial()	29
4.7	Exercise	32
5	Call-parking	32
6	Voice-mail	33
6.1	[general] Section	33
6.2	[CONTEXTS] section	34
6.3	Dial-plan entries for Voice-mail	35
6.3.1	Asterisk voicemail() command	35
6.4	Accessing the Voice-mail box	36
6.5	Exercises	37
7	Connecting to the outside world using ISDN	37
8	Connecting to another Asterisk system using the IAX Protocol	39
8.1	IAX Authentication [9]	43
8.2	Exercise	44
9	Integrating your existing ISDN equipment	44
9.1	Hardware setup	44
9.2	Configuring asterisk for ISDN phones	45
10	Softphones	46
10.1	Sip Phones	46
10.1.1	GnomeMeeting	46
10.1.2	KPhone	47
10.1.3	SFLPhone	48
10.1.4	X-Lite	48
10.1.5	SJphone	49
10.2	IAX Phones	49
10.2.1	KIAX	49
10.2.2	Idefisk	50

1 About Asterisk

Asterisk is an open source software PBX, created by Digium, Inc. and a continuously growing user and developer base. The source code of Asterisk has been licensed under the GPL. Asterisk can be downloaded at www.asterisk.org. It runs on Linux, BSD and MacOSX and provides many features needed on a PBX. Asterisk does voice over IP in many protocols, and can interoperate with almost all standards-based telephony equipment using relatively inexpensive hardware.

Asterisk can be considered as a star, it combines different technologies to communicate together. To achieve this, Asterisk supports many VoIP protocols and classical switched telephone network protocols like ISDN for example.

2 Installing Asterisk

2.1 Hardware considerations

Before starting installing Asterisk, some hardware considerations should be raised. If you plan to use Asterisk as the main PBX for your telephone system, you should consider that Asterisk is “only” a piece of software that runs on a simple PC. To guarantee a good service, this PC shouldn’t crash due to hardware or software problems. Even if Asterisk runs on “small” PCs, PIII, 800MHz, you should not use an old PC that has been recycled from the secretary. To rely on a stable and robust Asterisk installation, use a server having a RAID disk system and redundant power supplies.

The server should only run Asterisk software to avoid server crashes due to third party software, or due to system overload caused by other services (ex Databases). The operating system of the server should be stripped down. Install only the packages really needed for Asterisk. The kernel of the system should also be adapted for the use of Asterisk, see 2.4.6

2.2 Network considerations

2.3 Asterisk architecture[1]

Asterisk is carefully designed for maximum flexibility. Specific APIs are defined around a central PBX core system. This advanced core handles the internal interconnection of the PBX, cleanly abstracted from the specific protocols, codecs, and hardware interfaces from the telephony applications. This allows Asterisk to use any suitable hardware and technology available now or in the future to perform its essential functions, connecting hardware and applications.

The Asterisk core handles these items internally:

- PBX Switching - The essence of Asterisk, of course, is a Private Branch Exchange Switching (PBX) system, connecting calls together between various users and auto-

mated tasks. The Switching Core transparently connects callers arriving on various hardware and software interfaces.

- Application Launcher - launches applications which perform services for users, such as voice mail, file playback, and directory listing.
- Codec Translator - uses codec modules for the encoding and decoding of various audio compression formats used in the telephony industry. A number of codecs are available to suit diverse needs and arrive at the best balance between audio quality and bandwidth usage.
- Scheduler and I/O Manager - handles low-level task scheduling and system management for optimal performance under all load conditions.

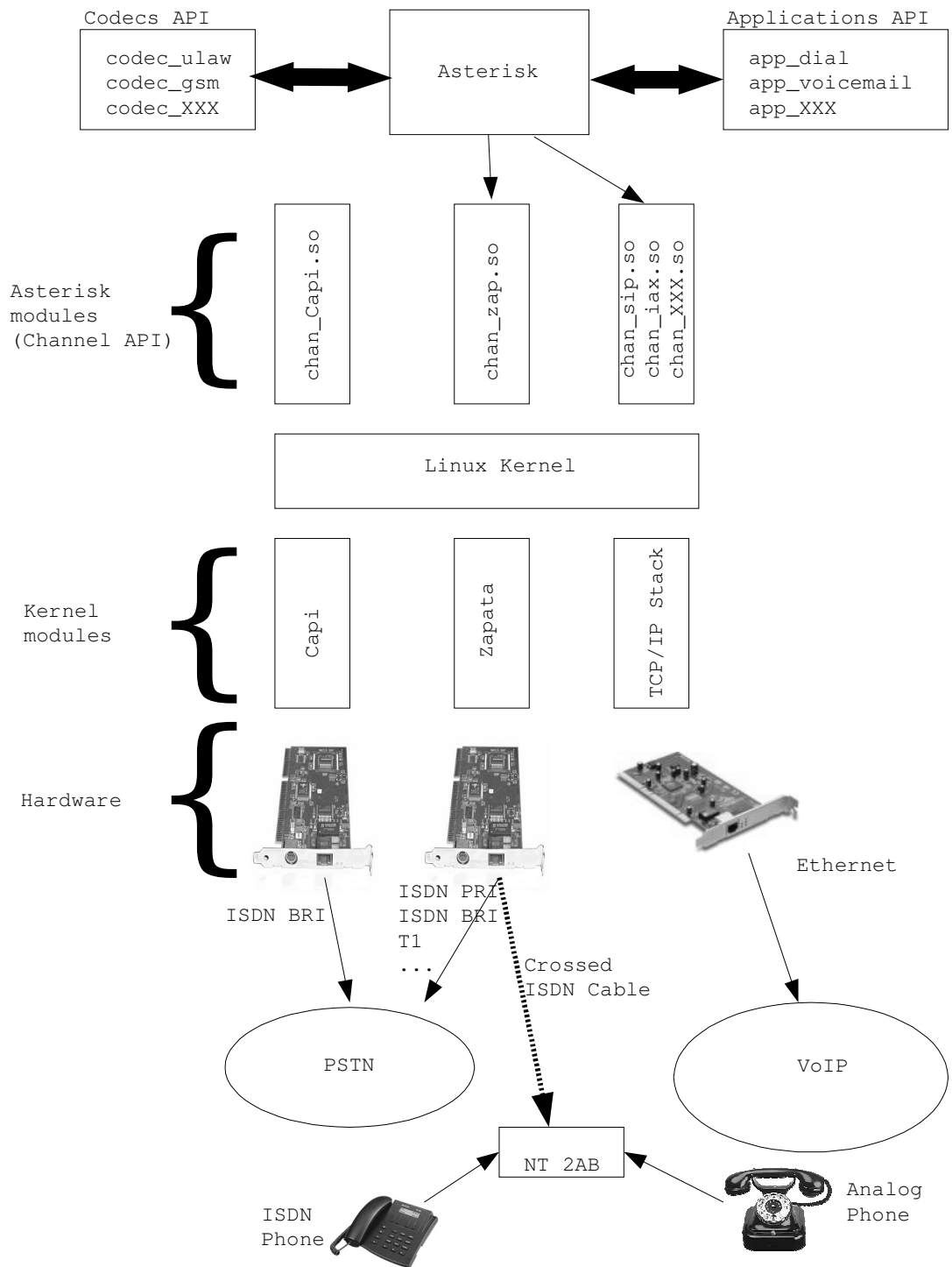
Loadable Module APIs: Four APIs are defined for loadable modules, facilitating hardware and protocol abstraction. Using this loadable module system, the Asterisk core does not have to worry about details of how a caller is connecting, what codecs are in use, etc.

- Channel API - the channel API handles the type of connection a caller is arriving on, be it a VoIP connection, ISDN, PRI, Robbed bit signaling, or some other technology. Dynamic modules are loaded to handle the lower layer details of these connections.
- Application API - the application API allows for various task modules to be run to perform various functions. Conferencing, Paging, Directory Listing. Voice mail, In-line data transmission, and any other task which a PBX system might perform now or in the future are handled by these separate modules.
- Codec Translator API - loads codec modules to support various audio encoding and decoding formats such as GSM, Mu-Law, A-law, and even MP3.
- File Format API - handles the reading and writing of various file formats for the storage of data in the file system.

Using these APIs Asterisk achieves a complete abstraction between its core functions as a PBX server system and the varied technologies existing (or in development) in the telephony arena. The modular form is what allows Asterisk to seamlessly integrate both currently implemented telephony switching hardware and the growing Packet Voice technologies emerging today. The ability to load codec modules allows Asterisk to support both the extremely compact codecs necessary for Packet Voice over slow connections such as a telephone modem while still providing high audio quality over less constricted connections.

The application API provides for flexible use of application modules to perform any function flexibly on demand, and allows for open development of new applications to suit unique needs and situations. In addition, loading all applications as modules allows for

a flexible system, allowing the administrator to design the best suited path for callers on the PBX system and modify call paths to suit the changing communication needs of a going concern.



The figure shows the architecture of Asterisk. On the hardware layer, the physical connections to the different networks are realized. The Ethernet card handles connectivity to the IP network. All VoIP protocols are using this interface to communicate. On top of the Ethernet interface are the Linux Ethernet drivers in the kernel. Asterisk communicates with the IP stack using the different VoIP protocols. For each protocol Asterisk has a specific module (`chan_XXX.so`).

The PSTN channels are handled the same way as the VoIP channels. On top of the ISDN cards are the different Linux Kernel drivers. Some cards can work using the standard CAPI driver, other cards are supported by the zaptel stack. On the Asterisk side these channels are handled using the `chan_capi` module for CAPI compatible cards and `chan_zapata` for the zaptel compatible cards.

2.4 Downloading Asterisk

Before compiling Asterisk, you have to define your needs. We will distinguish 3 special cases of an Asterisk configuration:

2.4.1 Asterisk as pure VoIP server

For this case you only need the Asterisk packages:

- `asterisk-1.2.X.tar.gz`, contains the source code to the base files of the Asterisk Open Source PBX.
- `asterisk-addons-1.2.X.tar.gz` contains the source code to various modules and addons for the Asterisk Open Source PBX.
- `asterisk-sounds-1.2.X.tar.gz` contains default IVR files and miscellaneous sound files.

2.4.2 Asterisk as VoIP server in combination with CAPI compatible ISDN cards (Fritz Hardware)

For this case you need to download the following files from www.asterisk.org:

- `asterisk-1.2.X.tar.gz`, contains the source code to the base files of the Asterisk Open Source PBX.
- `asterisk-addons-1.2.X.tar.gz` contains the source code to various modules and addons for the Asterisk Open Source PBX.
- `asterisk-sounds-1.2.X.tar.gz` contains default IVR files and miscellaneous sound files.

Additionally you need the `chan_capi` module, that is not part of the Asterisk package. Download it from sourceforge.net:

- `chan_capi-cm-0.6.1.tar.gz`

Older versions can be downloaded from junghanns.net.

Finally you need the CAPI driver for your card. It can be downloaded from <ftp://ftp.avm.de/cardware/fritzcrd.pci/linux/suse.93/fcpci-suse93-3.11-07.tar.gz>

For certain applications asterisk requires a timing device. This will be provided by the `ztdummy` driver of the `zaptel` package. see 2.4.6

- `zaptel-1.2.X.tar.gz` contains kernel interface device drivers for Analog and Digital interface cards for the Digium hardware

2.4.3 Asterisk as VoIP server in combination with hardware phones connected to an HFC-S ISDN card.

For this case you need to download the following files from www.asterisk.org:

- `asterisk-1.2.X.tar.gz`, contains the source code to the base files of the Asterisk Open Source PBX.
- `asterisk-addons-1.2.X.tar.gz` contains the source code to various modules and addons for the Asterisk Open Source PBX.
- `asterisk-sounds-1.2.X.tar.gz` contains default IVR files and miscellaneous sound files.
- `zaptel-1.2.X.tar.gz` contains kernel interface device drivers for Analog and Digital interface cards for the Digium hardware

Additionally you need to download the `bristuff-0.3.x.tar.gz` package from <http://www.junghanns.net>. This package contains:

- the `zaphc` kernel module to use ISDN cards with HFC-S chipset as Network Terminator (NT mode).
- patches for the asterisk source code.
- patches for the `zaptel` driver. unpack the `bristuff-0.3.x.tar.gz` package.

2.4.4 Asterisk as VoIP server in combination with Zaptel compatible ISDN cards (Digium Hardware)

For this case you need to download the following files from www.asterisk.org:

- `asterisk-1.2.X.tar.gz`, contains the source code to the base files of the Asterisk Open Source PBX.
- `asterisk-addons-1.2.X.tar.gz` contains the source code to various modules and addons for the Asterisk Open Source PBX.

- asterisk-sounds-1.2.X.tar.gz contains default IVR files and miscellaneous sound files.
- zaptel-1.2.X.tar.gz contains kernel interface device drivers for Analog and Digital interface cards for the Digium hardware

2.4.5 Asterisk as VoIP server in combination with Primary Rate ISDN (PRI) for T1/E1/J1 interfaces.

For this case you need to download the following files from www.asterisk.org:

- asterisk-1.2.X.tar.gz, contains the source code to the base files of the Asterisk Open Source PBX.
- asterisk-addons-1.2.X.tar.gz contains the source code to various modules and addons for the Asterisk Open Source PBX.
- asterisk-sounds-1.2.X.tar.gz contains default IVR files and miscellaneous sound files.
- libpri-1.2.X.tar.gz contains the drivers for PRI T1/E1/J1 interfaces

2.4.6 The ztdummy driver

In Asterisk certain applications and features requires a timing device in order to operate. If you are using Digium hardware, this timing will be guaranteed by the Digium cards. In our case, we don't have Digium cards, so we need the **ztdummy** driver included in the zaptel package. This driver then provides the timing to Asterisk. On Linux Kernel version 2.4.X, the **ztdummy** driver uses the clocking provided by the UHCI USB controller. If your motherboard lacks USB 2.0 support, you have to migrate your kernel to version 2.6.x. This kernel version provides 1-kHz timing with which the driver can interface.

2.5 Compiling Asterisk

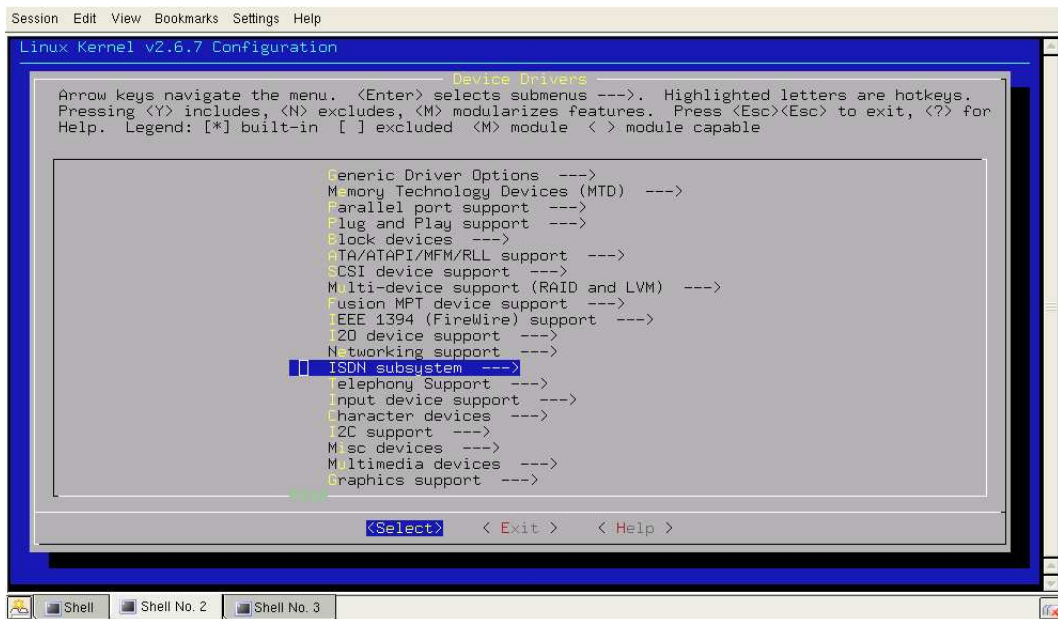
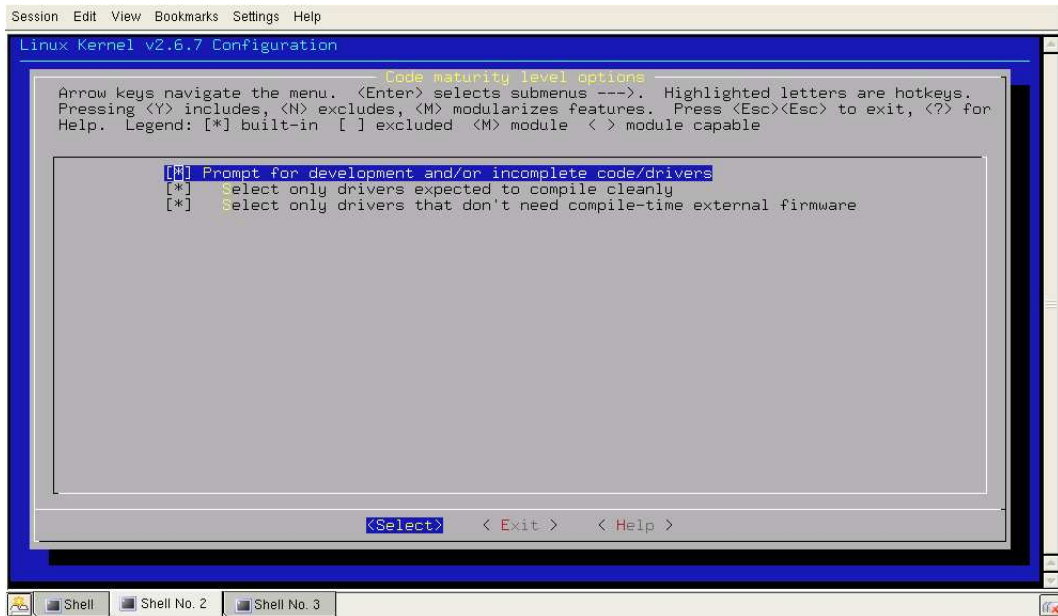
As this tutorials handles only basic rate ISDN cards (Fritz cards) (see:2.4.2) using the `chan_capi` module, this section only explains how to compile and install the corresponding packages.

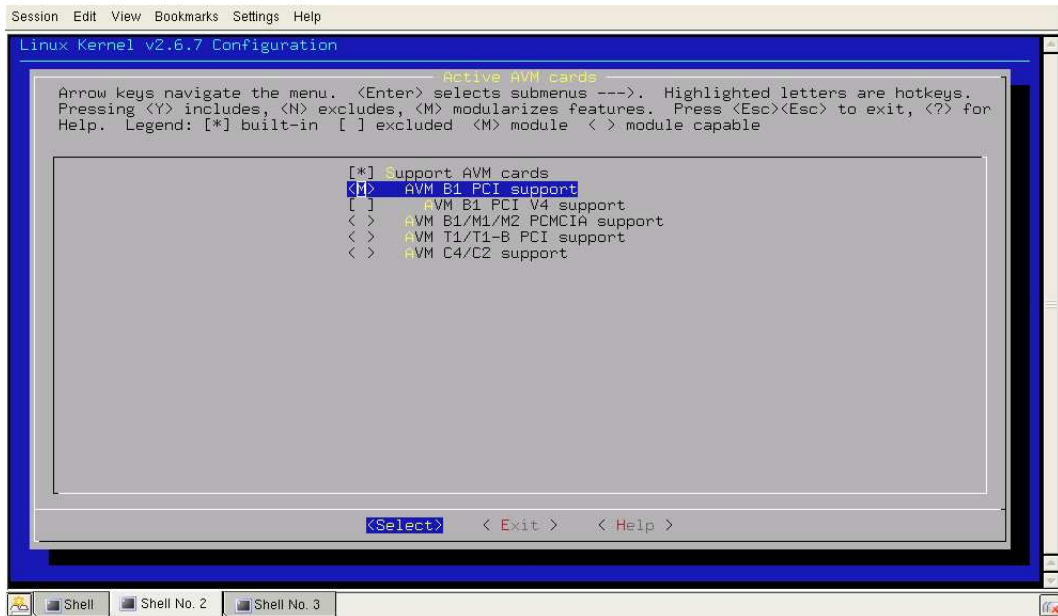
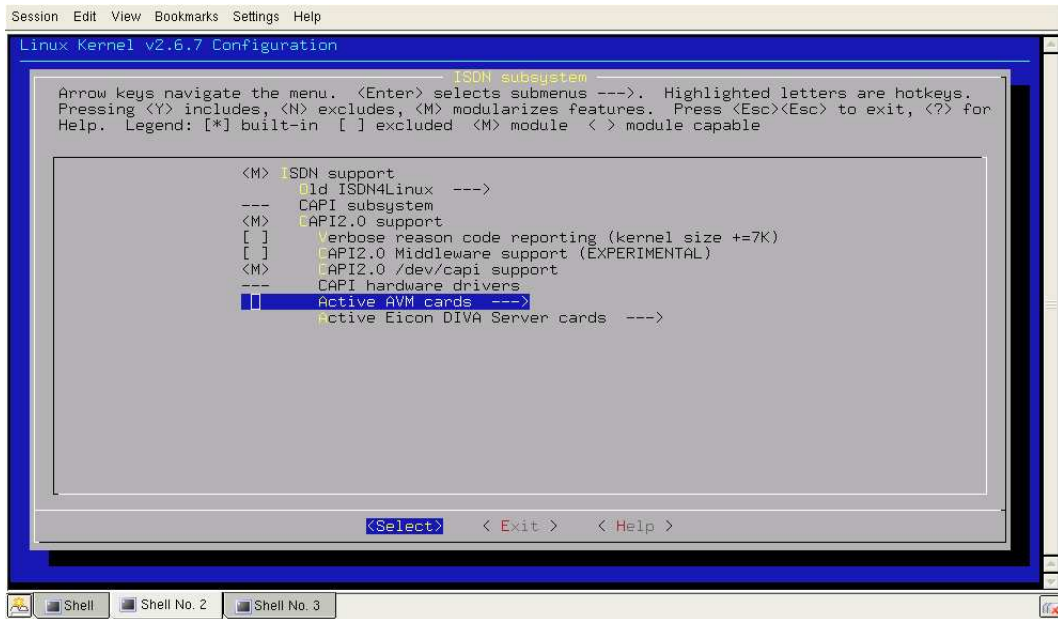
2.5.1 Configure your kernel

Download and install your kernel sources, that will be used in the CAPI compilation. If your kernel lacks CAPI support you have to build a new kernel including support for ISDN and CAPI.

- `cd /usr/src/linux`
- `make menuconfig`

- Check the different options shown in this screen shots:





After the kernel configuration run `make bzImage` followed by `make modules_install`. Copy the new kernel from `/usr/src/linux/arch/i386/boot/bzImage` to `/boot`. Depending on your bootloader, configure `lilo.conf` or `menu.lst`. Reboot the system. When the system comes up, login and try to load the CAPI drivers. With the command `modprobe capi`, the CAPI modules should load into the kernel. Using `lsmod` you can list the modules loaded.

2.5.2 Install the CAPI drivers for the Fritz card

If your kernel supports CAPI, then the second step is to compile the specific CAPI driver for your hardware. In our case we are using the Fritz PCI card.

- Download and unpack the Fritz drivers into `/usr/src/fritz`
- Compile the driver using `make`
- Install the modules using `make install`. This will install the driver into `/lib/modules/2.6.X/extra`
- `modprobe fcpci` should now load the driver for the Fritz hardware

2.5.3 Install the zaphfc drivers for the HFC-S compatible ISDN hardware

Download and unpack the drivers into `/usr/local/src`

- Compile the driver using `make`
- Install the modules using `make install`. This will install the driver into `/lib/modules/2.6.X/`

2.5.4 Loading and testing the zaphfc module

The `zaphfc` module needs to be loaded with special parameters. The Parameter `'modes=1'` to set the first found HFC-card to the NT-Mode. When the module is loaded we need to use the `ztcfg` config tool to initialize the card. We can view the card configuration by taking a look at `/proc/zaptel/1`.

```
modprobe zaphfc modes=1
ztcfg -v
cat /proc/zaptel/1
```

2.5.5 Compile and install Asterisk

Now it's time to go on and compile Asterisk. If you are using Debian Sarge distribution maybe you have to install the following additional packages to be able to compile Asterisk. Use `apt-get install`, or your favorite Debian package manager to install this packages:

- `libssl` developer files
- `zlib` developer files
- `libcapi20` developer files (only for `chan_capi`)

In our example we will unpack the Asterisk archives into `/usr/local/src`

```

# cp asterisk-* /usr/local/src
# cp zaptel* /usr/local/src
# tar -zxvf asterisk-1.2.X.tar.gz
# cd asterisk-1.2.X
# make clean

```

If you want to interate existing ISDN hardware to the Asterisk installation, you have to use the zaphfc kernel module. This driver enables your ISDN adapter to act as master on the ISDN S0 bus. We need to patch the source code of Asterisk to get the zaphfc module to work.

```

# cd /usr/local/src/
# tar -zxvf bristuff-0.3.x.tar.gz
# cd asterisk-1.2.x
# patch -p1 < ../bristuff-0.3.x/patches/asterisk.patch

```

In our configuration we want to install Asterisk into `/usr/local`. This has to be specified in the Makefile. Edit the Makefile and set `INSTALL_PREFIX?=/usr/local`

```

# make
# make install
# make samples

```

For the Asterisk addons package and for asterisk-sound use the same procedure; edit the Makefile and set `INSTALL_PREFIX?=/usr/local`, then run `make` and `make install`.

2.5.6 Compile and install zaptel (ztdummy driver)

The Zaptel package compilation is optional. If you don't have any Digium hardware or your don't need Asterisk applications that depends on exact timing, you can skip this part. In our example, we need the `ztdummy` driver to be able to use the `MeetMe()` application as well as to use the IAX protocol. Furthermore this package is needed to use the zaphfc module for integrating existing ISDN hardware. To use the zaphfc module we have to patch the zaptel package.

```

# cp zaptel-1.2.X.tar.gz /usr/local/src
# tar -zxvf zaptel-1.2.X.tar.gz
# cd zaptel-1.2.x
# patch -p1 < ../bristuff-0.3.x/patches/zaptel.patch
# make
# make install (copies the modules to /lib/modules/2.6.X/extra
# modprobe ztdummy
# lsmod
ztdummy          3176      -
zaptel           228100    -

```

```

capi          7520      -
fcpci        596408    -
kernelcapi   35328     -

```

2.5.7 Compile and install chan_capi module

To use the Fritz!CardPCI ISDN card for out and incoming calls, we have to use the chan_capi module.

- Download chan_capi-cm-0.6.1.tar.gz package from sourceforge.net
- run `make` to compile the package
- run `make install` to install the modules
- run `make config` to install a sample configuration. This step is optional
- In `/usr/local/etc/asterisk/modules.conf` insert the line: `load => chan_capi.so`
- and in the `[global]` section: `chan_capi.so=yes`
- Don't forget a trailing newline at the end of `modules.conf`!

2.6 Directory structure

The following table gives an overview about the Asterisk files:

Path	File types
<code>/usr/local/etc/asterisk</code>	Configuration files
<code>/usr/local/usr/sbin</code>	Asterisk binaries
<code>/usr/local/includes/asterisk</code>	Include files
<code>/usr/local/usr/lib/asterisk/modules</code>	Asterisk modules
<code>/usr/local/usr/share/man</code>	Asterisk man pages
<code>/usr/local/var/lib/asterisk/mohmp3</code>	Music on hold mp3 files
<code>/usr/local/var/lib/asterisk/sounds</code>	Sounds
<code>/usr/local/var/lib/asterisk/keys</code>	Keys needed for de/encryption
<code>/usr/local/var/log/asterisk</code>	Asterisk log files
<code>/usr/local/var/run</code>	Asterisk pid files
<code>/usr/local/var/spool/asterisk</code>	Different spool directories for voice mail, meet_me etc

Certain applications need to have a link from `/var/lib/asterisk` to `/usr/local/var/lib/asterisk`.

```
# ln -s /usr/local/var/lib/asterisk /var/lib/asterisk
```

2.7 Starting and managing Asterisk server

Asterisk can be started in several ways. The easiest way is to start Asterisk by running the binary located in `/usr/local/usr/sbin/asterisk`. However, the preferred way of starting Asterisk is to use the `safe_asterisk` script. This script will put asterisk in the

background. To connect to the Asterisk server, you can use the asterisk binary with the `-r` option:

```
# /usr/local/usr/sbin/safe_asterisk
# /usr/local/usr/sbin/asterisk -r
Asterisk 1.2.0, Copyright (C) 1999 - 2005 Digium.
Written by Mark Spencer <markster@digium.com>
=====
Connected to Asterisk 1.2.0 currently running on asterisk (pid = 13039)
Verbosity is at least 3
asterisk*CLI>
```

At the CLI prompt you can enter commands to manage and configure Asterisk.

One very simple way to start Asterisk at boot and restart it if it crashes, is via Linux's init process. Add the following line to `/etc/inittab`:

```
as:2:respawn:/usr/sbin/asterisk -f
```

The '2' in the entry above is the run level(s) in which you wish the asterisk process to run. You can specify multiple run levels, see `man inittab` for details. Many Debian systems default to run level 2, so setting asterisk to start in runlevel 2 via inittab will result in it always starting at boot time and also being respawned if the process dies for any reason.

Furthermore you have to load the Asterisk related kernel modules on system startup. On a Debian system add the following lines to `/etc/modules`

```
capi
fcpci
zaptel
ztdummy
```

3 Handling SIP Clients

3.1 SIP Channel Module [2]

The SIP Channel Module enables Asterisk to communicate via VOIP with SIP telephones and exchanges. Asterisk is able to act as

- a SIP client: This means that Asterisk registers as a client to another SIP server and receives and places calls to this server. Incoming calls are routed to an Asterisk extension.
- a SIP server: Asterisk can be configured so that SIP clients (phones, software clients) register to the Asterisk server and set up SIP sessions with the server, i.e. calls and answers incoming calls. This said, Asterisk is not a full-feature SIP server like SIP express router. If you are going to have thousands of SIP phones, you should use SER and forward calls to Asterisk for voice mail or PSTN access.

- a SIP gateway: Asterisk acts as a Media gateway between SIP, IAX, MGCP, H.323 and PSTN connections.

As an example, an Asterisk server can be connected to ISDN to give your SIP clients connectivity to the switched telephony network.

Defining SIP channels in `sip.conf`

Each SIP client and server is identified by a block of text that looks like:

```
[xxx]
type=yyy
parameter1=value
parameter2=value
```

Where xxx is the name associated with the SIP client, or is an arbitrary name used by other configuration files to refer to this SIP device. Typically if a SIP phone has an extension number of 123, then its corresponding entry in this file will begin with [123]. Note that you still have to enable an extension 123 in your dial plan to reach this phone. Type is either "user", "peer" or "friend". A "user" type is used to authenticate incoming calls, a "peer" type is used to for outgoing calls and a "friend" type is used for both.

Asterisk matches incoming calls to the name of a device with `type=user` based on the `From: user name` (ignoring the SIP domain). The other way that incoming SIP requests are matched to [xxx] sections in this file, is to examine the IP address that the request is coming from, and look for a peer [xxx] section that has a matching `Host=value`. If `Host=dynamic`, then no match is possible until the SIP client has registered.

Example

```
[general]
srvlookup=yes
disallow = all
allow = alaw
allow = ulaw
allow = gsm
; Johannes
[200]
type=friend
secret=welcome
qualify=yes
nat=no
host=dynamic
canreinvite=no
context=internal
mailbox=200
```

This example shows a sip.conf file where one sip client has been defined. The sip.conf file starts with a `[general]` section, which contains the channel settings and default options for all users and peers defined within sip.conf. You can override the default settings on a per-user/peer basis by configuring them within the user/peer definition.

The sip.conf files can define many parameters. This tutorial will not explain the whole parameter list. For a complete documentation about the parameters, please consult <http://www.voip-info.org/wiki/index.php?page=Asterisk+config+sip.conf>.

The `[general]` section is followed by the definitions of the clients. In our example one client with the extension 200 has been defined. The secret parameter defines the password for this client. Per default, passwords are in clear that might be a security problems. Consult <http://www.voip-info.org/wiki/view/Asterisk+sip+md5secret> on how to use md5 encrypted passwords.

If you turn on the qualify parameter, Asterisk will send a `SIP OPTIONS` command regularly to check that the device is still online. If the device does not answer within the configured (or default) period (in ms) Asterisk considers the device off-line for future calls.

The NAT parameter changes the behavior of Asterisk for clients behind a firewall. This does not solve the problem if Asterisk is behind the firewall and the client on the outside.

The host parameter defines how to find the client - IP or host name. If you want the phone to register itself, use the keyword `dynamic` instead of `Host IP`.

The `canreinvite` defines if the client is able to support SIP re-invites.

All defined extensions belong to one `CONTEXT`. These are defined in `extensions.conf` and referenced in many other configuration files. The `context` parameter defines in which context the client will be.

The `mailbox` parameter defines the Voice mail extension (for message waiting indications)

3.2 Exercise

We should now define one or more SIP clients that use our Asterisk system. All clients should be able to initiate and accept phone calls. They should all be in a context called "InternalSIP". The extensions should be 3 digits long.

- Edit the sip.conf file
- Connect to asterisk using `asterisk -r` command
- Reload the Asterisk configuration using the internal Asterisk `reload` command

4 Dial plan[3]

The configuration file "extensions.conf" contains the "dial plan" of Asterisk, the master plan of control or execution flow for all of its operations. It controls how incoming and outgoing calls are handled and routed. This is where you configure the behavior of all connections through your PBX.

The content of "extensions.conf" is organized in sections, which can be either for static settings and definitions, or for executable dial-plan components in which case they are referred to as contexts. The settings sections are general and globals and the names of contexts are entirely defined by the system administrator. A special type of contexts are macros, label by a user-defined name prefixed with `macro-`. These are reusable execution patterns, alike procedures in a programming language. Every section in extensions.conf starts with the name of the sections contained within square brackets. This gives the extensions.conf file a similar structure to the traditional .ini file format of the Windows world.

4.1 Dial-plan syntax

[general]

- At the top of your extensions.conf file, you configure a few general settings in the section headed [general].

[globals]

- Next, in the [globals] section, you may define global variables (or constants) and their initial values.

Contexts and Extensions

- After the [general] and [globals] categories, the remainder of the extensions.conf file is taken up by the definition of the Dial-plan. The Dial-plan consists of a collection of contexts. Each context consists of a collection of extensions.

Extension Patterns

- When you define the extensions within a context, you may not only use literal numbers, not only alphanumeric names, but also you may define extensions that match whole sets of dialed numbers by using extension patterns.

4.1.1 Pattern matching [4]

Dial-plan extensions can be simple numbers like "412" or "0". They can be alphanumeric names like "john" or "A93*". Although a typical telephone can't dial an extension called "john" (some can though), often your Dial-plan logic will involve jumping from one extension to a different extension, and for those jumps you may define extension names with any name you like, as you don't wish them to be dialed directly.

Of course, touch-tone telephones don't just have the digits 0 through 9, they also have * (star) and # ("pound" or "hash", depending on where in the world you live). And some touch-tone (DTMF) telephones have the extra four "digits", A, B, C and D. If you have such handsets within your organization, there's nothing stopping you making use of those extra buttons for some special purpose of your own.

Note: To have an extension that is triggered by dialing the # symbol, you must use an extension pattern (see below). Asterisk does not recognize # as an ordinary 'digit', even though it appears on all DTMF telephones.

Extension Patterns Extension names are not limited to single specific extension "numbers". A single extension can also match patterns. In the extensions.conf file, an extension name is a pattern if it starts with the underscore symbol (_). In an extension pattern, the following characters have special meanings:

Special Characters for Pattern Matching

- X matches any digit from 0-9
- Z matches any digit form 1-9
- N matches any digit from 2-9
- [1237-9] matches any digit or letter in the brackets (in this example, 1,2,3,7,8,9)
- . wild-card, matches one or more characters
- ! wild-card, matches zero or more characters immediately

Example

Consider the following context:
Context "routing":

Extension	Description
_61XX	Dallas Office
_63XX	Dallas Office
_62XX	Huntsville Office
_7[1-3]XX	San Jose Office
_7[04-9]XX	Los Angeles Office

This context, given the name "routing", sends calls to various servers according to their extension. This organization has decided that all of their telephone extensions will be 4 digits long. If a user dials an extension beginning with 61 or 63, it would be sent to the Dallas office; 62 would go to the Huntsville office; anything starting with 71, 72, or 73 would go to San Jose, and anything starting with 70, 74, 75, 76, 77, 78 or 79 would go to the Los Angeles office.

More Example Patterns

- _NXXXXXX matches a normal 7 digit telephone number
- _1NXXNXXXXXX matches an area code and phone number prekey-pressed by a one

- `_9011`. matches any string of at least five characters that starts with 9011, but it does not match the four-character string 9011 itself.
- `_#` matches a single # keypress

Warning Do not use a pattern of `_.` as this will match everything including Asterisk special extensions like `i`, `t`, `h`, etc. Instead use something like `_X`. or `_X` which will not match special extensions..

4.2 Introducing Contexts and Extensions [5]

The Dial-plan consists of collection of contexts. These context definitions are the most important part of the `extensions.conf` file and are the most important part of Asterisk configuration. A context is just a collection of extensions. Here is a diagrammatic representation of a simple example context:

Context "default":

Extension	Description
101	Mark Spencer
102	Wil Meadows
103	Greg Vance
104	Check voicemail
105	Conference Room
0	Operator

In this example context, which has been given the name "default", the first three extensions (101 to 103) all would be associated with ringing phones belonging to various employees. The fourth extension (104) would be associated with allowing someone to check their voicemail. The fifth extension (105) would be associated with a conference room. Finally, the "0" extension would be associated with the operator.

Here is another simple example of a context:

Context "mainmenu":

Extension	Description
s	Welcome message and instructions
1	Sales
2	Support
3	Accounting
9	Directory
#	Hangup

This example context, given the name "mainmenu", has only single-digit extensions. The "s" extension is the start extension, where the caller begins. This extension would

play a message along the lines of "Thank you for calling OurCompany. Press 1 for sales, 2 for support, 3 for accounting, 9 for a company directory, or # to hang-up." Each menu option is, in fact, an extension and could either dial someone's real extension or could do something like sending the caller to another menu.

Contexts can be used to implement a number of important features including:

- Security: Permit long distance calls from certain phones only
- Routing: Route calls based on extension
- Autoattendant: Greet callers and ask them to enter extensions
- Multilevel menus: Menus for sales, support, etc.
- Authentication: Ask for passwords for certain extensions
- Callback: Reduce long distance charges
- Privacy: Blacklist annoying callers from contacting you
- PBX Multihosting: Yes, you can have "virtual hosts" on your PBX
- Daytime/Nighttime: You can vary behavior after hours
- Macros: Create scripts for commonly used functions

How Are Contexts Used? When Asterisk receives a call connection, whether an incoming call from outside, or from an internal extension, that call belongs to a context. Which context the call belongs to depends on what channel the call came in on. When you configured the channels that you have on your Asterisk PBX, one of the things you had to do is to define what context an incoming connection on that channel would be put into, using a definition like:

```
context=incoming
```

So the first way contexts are used is to make Asterisk do different things depending on where a call is coming from. You pretty certainly will have at least one context defined for how Asterisk handles an incoming call on your telephone line: it might ring some of your extensions or play an announcement and record a voicemail message. You want Asterisk to handle connections from your internal extensions differently ? they will be permitted to dial a different extension, or make an outgoing call ? so you define that calls from those different channels go into different contexts.

4.3 Introduction to Asterisk Dial-plan Priorities [6]

Asterisk Priorities are used within a dial-plan, in the extensions.conf configuration file.

Priorities are numbered steps in the execution of each command that make up an extension. Each priority represents one specific application. Typically, priority numbers start at 1 and increment consecutively for each line in the context. Priority numbers are not always consecutive, but we will worry about that later. For now, just remember that for each extension, Asterisk runs each priority in numerical order ? as opposed to the order in which they appear in the file.

Example

```
exten => 555,1,Answer
exten => 555,2,Playback(tt-weasels)
exten => 555,3,Voicemail(44)
exten => 555,4,Hangup
```

This example is a definition of a single extension with the name "555". When a call is made to extension 555, Asterisk will answer the call itself, play a sound file called "tt-weasels", give the user an opportunity to leave a voicemail message for mailbox 44, and then hang-up.

- "exten =>" tells the dial-plan that the next thing it sees will be a command.
- "555" are the actual digits received (i.e. what the caller dialed, or the "extension" number dialed).
- "1", "2", "3", and "4" represent the priority, which determines the order in which commands for that extension will be executed.

Note that Asterisk does not care about the order in which you put the lines in the extensions.conf file. You could mix the lines into a different order, like this following example, and it would make no difference because Asterisk uses the priority of each line to determine order of execution:

```
exten => 555,4,Hangup
exten => 555,1,Answer
exten => 555,3,Voicemail(44)
exten => 555,2,Playback(tt-weasels)
```

The n Priority (Not supported by stable versions 1.0.9 or earlier) Instead of having to manually number (and renumber) priorities within a given extension, you may use "n" to represent the next priority. The "n" automatically increases a priority's value incrementally. Previously, if you wanted to insert a command between the third and fourth line of a 20-priority extension, you would need to manually renumber each priority after the inserted line.

Example

```
exten => 555,1,Answer
exten => 555,n,Playback(tt-weasels)
exten => 555,n,Voicemail(44)
exten => 555,n,Hangup
```

Using "n" priorities, there are two things you can do that make creating extension logic a lot easier.

The first is that you can set labels:

```
exten => s,n(Start),Answer
```

As well as making the dial-plan more readable, labels can be the target of gotos:

```
exten => s,n,Goto(Start)
```

The second feature that makes using priorities easier is that arbitrary increments can be defined:

```
exten => s,n+2,Dial(...)
```

Maybe its utility is not so obvious, but in conjunction with labels, it can be pretty handy. In a typical dial-plan, Asterisk must often handle the "+101" priority to handle the failure condition of an application, like `Dial()`. Without the `n` priority, if you were to change one of the lines within an extension, not only must the `Dial()` priority change, so must the corresponding +101 priority.

With `n` priority increments, the following expression is possible:

```
exten => s,n(MainDial),Dial(...) ; Dial the main numbers for this context
...
...
exten => s,MainDial+101,Voicemail(u100)
```

Now when new dial plan instructions are added before (MainDial) there is no need to update any priorities.

Note that the "+101" priority may also use a label:

```
exten => s,MainDial+101(MainDialNotAnswered),Voicemail(u100)
```

4.4 Using Variables in Asterisk Dial-plans [7]

Asterisk can make use of global and channel-specific variables for arguments to commands. Variables are referenced in the dial-plan (`extensions.conf`) using the syntax

```
${foo}
```

where `foo` is the name of the variable. A variable name may be any alphanumeric string beginning with a letter. User-defined variable names are not case sensitive, `#{FOO}` and `#{Foo}` refer to the same variable, but Asterisk-defined variables are case-sensitive `#{EXTEN}` works, but `#{exten}` doesn't.

There are three types of variables: global variables, channel variables, and environment variables.

- Global variables can be set either in the `[globals]` category of `extensions.conf` or by using the `SetGlobalVar()` command. Once defined, they can be referenced by any channel at any time.
- Channel variables are set using the `Set()` command (previously `setvar()`). Each channel gets its own variable space, so there is no chance of collisions between different calls, and the variable is automatically trashed when the channel is hungup.
- Environment variables provide a means to access unix environment variables from within Asterisk.

Predefined Channel Variables There are some channel variables set by Asterisk that you can refer to in your dial-plan definitions. Asterisk-defined variables, in contrast to user-defined variables, are case sensitive.

- `#{ACCOUNTCODE}`: Account code, if specified - see Asterisk billing
- `#{ANSWEREDTIME}`: Time when the call was answered.
- `#{BLINDTRANSFER}`: The active SIP channel that dialed the number. This will return the SIP Channel that dialed the number when doing blind transfers - see `BLINDTRANSFER`
- `#{CALLERID}`: The current Caller ID (name and number)
- `#{CALLERIDNAME}`: The current Caller ID name
- `#{CALLERIDNUM}`: The current Caller ID number (Note that this is not necessarily numeric as the name would indicate and can legitimately contain the space character. Commands acting on this variable (such as `'GotoIf'`, for example) should be aware of this).
- `#{CALLINGPRES}`: PRI Call ID Presentation variable for incoming calls (See `callingpres`)
- `#{CHANNEL}`: Current channel name
- `#{CONTEXT}`: The name of the current context
- `#{DATETIME}`: Current date time in the format: `DDMMYYYY-HH:MM:SS`

- `${DIALEDPEERNAME}`: Name of the called party. Broken for now, see `DIALEDPEERNAME`
- `${DIALEDPEERNUMBER}`: Number of the called party. Broken for now, see `DIALEDPEERNUMBER`
- `${DIALEDTIME}`: Time when the number was dialed.
- `${DIALSTATUS}`: Status of the call. See `DIALSTATUS`
- `${DNID}`: Dialed Number Identifier. Limitations apply, see `DNID`
- `${EPOCH}`: The current UNIX-style epoch (number of seconds since 1 Jan 1970)
- `${EXTEN}`: The current extension
- `${HANGUPCAUSE}`: The last hangup return code on a Zap channel connected to a PRI interface
- `${INVALID_EXTEN}`: The extension asked for when redirected to the i (invalid) extension
- `${LANGUAGE}`: The current language setting. See Asterisk multi-language
- `${MEETMESECS}`: Number of seconds a user participated in a MeetMe conference
- `${PRIORITY}`: The current priority
- `${RDNIS}`: The current redirecting DNIS, Caller ID that redirected the call. Limitations apply, see `RDNIS`
- `${SIPDOMAIN}`: SIP destination domain of an inbound call (if appropriate)
- `${SIP_CODEC}`: Used to set the SIP codec for a call (apparently broken in Ver 1.0.1, ok in Ver. 1.0.3 & 1.0.4, not sure about 1.0.2)
- `${SIPCALLID}`: The SIP dialog Call-ID: header
- `${SIPUSERAGENT}`: The SIP user agent header
- `${TIMESTAMP}`: Current date time in the format: `YYYYMMDD-HHMMSS`
- `${TXTCIDNAME}`: Result of application `TXTCIDName` (see below)
- `${UNIQUEID}`: Current call unique identifier
- `${TOUCH_MONITOR}`: used for "one touch record" (see `features.conf`, and `wW dial flags`). If is set on either side of the call then that var contains the `app_args` for `app_monitor` otherwise the default of `WAV|m` is used

4.5 Applications

Applications are the workhorses of the dial-plan. Each application performs a specific action on the current channel, such as playing a sound, accepting touch-tone input, or hanging up the call. In your dial-plan (extensions.conf file) you have to specify for each extension an application to execute.

Syntax:

```
exten => <extension>, <priority>, <application>
```

Example

```
[internal]
exten => 100,1,Answer()
exten => 100,1,Playback(hello-world)
exten => 100.2,hangup()
```

Explanation

If a call in the `[internal]` context with extension 100 is received by Asterisk, the first application (priority=1) launched is `Answer()`. This application picks up the phone call. Then we pass to the 2nd priority of the dial plan for extension 100 in the context `[internal]`. The application launched at this level is called `Playback(hello-world)`. This applications plays the sound file called "hello-world.gsm". At the last level for this call, the `hangup()` application will end the call.

For a complete list and explanations of the Asterisk applications please visit <http://www.voip-info.org/wiki-Asterisk+-+documentation+of+application+commands>. In this tutorial we want to give a focus only on a few very important applications:

4.6 List of important applications

4.6.1 Answer()

If the channel is ringing, answer it, otherwise do nothing.

Return code Returns 0 unless it tries to answer the channel and fails.

4.6.2 Wait()

The Wait command takes one argument, the number of seconds to wait. During the time waited, all sound input received on the channel, including DTMF tones, are silently ignored. The Wait command is typically used before answering a channel.

How to Accept DTMF Keypresses While Waiting The `Wait` command will ignore any DTMF input from buttons pressed by callers on the line. If you want to accept DTMF input while waiting, set the wait time by calling `ResponseTimeout()` and do not define a step at the next highest priority. Asterisk will silently wait up to `ResponseTimeout()` seconds for the user to dial an extension number valid in the current context.

Return Code `Wait` normally returns 0, or -1 if interrupted.

4.6.3 Hangup()

Unconditionally hangs up a given channel by returning -1 always.

4.6.4 Playback(filename,options...)

Plays the specified sound file (you need to omit the filename extension). Sound files are stored in the `/var/lib/asterisk/sounds` directory by default (the directory path can be changed in `asterisk.conf`). Playback is Multi-Language-compliant. It will look in a subdirectory corresponding with the current language code (as set by the `SetLanguage()` command, or the channel's default language code. Failing that, it will play the non-language-specific edition.

`Playback()` will play the whole sound file, and when complete, return control. Compare with the `Background()` command, which plays a sound file but returns control immediately, allowing Asterisk to perform other commands on this channel while the sound file is playing.

Options

- **skip**: Play the sound file only if the channel has already been answered. If the channel has not yet been answered, the Playback command will return immediately without playing anything.
- **noanswer**: Play the sound file, but don't answer the channel first (if hasn't been answered already). Not all channels support playing messages while still on hook.

If neither `skip` nor `noanswer` options are specified, then the `Playback()` command will first answer the channel (if it hasn't been answered already) and then play the sound file.

Note: The options won't work if there are spaces between the filename, the comma and the option(s).

Example

```
exten => 500,1,Playback(tt-weasels,skip)
```

Return Code

Returns -1 if the channel was hung up, or if the file does not exist. Returns 0 otherwise.

4.6.5 Dial()

Attempts to establish a new outgoing connection on a channel, and then link it to the existing input channel.

Description

```
Dial(type/identifier,timeout,options,URL)
```

```
Dial(type1/identifier1&type2/identifier2&type3/identifier3...,timeout,options,URL)
```

Attempts to "dial out" on all the specified channels (each specified by a type and identifier) simultaneously. The first channel that answers "wins", and all the other outgoing channels are hung up. The originating channel that triggered this `Dial()` command is then answered, if necessary, and the two channels are connected together ("bridged") allowing a conversation to take place between them. When the channel that triggered the `Dial()` command hangs up, the `Dial()` command exits.

`RetryDial()` was added in Asterisk v1.2 together with the 'd' flag.

Parameters

- New in Asterisk v1.2.0: If you don't want to modify options on each app that used to have jumping behavior, you can set "`priorityjumping=yes`" in the [general] section of `extensions.conf` which will enforce the old behavior globally. As far as the `Dial()` application is concerned you can control the behavior with the 'j' option (see below).
- New in Asterisk v1.2.0: The Caller*ID of the outbound leg is now the extension that was called, rather than the Caller
- ID of the inbound leg of the call. The "o" flag for Dial can be used to restore the original behavior if desired. Note that if you are looking for the originating callerid from the manager event, there is a new manager event "Dial" which provides the source and destination channels and callerid.
- type specifies the channel type. It should be one of the registered channel types, such as "Zap", "SIP", "IAX2", and so on.
- identifier specifies the "phone number" to dial on that channel. The format of the "phone number" depends on the channel, and may contain additional parameters (e.g. a distinctive ring parameter) specific to the channel module in question; the Dial command simply passes identifier to the channel module to process in whatever way is appropriate. See the documentation for the individual channel modules to learn about the correct format for specifying the identifier for the Dial command, and the options available to you when doing so. If you need a pause while dialing a number you can insert a w in the appropriate place.

- If you wish to specify more than one channel for the Dial command to try ? remembering that it will dial out on all of them simultaneously ? separate them with the & symbol. The channels can be different types. See Examples, below.
- The timeout parameter is optional. If not specified, the Dial command will wait indefinitely, exiting only when the originating channel hangs up, or all the dialed channels return a busy or error condition. Otherwise it specifies a maximum time, in seconds, that the Dial command is to wait for a channel to answer.
- The options parameter, which is optional, is a string containing zero or more of the following flags and parameters:
 - t: Allow the called user to transfer the call by hitting #
 - T: Allow the calling user to transfer the call by hitting #
 - r: Generate a ringing tone for the calling party, passing no audio from the called channel(s) until one answers. Use with care and don't insert this by default into all your dial statements as you are killing call progress information for the user. Really, you almost certainly do not want to use this. Asterisk will generate ring tones automatically where it is appropriate to do so. "r" makes it go the next step and additionally generate ring tones where it is probably not appropriate to do so.
 - R: Indicate ringing to the calling party when the called party indicates ringing, pass no audio until answered. This is available only if you are using kapejod's bristuff.
 - m: Provide Music on Hold to the calling party until the called channel answers. This is mutually exclusive with option 'r', obviously. Use m(class) to specify a class for the music on hold.
 - n: (Asterisk 1.1 and later) July 2005 bug 752 was included in CVS (Asterisk 1.1) and enhances the privacy manager considerably. As part of this patch, the 'n' flag to Dial got changed to be used as part of the privacy features, instead of being the 'dont jump to +101' flag. That flag is now 'j'.
 - o: Restore the Asterisk v1.0 CallerId behaviour (send the original caller's ID) in Asterisk v1.2 (default: send this extension's number)
 - j: Asterisk 1.2 and later: Jump to priority n+101 if all of the requested channels were busy (just like behaviour in Asterisk 1.0.x)
 - M(x): Executes the macro (x) upon connect of the call
 - h: Allow the callee to hang up by dialing *
 - H: Allow the caller to hang up by dialing *
 - C: Reset the CDR (Call Detail Record) for this call. This is like using the NoCDR command
 - P(x): Use the PrivacyManager, using x as the database (x is optional)

- g: When the called party hangs up, exit to execute more commands in the current context.
 - G: G(context^exten^pri) - If the call is answered, transfer both parties to the specified priority; however it seems the calling party is transferred to priority x, and the called party to priority x+1 (new in v1.2)
 - A(x): Play an announcement (x.gsm) to the called party.
 - S(n): Hangup the call n seconds AFTER called party picks up.
 - d: This flag trumps the 'H' flag and intercepts any dtmf while waiting for the call to be answered and returns that value on the spot. This allows you to dial an exit extension while waiting for the call to be answered - see also RetryDial o D(digits): After the called party answers, send digits as a DTMF stream, then connect the call to the originating channel.
 - L(x[:y][:z]): Limit the call to 'x' ms, warning when 'y' ms are left, repeated every 'z' ms) Only 'x' is required, 'y' and 'z' are optional. The following special variables are optional for limit calls: (pasted from app_dial.c)
 - * LIMIT_PLAYAUDIO_CALLER - yes|no (default yes) - Play sounds to the caller.
 - * LIMIT_PLAYAUDIO_CALLEE - yes|no - Play sounds to the callee.
 - * LIMIT_TIMEOUT_FILE - File to play when time is up.
 - * LIMIT_CONNECT_FILE - File to play when call begins.
 - * LIMIT_WARNING_FILE - File to play as warning if 'y' is defined. If LIMIT_WARNING_FILE is not defined, then the default behaviour is to announce ("You have [XX minutes] YY seconds").
 - f: forces callerid to be set as the extension of the line making/redirecting the outgoing call. For example, some PSTNs don't allow callerids from other extensions than the ones that are assigned to you.
 - w: Allow the called user to start recording after pressing *1 or what defined in features.conf (Asterisk v1.2.x); requires Set(DYNAMIC_FEATURES=automon)
 - W: Allow the calling user to start recording after pressing *1 or what defined in features.conf (Asterisk v1.2.x); requires Set(DYNAMIC_FEATURES=automon)
- The optional URL parameter will also be sent to the called party upon successful connection, if the channel technology supports the sending of URLs in this way.

Return codes If all the called channels are busy, Dial() will continue in the current context at priority n+101, if it exists, where n is the priority of the Dial() command. This allows you to set up different behavior for a busy response as opposed to a no-answer response.

If Dial() was unable to place the call for some other reason, such as timing out before any channel answered successfully, then Dial() will execute the next higher priority (n+1) command.

If the `g` option is specified, and the called party hangs up before the calling party, then `Dial()` continues execution at priority `n+1`.

Example

```
exten => 100,1,Answer()
exten => 100,2,Dial(SIP/${EXTEN},40)
exten => 100,2,Hangup()
```

On an incoming call from extension 100, this dial plan first answers the call, then establishes a new outgoing call on the SIP channel using the extension specified in the variable `${EXTEN}`. If there is now answer in less than 40 seconds, the call will be terminated.

4.7 Exercise

We should now create our first dial plan. Edit the `extension.conf` file to achieve the following points:

1. Create a simple entry to hear a welcome message (hello-world) and then hangup
2. Create a static entry to handle incoming calls
3. Create a generic entry to replace the static entry and to be able to handle all connected SIP clients

Use your softphone to test the configuration.

5 Call-parking

Imaging you want to have a receptionist that answers all incoming calls for your company, then forwards the calls to the person the caller wants to reach. To implment this functionality, you can use the call-parking feature of Asterisk. An incoming call will be answered by the recptionist, then he/she has to push the `#` button on the phone. At this moment the caller will hear MOH (music on hold), and the receptionist dials the requested extension. When the dialed extension hooks-up, the caller will be inline with him. To configure call-parking you have to do the following steps:

- edit the `features.conf` file and make sure the following parameters are in:

```
[general]
parkext => 700           ; What ext. to dial to park
parkpos => 701-720      ; What extensions to park calls on
context => parkedcalls  ; Which context parked calls are in ;
parkingtime => 45       ; Number of seconds a call can be parked for
```

- edit the `extension.conf` file and include the `parkedcalls` file:

```
[internal]
include => parkedcalls
```

- Add the 't' and 'r' flag to the corresponding `Dial()` commands in the `extensions.conf` file. The 't' means that the called user can transfer the call. The 'r' tells the calling party that the extension is ringing:

```
exten => _XXX,1,Answer()
exten => _XXX,2,Dial(SIP/${EXTEN},40,tr)
exten => _XXX,3,Voicemail(u${EXTEN})
exten => _XXX,4,Hangup()
```

6 Voice-mail

One important application of Asterisk is the voice-mail functionality. This feature allows to create voice-mail boxes for each user.

To configure voice-mail, you have to first edit the `voicemail.conf` file. Specify the voice-mail boxes for your users here.

Example:

```
[general]
format=gsm|wav
attach=yes
maxgreet=30
maxmessage=90
[default]
100 => 1111,Patrick, harpes@localhost
200 => 1111,johannes, johannes@localhost
300 => 1234,eric
```

6.1 [general] Section

Lets first discuss the contents of the general section. This section contains configuration settings which will apply as a common policy across all users. The general section must be defined and present in `voicemail.conf`. Configuration entries are coded in `setting=value` format. The configuration settings available in the general section are as follows:

attach

Attach causes Asterisk to copy a voicemail message to an audio file and send it to the user as an attachment in an e-mail voicemail notification message. The default is not to do this. Attach takes two values yes or no.

format

The format setting selects audio file format(s) to use when storing voice mail messages. The value is a string defining the audio format(s) of the message file. The default format string is wav49|gsm|wav, meaning that Asterisk will save the voicemail message in all three supported formats. When emailing the attachment, however, it will send only the first of the formats defined here.

maxgreet

This setting allows the administrator to limit the length of the user-recordable voicemail greeting. Use this option on systems with a large number of users and limited disk space. The value is an integer defining the maximum length in seconds of a greeting message. The default value is 0 which means that the greeting message can be any length. This setting will control the lengths of the unavailable greeting, busy greeting, and user name messages.

maxmessage

This defines the maximum amount of time in seconds of an incoming message. Use this when there are many users and disk space is limited. The default value for this setting is 0 which means there will be no maximum time limit enforced.

6.2 [CONTEXTS] section

The final part of the voicemail.conf configuration file contains one or more context sections where voice mail boxes are defined and configured. (Remember that context sections are sections which are named after contexts defined elsewhere in Asterisk). Multiple voice mail boxes may be defined in a context section. The format of context section is:

```
[context_section] extension_number => voicemail_password,  
user_name,  
user_email_address,  
user_pager_email_address,  
user_option(s) ...
```

After the context section definition, an entry for each voice mail boxes you want to have in this context. The **extension_number** is the extension number which will be assigned this voice mail box. There are 5 parameters which define the configuration for this voice mail box entry:

- The **voicemail_password** field contains the numeric password for this voice mail box entry. The voicemail system will compare the password entered by the user against this value.
- The **user_name** is an alpha field which is usually set to the first and last name of the user of this voice mail box.

- The `user_email_address` can be set to the email address of the user so that when a voice mail message is left by a caller, an e-mail notification will be sent to the address defined in this field.
- The `pager_email_address` can be set to the email address of a pager so that when a voice mail message is left by a caller, a page will be sent to the pager email address defined in this field.
- The `user_option(s)` field can be used to override default settings defined in the general section, or set a specific time zone for this user. Specifically, there are 9 `setting=value` pairs which can be specified in the `user_option(s)` field. There can be multiple `setting=value` pairs defined in the `user_option(s)` field. Each `setting=value` pair after the first must be delimited with a vertical bar (`|`). The nine settings which may be used are: `attach`, `serveremail`, `tz`, `saycid`, `review`, `operator`, `callback`, `dialout` and `exitcontext`. With the exception of `tz`, the functions of the 8 remaining settings are exactly the same as those defined in the `[general]` section. The `tz` setting is used to override the default time zone and it must be set to a custom time zone defined in the `[zonemessages]` section.

For a complete list of parameters for `voicemail.conf` please visit [8]

6.3 Dial-plan entries for Voice-mail

To access the voice-mail box, we have to define an entry for it in our dial-plan. Consider the following `extensions.conf` lines:

```

exten => _XXX,1,Dial(SIP/${EXTEN},40)
exten => _XXX,2,Voicemail(u${EXTEN})
exten => _XXX,3,Hangup()

```

The first line calls the `Dial()` application to dial the given number. If there is no answer in within 40 seconds, the second line will be executed. This calls the `voicemail()` application, to let the caller leave a message. The third line hangs-up the call.

6.3.1 Asterisk voicemail() command

`VoiceMail([flags]boxnumber[@context][&boxnumber2[@context]][&boxnumber3])`

Records the channel, saving an audio file in a given voice-mail box number, which must be configured in `voicemail.conf`. The box-number may be preceded by one or more flags:

- `s`: The letter `s`, if it is not present, causes the instructions ("Please leave your message after the tone. When done, hang up, or press the pound key.") to be skipped.
- `u`: The letter `u`, if present, causes the unavailable message to be played. By default, the message says, "The person at extension ... 1234 ... is unavailable," but the mailbox owner may record their own unavailable message with the `VoicemailMain` command.

- b: The letter b, if present, causes the busy message to be played. By default, the message says, "The person at extension ... 1234 ... is busy."

You may not specify both u and b flags together. You may, however, combine them with s, giving six possibilities:

1. s: Play nothing.
2. (no flags): Play instructions.
3. su: Play unavailable message.
4. u: Play unavailable message, then instructions.
5. sb: Play busy message.
6. b: Play busy message, then instructions.

In all cases, the beep.gsm file will also be played, prior to starting to record.

The voice-mail messages will be saved into the inbox directory for that voice-mail box-number:

```
/var/spool/asterisk/voicemail/context/boxnumber/INBOX/
```

6.4 Accessing the Voice-mail box

Until now we explained how to configure the voice-mail application to let the caller leave messages. Now we want to explain how to configure Asterisk to let the owner of the voice-mail box access his voice-mail box to listen to the messages. As you can imagine, this configuration has to be done in the extensions.conf file. We have to add entries to start the VoiceMailMain() application that handles the voice-mail boxes.

```
VoiceMailMain([[s]mailbox]@context)
```

enters the main voice-mail system for the checking of voice-mail. The mailbox can be passed as the option, which will stop the voice-mail system from prompting the user for the mailbox.

If the mailbox is preceded by 's' then the password check will be skipped. If a context is specified, logins are considered in that context only.

Example

```
; To let the user listen his voice-mail ;
exten => 5000,1,VoiceMailMain()
; To give direct access to the mailbox
exten => _5XXX,1,VoiceMailMain(${EXTEN:1})
```

In the first example, the users can access their mailbox by dialing 5000. Then the `VoiceMailMain()` application asks the caller to provide mailbox number and password. In the second example, the user can access his mailbox by dialing his number preceded by '5'. In this case the `VoiceMailMain()` application asks only the password to access the mailbox.

6.5 Exercises

1. Modify the dial-plan in a manner to switch to the voicemail if the user is not available
2. Extend the dial-plan to let the users interrogate their voice-mail box

7 Connecting to the outside world using ISDN

At this point we have a running PBX which is able to serve SIP phones. We want now enhance this configuration in a way that the internal users have the possibility to make phone calls to the outside world. This can be done using a standard ISDN card. In our case we use the Fritz PCI ISDN card. The installation of the drivers for the card has been discussed in chapter 2.5.7.

After the successful installation of the card and the drivers, we have to configure the CAPI interface. This can be done by editing the `capi.conf` file.

```
[general]
nationalprefix = 00
internationalprefix = 000
rxgain = 0.8
txgain = 0.8
[ISDN1]
isdnmode=msn
msn = 435253
incomingmsn = *
context = capi-in
softdtmf = 0
controller = 1
devices = 2
echocancel=yes
echocancelold=yes
```

Important parameters in this file are the definition of the context and the definition of the number of available ISDN channels (device parameter). The `msn` parameter defines the number for outgoing calls.

To activate this configuration you have to restart the Asterisk server. The second step of the configuration for the ISDN connection is to update the dial-plan. We have to add rules to handle calls from and to the ISDN channel.

Example of the extension.conf file:

```
[Internal]
; To let the SIP users phone via ISDN, prefix=0
exten => _0.,1,Dial(CAPI/contr1/1234:${EXTEN:1})
exten => _0.,2,Congestion()
exten => _0.,3,Hangup()
[capi-in]
exten => _X.,1,Answer()
exten => _X.,2,Dial(SIP/100,40)
exten => _X.,3,hangup()
```

The first part of the extension.conf file defines the context [Internal], that is used for all SIP calls from internal users. If an internal SIP user wants to use the ISDN line he has to add 0 in the beginning of the phone number. If the number contains a preceding 0, the dial command is executed. The parameter of the Dial() command specifies the CAPI channel followed by the phone number to dial.

The second part of the configuration defines the [capi-in] context. As we defined [capi-in] as context for incoming ISDN calls in the file capi.conf, this section handles incoming ISDN calls. In this example, each incoming call from an ISDN line will be forwarded to the internal SIP user with the number 100. As this configuration is not too much useful, we should now consider a more complex example of a dial-plan:

```
[internal]
exten => _XXX,1,Answer()
exten => _XXX,2,Dial(SIP/${EXTEN},40)
exten => _XXX,3,Voicemail(u${EXTEN})
exten => _XXX,4,Hangup()
[capi-in]
; Accept a call from ISDN line (msn=123456)
exten => 123456,1,Answer()
exten => 123456,2,Wait(1)
exten => 123456,3,Playback(enter-ext-of-person)
; Handle the extension, the user enters using DTMF tones
exten => _XXX,2,Goto(internal,${EXTEN},1)
```

In this configuration an incoming call on the ISDN channel with msn=123456, starts the Playback() applications. This applications tells the user to enter the extension of the person that the caller wants to reach. The user has to enter the extension using DTMF tones feature. The last line of the configuration, catches this extension and forwards it into the [internal] context. In this context, the corresponding SIP user will be called.

8 Connecting to another Asterisk system using the IAX Protocol

This chapter describes how to configure Asterisk to connect to another Asterisk server using the Asterisk IAX protocol. The IAX revision 2 protocol is used by Asterisk as an alternative to SIP, H.323, etc. when connecting to other devices that support IAX. In our context we want to use IAX to connect one Asterisk server to another Asterisk server, but it's also possible to use the IAX protocol to connect to IAX compatible VoIP phones (software or hardware).

The first step to configure the IAX protocol is to define IAX channels. This has to be done in the file `iax.conf`.

Example of an `iax.conf` file:

```
[general]
port=5036
disallow=all
allow=ulaw
allow=alaw
allow=gsm
bandwidth=high
[asterisk-server1]
type=peer
user=user1
secret=1234
host=192.168.1.3
[asterisk-server2]
type=user
user = user2
secret=4321
context=from_server1
```

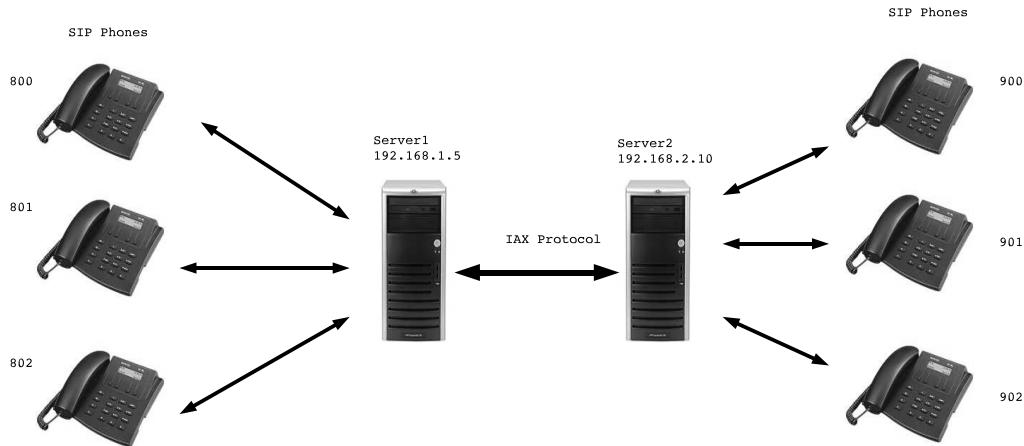
The first non-comment line in your `iax.conf` file must be the heading `[general]`. The parameters in this section will apply to all connections using this protocol, unless defined differently in a specific channel's definition. In our example the 1st line in the general section defines the port number, the following 4 lines defines the codecs to use on the IAX protocol. The bandwidth parameter helps to specify which codecs to use on this channel.

After the `[general]` section, we can now define our channels. Each connection is defined by the `type` parameter as a user, peer or friend. A user type is used to authenticate incoming calls, a peer type is used for outgoing calls, and a friend type is used for both. The first channel is called `[asterisk-server1]`, and it's type has been defined as peer. The user name and password have been defined as `user1` and `1234`. The `host` parameter specifies the IP address of the remote server to forward the calls. The second channel

specifies an incoming channel (type=user). As this channel is an incoming channel, we don't have to specify a host, but we have to define a context to which the calls are routed.

Concrete Example

Consider the following situation:



As on the figure, we have 2 servers to which are connected local SIP users. Now we want to connect the 2 sites together, and the users must have the possibility to make calls to everybody on the system. To achieve this, we want to connect the servers using the IAX protocol. On each server, the SIP clients have to be defined. Furthermore the `iax.conf` file and the `extensions.conf` have to be configured .

The SIP users of site 1 use the phone numbers from 800 - 899, the users of the site 2 use from 900-999.

Configuration on server 1:

`sip.conf`

```
[general]
context=default
srvlookup=yes
disallow = all
allow = alaw
allow = ulaw
```

```
allow = gsm
[8XX]
type=friend
secret=welcome
qualify=yes
nat=no
host=dynamic
canreinvite=no
context=internal
```

iax.conf

```
[general]
port=5036
disallow=all
allow=ulaw
allow=alaw
allow=gsm
bandwidth=high
[server1]
type=peer ; outgoing
user=server1
secret=1234
host=192.168.2.10
[server2]
type=user ; incoming
secret=4321
context=from_server2
```

extensions.conf

```
[internal]
; Extension that starts with 9 should be forwarded to server2
exten => _9XX,1,Answer()
exten => _9XX,2,Dial(IAX2/server2/${EXTEN},40)
exten => _9XX,3,Hangup()
; For internal SIP phones
exten => _XXX,1,Answer()
exten => _XXX,2,Dial(SIP/${EXTEN},40)
exten => _XXX,3,Voicemail(u${EXTEN})
exten => _XXX,4,Hangup()
[from_server2]
; Forward everything incoming into this context to context [internal]
exten => _XXX,1,Goto(internal,${EXTEN},1)
```

Configuration on server 2:

```
sip.conf
[general]
context=default
srvlookup=yes
disallow = all
allow = alaw
allow = ulaw
allow = gsm
[9XX]
type=friend
secret=welcome
qualify=yes
nat=no
host=dynamic
canreinvite=no
context=internal
```

iax.conf

```
[general]
port=5036
disallow=all
allow=ulaw
allow=alaw
allow=gsm
bandwidth=high
[server1]
type=user ; incoming
secret=1234
context=from_server2
[server2]
type=peer ; outgoing
user = server1
secret=4321
host=192.168.1.5
```

extensions.conf

```
[internal]
; Extension that starts with 9 should be forwarded to server2
exten => _9XX,1,Answer()
exten => _9XX,2,Dial(IAX2/server2/${EXTEN},40)
exten => _9XX,3,Hangup()
```

```

; For internal SIP phones
exten => _XXX,1,Answer()
exten => _XXX,2,Dial(SIP/${EXTEN},40)
exten => _XXX,3,Voicemail(u${EXTEN})
exten => _XXX,4,Hangup()
[from_server2]
; Forward everything incoming into this context to context [internal]
exten => _XXX,1,Goto(internal,${EXTEN},1)

```

8.1 IAX Authentication [9]

The IAX protocol IAX supports three methods of authentication. The first (and least secure) is plaintext. The passwords (or secrets) are sent in clear text over the network. The second is md5, which uses an md5 challenge response algorithm. Both machines will have clear-text access to the passwords, but they will be confirmed using an md5 hash while passing over the network. The most secure option is to use RSA public/private key encryption to store and transmit the secret. Public/private key pairs can be generated using the included program `astgenkey`. The public key will need to be manually transferred to the server and stored in `/var/lib/asterisk/keys/name.pub`. Server private keys are stored in the same location as `name.key`.

Important Note: In order use RSA keys with Asterisk, you will have to `init keys` at the console during startup. Asterisk will prompt you to do so every time it is launched.

Authentication parameters of `iax.conf`:

- `inkeys`: The public keys to use to decrypt authentication for an incoming client request or registration.
- `outkey`: The private key to encrypt outgoing authentication communication for this client.

md5 Example:

```

auth=md5
secret=password

```

rsa Example:

```

auth=rsa
inkeys=theirkey
outkey=mykey

```

8.2 Exercise

Try to setup a link between your Asterisk installation and the main Asterisk server located at the desk. The main server is configured to handle only outgoing calls. The configuration of the main server is the following:

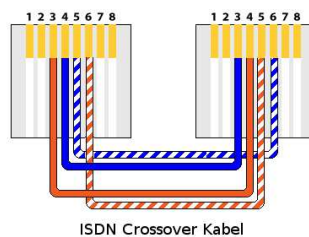
iax.conf

```
[general]
port=5036
disallow=all
allow=ulaw
allow=alaw
allow=gsm
bandwidth=high
[asterisk]
type=user
user=tux
secret=1234
host=192.168.2.10
```

9 Integrating your existing ISDN equipment

9.1 Hardware setup

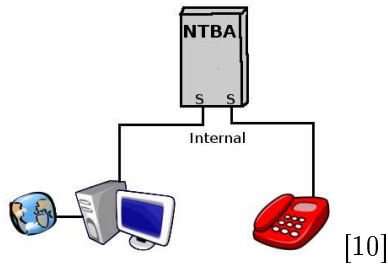
What we want to do is to use a normal hardware ISDN phone or even a normal telephone (connected to an ISDN-analog changer) for our voice over ip calls. As the installed HFC card is capable of working in NT-mode, we need a special crossover ISDN cable to connect the phones to the card. This cable crosses the RX with the TX cable pairs of the connection.



ISDN Crossover Kabel

[10]

Additionally we need a power supply for our new ISDN-bus to power the connected phones and devices. We can use an old NTBA which is not connected to the phone network to act as patchbay and power supply. The ISDN-card is connected to the first 'S' connector of the NTBA, and the telephones to the second one.



After all phones are connected and powered we have to set the Multiple Subscriber Number of the phones. In our example we use the number interval of 9xx for all phones on the zaptel card. The connected Tectra ISDN phone has MSN 999.

9.2 Configuring asterisk for ISDN phones

Now that the hardware part is done, we need to configure our asterisk server to use the connected phones. This configuration is done in the zapata.conf file.

Example of a zapata.conf file:

```
[channels]
channel => 1-2
group = 1
switchtype = euroisdn
signalling = bri_net_ptmp
pridialplan = local
immediate = no
overlapdial = yes
echocancel = yes
context = default
```

The [channel] heading in this example uses the channels 1-2 (which is the first ISDN card) of the zaphfc driver. We need to set the switchtype to 'euroisdn' and signalling to 'bri_net_ptmp' as we want to use phones that are specified for the european ISDN network. The parameters `pridialplan`, `immediate` and `overlapdial` define the dial-behavior of the connected phones. In our case the ISDN phone acts as a normal phone which allows us to dial a number after picking up the handset. The `echocancel` option is optional, but worked really good for us. All incoming calls are routed to the default context of the extensions.conf.

Further we have to edit the extensions.conf to handle incoming and outgoing calls for the ISDN phones.

Example of the extensions.conf:

```

; all connections start here
[default]
; 9xx numbers belong to local ISDN phones
exten => _9XX,1,GoTo(isdn-phones,${EXTEN},1)
; all other numbers go to master asterisk server
exten => _X.,1,GoTo(master-asterisk-server,${EXTEN},1)
; Dial IAX Connection to master asterisk server
[master-asterisk-server]
exten => _X.,1,Dial(IAX2/asterisk-server/${EXTEN})
exten => _X.,2,hangup()
; Dial local ISDN Phones
[isdn-phones]
exten => _X.,1,Answer()
exten => _X.,2,Dial(Zap/g1/${EXTEN})
exten => _X.,3,Hangup()

```

We have splitted the extensions.conf to a [default] section, where all incoming calls are dropped in, and special sections for each outgoing interface. The [default] section decides where to route the call by using pattern matching on the dialed number. In this case all three-digit numbers starting with 9 are routed to the [isdn-phones] section. This section dials the phones on the zaptel ISDN card. All other numbers are forwarded to the master asterisk server via an IAX connection.

10 Softphones

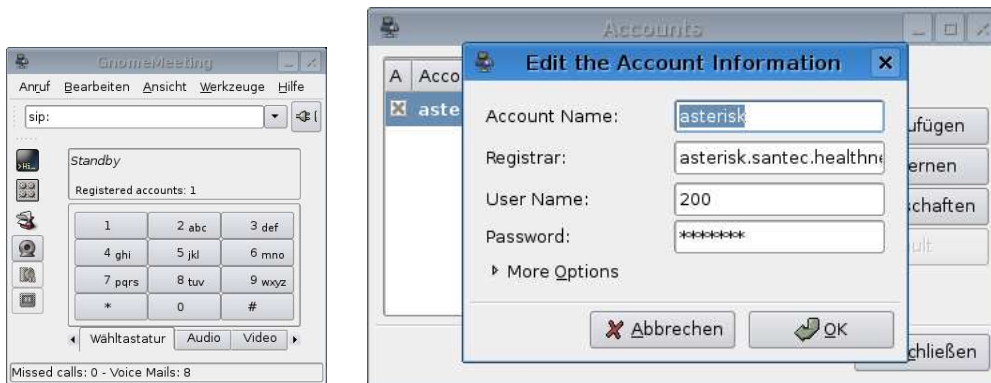
“A soft phone is software that simulates a real phone and runs on a general purpose computer, rather than a dedicated device. It is usually used with a headset connected to the sound card of the PC or USB phone.”[11]

There are a lot of softphones available, which differ in their destined operation systems, their supported network and audio protocols and their software license. This paragraph shows up some free available SIP and IAX capable softphones.

10.1 Sip Phones

10.1.1 GnomeMeeting

“GnomeMeeting is an H.323 compatible videoconferencing and VOIP/IP-Telephony application that allows you to make audio and video calls to remote users with H.323 hardware or software (such as Microsoft Netmeeting).”



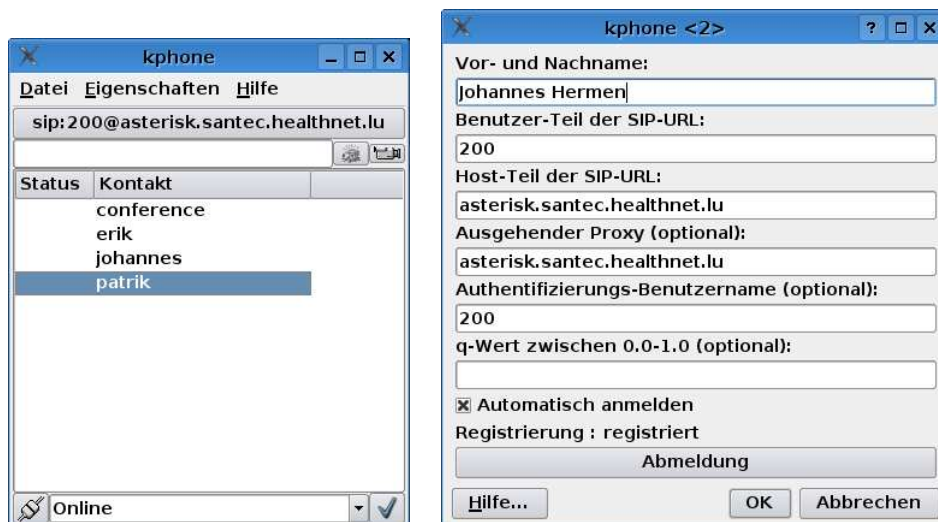
We installed the latest version of GnomeMeeting (2.0) which will be officially available in the next weeks is capable of connecting to SIP servers. The program runs under Linux and places a trayicon into the KDE and Gnome taskbar. Even if the program is still a beta version, it worked stable and had a lot of working features. The user interface is small and clear, and the GTK2 toolkit integrates good with Gnome and KDE. GnomeMeeting is shipped with most Linux distributions and even the beta version is available precompiled for several distributions.

License: GnomeMeeting is released under the GNU/GPL license .

Homepage: <http://www.gnomemeeting.org/>

10.1.2 KPhone

“KPhone is a SIP user agent for Linux, with which you can initiate VoIP (Voice over IP) connections over the Internet. It supports Presence and Instant Messaging, and to some extent also video calls between two hosts.”

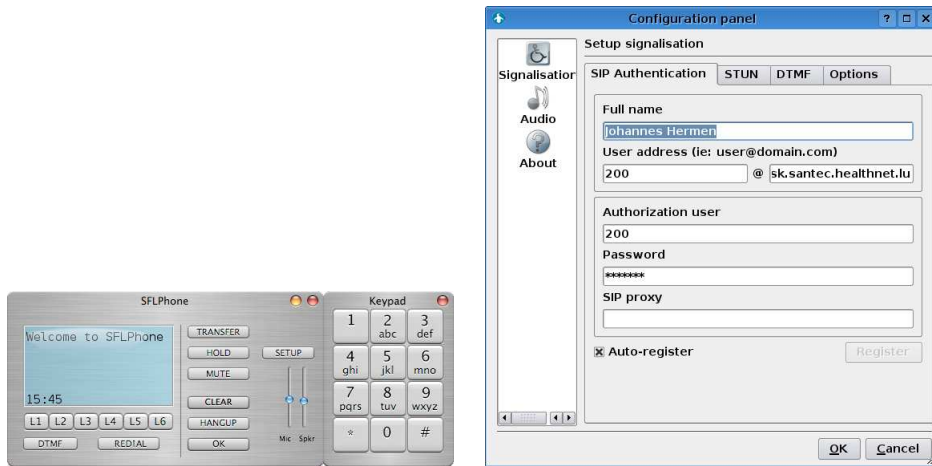


We tested KPhone in version 4.2, which is packaged with Kubuntu Linux 5.10. KPhone worked, but had a very rudimentary user interface. It would be nice if later versions of KPhone would integrate better into KDE e.g. use the KDE internal address book.

License: KPhone is released under the GNU/GPL License.
Homepage: <http://www.wirlab.net/kphone/>

10.1.3 SFLPhone

“SFLphone is a multiplatform softphone for IP telephony. It’s currently available on Linux but we hope that Windows and MacOS porting will come soon. SFLphone aims to become your desktop’s VoIP companion.”

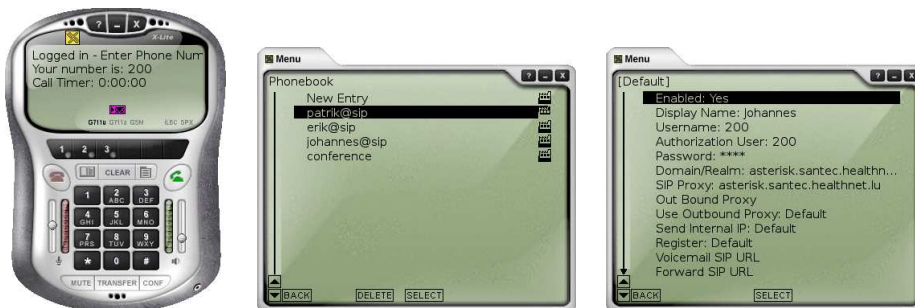


SFLPhone has a really nice User interface, but as it is not that familiar, packages are only available for Fedora Core and SuSE Linux 9.3. After taking some difficulties on building and installing SFLphone, the software worked really good. Even the transfer of answered calls to other phones worked well. SFLphone lacks of a good address book and a tray integration for Gnome and KDE, but maybe more features will be available in later versions.

License: SFLPhone is released under the GNU/GPL License.
Homepage: <http://www.sfphone.org/>

10.1.4 X-Lite

“X-Lite is a SIP-based user agent (IP phone) with all the features of a standard business telephone. It offers line hold, three lines, redial, transfer, recent numbers list, caller ID, mute, touchtone, a call timer, and more.”



X-Lite is a free (free like free beer not like free and open source software) available and funktionally reduced version of the commercial “X-Pro” and “eye-beam” software from Xten Networks. It is available for Linux, Windows, Mac OS X and Pocket PC.

X-Lite disabled the nat-proxy feature in the free version, which makes it impossible to use this program behind a router. While testing the program we had problems with the “music on hold” feature of asterisk. X-Lite was unable to reproduce the played sounds correctly.

License: X-Lite is available as Freeware.

Homepage: <http://www.globalipphones.com/>

10.1.5 SJphone

SJphone is a softphone which allows you to speak over Internet using any desktops, notebooks, PDAs, stand-alone IP phones, and even any traditional landline or mobile phones. It supports both SIP and H.323 industry open standards and is fully interoperable with most major Internet Telephony Service Providers (ITSP) and software and hardware manufacturers.



SJphone worked really good under Windows XP, but the Linux version, which seemed to be older, and did not had this nice GUI, did not work as expected. Like X-Lite, SJPhone is also available for Linux, Windows, Mac OS X and Pocket PC. There is although a commercial version of SJphone available which can be customized by SJLabs to fit the customers needs.

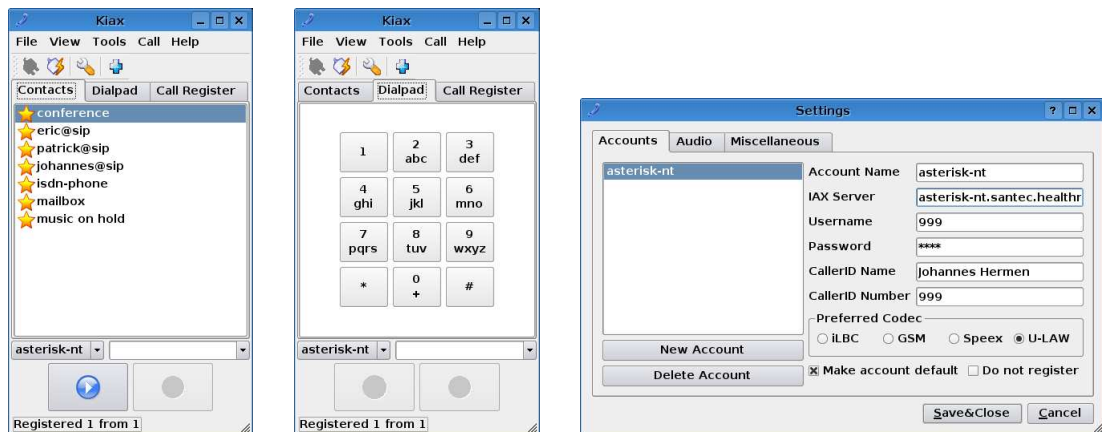
License: SJPhone is available as Freeware.

Homepage: <http://www.sjlabs.com>

10.2 IAX Phones

10.2.1 KIAX

“KiAx is an IAX client application (a so called Softphone) which allows PC users to make ordinary VoIP calls to Asterisk servers, the same way as they do it with their hardware telephone. It aims to provide a simple and user-friendly graphical interface and desktop integration for calling, contact list, call register management and easy configuration.”



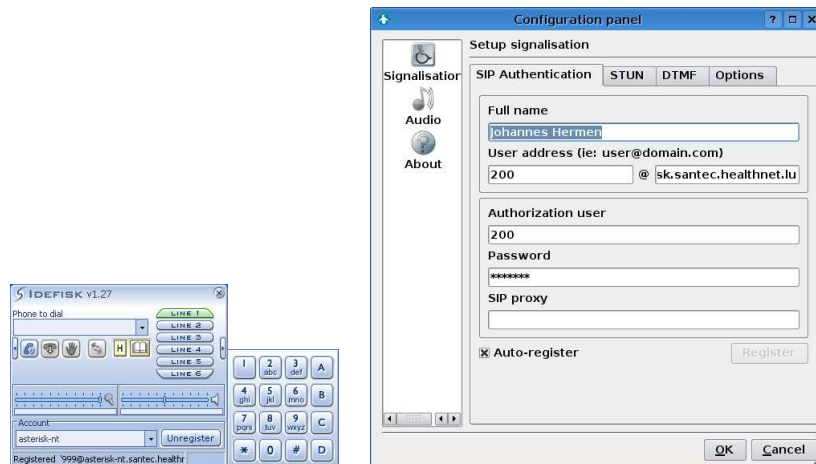
KIAX, which is available in version 0.8.4 at the moment, can be downloaded as binary package, that only has to be extracted to get KIAX to work. It uses the QT framework, and integrates very well into KDE. As it uses the Asterisk internal IAX protocol, it was easy to connect it to our asterisk server. KIAX is available for Linux only.

License: KIAX is available under the GNU/GPL License

Homepage: <http://kiax.sourceforge.net/>

10.2.2 Idefisk

“Idefisk is a softphone client, which supports the Inter-Asterisk Exchange (IAX2) protocol. It can be downloaded for free. So far, It is supported by the Windows operating system. In order to use the phone you will need an installed Asterisk PBX.”



We tested Idefisk version 1.27, which worked good on Windows XP. The GUI looks nice and it offers a lot of Asterisk features.

License: Idefisk is available for free in personal or commercial purposes.

Homepage: http://www.asteriskguru.com/tools/idefisk_beta.php

References

- [1] <http://www.asterisk.org/architecture>
- [2] <http://www.voip-info.org/wiki-Asterisk+SIP+channels>
- [3] <http://www.voip-info.org/tiki-index.php?page=Asterisk%20config%20extensions.conf>
- [4] <http://www.voip-info.org/wiki/index.php?page=Asterisk+Dialplan+Patterns>
- [5] <http://www.voip-info.org/wiki/index.php?page=Asterisk+Dialplan+Introduction>
- [6] <http://www.voip-info.org/wiki/index.php?page=Asterisk+priorities>
- [7] <http://www.voip-info.org/wiki-Asterisk+variables>
- [8] <http://www.voip-info.org/wiki/index.php?page=Asterisk+config+voicemail.conf>
- [9] The Asterisk Handbook Version 2, Spencer, Allison, Rhodes
- [10] <http://www.pro-linux.de/work/asterisk/asterisk-1.html>
- [11] <http://en.wikipedia.org/wiki/Softphone>