CapiSuite 0.5.cvs

Gernot Hillier

gernot@hillier.de

CapiSuite 0.5.cvs
by Gernot Hillier

Table of Contents

List of Examples

Introduction

1. Welcome to CapiSuite

Welcome to CapiSuite, a Python-scriptable ISDN telecommunication suite. It uses the new CAPI interface for accessing your ISDN-hardware - so you'll need a card for which a CAPI compatible driver is available. Currently these are all cards manufactured by AVM and some Eicon cards.

This manual should help you to be able to use CapiSuite as quick as possible. As I hate reading long documentation just as much as you do, let's jump right in.

2. What the heck is "CapiSuite"?!

CapiSuite tries to give the user the ability to code his own ISDN applications without having to fiddle around with all the dirty programming details like callback functions, data buffers, protocol settings and so on.

I took a scripting language which is (in my opinion) very easy to understand, to use and to learn - especially for beginners: Python. I extended it with some functions providing the basic ISDN "building blocks" for the users application. Behind these functions the heart of CapiSuite implements all the dirty details a user isn't interested in. My goal was to make script-coding as simple as possible but to also give you the flexibility to realize what you want.

To give you an impression, coding a simple answering machine is as easy as:

```
def callIncoming (call, service, call_from, call_to):
    connect_voice (call, 10)            # answer call after 10 secs
    audio_send (call, "announcemnt.la") # play announcement
    audio_send (call, "beep.la")        # play beep
    audio_receive (call, "call.la", 10) # record call
```

Of course some details are missing like creating a unique filename or storing the additional information (called and calling party numbers, time, ...) - but I assume you got my idea.

And - don't be afraid - if you just want to have a normal answering machine or send and receive some fax documents, you can use the default scripts distributed with CapiSuite. They give you already some nice features - e.g. the answering machine is multi-user ready, supports automatic fax detection and remote inquiry functions. You'll only need to tell CapiSuite some details like your own number, record an own announcement and that's it.

So CapiSuite is already equipped for your daily telecommunication needs - but if you don’t like to do the things the way I do - just change it or completely do it on your own. And if you write nice scripts or have changes to my default scripts, I would love to get and perhaps make them available for all users if you don’t mind.

3. Structure of the manual

This manual is split into three big parts.

The first part (Chapter 1) explains how to install CapiSuite, what you can do with the default scripts you have after installing it and how to configure them. No line of code will be presented here. If you just want to use the default scripts that should be all the reading you need.

The second part (Chapter 2) will tell you how to write your own scripts. It will give you a very, very small introduction into Python and a complete reference of the commands CapiSuite adds to it. Last, an overview over the default scripts is given which will tell you how they work so you can easily take them as starting points and/or examples for your own application.

The last part is intended for programmers who want to help in developing the CapiSuite core. It provides an overview of the system and a detailled description for each single class, method and attribute. As it’s autocreated from the sources of CapiSuite, it’s not included in this document. You’ll find it locally in ../reference/index.html or online at http://www.capisuite.de/capisuite/reference/index.html.

There are also some additional parts containing "what I also wanted to mention":

As CapiSuite started as a diploma thesis, I want to thank all who helped me so far in Appendix A

When you want to code your own scripts or want to help in developing the CapiSuite core, you’ll soon stumble upon some special ISDN and CAPI error codes, which are explained in Appendix B.

If you need further information or support, please have a look at the CapiSuite home page on http://www.capisuite.de. You’ll find links to a bug tracker, up-to-date documentation, downloads and other ressources there. If you have questions, need support or want to tell us your ideas concerning CapiSuite or your opinion, you’re more than welcome on the CapiSuite mailing lists. Please don’t write me personal mails with such questions as this won’t help other users and I can’t answer the same question ten times a day, sorry. For informations on how to subscribe to the lists and where to find the archives, please also refer to the home page.

Hope I managed to whet your appetite - so let’s now really start over to get you ready to use it.

Chapter 1. Getting Started

1.1. Requirements and installation of CapiSuite

1.1.1. Requirements

1.1.1.1. Hardware and drivers

As CapiSuite uses the CAPI (Common ISDN Application Programming Interface) for accessing your ISDN-hardware, you'll need a card for which a CAPI compatible driver is available.

Currently these are all cards manufactured by AVM and some Eicon cards. If you have one of the passive cards of AVM, you'll have to download and install their CAPI drivers.

There are also some distributions (e.g. current versions of SuSE) which include the Capi4Linux drivers from AVM already - you'll only have to activate them (use YaST2 in SuSE Linux). If you own an active card of AVM (e.g. the B1, C2 or C4), then you'll have everything you need already installed.

No, there's currently no way to get it working with the old ISDN4Linux interface. Perhaps there never will be one as the ISDN4Linux project will provide a CAPI compatible interface some near day in the future - so all supported ISDN cards will work with CapiSuite then. If you nevertheless want to write a backend for ISDN4Linux, just contact me - I'll be more than happy to help you with that.

CapiSuite has mainly been tested on AVM ISDN cards, esp. the Fritz!PCI, the Fritz!USB and the B1 on the i386 platform but there should be no problem with other CAPI-compatible drivers for other cards or on other platforms. Nevertheless, some features aren't mandatory for all CAPI-compatible cards, so perhaps you may not be able to fax or to switch from voice to fax mode with all cards.

1.1.1.2. Software

CapiSuite depends on some packages which must be installed before CapiSuite can be used.

I will list them here with a short information why this packages are needed and where to find further information on how to install them. It may be always a good idea to check the installation tool of your favourite distribution first and see if they're included with it before trying to download and install them from the net. Don't be afraid, because there are so many - most of them are included in nearly every distribution and perhaps are already installed on your system.

Python >= 2.2

CapiSuite uses an embedded Python interpreter to interpret the given scripts - so you'll need an installed and working version of Python. This should be included in mostly every up-to-date Linux distribution. For further infos on Python, a nice tutorial and much more, please go to http://www.python.org

sox >= 12.17.3

This is the swiss-knife for converting audio formats. It's not required by the CapiSuite core, but will be very helpful if you want to hear or record the voice files used for calls on your machine. It's also required if you want to use the default scripts of CapiSuite. I'll bet this is included in your distribution and most likely already installed on your system. Just try to start **sox** to get sure. As Helmut Gruber pointed out, you need at least version 12.17.3, as this version started to handle inverse A-Law files. You'll find more details on http://sox.sourceforge.net

sfftobmp

CapiSuite will save fax files in the CAPI specific format Structured Fax File (SFF). sfftobmp is a small but useful converter to convert this files to more common formats like JPEG, TIFF or BMP. Get it on http://sfftools.sourceforge.net/sfftobmp.html. It's again not needed by the CapiSuite core, but by the default scripts.

sffview

This tool is a simple but useful SFF viewer. It's not needed by any CapiSuite component, but very useful if you just want to see a fax file without the need to convert it first. You can get it from http://sfftools.sourceforge.net/sffview.html.

tiff2ps

A small utility to convert TIFF files to the Postscript format. It's needed by the default script to convert faxes to PDF files (SFF->TIFF->PS->PDF :-}). It's often included in a package called `tiff` or `tifftools`. Details on http://www.libtiff.org

ps2pdf

Again a small utility for the SFF->PDF chain - this time for the conversion of Adobe PostScript to Adobe PDF. It's part of Ghostscript, so you most likely have it already. (http://www.gnu.org/software/ghostscript/ghostscript.html)

current Ghostscript with cfax patch

Current Ghostscript versions will include a device to create the above mentioned SFF files. If you have an older version, you'll need the patch from http://sfftools.sourceforge.net/ghostscript.html. To see if your GhostScript version already has this patch, please call **gs --help** and see if you can find the device `cfax` in the long list of supported devices.

jpeg2ps

The **jpeg2ps** command is used to convert color fax files to the PostScript format for mail delivery. It's not so important, unless you want to be able to receive color faxes. Unfortunately, there's currently no way to disable the reception of color faxes with AVM cards due to a bug in the AVM CAPI driver. So if someone sends you a color fax (which seems to be a very rare case), you'll need

this package - unless you'll get a mail stating this error. If your distribution doesn't have this packages, you can download it from http://www.pdflib.com/products/more/jpeg2ps.html.

As the color fax protocol uses concatenated JPEG files for transferring multiple pages, you should also download and apply my multipleJPEG patch from
http://www.hillier.de/linux/jpeg2ps-multi.php3

glibc locales

If you want to use an AVM card (which most of you will do ;-)) to send or receive faxes, you'll need installed glibc locales because CapiSuite needs to convert your fax headline to the historic CP437 encoding which AVM drivers expect. The glibc locales are a part of the default C library glibc which you definitely already have installed, but some distributors make the locales installable separately. SuSE e.g. puts them in a package called glibc-locale which you probably have to install. To see if you already have them, look for a file called "IBM437.so" in your library paths (in SuSE it's located in /usr/lib/gconv). If you don't find it, look for sub-packages of glibc which could include them.

1.1.2. Installation

First of all, I would suggest to check if your CAPI-driver is setup correctly. To do this, simply run **capiinfo** on a root shell.

If you get many lines of output, your CAPI driver works. If you just get an error message, you'll have to install CAPI-compatible drivers. Refer to the documentation of your ISDN card vendor, your Linux distribution and/or some ISDN mailing lists for this, please. If you really can't find anyone to support you in doing this, you may ask on the CapiSuite mailing lists for support *as last resort*.

The rest of the installation depends on wether you use binary or source packages for installing CapiSuite. If you don't want to change the CapiSuite sources, I would recommend you to use the binary packages when available for your distribution and platform.

You can download both binary packages and sources from the download section on http://www.capisuite.de/download. If you built up your own packages for other distributions, please send me them and I'll copy them there...

1.1.2.1. Installation from binary packages

If you can get binary packages for your distribution and platform, I would advise to use them. I'm building RPM packages for SuSE Linux regularly, as this is the distribution I use (and BTW the company which paid and supported me to write CapiSuite as diploma thesis ;-)). Debian packages are maintained by Achim Bohnet.

If you managed to install CapiSuite on a system not mentioned below, please tell me and I'll include the instructions here. If you have created binary packages for other distributions, I'll be also happy to point to your download section or make them available on my page.

Now everything should be setup ready to run. So please read on in Section 1.2.

1.1.2.1.1. Installation from RPM packages (SuSE)

To install the CapiSuite RPM packages you can either use your favorite setup tool - either provided by your distributor or the community - or you can do manually (as root):

```
rpm -Uvh capisuite-version.rpm
```

1.1.2.1.2. Installation from Debian packages

Thanks to Achim Bohnet, we also have binary and source packages of CapiSuite and the necessary tools for Debian (woody).

To access them, please add the following lines to your `/etc/apt/sources.list` :

```
deb     http://www.mpe.mpg.de/~ach/debian ./
deb-src http://www.mpe.mpg.de/~ach/debian ./
```

Now, issue the following commands:

```
apt-get update            # add package list from repository
apt-get install capisuite # install and resolve dependencies
```

When a new package is available, **apt-get update** grabs the new package listing and

```
apt-get -u (dist-)upgrade
```

updates all out-of-date packages (and new additional package dependencies in the dist-upgrade version).

Similar procedures are valid for the menu/gui pkg installers **dselect**, **aptitude**, **kpackage**, ...

For further details, please have a look at the Debian documentation, for example the APT HowTo on http://www.debian.org/doc/manuals/apt-howto/index.en.html (english) or http://www.de.debian.org/doc/user-manuals#apt-howto (german).

1.1.2.2. Installation from the source packages

If there are no binary packages you can use or if you like to do everything on your own, you can get the sources from the download section.

Download the newest source tarball (capisuite-X.Y.tar.gz) from the CapiSuite homepage and copy it to some location. Go there and issue the following commands:

```
./configure
make
su # get root now
make install
```

This will install CapiSuite completely in the `/usr/local`-tree. If you want it to stay in other directories, please see the commandline-help printed by

```
./configure --help
```

for options to customize the installation directories.

1.1.2.3. Installation from CVS

If you want to live on the bleeding edge and always test the newest features, you may also checkout the current sources of CapiSuite from the CVS repository.

This is not recommended unless you want to test the newest features or want to help in developing CapiSuite! The sources in CVS may do anything, may not work or not even compile. Do this on your own risk!

You'll need installed and working versions of the usual development tools like autoconf, autoheader, automake, GNU make, gcc/g++ and also the components described above (esp. development packages of Python).

If you want to build the documentation out of the sources, you'll also need Doxygen.

For instructions on where to find the CVS repository and how to checkout the sources, please refer to the download section on the CapiSuite homepage on http://www.capisuite.de.

After you checked out the sources to some directory, please do

```
make -f Makefile.cvs
```

Now, you can continue with the normal installation process as described in Section 1.1.2.2.

1.2. How CapiSuite works, how it is configured and started

First, let's start with a short introduction what CapiSuite actually is and how it works. After that, the configuration and startup of CapiSuite will be explained in short.

1.2.1. How does CapiSuite work?

CapiSuite is a daemon (program which runs in the background) whos main task is to sit around and wait until a call is incoming. If this happens it will start a special Python script - the *incoming script* - and do what this script tells it, for example record a voice call to implement an answering machine.

To also be able to issue outgoing calls, another script is called at regular intervals - the *idle script*. It can check any resource to get instructions for placing a call - one can for example imagine to check a special mail account or watch a special directory where tasks are placed by the user.

So all user-visible actions and the behaviour of CapiSuite are defined in these two scripts.

You'll need to do two things now:

- provide scripts by either
  - using and configuring the default scripts distributed with CapiSuite or
  - writing your own scripts (perhaps by using the default ones as templates)
- configure CapiSuite itself and tell it where to find the two scripts

This page concentrates on the general configuration of CapiSuite - that consists mainly of options telling it which scripts to use and where and how to log its activities. After that, some details about starting CapiSuite are described.

The next pages will then introduce the standard scripts you already installed along with CapiSuite and tell you how to use the answering maching and fax functions provided by them.

The details on how to write your own scripts are covered in another part of the documentation (Chapter 2).

1.2.2. Configuration of CapiSuite

CapiSuite uses a general configuration file for the core functions. This file should be located in `/etc/capisuite/capisuite.conf` or `/usr/local/etc/capisuite/capisuite.conf` depending on how you installed CapiSuite.

Most options are set to reasonable defaults already for using the standard scripts - so if you want you can also skip this section and read on in Section 1.2.3

The options will be presented in brief here - for further details please refer to the comments in the configuration file itself.

Options available in capisuite.conf

`incoming_script="/path/to/incoming.py"`

This option tells CapiSuite which script should be executed at incoming calls. Only change this if you want to use your own script.

`idle_script="/path/to/idle.py"`

This option reflects the path and name of the idle script. This script is called in regular intervalls to check if any outgoing call should be done. As above, the default should be ok if you don’t use your own script.

`idle_script_interval="30"`

Here you can define how often the idle script should be executed. The number given is the interval between subsequent invocations in seconds. Lesser numbers give you quicker response to queued jobs but also a higher system load. The default should be ok in most cases.

`log_file="/path/to/capisuite.log"`

This file will be used for all "normal" messages printed by CapiSuite telling you what it does. Error messages are written to a special log (see below).

`log_level="1"`

You can define how detailled the log output of CapiSuite will be. The default will give you some informational messages for each incoming and outgoing call and should be enough for normal use. I would recommend to only increase it if you encounter some problems. Logs of higher level are mainly intended for developers, so just use them if you want to report a problem or have some know-how of the CAPI interface and the internals of CapiSuite.

`log_error="/path/to/capisuite.error"`

All errors which CapiSuite detects internally and in your scripts will end up here. They are written to an extra file so that they don’t get lost in the normal log. Please check this log regularly for any messages - especially when you encounter problems. Please report all messages you don’t understand and which aren’t caused by your own script-modifications to the CapiSuite team.

1.2.3. Startup of CapiSuite

As CapiSuite is a daemon, it is normally activated during the system startup process. Just add a call to

```
/path/to/capisuite -d
```

in your startup scripts. In LSB conforming Linux distributions, you'll find the startup scripts in `/etc/init.d`. For detailled documentation how to add a service there please refer to the documentation of your distribution. There's an example startup script written for SuSE Linux included in the source distribution (see `rc.capisuite`) which should (hopefully) work with other LSB compliant distributions, too. If you need to modify it, I'll welcome your feedback and happily add instructions for other distributions here.

If you use the right RPM packages of CapiSuite, the necessary scripts should already be included. For activating them, please use your distributors config tool. If you use the RPM distributed with SuSE Linux and want to stay with the default scripts, everything should work "out of the box". As soon as you have configured the default scripts, simply run **rccapisuite restart**.

For debug purposes, you can also start CapiSuite manually at any time by just calling

```
/path/to/capisuite
```

There are also some other commandline options available:

commandline options of CapiSuite

`--help, -h`

show a short summary of commandline options

`--config=file, -c file`

use a custom configuration file instead of `/etc/capisuite/capisuite.conf` or `/usr/local/etc/capisuite/capisuite.conf`.

`--daemon, -d`

run as daemon (used in your startup script, see above)

CapiSuite can run as any user you want theoretically. It only needs read/write permissions to `/dev/capi20`. If you use the default scripts, however, CapiSuite *must* run as `root`.

1.3. Features and configuration of the default scripts

As already written above, CapiSuite comes with default scripts giving you the most used communication functions of an answering machine and a fax device.

This section should help you to use them for your daily needs.

1.3.1. Script features

The scripts distributed with CapiSuite give you the following main functions:

- multi-user answering machine
  - different users using different numbers and different announcements are supported
  - incoming calls are saved and sent to the user by email
  - the delay until a call is accepted and the maximum record length are freely adjustable
  - silence is detected and the call terminated after an adjustable silence period
  - incoming fax calls are automatically detected and received
  - comfortable, menu-controlled remote inquiry functions are supported telling you the date/time when the call was received and the called and calling numbers.
  - record your own announcement via the remote inquiry menu
  - nearly each setting is configurable globally but can be overwritten for each user

- fax machine
  - different users using different numbers are supported
  - incoming faxes are stored and sent to the user by email
  - command line tool for faxing PostScript documents included
  - number of tries and delays for sending faxes freely configurable
  - currently supports only one ISDN controller for outgoing faxes

As my native language is german, all waves distributed with CapiSuite are in german only. If someone wants to provide waves in english (or any other language), please contact me. Thx!

1.3.2. How the scripts work

Here follows a rough overview of how the scripts work in general. I will only explain the behaviour which is important for the user here. If you want to understand the internals, please refer to Section 2.6.

When an incoming call is received, several lists for the different users are searched for the called number. The different users can define their own numbers in the configuration (see below). So the scripts decide by looking on the called number to which user the call destinates. If they find the number in the voice- or fax-number list of any user, they'll answer the call with this service and give the caller the possibility to leave his message or send his fax.

The received document is then saved to a local directory in some native format and also converted to a well-known format and mailed to the user along with some details of the call. Voice calls are sent as a WAV attachment, while fax calls are sent as PDF documents attached to the mail.

So you'll normally get your incoming calls as a mail to a specified address - but they're also saved in the local filesystem to be on the safe side. It's your task to delete old files you don't need any more. For further instructions, please see Section 1.3.4.

There's also the possibility to do a remote inquiry on the answering machine. The caller is presented a menu where he can choose to record his announcement or to hear the saved voice calls. He will be told how many calls are available, from whom and when they were received and so on. He'll also be able to delete recorded calls he doesn't need any more.

Another script will check special queue directories for fax send jobs regularly. To put jobs in this directory, the commandline tool **capisuitefax** is provided. See Section 1.4 for further details on this.

1.3.3. Script configuration

There are some important options which the scripts need to know before you can use them - things like the users' numbers and some details of how to handle the calls.

These options are read from two configuration files. Each configuration file is divided into one or more sections. A section begins with the section name in square brackets like `[section]` while the options are `key="value"` lines.

Each file must have a special section called `[GLOBAL]` and one section for each user called `[<username>]` (with `<username>` being a valid system user).

The `[GLOBAL]`-section defines some global options like pathnames and default settings for options that can be overridden in the user-sections. The user-sections hold all the options which belong to a particular user.

All options for the two files are described in short below. For all details, please see the comments in the sample configuration files installed with CapiSuite.

1.3.3.1. Configuration for fax service

This file holds all available config options for the fax services (fax receive and send).

It's read from `/etc/capisuite/fax.conf` or `/usr/local/etc/capisuite/fax.conf` (depending on the installation).

1.3.3.1.1. The [GLOBAL] section

`spool_dir="/path/to/spooldir/"`

This directory is used to archive sent (or failed) jobs. It must exist and the user CapiSuite runs as must have write permission to its subdirectories. Two subdirectories are used:

`spooldir/done/`

Jobs finished successfully are moved to this directory.

`spooldir/failed/`

Job which have failed finally end up here.

This option is mandatory.

`fax_user_dir="/path/to/userdir/"`

This directory is used to store fax jobs and received documents to. It must exist and the user CapiSuite runs as must have write permission to it. It will contain one subdirectory for each configured user (named like his userid). The following subdirectories are used below the user-specific dir:

`user_dir/username/received/`

Received faxes are saved here.

`user_dir/username/sendq/`

Fax files to be sent are queued here by **capisuitefax**.

This option is mandatory.

`send_tries="10"`

When a fax can't be sent to the destination for any reason, it's tried for several times. This setting limits the number of tries. If all tries failed, the job will be moved to the failed dir (see `fax_spool_dir`) and the user will get a mail.

This option is optional. If not given, it defaults to 10 tries.

`send_delays="60,60,60,300,300,3600,3600,18000,36000"`

When a fax can't be sent to the destination for any reason, it's tried again. This setting specifies the delays in seconds between subsequent tries. The different values are separated with commas and *no blanks*. The list should have send_tries-1 (see `fax_send_tries`) values - if not, surplus entries are ignored and missing entries are filled up with the last value. The default should just be ok giving you increasing delays for up to 10 tries.

This option is optional. If not given, it defaults to the list shown above.

`send_controller="1"`

If you have more than one ISDN controller installed (some active cards for more than one basic rate interface like the AVM C2 or C4 are also represented as multiple controllers for CAPI applications like CapiSuite), you can decide which controller (and therefore which basic rate interface) should be used for sending your faxes. All controllers are numbered starting with 1. If you're not sure which controller has which number, increase the log level to at least 2 in CapiSuite (see Section 1.2.2), restart it and have a look in the log file where all controllers will be listed then. Unfortunately, CapiSuite isn't able to use more than one controller for sending faxes at the moment, so no list is allowed here. If you have only one controller, just leave it at `1`

This option is optional. If not given, it defaults controller 1.

`outgoing_MSN="<your MSN>"`

This number is used as our own number for outgoing calls. If it's not given, the first number of fax_numbers is used (see Section 1.3.3.1.2). If this one is also empty, the user can't send faxes. Please replace with one valid MSN of your ISDN interface or leave empty. This value can be overwritten in the user sections individually.

This option is optional. If not given, it defaults to empty.

`outgoing_timeout="60"`

Default setting which defines how many seconds we will wait for a successful connection after dialing the number. This value can be overwritten in the user sections individually.

This option is optional. If not given, it defaults to 60 seconds.

`dial_prefix=""`

If anything is entered here, it will be used as a prefix which is added to any number given to **capisuitefax** as prefix. This is e.g. very helpful if your ISDN adapter is connected to a PBX which needs "0" for external calls. It's also possible to disable its usage later for a certain fax document, so setting this will certainly not prevent you from placing internal calls without prefix.

This option is optional. If not given, it defaults to an empty prefix.

`fax_stationID="<your faxID>"`

Default fax station ID to use when sending a fax document. The station ID is usually the number of your fax station in international format, so an example would be "+49 89 123456" for a number in Munich, Germany. Station IDs may only consist of the "+"-sign, spaces and the digits 0-9. The maximal length is 20. This value can be overwritten in the user sections individually.

This option is mandatory.

`fax_headline="<your faxheadline>"`

Default fax headline to use when sending a fax document. Where and if this headline will be presented depends on the implementation of your CAPI driver. The headline should have a reasonable length to fit on the top of a page, but there's no definite limit given.

This option is optional. If not given, it defaults to an empty headline.

`fax_email_from="<mailaddress>"`

You can set a default originator ("From"-address) for the e-mails CapiSuite sends here.

This option is optional. If you set this to an empty string, the destinator is used as originator (i.e. if "gernot" receives a fax, the mail comes from "gernot" to "gernot").

1.3.3.1.2. The user sections

`outgoing_MSN`

User specific value for the global option Section 1.3.3.1.1 above

`outgoing_timeout`

User specific value for the global option Section 1.3.3.1.1 above

`fax_stationID`

User specific value for the global option Section 1.3.3.1.1 above

`fax_headline`

User specific value for the global option Section 1.3.3.1.1 above

`fax_email_from`

User specific value for the global option Section 1.3.3.1.1 above

`fax_numbers="<number1>,<number2>,..."`

A list containing the numbers on which this user wants to receive incoming fax calls. These numbers are used to differ between users - so the same number must not appear in more than one user section! The numbers are separated with commas and *no blanks* are allowed. The first number of the list also serves as our own number for sending a fax if outgoing_MSN is not set (see `outgoing_MSN`).

If you want to use the same number for receiving fax and voice calls, please *do not* enter it here. Use the voice_numbers option instead (see Section 1.3.3.2.2) - the answering machine has a built in fax detection and can also receive faxes.

When this list is set to `*`, *all* incoming calls will be accepted for this user (use with care!). This is only useful for a setup with only one user which wants to receive any call as fax.

If for any reason *no destination* number is signalled for special MSNs (austrian telecom seems to do this for the main MSN, where it is called "Global Call"), you can use the special sign `-` which means "no destination number available".

This option is optional. If not given, the user can’t receive fax documents.

`fax_email=""`

If given, this string indicates email-addresses where the received faxes will be sent to. More addresses are separated by commas. If it is empty, they will be sent to the user account on the system CapiSuite is running on. The address is also used to send status reports for sent fax jobs to. If you don’t want emails to be sent at all, use the action option (see option `fax_action`) below.

This option is optional. If not given, the mail is sent to the system account.

`fax_action="MailAndSave"`

Here you can define what action will be taken when a call is received. Currently, two possible actions are supported:

`MailAndSave`

The received call will be mailed to the given address (see `fax_email` above) and saved to the `fax_user_dir` (see Section 1.3.3.1.1)

`SaveOnly`

The call will be only saved to the fax_user_dir (see Section 1.3.3.1.1)

This option is mandatory.

1.3.3.2. Configuration for the answering machine

This file holds all available config options for the answering machine.

It's read from `/etc/capisuite/answering_machine.conf` or `/usr/local/etc/capisuite/answering_machine.conf` (depending on the installation).

1.3.3.2.1. The [GLOBAL] section

`audio_dir="/path/to/audiodir/"`

The answering machine script uses several wave files, for example a global announcement if the user hasn't set his own and some spoken word fragments for the remote inquiry and the menu presented there. These audio files are searched in this directory. If user_audio_files is enabled (see `user_audio_files`), each user can also provide his own audio snippets in his user_dir (see `voice_user_dir`).

This option is mandatory.

`voice_user_dir="/path/to/userdir/"`

This directory is used to save user specific data to. It must exist and the user CapiSuite runs as must have write permission to it. It will contain one subdirectory for each configured user (named like his userid). The following subdirectories are used below the user-specific dir:

`user_dir/username/`

Here the user may provide his own audio_files (see also option `user_audio_files` below). The user defined announcement is also saved here.

`user_dir/username/received/`

Received voice calls are saved here.

This option is mandatory.

`user_audio_files="0"`

If set to `1`, each user may provide his own audio files in his user directory (see `voice_user_dir`). If set to `0`, only the audio_dir (see `voice_audio_dir`) will be searched.

This option is optional. If not set, it defaults to not reading own user audio files (0).

`voice_delay="15"`

Sets the default value for the delay for accepting an incoming call in (in seconds). A value of `10` means that the answering machine accepts incoming calls 10 seconds after the incoming connection request. This value can be overwritten in the user sections individually.

This option is mandatory.

`announcement="announcement.la"`

Sets the default name to use for user announcements. The announcements are searched in `user_dir/username/announcement` then. If not found, a global announcement containing the called MSN will be played. This value can be overwritten in the user sections individually.

This option is optional. If not set, it defaults to "announcement.la".

`record_length="60"`

Default setting for the maximum record length in seconds. This value can be overwritten in the user sections individually.

This option is optional. If not set, it defaults to 60 seconds.

`record_silence_timeout="5"`

Default setting for the record silence timeout in seconds. When set to a value greater than 0, the recording will be aborted if silence is detected for the given amount of seconds. Set this to 0 to disable it. This value can be overwritten in the user sections individually.

This option is optional. If not set, it defaults to 5 seconds.

`voice_email_from="<mailaddress>"`

You can set a default originator ("From"-address) for the e-mails CapiSuite sends here.

This option is optional. If you set this to an empty string, the destinator is used as originator (i.e. if "gernot" receives a voice call, the mail comes from "gernot" to "gernot").

1.3.3.2.2. The user sections

`voice_delay`

User specific value for the global option Section 1.3.3.2.1 above

`announcement`

User specific value for the global option Section 1.3.3.2.1 above

`record_length`

User specific value for the global option Section 1.3.3.2.1 above

`record_silence_timeout`

User specific value for the global option Section 1.3.3.2.1 above

`voice_email_from`

User specific value for the global option Section 1.3.3.2.1 above

`voice_numbers="<number1>,<number2>,..."`

A list containing the numbers on which this user wants to receive incoming voice calls. These numbers are used to differ between users - so the same number must not appear in more than one user section! The numbers are separated with commas and *no blanks* are allowed. The answering machine script does also automatic fax detection, so a fax can be sent to this number. When this list is set to `*`, *all* incoming calls will be accepted for this user (use with care!). This is only useful for a setup with only one user which wants to receive any call.

If for any reason *no destination* number is signalled for special MSNs (austrian telecom seems to do this for the main MSN, where it is called "Global Call"), you can use the special sign `-` which means "no destination number available".

This option is optional. If not set, the user won’t receive voice calls.

`voice_email=""`

If given, this string indicates email-addresses where the received faxes and voice calls will be sent to. If it is empty, they will be sent to the user account on the system CapiSuite is running on. More addresses are separated by commas. If you don’t want emails to be sent at all, use the action option (see `voice_action`).

This option is optional. If not set, the calls are mailed to the system account.

`pin="<your PIN>"`

The answering machine also supports a remote inquiry function. This function is used by entering a PIN (Personal Identification Number) while the announcement is played. This PIN can be setup here. If you don't want to use the remote inquiry function, just use an empty PIN setting. The PIN doesn't have a maximal length - but perhaps you should not use 200 digits or you perhaps won't be able to remember them (I won't at least). ;-)

This option is optional. If not set, remote inquiry is disabled.

`voice_action="MailAndSave"`

Here you can define what action will be taken when a call is received. Currently, three possible actions are supported:

`MailAndSave`

The received call will be mailed to the given address (see `voice_email` above) and saved to the `voice_user_dir` (see Section 1.3.3.2.1)

`SaveOnly`

The call will be only saved to the `voice_user_dir` (see Section 1.3.3.2.1)

`None`

Only the announcement will be played - no recording is done.

This option is mandatory.

1.3.4. Deleting old files

As written above, all incoming and outgoing calls will be saved on the local file system to assure nothing gets lost. There's no cleaning up done by CapiSuite, so these files will stay forever on your system if you don't clean them up from time to time.

As it's not very convenient to do this manually, I would advise to automate this process. cron is predestinated for such a task. On most modern GNU/Linux distributions, you can simply place scripts in `/etc/cron.daily` and they will be called automatically once a day.

An example for a bash script you can use is included in the CapiSuite distribution. Just copy `capisuite.cron` to `/etc/cron.daily/capisuite` and assure it has correct permissions (owner root, executable bit set).

Now edit the file `cronjob.conf` and copy it to your CapiSuite configuration directory (usually `/etc/capisuite` or `/usr/local/etc/capisuite`). It tells the cron job how long the files should be stored in the different dirs.

The following options are available:

`MAX_DAYS_RCVD="<value>"`

Files stored in the user receive directories which weren't accessed in the last `<value>` days are deleted. Set to `0` to disable this automatic deletion.

`MAX_DAYS_DONE="<value>"`

Files stored in the global done directory which weren't accessed in the last `<value>` days are deleted. Set to `0` to disable this automatic deletion.

`MAX_DAYS_FAILED="<value>"`

Files stored in the global failed directory which weren't accessed in the last `<value>` days are deleted. Set to `0` to disable this automatic deletion.

1.4. Using CapiSuite together with the default scripts

1.4.1. Receiving calls

Now this is a nice, short section. Once you have configured CapiSuite, the scripts and started CapiSuite successfully, there's nothing more you have to do. You'll get your mails as described in Section 1.3.2 and that's it. You only have to setup your mail program to receive local mails. Enjoy! :-)

1.4.2. Doing a remote inquiry

To do a remote inquiry, please enter your PIN (see Section 1.3.3.2.2) while the announcement of the answering machine is played. After some seconds you will get a "voice menu" telling you how to record your own announcement for your answering machine or how to playback the received calls.

1.4.3. Sending fax jobs

The default scripts for CapiSuite also include a commandline tool for sending faxes called **capisuitefax**.

capisuitefax will be called with some parameters telling it which file to send (it currently only supports PostScript files) and to which number. It will then enqueue the job converted to the right format into the

send queue from which it's collected by another CapiSuite script and sent to the destination. If the sending was completed successfully or failed finally after trying for some time, the according user will get an email telling him/her what has happened.

The following options are recognized by **capisuitefax**:

```
capisuitefax [-q] [-n] [-u user] [-A adr] [-S subj] -d number file1 [file2 ...]

capisuitefax [-q] -a id

capisuitefax -h

capisuitefax -l
```

`-a id`

Abort the job with the given id. To get a job id, use the `-l` option.

`-A adr`

The addressee of the fax. This option is (currently) only for informational purposes and will be quoted in the sent status mail.

`-d number`

The number which should be called (destination of the fax)

`-h`

Show a short commandline help

`-l`

Shows the jobs which are currently in the send queue.

`-n`

Don't use the configured dial prefix for this job. Useful for internal jobs.

`-q`

Be quiet, don't output informational messages

`-S subj`

A subject for the fax. This option is (currently) only for informational purposes and will be quoted in the sent status mail.

`-u user`

Send fax as another user. Only allowed if **capisuitefax** is called as user `root`. This is mainly helpful for realizing extensions to e.g. do network faxing.

`file1 [file2...]`

One or more PostScript files to send to this destination. More than one PostScript file will produce several separate fax jobs.

Chapter 2. Users Guide

In the last chapter you've seen how to use the default scripts distributed with CapiSuite. But the main goal in developing CapiSuite was not to provide a perfect ready-to-use application. I intended to develop a tool where you can write your *own* applications very easyly. I'll show you how to do this in the next sections.

2.1. Introduction to Python

As I thought about the scripting language I wanted to integrate into CapiSuite, my first idea was to develop an own, simple one. But as more as I looked into it, the more I found that a general purpose language will be much more helpful than re-inventing every wheel that I would need. So I looked for some easy to integrate (and to learn) language. The one I liked most was Python - and it also had a nice documentation about embedding, so I chose it and I'm still happy about that decision. :-)

So the first thing you'll have to do is to learn Python. Don't be afraid - it was developed as a beginners language and Guido (Guido van Rossum, the inventor of Python) has done very well in my opinion.

In the next few sections, I'll give you a short introduction to the features of Python you most probably will need for CapiSuite. As this shouldn't be a manual about Python or a tutorial in computer programming, I assume you're already familiar with the basic concepts of todays wide-spread procedural and object-oriented languages.

If not, I would advise you to get and read a book for learning Python - there are many available in different languages. The Python home page on http://www.python.org has also nice and comprehensive manuals and tutorials available for free.

2.1.1. Python Basics

Python supports most features you know from other common languages. Here's the syntax of the basic operations shown in a Python session. A Python session is another fine feature of its interpreter: just start it by typing **python** in a shell and you'll get a prompt:

```
gernot@linux:~> python
Python 2.2.1 (#1, Sep 10 2002, 17:49:17)
[GCC 3.2] on linux2
Type "help", "copyright", "credits" or "license" for more
information.
>>>
```

As you can see, the Python prompt is >>>. If you enter commands that span multiple lines, Python shows a second prompt: ...

```
>>> if (1==2):
...     print "Now THAT’s interesting!"
...
```

Ok, now let’s go on:

```
>>> # comments start with # at the begin of a line
>>> # now the usual first steps
>>> print "hello world"
hello world
>>> # variables
>>> a=5 # no separate declarations necessary
>>> b=a*2
>>> print b
10
>>> b=’hello’
>>> print b,’world’
hello world
>>> # python is very powerful in handling sequences
>>> a=(1,2,3) # defines a tuple (not changeable!)
>>> print a
(1, 2, 3)
>>> a[1]=2  # this must fail
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
TypeError: object doesn’t support item assignment
>>> a=[1,2,3] # defines a list (changeable)
>>> a[1]=7
>>> print a
[1, 7, 3]
>>> # control structures
>>> if (b==’hello’):
...     print "b is hello"
... else:
...     print "????"
...
b is hello
>>> # the for statement can iterate over sequences
>>> for i in a:
...     print i
...
1
7
3
>>> # replace positions 1 to 3 (without 3) with 0
>>> a[1:3]=[0]
>>> a
[1, 0]
>>> # a[-i] is the i-the element counted from the back
```

```
>>> a[-1]=7; a[-2]=8
>>> a
[8, 7]
```

2.1.2. Blocks, Functions and Exceptions

Blocks are grouped only by identation. No `begin`, `end`, braces (`{`, `}`) or the like are needed. This sounds very uncomfortable at the first sight, but it's really nice - you must always structure your code exactly how you *mean* it:

```
>>> for i in [1,2,3]:
...     print 2*i
...
2
4
6
>>> i=0
>>> while (i!=3):
...     print i
...     i+=1
...
0
1
2
```

Now let's see how to define functions and how to work with exceptions:

```
>>> def double_it(a):
...     return (2*a)
...
>>> print double_it(9)
18
>>> print double_it("hello")
hellohello
>>>
>>> # let's trigger a exception
>>> a=1/0
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
ZeroDivisionError: integer division or modulo by zero
>>>
>>> # now let's catch it
>>> try:
...     a=1/0
... except ZeroDivisionError,e:
...     print "You divided by zero, message was:",e
...
You divided by zero, message was: integer division or modulo by zero
```

2.1.3. Working with modules

Modules are a way to group functions together. They must be imported before you can use them and they give you a new object containing all functions. Let's play around with some of them:

```
>>> import time
>>> # what is in time?
>>> dir(time)
['__doc__', '__file__', '__name__', 'accept2dyear', ...]
>>> # So - what do all these functions do? Python can tell...
>>> print time.__doc__
This module provides various functions to manipulate time values.

[...]

Variables:

[...]

Functions:

time() -- return current time in seconds since the Epoch as a float
ctime() -- convert time in seconds to string
[...]

>>> # Could you please explain ctime in more detail?
>>> print time.ctime.__doc__
ctime(seconds) -> string

Convert a time in seconds since the Epoch to a string in local time.
This is equivalent to asctime(localtime(seconds)). When the time tuple
is not present, current time as returned by localtime() is used.

>>> time.time()
1044380131.186987
>>> time.ctime()
'Tue Feb  4 18:35:36 2003'
>>> import os
>>> os.getuid()
500
>>> import pwd
>>> pwd.getpwuid(500)
('hans', 'x', 500, 100, 'Hans Meier', '/home/hans', '/bin/bash')
```

Ok, now I hope you got a small idea of Python. Have fun with it. I had... :-)

If you have further questions, I would *really* advise you to continue with a good book or the documentation on http://www.python.org. Please don't ask general Python questions on the CapiSuite lists...

2.2. A first look on the incoming and idle scripts

In Section 1.2.1 I already told you that there are two kinds of scripts used in CapiSuite. Now let's have a closer look on them.

2.2.1. The incoming script

Every time a call is received by CapiSuite, it will call the incoming script - to be precise it will call a function named `callIncoming` in a python script located somewhere on your disk. This "somewhere" was defined in Section 1.2.2, remember?

So the given script must always define a function with the following signature:

```
def callIncoming(call,service,call_from,call_to):
 # function body
 ...
```

The parameters given by CapiSuite are:

call

reference to the incoming call. This will be used later in all CapiSuite-functions you call to tell the system which call you mean. You'll only pass this parameter on to other functions - the script can't do anything other with it (it's *opaque*).

service (integer)

Service of the incoming call as signalled by the ISDN, set to one of the following values:

- `SERVICE_VOICE`: voice call
- `SERVICE_FAXG3`: analog fax call
- `SERVICE_OTHER`: other service not listed above

call_from (string)

the number of the calling party (source of the call) as Python string

call_to (string)

the number of the called party (destination of the call) as Python string

The first task of the function should be to decide if it wants to accept or reject the call. If it accepts it, it will normally do something with it (receive a fax, record a voice call, play nice announcements, ...) and then disconnect. After it has done all necessary work, it should finish immidiately. In a later chapter, I'll present you some examples which should make things clearer.

Naturally, you can break down your application in more functions and perhaps more scripts, which will be called and/or imported recursively - but the starting point is always the *incoming script* containing `callIncoming`. If Python and CapiSuite are correctly installed, you should also be able to import and use any Python module.

2.2.2. The idle script

As the incoming script will only be started when a call comes in, we need another mechanism to initiate an outgoing call. As CapiSuite can't know when you plan to do so, it will just call a function named `idle` in the so called "idle script" in regular intervals. For configuring the intervals and where this script is located, please refer to Section 1.2.2.

The called function must have the following signature:

```
def idle(capi):
 # function body
 ...
```

The only parameter given by CapiSuite is:

capi

> This is a reference to an internal class of CapiSuite which handles the communication with the CAPI interface. You'll have to pass on this parameter to some CapiSuite functions. Nothing else useful you can do with it in your script. This parameter has internal reasons and will possibly (hopefully) go away some day in the future. Just pass it on when told to do so for now.

Now you can do what you want in this function. Most likely, you'll check for a job in an email account, look for a file to send in a special directory or so and place a call to send the job to the right destination.

Theoretically, you could also accomplish every other periodical task on your system in the idle script - but perhaps we should leave such general things to applications which were designed for this like cron. ;-)

As above, `idle` can call other functions or scripts if you like to and all Python modules are available for import.

2.3. Used file formats

Before we'll continue with writing scripts, please let me tell you some words about the file formats the CapiSuite core uses.

CapiSuite always reads and saves files in the native format as they are be expected and given by the CAPI ISDN drivers. This preserves it from having to convert everything from and to other formats thus reducing unnecessary overhead.

As these formats aren't that well-known and you will need special tools to convert or view/play them, I'll give you a short overview of how you can do this.

Most likely, your scripts will convert the special ISDN file formats to well-known ones for sending them to you via e-mail for example. Nevertheless, I'd advice you to store the received and sent files in the native CapiSuite formats somewhere. This will protect you from losing data in the case the conversion fails and will help you in debugging problems which may arise with your scripts.

All tools which I refer to here are described in Section 1.1.1.2. See there for informations how to get them.

2.3.1. Format of voice files (inversed A-Law, 8kHz, mono)

ISDN transmits voice data as waves with a sample-rate of 8kHz in mono. To save bandwith, a compression called A-Law is used (at least in Europe, other countries like the USA use u-Law which is quite similar to A-Law). For any reason beyond my understanding, they use a bit-reversed form of A-Law called "inversed A-Law".

2.3.1.1. Creating A-Law files

There are two possible ways to create A-Law files.

The first one is to call your computer with your phone (either use the default answering machine script and configure it as described in Section 1.3.3.2 or write a simple script yourself). Now, record whatever you want and take the created output file (when you use the default scripts please take the file from the user_dir, not the attachment of the mail as this is already converted) and use it.

You eventually want to trim the recorded file and remove unwanted noise and silence at the beginning and the end. This can easily be done by **sox** and **play** (both come together with the **sox** package).

sox is used to convert a file while **play** is used to just play it. Both support the same effects including the trim option. Both also detect what type of file you are using by looking at the suffix of your file name. So all your inversed A-Law files should be named like `something.la` (`.la` is the inversed form of `.al` which stands for A-Law).

So let's first try to find the optimal values for the trim effect by calling **play:**

```
play myfile.la trim <start-offset> <duration>
```

Now play around with start-offset and duration (both given in seconds) until you know the right values. If you found them, you can use **sox** to actually produce the needed file:

```
sox myfile.la outfile.la trim <start-offset> <duration>
```

You'll now get a file named `outfile.la` which should contain what you want.

The second way to create an inversed A-Law file is to record a normal WAV-file with your favourite sound-tools and convert it to the destination format using **sox**. You'll get the best results when your WAV file already is in 8kHz, mono, 8 bit format. **sox** is able to convert other waves if necessary but this usually will result in worse quality. You should also normalize your sound file to about 50% of the maximum amplitude.

You can convert WAV to inversed A-Law by calling (thx to Carsten Heesch for the tip):

```
sox myfile.wav -r 8000 -c 1 -b outfile.la resample -ql
```

If you used vbox in the past and want to convert your old message files, you can use the following command:

```
vboxtoau < infile.msg | sox -tau - outfile.la
```

2.3.1.2. Playing A-Law files

Again, there are two possibilities. The **play** command of **sox** is able to just play the inversed A-Law format without any conversion. Just call **play** with the filename as parameter:

```
play myfile.la
```

But you can also use sox to convert the A-Law files to the more common WAV format by just invoking:

```
sox myfile.la outfile.wav
```

The created `outfile.wav` can be played by nearly any audio player without problems.

2.3.2. Format of fax files (Structured Fax Files)

CAPI-compliant drivers will expect and provide fax files in a so called Structured Fax File (SFF). As this seems to be a CAPI-specific format, there are not much tools out there for GNU/Linux which are capable of handling it. Finally I found some small tools written by Peter Schäfer, which we can use here.

CapiSuite is also able to receive color fax files which will be stored in a special file format I called CFF.

2.3.2.1. Creating a SFF

In current Ghostscript releases, a patch from Peter has been included to produce SF files. To see if your Ghostscript already supports it, enter **gs --help** and look for the so-called **cfax**-device in the long device list presented to you. If it's not listed, you have to take a newer Ghostscript or recompile it, sorry. I don't know any other way to produce SFF currently.

You need a PostScript file (as produced by nearly every Linux program when you choose "print to file") first. Now you can call GhostScript to convert it to a SFF:

```
gs -dNOPAUSE -dBATCH -sDEVICE=cfax -sOutputFile=outfile.sff file.ps
```

If you're not sure if it worked you can use **sffview** as described below.

2.3.2.2. Viewing / converting from SFF

To simply view a received SFF, you can use the **sffview** program. It's a simple but useful tool for viewing SF files without the need to convert them. Just start it and you will get a GUI where you can open the desired file.

If you want to convert a fax file to a more common format, I recommend using **sfftobmp**. It supports quite some output formats like JPEG, TIFF, PBM or BMP. I prefer multipage TIFF files as this is the only format being able to store several pages in one file. To convert SFF to multipage TIFF, call:

```
sfftobmp -tif myfile.sff outfile.tiff
```

This will give you a TIFF file which you can convert now to nearly any other useful format with the TIFF tools, for example **tiff2ps**.

2.3.2.3. Color faxes - the CFF format

There exists an enhancement to the fax standard which allows to transfer documents in color. It's not very widely used, but as some people wanted it for CapiSuite, I added support for receiving this faxes with CapiSuite.

The CFF format (I don't know if this is an official name for the format) seems to be some sort of JPEG file with a special encoding. Most programs who can handle JPEG files should be able to open it. Perhaps you must rename it from `.cff` to `.jpg` first, before it will be recognized.

Currently, I don't know a nice way to create this format manually. Therefore, CapiSuite currently only supports the reception of these files. If someone knows more about it or knows the JPEG standards well, please contace me!

2.4. Tutorial: writing an incoming script

In this section, I'll show you how to code your own incoming script step by step. We begin with simply accepting every incoming call, playing a beep. The last example is a very simple but useful answering machine with fax recognition and receiving.

2.4.1. Basics and a really dumb answering machine.

Let's start with a very simple case: accept all incoming calls, beep and record something so we have an audio file to play with later. First of all, create a new directory somewhere which must be writable to `root`. We also need some test audio file for sending it. Let's take the beep which is distributed with CapiSuite.

```
mkdir capisuite-examples
chmod 777 capisuite-examples # make it world-writeable
cd capisuite-examples
cp /usr/local/share/capisuite/beep.la .
```

Perhaps you must change the path in the last line to reflect your installation.

Now copy and paste the example shown here to a file called `example.py` in this directory. Don't forget to change the `my_path`-setting.

Example 2-1. example.py

```
import capisuite❶

my_path="/path/to/the/just/created/capisuite-examples/"❷

def callIncoming(call,service,call_from,call_to):❸
 capisuite.connect_voice(call,10)❹
 capisuite.audio_send(call,my_path+"beep.la")❺
 capisuite.audio_receive(call,my_path+"recorded.la",20,3)❻
 capisuite.disconnect(call)❼
```

Let's walk through the script line by line:

❶ Import the capisuite module which holds all CapiSuite specific functions. All CapiSuite objects (functions, constants) in this module can be referenced by `capisuite.objectname` now. You

could also do a "`from capisuite import *`", which will insert all objects in the current namespace - but this isn't recommended as they may collide with other global objects.

> **Note:** The imported module `capisuite` isn't available as extra module, so you can't do this in an interactive Python session. It's included in the CapiSuite binary and only available in scripts interpreted by CapiSuite.

❷ Please change this to the real path you use for running these examples.

❸ Define the necessary function as explained in Section 2.2.1

❹ That's the first CapiSuite function we use: it accepts the pending call. The first parameter tells CapiSuite which call you mean. This parameter is necessary for nearly all CapiSuite functions. Ok, we only have one call now - but please think about an incoming script which also wants to place an outgoing call at the same time (for example to transfer a call). In this case CapiSuite wouldn't know which call you mean - so you must pass the reference you got to all connection related functions.

You can also tell CapiSuite to wait for an arbitrary time before accepting a call - that's what the second parameter is used for. So this script will wait 10 seconds before connecting with the caller. Don't think this parameter is useless and you could call a Python function (like `time.sleep()`) to wait instead. This won't work for any delay longer than 4 (or 8, depending on your ISDN setup) seconds as the call will timeout if an ISDN device doesn't "pre-accept" it by telling your network provider that it's ringing. CapiSuite will do so if necessary - so please just use this parameter.

❺ This call should be fairly self-explainig. Send the audio file stored in `beep.la`.

❻ Record an audio file for maximal 20 seconds - stopping earlier if more than 3 seconds of silence are recognized.

❼ Last, but not least - disconnect. Hang up. Finish. It's over.

CapiSuite configuration must be changed to use the just created script. Do this by editing your `capisuite.conf` and replacing the `incoming_script` value by the path to the file you just created (see Section 1.2.2) and restart CapiSuite.

Now test it: call any number which ends up at your ISDN card - if you have connected it to your ISDN interface, than any number (MSN) will do - if it's connected to a PBX, then you must call a number which was configured for the card in your PBX.

You should hear a beep and then you can speak something into this primitive answering machine. Please don't hangup before the script does as this case isn't handled yet. Just wait 3 seconds after saying something - it should disconnect after this period of silence.

If it doesn't work, you perhaps made an error when copying the script. In this case, please have a look at the CapiSuite log and error log, which you'll find in `/var/log/capisuite` or `/usr/local/var/log/capisuite` if you haven't changed the path.

A good trick to check for syntax errors is also to run your script through the normal Python interpreter. Do this by calling **python /path/to/your/example.py**. Naturally, it will complain about the `import capisuite` as this is no standard Python module. But before it does this, it will check the syntax of your script - so if you get any *other* error, please fix it and try again. If you only get

```
Traceback (most recent call last):
  File "../scripts/incoming.py", line 16, in ?
    import capisuite,cs_helpers
ImportError: No module named capisuite
```

then your script has a correct syntax.

I hope you got your script working by now - if not, don't hesitate to ask on the CapiSuite mailing lists *if* you have read a Python tutorial before.

In the next section we want to use an announcement, so please record some words with this simple script and move the created file `recorded.la` to `announce.la`.

2.4.2. Improving it to a useful (?) state

Well, it's really not nice that the caller mustn't hangup - and it's even worse that we do accept all incoming calls - perhaps by taking away your mothers important calls?

Let's quickly improve this.

Example 2-2. example.py, improved

```
import capisuite

my_path="/path/to/the/just/created/capisuite-examples/"

def callIncoming(call,service,call_from,call_to):
 try:❶
  if (call_to=="123"):❷
   capisuite.connect_voice(call,10)
   capisuite.audio_send(call,my_path+"announce.la")❸
   capisuite.audio_send(call,my_path+"beep.la")
   capisuite.audio_receive(call,my_path+"recorded.la",20,3)
   capisuite.disconnect(call)
  else:
   capisuite.reject(call,1)❹
   capisuite.disconnect(call)❺
 except capisuite.CallGoneError:
  capisuite.disconnect(call)❻
```

❶ CapiSuite will tell the script that the other party has disconnected by raising an exception named `CallGoneError`. So you should always put your code in a `try` statement and catch the raised

exception at the end of your script (or perhaps earlier if needed). This exception can be raised by call to a CapiSuite command.

❷ Have a look at the called number (please replace `123` with the number CapiSuite should accept)...

❸ Play the announcement we recorded in the last section. If you don't like it, simply record a new one and move the `recorded.la` again to `announce.la`.

❹ Ignore the call. The second parameter tells the exact reason for the reject - you can ignore a call (any other ISDN device or phone will still be ringing for that number) by using `1`, actively disconnect by using `2` or any error condition which is available in the ISDN specification (see Section B.1.2 for available codes).

❺ You always have to call `disconnect` at the end of your script, as this will wait for the end of the call, while `reject` only initiates the call reject. Otherwise you'll get a warning in the error log.

❻ This is the exception handler for `CallGoneError` - the exception CapiSuite raises when the call is disconnected by the other party. You should also call `disconnect` here to wait until the call is completely disconnected.

Save this to `example.py` again and test it. It's not necessary to restart CapiSuite as all scripts will be read at each time they're executed. Now you're allowed to hang up, too ;-).

2.4.3. Using sensible file names

We always used the same name to save the recorded message to which clearly isn't reasonable. We should really choose a new name for every new call. This isn't as simple as it may sound - you must assure that the used algorithm will also work for multiple calls arriving at the same time. Fortunately, the helpful programmer of CapiSuite had the same problem and so we can use the code he (hmmm... I?) has written.

The Python module `cs_helpers.py` contains some useful functions which are needed by the default scripts provided with CapiSuite but may be also helpful for the use in your own scripts. It contains the function `uniqueName` which does exactly what we need here. The syntax is:

```
filename=cs_helpers.uniqueName(directory,prefix,sufix)
```

The function will find a new unique filename in the given `directory`. The created filename will be "`prefix-XXX.suffix`" where `XXX` is the next free number started at 0. The next free number is remembered in a file `prefix-nextnr` and the created name is returned.

We can simply add this call to our script:

Example 2-3. using unique filenames

```
import capisuite,cs_helpers

my_path="/path/to/the/just/created/capisuite-examples/"
```

```
def callIncoming(call,service,call_from,call_to):
 try:
  if (call_to=="123"):
   filename=cs_helpers.uniqueName(my_path,"voice","la")
   capisuite.connect_voice(call,10)
   capisuite.audio_send(call,my_path+"announce.la")
   capisuite.audio_send(call,my_path+"beep.la")
   capisuite.audio_receive(call,filename,20,3)
   capisuite.disconnect(call)
  else:
   capisuite.reject(call,1)
 except capisuite.CallGoneError:
  capisuite.disconnect(call)
```

If you're interested in other functions which `cs_helpers.py` defines, just have a look at the reference at Section 2.6.4.

2.4.4. Automatic fax recognition and receiving

As last step, I want to show you how fax recognition and receiving works and how to switch from voice to fax mode.

Here's the last and most complicated example of this section. It'll introduce four new CapiSuite functions and shows how to split up the functionality in another function which is used by `callIncoming`. There are much changes which are described below - but most of them should be nearly self-explanatory. So I don't think this last step is too big. And you don't want to read 10 more steps here, do you? ;-)

Example 2-4. Adding fax functions

```
import capisuite,cs_helpers,os❶

my_path="/path/to/the/just/created/capisuite-examples/"

def callIncoming(call,service,call_from,call_to):
 try:
  if (call_to=="123"):
   filename=cs_helpers.uniqueName(my_path,"voice","la")
   capisuite.connect_voice(call,10)
   capisuite.enable_DTMF(call)❷
   capisuite.audio_send(call,my_path+"announce.la",1)❸
   capisuite.audio_send(call,my_path+"beep.la",1)
   capisuite.audio_receive(call,filename,20,3,1)
   dtmf=capisuite.read_DTMF(call,0)❹
   if (dtmf=="X"):❺
    if (os.access(filename,os.R_OK)):❻
     os.unlink(filename)
    faxIncoming(call)❼
   capisuite.disconnect(call)
```

```
  else:
   capisuite.reject(call,1)
 except capisuite.CallGoneError:
  capisuite.disconnect(call)

def faxIncoming(call):
 capisuite.switch_to_faxG3(call,"+49 123 45678","Test headline")❽
 filename=cs_helpers.uniqueName(my_path,"fax","sff")
 capisuite.fax_receive(call,filename)❾
```

❶ In this example, we need a normal Python module for the first time. The `os` module holds functions for all kinds of operation system services and is needed for deleting a file here.

❷ DTMF is the abbreviation for Dual Tone Multi Frequency. These are the tones which are generated when you press the digits on your phone and are usually used for dialling. They're also sent by modern fax machines before the transmission starts. Therefore, the same functions can be used for recognizing pressed digits and fax machines.

Before any DTMF is recognized by CapiSuite, the according function must be enabled by `enable_DTMF`.

❸ All audio send and receive functions support abortion when a DTMF tone is recognized. This is enabled by passing "`1`" as last parameter. It will also prevent the function from starting if a DTMF char was recognized *before* but not yet read by the script.

❹ CapiSuite stores all received DTMF signals in a buffer from where they can be read at any time. Reading is done by `read_DTMF` which also clears the buffer. It will return all received characters in a string, so if the caller presses "3","5","*", you'll get "35*".

The `0` tells CapiSuite not to wait for DTMF signals - if none are available, it will simply return an empty string. It's also possible to specify that it should wait for a certain amount of time or until a certain number of signals have been received.

> **Note:** Please note that it's not necessary to check for received DTMF after each audio send or receive function. Simply enable the DTMF abortion in all commands in a block and check for received tones after the whole block.

❺ Fax machines send a special tone which is represented as "`X`" by the CAPI. So if you receive the string "X", a fax machine is calling and we should start our fax handling routines.

❻ Possibly, the announcement was so short that the recording has started already before the fax is recognized. We won't save a file containing only the fax beep and so we test if it was created (`os.access` checks for the existence of a file) and delete it if needed by calling `os.unlink`.

❼ Fax handling was realized in a separate function which is called here.

❽ So far, this connection is in voice mode (which was set by using `connect_voice`). If we want to receive a fax now, the mode must be changed to fax. This is done by `switch_to_faxG3`. As the fax protocol needs some additional parameters, they must be given here. The first string is the so called *fax station ID* which is sent to the calling fax and shown in it's protocol, while the second one is a

fax headline. This headline is mainly used for sending faxes. To be honest, I personally don't know if it has any sense to specify this if you only want to receive a fax. But it surely won't harm ;-). If someone knows this for sure, please tell me.

> **Note:** If you want to use an own number solely for fax purposes, you should *not* use `switch_to_faxG3`. Use `connect_faxG3` instead.

❾ After the connection has been set to fax mode succesfully, we can receive the fax document finally. The used function `fax_receive` gets a new name which is again created by calling `cs_helpers.uniqueName` as above.

Congrats. You've finished my small tutorial. Now it's up to you - you can play with the created script and try to make it more complete. There's still much to do - sending received calls to a user via e-mail, log connections, ... If you want to complete this script, Section 2.7 will be helpful. You can also read on here to have a short look on the idle scripts, followed by a quick overview of the structure of the default scripts shipped with CapiSuite.

2.5. Example for an idle script

After we've seen how to handle incoming calls, a very short introduction to initiate outgoing calls by using the idle script will follow.

As written before, the idle script will be called by CapiSuite in regular intervals allowing you to look for stored jobs somewhere and sending them to the destinations.

The example shown here will look for a file `job-XXXX.sff` in the example directory we created in the last section. This file will be faxed to the destination indicated by `XXXX`. If you have no valid destination where you can send test faxes to, how about using CapiSuite as source and destination at the same time? In this case, replace `XXXX` by the number your incoming script handles. This won't work if your ISDN card can't handle two fax transfers in parallel (some old AVM B1 cards have this limitation, for example).

We now need one or more fax files in the SFF format for our tests, so please create some with a name like the one shown above. If you don't know how to do this, please refer to Section 2.3.2.1.

If I want to develop a CapiSuite script but am not really sure how to do it, I often start by coding a normal script which I can test without CapiSuite. So let's create a script which searches the files and extracts the destination numbers first. If this works, we can continue by adding the CapiSuite specific calls later.

Example 2-5. idle_example.py

```
import os,re❶
```

```
my_path="/path/to/your/capisuite-examples/"

files=os.listdir(my_path)❷
files=filter (lambda s: re.match("job-.*\.sff",s),files)❸

for job in files:❹
 destination=job[4:-3]❺ # Hmmm.. Is this right?
 print "found",job,"to destination",destination
```

❶ We know the `os` module already. `re` provides functions for searching for regular expressions. If you don't know what regular expressions are, please read for example the Python documentation for the `re`-module or some other documentation about them. It's too complicated to explain it here.

❷ `os.listdir` returns the files in a given directory as list.

❸ This line is a little bit more tricky. It filters out all filenames which doesn't follow the rule *starting with "job-", then any number of chars, ending with ".sff"* from the list. This is done by the `filter` function. The function expects the name of a function which checks the rule as first parameter and the list to filter (`files`) as second one.

We could now define a new function and use its name here, but the `lambda` keyword allows a much more elegant solution: it defines a "nameless function" with the parameter `s`. The function body follows directly behind and consists of a call to `re.match` which checks if the given string `s` matches the expression.

❹ Iterate over all found filenames.

❺ The destination is extracted from the given filename by using string indexes.

Now, save the script as `idle_example.py` in our example dir and run it by calling **python idle_example.py**.

If you have provided SFF files with the right names they should be shown line by line now. But... Obviously something doesn't work right here. The destination includes the "`.`". Indeed, I've made a mistake when indexing the string. It should be `destination=job[4:-4]` instead of `[4:-3]`. So let's change that and test again. It should work now. That's the reason why I prefer to code such scripts outside of CapiSuite first. Debugging is much faster this way...

As we know now that the basic parts work, we can add the real communication functions.

Please save this example to `idle_example.py` in your example directory, again.

Example 2-6. idle_example.py, version for CapiSuite

```
import os,re,capisuite

my_path="/path/to/your/capisuite-examples/"
my_number="678"❶
my_stationID="+49 123 45678"
```

```
my_headline="example headline"

def idle(capi):❷
 files=os.listdir(my_path)
 files=filter (lambda s: re.match("job-.*\.sff",s),files)

 for job in files:
  destination=job[4:-4]
  capisuite.log("sending "+job+" to destination "+destination,1)❸
  try:
   (call,result)=capisuite.call_faxG3(capi,1,my_number,destination,
                60,my_stationID,my_headline)❹
   if (result!=0):❺
    capisuite.log("job "+job+" failed at call setup with reason "
                  +str(hex(result)),1)
    os.rename(my_path+job,my_path+"failed-"+job)❻
    return❼
   capisuite.fax_send(call,my_path+job)❽
   (result,resultB3)=capisuite.disconnect(call)❾
  except capisuite.CallGoneError:
   (result,resultB3)=capisuite.disconnect(call)

  if (result in (0,0x3400,0x3480,0x3490) and resultB3==0):(10)
   capisuite.log("job "+job+" was successful",1)
   os.rename(my_path+job,my_path+"done-"+job)
   return
  else:
   capisuite.log("job "+job+" failed during send with reasons "
                 +str(hex(result))+","+str(hex(resultB3)),1)
   os.rename(my_path+job,my_path+"failed-"+job)
```

❶ Some parameters for sending the fax are set here. `my_number` is your own number which is used for sending the fax. `my_stationID` is the fax station ID, which will be transmitted to the other fax machine and shown on the sent fax page. Only digits and "+" are allowed. You can also define a short text which will show up in the fax headline in `fax_headline`.

❷ As explained in Section 2.2.2, you have to define a function called `idle` which will be executed in regular intervals by CapiSuite then. So all code has been moved to this function.

❸ We can't print messages to stdout as the script will run in the context of a daemon. So CapiSuite provides functions for creating entries in the CapiSuite log file. `log` expects at least two parameters: the message and a log level. This level corresponds to the log level setting in the global CapiSuite configuration (see Section 1.2.2). If the level of the message is *less or equal* the level set in the configuration, it is printed to the logs. So you can insert messages for debug purposes which aren't printed to the logs in normal operation by using levels higher than 1.

❹ This function initiates an outgoing call using the fax service. The parameters are:

- reference to the Capi object you got from CapiSuite (parameter to `idle`).
- the (number of the) controller to use for outgoing calls. The first controller has always number "1".
- own number to use for the outgoing call

- destination number to call
- maximum time to wait for a successful connection in seconds
- the fax station ID
- fax headline

The function returns a tuple containing a reference to the created call and an error value.

❺ This block checks if the connection was successful. For a detailled description of possible error values, please see the Section 2.7. 0 means "everything was ok, call is established".

❻ If the call wasn't successful, rename the fax file to prevent the script from sending the same file over and over.

❼ Don't forget to exit the `idle` function if the call couldn't be established!

❽ Another very simple CapiSuite command: send the given file as fax document.

❾ We previously ignored the reasons *why* a call was disconnected. Now we have to analyze them because we need to know if the file was transferred succesful. Therefore, `disconnect` returns a tuple containing of the physical and logical error value. Every ISDN connection contains one physical and (at least) one logical connection. One could imagine the physical connection as "the wire" connecting us to our destination, while the logical connection refers to the fax protocol which uses this "wire". You have to look on both values to see if everything was ok.

(10) Allowed values for the physical disconnection are 0,0x3400,0x3480 and 0x3490. These all mean "no error occured, call was disconnected normally". The logical value may only be 0 if everything went ok. For further information on error values, please refer to Section 2.7.

After you've saved the file and changed the default values to your own configuration, please alter the value of `idle_script` in the CapiSuite configuration to point to this script as described in Section 1.2.2.

Restart CapiSuite and watch the logs. Some minutes later, the `job-XXX.sff` files should've been sent and renamed to either `done-job-XXX.sff` or `failed-job-XXX.sff`. If the job failed, please consult the error log and the error values explained in Section 2.7 and Appendix B.

Hopefully, this tutorial helped you in understanding how to code your own scripts. Please continue with changing the examples or the files distributed with CapiSuite (read Section 2.6 before). You will find a complete reference of the available commands in Section 2.7.

If you have any trouble in getting your scripts up and running, please use the CapiSuite mailing lists. And don't forget to have fun. ;-)

2.6. Structural overview of the default scripts

2.6.1. incoming.py

The incoming script handles all incoming connections. It reads two configuration files containing all necessary data which were described in detail in Section 1.3.3. The overall structure will be described here giving you an overview of how it is implemented.

Firstly (after importing some necessary modules), it defines the necessary function `callIncoming` which will call all other functions if needed.

2.6.1.1. function `callIncoming`

This function starts with a call to `cs_helpers.readConfig` to read the configuration. It then iterates through all sections representing the configured users (except `GLOBAL`) to see if the called number belongs to any user. If a match is found, the user and the defined service are saved to `curr_user` and `curr_service`.

If no match was found (`curr_user` is empty), the call is rejected and the function returns. Otherwise the directory to use for incoming fax or voice data is determined and created if not existing yet.

The last thing the function does, is to produce a log entry, accept the call with the right service (fax or voice) and call either `faxIncoming` or `voiceIncoming`. It also defines an exeception handler for `capisuite.CallGoneError`.

2.6.1.2. function `faxIncoming`

`faxIncoming` is quite straight forward: it creates a uniqe filename, calls `capisuite.fax_receive`, disconnects and logs the disconnection reasons. Then it checks if a fax have been really received succesfully (i.e. if the file exists). If yes, it creates a description file for it, **chown**'s the file to the right user and sends the file as mail.

2.6.1.3. function `voiceIncoming`

`voiceIncoming` has much more features to accomplish like fax recognition and switching to fax mode, starting a remote inquiry etc.

It starts by determining the directory to use and creating a unique filename. Also, the PIN for remote inquiry is saved to a private variable. There are two possibilites now: the user has already an own announcement - in this case it's played now. Otherwise, a predefined announcement containing of a

general announcement and the number which was called is played. If recording a message wasn't disabled (by setting `voice_action` to `None`), it starts now after a beep.

All `audio_send` and `audio_receive` calls used so far had DTMF abortion enabled and so the script "falls through" all these calls after a DTMF signal was recognized. After them, `read_DTMF` is used to see if any such signal have been found. "`X`" represents the fax tone and triggers a switch to fax protocols and a call to `faxIncoming`. Any other received signals are interpreted to be a part of the PIN for remote inquiry and so a loop which waits 3 seconds after each tone for the next one is entered. If a valid PIN is entered, it starts the `remoteInquiry`. After three wrong attempts, it will disconnect.

After disconnecting and logging, a description file is written (if the recorded file exists), both files will be **chown**'ed to the right user and the recorded message will be mailed to him/her.

2.6.1.4. function `remoteInquiry`

The `remoteInquiry` starts by creating a lock file and acquiring an exclusive lock on it to prevent two parallel remote inquiries for the same user. If the lock can't be acquired, an error message is played and the function returns. If locking has succeeded, a list of the recorded voice calls is compiled by listing the user directory, filtering and sorting it. Now, a file called `last_inquiry` is read when it exists. It contains the number of the last heard message. With this information, the old messages can be filtered out to a separate list and thus the caller can listen to messages he doesn't know already first.

The number of new messages is said, followed by a small menu where the caller can choose to either record an announcment or hear the recorded messages. If he chooses announcement recording, the function `newAnnouncement` is called, otherwise `remoteInquiry` will continue.

Now, a loop will first iterate over the new and then the all old messages. It starts by telling the caller how much messages have been found. Then all messages will be played, repeating the following steps for each one:

- read the description file of the current message
- play an information block containing the current message number, source, destination, date and time of the call.
- play the message
- provide a menu where the caller can go on to the next or last message, repeat the current message or delete it

At the end, the caller will be informed that no more messages are available and the connection will be finished, followed by releasing the lock file and deleting it.

2.6.1.5. function `newAnnouncement`

`newAnnouncement` presents some instructions to the caller first. Then, the new announcement will be recorded to a temporary file. To give the user the ability to check it, it will be played, followed by a menu allowing him/her to save it or to repeat the recording. If the user has chosen to save it, it will be moved from the temporary file to `announcement.la` in the users voice directory and **chown**'ed to him/her. The call will be finished with an approval to the caller that it has been saved succesfully.

2.6.2. idle.py

The idle script is responsible for collecting jobs from the send queues (where they're stored by **capisuitefax**) and sending them to the given destinations. It reads its configuration from the files presented in Section 1.3.3, too.

2.6.2.1. function `idle`

After reading the configuration by calling `cs_helpers.readConfig` and testing for the existence of the archive directories needed, the userlist is compiled from the list of available sections.

For each user who has a valid fax setup (otherwise this user will be skipped), the send queue will be looked at. If the necessary queue directories don't exist, they'll be created. After that, a list called `files` with the names of all files in the send queue is created and filtered to only contain fax jobs.

For each found job, a security check is done to see if it was created by the right user. If this check was successful, a lock file is created and a lock on it is acquired. This prevents the **capisuitefax** command to abort a job while it is in transfer. After that, the existence of the file is checked (perhaps the job has been cancelled before we could acquire the lock?).

Now, the description file for this job is read and the starttime is checked. If it's not reached, the script will go on with the next job. Otherwise, some parameters are taken from the configuration and a log message is created. The file is transferred by calling `sendfax`. The results are stored and logged. If the job was successful, it is moved to the done dir and an approval is mailed to the user. If it wasn't succesful, the delay interval will be determined from the configuration and the new starttime is calculated by increasing the old starttime by this interval. A counter for the used tries is increased and the description file is rewritten with the new values. If the number of tries exceeds a given maximum, the job is moved to the failed dir and the error is reported to the user by mail.

Finally, the lock file will be unlocked and deleted.

2.6.2.2. function `sendfax`

This function handles the send process. After determining the MSN to use from either the `outgoing_MSN` setting or from the `fax_numbers` list, a call to the destination is initiated. If it fails, the function returns; otherwise the file will be sent and the connection finished.

2.6.2.3. function `movejob`

This is a small helper function used for moving a job and its accompanying description file to another directory.

2.6.3. capisuitefax

capisuitefax allows to enqueue fax jobs, list the current queue and abort jobs. It's not used directly by the CapiSuite system - it's a frontend for the users send queue directory. It has several commandline options - for an explanation of its usage, please refer to Section 1.4.3.

There are three helper functions defined first. *`usage`* prints out a small help if "`--help`" or "`-h`" was given as parameter or if a parameter isn't understood. *`showlist`* gets a listing from the users send queue directory and prints it nicely formatted as table. *`abortjob`* removes a job from the queue. It does this safely by using a lock file to not interfere with the sending process.

The main code of this script checks the given commandline options first. It sets several variables to the given values. After some checks of the validity of the options, the rights of the user to send faxes and the existence of the necessary directories, it will fulfill the requested task. Either, `listqueue` will be called to show a listing of active jobs, `abortjob` to abort a job or the given files are processed and put to the queue.

To process a job, the existence of it and its format will be checked. Currently, only PostScript is allowed. The CapiSuite core itself only supports the SF format. Therefore, the files are coverted from PostScript to it by calling **ghostscript**. Finally, the description file for this job is created containing the given parameters like the destination number.

2.6.4. cs_helpers.py

The `cs_helpers.py` script contains many small helper functions used in the other scripts. These are:

`readConfig`

> Reads either the configuration files described in Section 1.3.3 or an arbitrary config file like the description files accompanying each received file or job to send.

`getOption`

Get an option from the given user section and fall back to the global section if it's not found.

`getAudio`

Get an audio file from the users directory or fall back to the global CapiSuite directory.

`uniqueName`

Construct a new file name in a given directory by using a given prefix & suffix and adding a counter. See also Section 2.4.3.

`sendMIMEMail`

Send an e-mail with attachment to a given user. Supports also automatic format conversion SFF -> PDF and inversed A-Law -> WAV.

`sendSimpleMail`

Send a normal e-mail without attachment to a given user.

`writeDescription`

Create a description file which can be read by `readConfig` later.

`sayNumber`

Supports saying a number using various wave fragments. Works only for german output currently.

For a detailled description of each function and its usage, please have a look at the script file itself. There are comments describing each function in detail.

2.7. CapiSuite command reference

CapiSuite provides an internal Python module called `capisuite` which can be imported as usual by `import capisuite`. Internal means, it's compiled in the CapiSuite binary and will only be found if CapiSuite interpretes the script.

A complete reference of all functions of this module is auto-generated from the CapiSuite sources and so you'll find it in the reference manual available locally on ../reference/group__python.html or online on http://www.capisuite.de/capisuite/reference/group__python.html.

As it doesn't make sense to duplicate the information here, please refer to it.

Note: These functions are implemented in C internally and so the reference document shows the C function header instead of the header how it would be defined in Python. So please ignore the function header shown there and only have a look at the description and the parameters given under

args. If this is too confusing, please tell me and perhaps I'll find a better way to auto-create this document someday then...

Appendix A. Acknowledgements

CapiSuite started as diploma thesis in winter semester 2002/03. I want to thank the following people for helping me with it:

- Karsten Keil from SuSE Linux AG (my tutor for the thesis) for his invaluable support and patience when answering me many ISDN questions, doing tests and for his suggestions concerning the architecture of CapiSuite
- Prof. Dr. Wolfgang Jürgensen from the UAS Landshut (my tutor from the UAS) for his help in ISDN questions during the thesis and for teaching me the ISDN basics in his lecture before
- Prof. Dr. Peter Scholz from the UAS Landshut (second tutor from the UAS) for his support and his suggestions
- my girl friend Claudia and her sister Bethina for proof-reading my thesis
- Peter Reinhart from SuSE for proof-reading my thesis
- many colleagues from SuSE for helping me with technical problems, esp. Andreas Jaeger, Andreas Schwab, Thorsten Kukuk and Andi Kleen
- Achim Bohnet for being the first one from the community trying to compile the CVS version and making me quite some suggestions how to improve it

Appendix B. CAPI 2.0 Error Codes

The CAPI interface used here has its own coding of standard ISDN error codes. Most of the errors described in Section B.2 are only important for developers of the CapiSuite core. As user, you only need to know the codes shown in Section B.1 as they'll be used in the CapiSuite Python functions like `capisuite.disconnect`.

You'll find a list of all the codes and a short description below. A detailled description of the CAPI codes can be found in the CAPI specification available at http://www.capi.org.

All numbers are given *hexadecimal*!

B.1. CAPI errors describing connection problems

All errors described here indicate some problem with the connection. These errors are also important for script writers as they're returned by some CapiSuite Python functions. See Section 2.7 for further details.

B.1.1. Protocol errors

Protocol errors indicate some problem during data transfer. Only messages for the transparent (voice) and fax protocols spoken by CapiSuite are shown here.

- `0` - Normal call clearing, no error
- `3301` - Protocol error layer 1 (broken line or B-channel removed by signalling protocol)
- `3302` - Protocol error layer 2
- `3303` - Protocol error layer 3
- `3304` - Another application got that call
- `3311` - T.30 (fax) error: Connection not successful (remote station is not a G3 fax device)
- `3312` - T.30 (fax) error: Connection not successful (training error)
- `3313` - T.30 (fax) error: Disconnect before transfer (remote station doesn't support transfer mode, e.g. wrong resolution)
- `3314` - T.30 (fax) error: Disconnect during transfer (remote abort)
- `3315` - T.30 (fax) error: Disconnect during transfer (remote procedure error)
- `3316` - T.30 (fax) error: Disconnect during transfer (local transmit data underflow)
- `3317` - T.30 (fax) error: Disconnect during transfer (local receive data overflow)
- `3318` - T.30 (fax) error: Disconnect during transfer (local abort)
- `3319` - T.30 (fax) error: Illegal parameter coding (e.g. defective SFF file)

B.1.2. ISDN error codes

These codes are ISDN error codes which are described by the ETS 300 102-01 standard in more detail. It's currently available for private use at http://www.etsi.org without fee. For details how the ISDN codes are mapped to the CAPI numbers see the CAPI specification, parameter "Info".

- `3400` - Normal termination, no reason available
- `3480` - Normal termination
- `3481` - Unallocated (unassigned) number
- `3482` - No route to specified transit network
- `3483` - No route to destination
- `3486` - Channel unacceptable
- `3487` - Call awarded and being delivered in an established channel
- `3490` - Normal call clearing
- `3491` - User busy
- `3492` - No user responding
- `3493` - No answer from user (user alerted)
- `3495` - Call rejected
- `3496` - Number changed
- `349A` - Non-selected user clearing
- `349B` - Destination out of order
- `349C` - Invalid number format
- `349D` - Facility rejected
- `349E` - Response to STATUS ENQUIRY
- `349F` - Normal, unspecified
- `34A2` - No circuit / channel available
- `34A6` - Network out of order
- `34A9` - Temporary failure
- `34AA` - Switching equipment congestion
- `34AB` - Access information discarded
- `34AC` - Requested circuit / channel not available
- `34AF` - Resources unavailable, unspecified
- `34B1` - Quality of service unavailable
- `34B2` - Requested facility not subscribed
- `34B9` - Bearer capability not authorized
- `34BA` - Bearer capability not presently available

- `34BF` - Service or option not available, unspecified
- `34C1` - Bearer capability not implemented
- `34C2` - Channel type not implemented
- `34C5` - Requested facility not implemented
- `34C6` - Only restricted digital information bearer capability is available
- `34CF` - Service or option not implemented, unspecified
- `34D1` - Invalid call reference value
- `34D2` - Identified channel does not exist
- `34D3` - A suspended call exists, but this call identity does not
- `34D4` - Call identity in use
- `34D5` - No call suspended
- `34D6` - Call having the requested call identity has been cleared
- `34D8` - Incompatible destination
- `34DB` - Invalid transit network selection
- `34DF` - Invalid message, unspecified
- `34E0` - Mandatory information element is missing
- `34E1` - Message type non-existent or not implemented
- `34E2` - Message not compatible with call state or message type non-existent or not implemented
- `34E3` - Information element non-existent or not implemented
- `34E4` - Invalid information element contents
- `34E5` - Message not compatible with call state
- `34E6` - Recovery on timer expiry
- `34EF` - Protocol error, unspecified
- `34FF` - Interworking, unspecified

B.2. Internal CAPI errors

These errors are mainly of interest for developers of the CapiSuite core. If you’re a user, you normally won’t need them.

B.2.1. Informative values (no error)

These values are only warnings and may appear in the extensive CapiSuite log in messages from the CAPI.

- `0000` - No error, request accepted
- `0001` - NCPI not supported by current protocol, NCPI ignored
- `0002` - Flags not supported by current protocol, flags ignored
- `0003` - Alert already sent by another application

B.2.2. Errors concerning CAPI_REGISTER

These errors may appear when the application starts and mostly indicate problems with your driver installation.

- `1001` - Too many applications.
- `1002` - Logical Block size too small; must be at least 128 bytes.
- `1003` - Buffer exceeds 64 kbytes.
- `1004` - Message buffer size too small, must be at least 1024 bytes.
- `1005` - Max. number of logical connections not supported.
- `1006` - reserved (unknown error).
- `1007` - The message could not be accepted because of an internal busy condition.
- `1008` - OS Resource error (out of memory?).
- `1009` - CAPI not installed.
- `100A` - Controller does not support external equipment.
- `100B` - Controller does only support external equipment.

B.2.3. Message exchange errors

These errors are really internal: they’re raised if the application calls CAPI in a wrong way. If they occur, it’s usually a bug which you should tell the CapiSuite developers.

- `1101` - Illegal application number.
- `1102` - Illegal command or subcommand, or message length less than 12 octets.
- `1103` - The message could not be accepted because of a queue full condition.
- `1104` - Queue is empty.
- `1105` - Queue overflow: a message was lost!!
- `1106` - Unknown notification parameter.
- `1107` - The message could not be accepted because on an internal busy condition.
- `1108` - OS resource error (out of memory?).
- `1109` - CAPI not installed.

- `110A` - Controller does not support external equipment.
- `110B` - Controller does only support external equipment.

B.2.4. Resource/Coding Errors

The errors described here are issued when the application tries to use a ressource which isn't available. These are mostly also bugs in the application. Please tell us.

- `2001` - Message not supported in current state
- `2002` - Illegal Controller / PLCI / NCCI
- `2003` - Out of PLCI
- `2004` - Out of NCCI
- `2005` - Out of LISTEN
- `2007` - llegal message parameter coding

B.2.5. Errors concerning requested services

The errors described here are issued when the application tries to request a service in a wrong way. Again, these are mostly bugs you should tell us.

- `3001` - B1 protocol not supported
- `3002` - B2 protocol not supported
- `3003` - B3 protocol not supported
- `3004` - B1 protocol parameter not supported
- `3005` - B2 protocol parameter not supported
- `3006` - B3 protocol parameter not supported
- `3007` - B protocol combination not supported
- `3008` - NCPI not supported
- `3009` - CIP Value unknown
- `300A` - Flags not supported (reserved bits)
- `300B` - Facility not supported
- `300C` - Data length not supported by current protocol
- `300D` - Reset procedure not supported by current protocol