SMS Applications

Stefan Frings, December 2005-January 2011

A book about setting up commercial SMS applications in style of a training course:

Description of recommended hardware

Description of recommended operating systems and software

Use of GSM modems for sending and receiving

Beginner instructions and example applications for:

Shell Scripts, PHP, Awk

MySQL

Apache Webserver

Email Server

Hints for enhancing the SMS Server Tools

Table Of Content

1. Introduction

In this chapter, I describe the intention of this book.

I write something about myself and how SMS and MMS services work in general.

1.1. About This Book

SMS is one of many phenomenons of the 20th century.

First ignored, these short messages became very popular. The phone network provider and many other companies reached significant revenues.

Many owners of mobile phones invest more money in SSM services, than in voice calls. In Germany there are about 62 million users of mobile phones, 85% of them also use SMS services.

Engineers collect measurements via SMS in a central place for investigation and car owners find stolen cars because they send their position via SMS. Alarm devices and machines notify their owners via SMS.

SMS is available since about 10 years, but not many people think about how many things can be done with it. I like to show you some practical examples from real live, starting with the idea, continuing with the physical construction and ending with the operation of the services.

I tried to keep this book as short as possible. I know that time is money and you probably don't want to wast your time in reading meaningless sentences. This business is fast and everybody who is not fast will not survive in that business.

Please read the book from the beginning to the end. The complexity increases from page to page and every chapter is based on the previous chapters.

Please apologize my bad English. I'm not a professional translator. But I tried to translate it as good as I can.

A lot of money is earned with SMS services

Start to get your part from this money!

You will find some registered company names, product names, logos and photos. They are all owned by the corresponding companies.

1.2. About The Author

My name is Stefan Frings. I was born in Germany 1974 and I learned a lot about electronics and computer starting as I was a child. In school, I developed experimental electronical sets together with my technical teacher for the following classes, and I repaired broken devices.

After job training to an engineer for telecommunications, I started working as a general EDV technician, changing to a programmer of SMS services and databases. Later I worked as a hardware developer.

I learned enough to get a job in the largest European company for mobile communication where I'm now responsible for about 250 central computers. In addition I teach my colleges sometimes.

My cost-free program SMS Server Tools became very popular. Many companies use it for different commercial applications.

If you have question about the book or if you like to tell me what I could do better, then please contact me via eMail. My address is smstools@meinemullemaus.de.

1.3. What is SMS?

SMS is the shortcut for

- Short

- Message

- Service

These are short text messages of up to 160 characters (7 Bit), that can be sent from one mobile phone to another.

Some devices support binary messages with 8 Bit for transferring small pictures, ringtones and configuration files for the mobile phone.
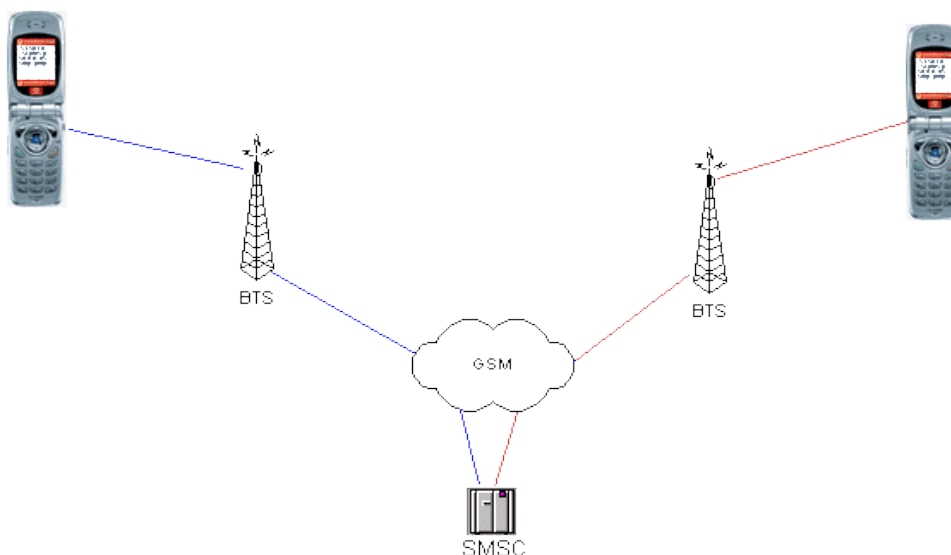
The short message is normally entered via the keypad of the mobile phone and stored onto the SIM card. Then is is sent through the signalling channel to the SMSC. This channel is also used for call setup.

There are many SMSC, therefore you can configure in your phone which one you like to use when sending a message.

The SMSC stores the messages for some days (in Germany max. 2 days) and sends them through the signalling channel to the receiver. If the sender requested it, the SMSC sends him a status report back to keep him updated about the status.

1.4. What Is MMS?

MMS is the short name for

- Multimedia

- Message

- Service

Multimedia messages can contain text, pictures, videos and audio. The latest generation of mobile phones, like the Sharp GX-10 on the right side support this function.

MMS is not part of this book, but this service is just coming up and has some things together with SMS. Therefore I like to write a little bit about it.

An MMS consists always of a text part with file attachments like an eMail. The MMSC transfers these message without modification to the destination phone.

When somebody sends an MMS, the sending phone connects to the internet (WAP) and send the text and attached files to the MMSC.

The MMSC stores it and checks, if the destination phone supports MMS. If not, the receiver gets a regular text SMS with the URL, where he can download the message with a computer and a webbrowser.

If the destination phone supports MMS, it gets a notification SMS. This SMS tells the phone to go online now and fetch the message through the internet (WAP).

At the moment you cannot create MMS applications without direct interwork with your local GSM network providers. Therefore I do not develop an MMS software at the moment.

2. Hardware And Software

In this chapter I will show you some recommended hardware and software that I use for commercial SMS applications.

2.1. SUN Server

Sun Microsystems delivers hardware of the highest quality level. SUN servers are known as very fast and they are also known for their stability. Compared to Intel based computers (PC), their multitasking is very smoothly. SUN server can run much more than 1000 programs at the same time without any problems.

The support of sun is excellent. They answer quickly to problem reports and their personal is very component.

2.2. PC Server

PC Server run with Microsoft Windows or Linux.

If you buy a PC server then take care about the quality of the hardware. The newest computers are often a bad solution because they often run unstable. For SMS applications the speed is less important that the quality.

Ask yourself these questions:

- Are there all connectors that I need?
- Is the mechanical construction stable and clean?
- Does the vendor offer enough support?
- Are spare parts long enough available?
- Can you put standard hardware into the device or does it only work with "original" parts from the same Manufacturer?
- Is the hardware fully compatible to the desired operating system (Windows/Linux)?

2.3. Serial Interfaces

In this chapter I describe how serial interfaces work. You will also find the layouts of different serial connectors and a device that enhances serial ports via ethernet.

2.3.1. How Serial Interfaces Work

Serial interfaces are called so because they send characters in a serial way. Each character is made of 8 bits and all the bits are transferred through one single wire one after the other.

Example:

The character A has the code 65 because it is the 65th character in the character-set table. This is the number 01000001 in binary notation.

To separate multiple characters they are surrounded by start bits with the fixed value 0 and one or two stop bits with the fixed value 1. The eight data bits are transferred from the right end to the left end, that means that the least significant bit is transferred first.

The data stream of one single "A" looks like this:

```
........ 0 1 0 0 0 0 0 1 0 1 ........
```

And two "AA" look like this:

```
........ 0 1 0 0 0 0 0 1 0 1 0 1 0 0 0 0 0 1 0 1 ........
```

The 0 is transferred as +12V and the 1 is transferred as -12V. If the interface is idle (shown as dots), the line remains at -12V.

The transfer of a single "A" looks like this:

```
+ + + + + + + + − + + + + + − + − + + + + + + + + +
```

Every serial port has two physical wires tro transfer data. The wire TxD sends data and and the wire RxD received them

With such a 2-wire connection (plus one ground wire) there is a method missing to control the data flow between two computers, that is called handshake. The sending computer never knows when the receiving computer is ready to receive and when it is to busy.

Therefore somebody reserved two characters for this signalling. Their name is Xon and Xoff. Anyway, for modem control they decided to add some more wires to allow modems to transfer every possible byte. This keeps programming easy. The additional lines are called handshake lines:

- With CTS the modem shows, that it is ready to receive data.

- With DTR the modem shows, that it is switched on.

- With DCD the modem shows, that it has a connection to the destination. Not all SUN computers have this DCD line but this is not important for SMS applications. They don't need this signal.

- With DSR the computer shows, that the serial port is in use

- With RTS the computer asks the modem to prepoare for sending data **and** the computer shows that it is ready to receive data from the modem. So this line has two meanings.

The handshake lines are controlled by the operating system. In addition the operating system creates a buffer that holds all received characters until the running program reads them. When the buffer of the computer or modem becomes full, they tell each other to stop sending data using the RTS and CTS line.

The specification of the serial RS232 port says that the signals 1 and 0 should be represented by -12V and +12V but voltages between 3-15V are acceptable. In practice voltages between 9 and 12V are used by most computers.

2.3.2. Connector Layouts

Please note: SUN computers have oft a very special pin layout!

25 pin SUB-D connector (Modem and PC)
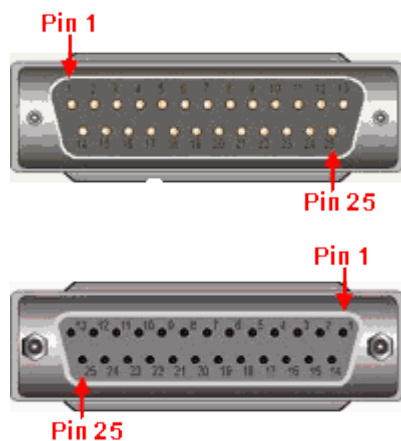
2 TxD

3 RxD

4 RTS

5 CTS

6 DSR

7 GND

8 DCD

20 DTR

9 pin SUB-D connector (Modem and PC)
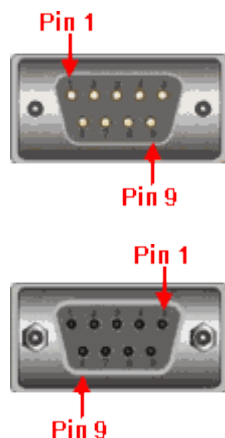
1 DCD

2 RxD

3 TxD

4 DTR

5 GND

6 DSR

7 RTS

8 CTS

RJ45 connector (Sun Fire V120 and V480)

Those servers do not have a DCD line at the serial port!
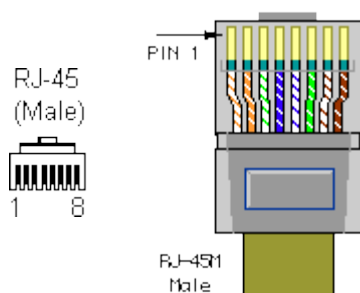
1 RTS

2 DTR

3 TxD

4 GND

5 GND

6 RXD

7 DSR

8 CTS

25 pin SUB-D connector (SUN Fire V880)

This server has two serial ports in one connector. There is no DSR line, therefore you have to configure the modem to ignore the missing DSR signal or you have to connect DSR with RTS at the modem end of the cable.
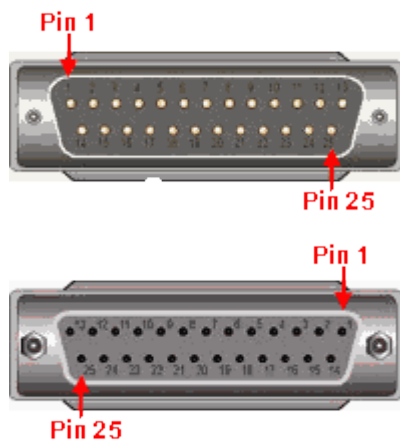
2 TxD Port A
3 RxD Port A
4 RTS Port A
5 CTS Port A
8 DCD Port A
20 DTR Port A

7 GND

11 DTR Port B
12 DCD Port B
13 CTS Port B
14 TxD Port B
16 RxD Port B
19 RTS Port B

RJ45 Anschluss (Digi Etherlite)
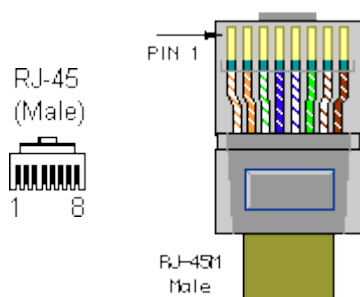
1 RTS

2 DSR

3 DCD

4 RxD

5 TxD

6 GND

7 DTR

8 CTS

2.3.3. Enhancing Serial Ports

If your server has not enough serial ports you can select between several methods of enhancing them:

- Use more servers

- Use manual switches

- Use serial port extender via USB

- Use serial port extender as PCI card

- use serial port extender via ethernet

The first method is surely to expensive and the second is not useful because your server shall work automatically. Serial ports can easily be expanded with USB converters but these devices are only delivered with Windows driver and therefore not usable with Linux or Solaris.

You could buy PCI cards with two or much more serial ports. But the number of free PCI slots may be very limited and you may not want to open the computer because you will loose warranty if you do that.

The last method, enhancing serial ports via ethernet, is my recommended method for all operating systems. There are different devices available. They have up to 32 serial ports and one ethernet connector that goes to your server. These serial port work exactly like local ports within the server. There are good devices available for every popular operating systems.

I would like to show you devices from the manufacturer Digi because they worked fine in my systems.

2.3.4. Digi Etherlite

Digi Etherlite devices enhance the serial ports of a server by two, eight, sixteen or thirty-two interfaces. If that is not enough for you, you could connect several Etherlites to the same computer.

For SMS applications I cannot imagine that somebody needs more than thirty-two interfaces.

Digi Etherlite devices work fine under Linux, Solaris and Windows. The software does not need to know that these ports are external. They work exactly as they were internal serial ports.

Because the connection between the Etherlite and the server is ethernet, you can place the modems far away from the server without the need of many expensive cables.

Digi Etherlite devices have an internal hardware to suppress ground-loops.

You may know these loops when you think about your home PC connected to your stereo HiFi and TV system. Many people notice a noise of 50 or 60 HZ in the speakers as soon as the stereo system is connected to the computer.

A ring-loop in the ground lines of all your cables introduce that noise. So you can break the loop using small transformators in the antenna cable of the TV and Radio or in the audio line to the computer.

On serial connections this noise can be so loud that it disturbs the data flow between modem and computer. Digicom Etherlite has internal hardware against this effect.

Please notice that you need a Windows PC to set up the IP-Address of these devices. There is no Linux or Solaris program available for this configuration. After this setup you can use Digi Etherlite devices also under Solaris and Linux.

You can get more information about Digi Etherlite on http://www.digi.com. There is also a list of vendors.

2.4. GSM Modems And Mobile Phones

In this chapter I show you some GSM modems that work fine for SMS applications. You will also learn which criteria are important when want to use a mobile phone and why they are not recommended for professional use.

2.4.1. Siemens GSM Modems

To operate Siemens GSM modems you need an addition 12V power supply with min. 300mA strength, a GSM antenna and a SIM card. If the power supply is not stabilized you should connect a capaciator of 2200yF 25V to it. This makes the operation more stable.

The Siemens TC35 terminal is the actual widely used GSM modem from Siemens.

If you want to receive status reports, you need to initialize it with AT+CNMI=2,0,0,2,1 otherwise it will discard every status report. This is special to Siemens modems.

2.4.2. Falcom GSM Modems

Falcom modems work really good. They are stable and I had never problems with them. I used always Falcom modems and changed all others as soon as the first GSM modem of this manufacturer came into the market.

Falcom has many different GSM modules and GSM modems for any special need. There are devices available with user programmable firmware and GPS antenna that can send GPS positions via SMS.

Falcom has good personnel that help if you have problems, and they also help well to decide which Falcom device is the best one for your need.

Currently, the Falcom Tango is the best choice for SMS Applications.

Falcom Samba

Initialize the Falcom Samba with the command **AT^SSMSS=1**, to be able to read out received short messages. This small USB stick has no external antenna, therefore it is not useable in some locations.

Vodafone Conect Card GPRS/UMTS (Manufactured by Option)

This Modem card has got an internal USB Controller and a USB2Serial adapter. The modem inside this card is a serial modem. This card can be used under Linux by loading the driver with the following commands:

modprobe ohci
modprobe usbserial vendor=0x0af0 product=0x5000

Das Device heisst dann /dev/usb/ttyUSB0

2.4.3. Mobile Phones

In general you can use mobile phones for SMS applications. But I learned from practice that they are not stable enough for commercial use. Mobile phones are typically absolutely stable as long as you do not connect them to a PC but they tend to hang often when they are connected.

It seems that mobile phones are not made for permanent use by a computer. The computer interface is made for notebooks and only for temporary use. 90% of all support questions about SMS are problems with mobile phones!

Therefore I recommend not to connect mobile phones to computers for commercial use. This may be fine for quick testing but nor for professional applications.

As long as the products change every few month I cannot describe actual devices in this book. So please go to a local phone shop If you want to buy a new one for SMS applications.

Please not that nearly every mobile phone receive status reports but they are not visible for an attached computer and cannot be read through the computer interface. Sending and receiving 8 bit messages is also not possible with many phones.

If you use a serial cable this cable must support RS232 protocol. FBUS and MBUS cables are not usable. Many not original cables have no hardware handshake wires but original cables have them always. Hardware handshake is necessary to ensure that the computer does not talk to fast to the phone.

The infrared interface is easy to use under Linux and seems to have less problems than serial cables. They are also more flexible because you can use them with every phone.

You cannot use the infrared interface with Windows because CygWin does not support them.

Not all mobile phones can send SMS over the infrared or serial interface. If you buy a mobile phone check the product description for GSM 07.05 compatibility or for the keyword "internal modem".

2.4.4. GSM Antenna

You can surely use every GSM antenna with GSM modems as long as their frequency matches the modems frequency (in europe 900/1800 Mhz and in USA 1900 Mhz).

I think the antennas in the photos below are very useful because they have a magnetic socket for setting them on top of cars. They can also easily set on top of 19 inch racks.

If you have problems with weak signal you should use the largest antenna that you can get, and keep the distance to other electrical devices as large as possible.

Please keep a distance of at least 10cm between many antennas and check if they do not touch anything except at the socket.

Do never switch a GSM modem on without an antenna or with a broken antenna-cable. This may destroy the modem. If you dont have an antenna and need an urgent workaround you could plug a simple wire of exactly 33cm length into the antenna connector.

2.5. Operating Systems

In this chapter I describe operating systems that I use for SMS applications.

After reading it, you should be able to decide which operating system you will use for your application.

I compare Linux, Solaris and Windows, keeping in mind that we want to run SMS applications on them.

2.5.1. Microsoft Windows

Windows is available in many different versions. For SMS applications the following versions are fine:

- Windows NT 4

- Windows 2000 Server

- Windows XP

, where you should note that Microsoft does not support Windows NT anymore.

Please do not use a workstation version of Windows for professional SMS applications, like Windows 95, 98 or Me. All programs of this book run fine on all Windows versions but is is not a good idea to let computers running 24h each day with a workstation version of Windows.

Many people complain about a continuously decreasing performance and sometimes the computers stop running suddenly. I can explain this:

The workstation versions have a much easier memory control mechanism. They are made to be as fast as possible giving as much CPU power and memory to the running programs (primarily games) as possible. The disadvantage is that these Windows versions do not defragment the memory which causes sooner or later a system slowdown and sometimes programs cannot allocate enough coherent memory blocks.

On Windows 95 and 98 I noticed many times that the clock in the taskbar ran slightly slower than my watch but the clock chip on the mainboard ran correct. After a restart of windows the taskbar showed again the correct time for a while. This affects all running programs.

Windows is a good thing if you want to set up a simple fileserver. But in many other cases you are forced to buy very expensive software (for example MS Exchange Server, Microsoft SQL, fax software, statistic programs and timed programs).

Software development for windows is typically not open-source, therefore you cannot modify programs if you want. The interconnections between programs of different manufacturer is typically problematic or impossible. Software manufacturers tend to separate from others so their products do not work very well together.

Of course you can get nearly everything that you need from Microsoft, but do you really want to depend on one single manufacturer?

Please keep also in mind that Windows and other Microsoft programs are very often the goal of virus attacks.

2.5.2. SUN Solaris

SUN offers the operating system Solaris for their own computers and for PC's. As I wrote this book, version 9 (also called 2.9) was actual.

Unfortunately the manufacturers of PC hardware do normally not develop Solaris drivers and it seems that they are not interested in operating systems different than Windows. Therefore using Solaris on a PC may be difficult.

A SUN Server runs with Solaris very stable and fast. SUN offers a good (and expensive) support, therefore I recommend this manufacturer if you need a system with less outages, best stability and professional support.

Solaris has not as many utilities as Linux. But it has everything that is necessary for daily work. And if you need some few additional programs you can get them normally free by downloading them from the internet.

A large directory with free Solaris software is on http://www.sunfreeware.com.

Most applications of this book are primarily developed under Linux, but Solaris is a Unix-style operating system like Linux and therefore you can run most of them also under Solaris. As long as you can get all software that you need for Solaris, using this operating system is a good decision. All applications of this book run on Solaris 6 and newer versions.

Before you start working with Solaris servers you should visit the following trainings from SUN:

- Basics of Solaris

- Solaris Administration Part 1

- Solaris Administration Part 2

- Solaris Network Administration

- Shell Programming For System Administrators

In these trainings the EDV specialist learns all important commands and configuration files of Solaris. These trainings are also useful for Linux administrators because there are only small differences between Solaris and Linux.

My opinion is: SUN is expensive but very good. Use it if stability and support is more important than money.

2.5.3. Linux

Linux is an operating system of Linus Torvalds and a lot of other developers. The name is a combination of Unix and Linus, the developer of the Linux core part (kernel).

Linux is a Unix-style operating system for PC's. The developers started when DOS was actual, and they continued their development quickly. Linux was started as a multitasking operating system with network functions and in some points it is more powerful than Windows.

The name Linux refers only to the Linux core. A lot of professional and hobby-programmers added more and more tools, drivers and applications. Some companies collect them together with the Linux kernel and add a comfortable installation program.

Every Linux installation is based on such a "distribution" and therefore the meaning of the word "Linux" has changed. The Linux distributions differ in support options and their size. I recommend Linux distributions from RedHat and SuSE.

RedHat is the model of all Linux distributions. RedHat is not as large as SuSE Linux. Another difference is that SuSE is very active in germany offering hardware, software and trainings. For international use RedHat is more popular. RedHat keeps all the programs normally in their original state while SuSE often changes them a little bit to simplify installation and configuration.

Decide for RedHat if you write programs that should also run on other Linux distributions or if you want English trainings. Decide for SuSE if you want an easy installation and/or German configuration.

Linux is the ideal operating system for programmers because it includes nearly every possible programming language and programming tools. Nearly all Linux programs are open-source, that means that their source code is free available for everybody. This allows programmers, changing things and analysing source codes. They can learn from each other and develop own programs based on existing ones.

As the opposite to Microsoft, who enforces own ideas as world-standard, the Linux programmers work together. This ensures that their program fit together.

Linux is free, therefore you get normally no support from the developers. But companies like SuSE and RedHat offer commercial support. You will need their help only seldom because the program authors are often reachable via eMail and they help as long as they have time for your problem.

There is also a lot of information in the internet that should help in most cases. Many people wrote pretty good guildelines about everything that a Linux administrator should know.

You can find Linux programs on http://www.freshmeat.net.

Normally Linux is much faster than Windows (after the boot process has finished). You can easily test this by compiling the Apache Webserver from the CD, that came with this book, under Linux and Windows and then comparing the times.

If you plan to develop applications for Linux, you need somebody with appropriate Linux knowledge. In Windows you can typically try to klick some buttons and see what happens and it is much easier to get help from somebody else. But for Linux you need to be familiar with its command line and you should be prepared to read many manuals. There are not many computer dealers with Linux knowledge and most of them sell their knowledge very expensive. There is free help available in the internet but without basic knowledge you will not understand most of it.

My opinion is: Linux is cheap and good. There is only limited commercial support. Linux users should have fun experimenting with software and reading manuals and they should have no problem searching help in the internet.

All applications in this book were tested under RedHat 7.3, SuSE 7.3 and SuSE 8.1. I'm pretty sure that they will also work fine on newer Linux versions but I don't know if they work on older Linux versions.
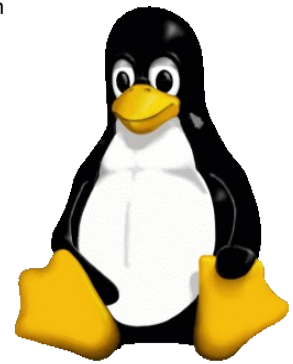
2.6. Software

This chapter shows you the programs that I use for SMS applications.

You will see that most programs were primarily developed for Linux and modified to run also on Solaris and Windows. Therefore you may miss colourful buttons and other graphical elements.

Anyway these programs work well under all three operating systems. Most documentation is available in the internet instead of printed books.

I recommend to read the original documentation of every program before you work with it.

2.6.1. PDU Spy

PDU Spy is a useful tool for programmers to decode and encode SMS messages. The program can also directly send and receive messages using a GSM modem.

Programmers can use this tool to find bugs in their own programs.

You can download it for free on the authors homepage http://www.nobbi.com.

There is no Linux version available, but you can start the Windows version on Linux using the Windows emulator wine. Solaris owner cannot use PDU Spy.

When you want to see all the AT commands going from the program to the modem, you can activate a logfile. Start PDU Spy with the option -d, then it will write a detailed logfile into the working directory.
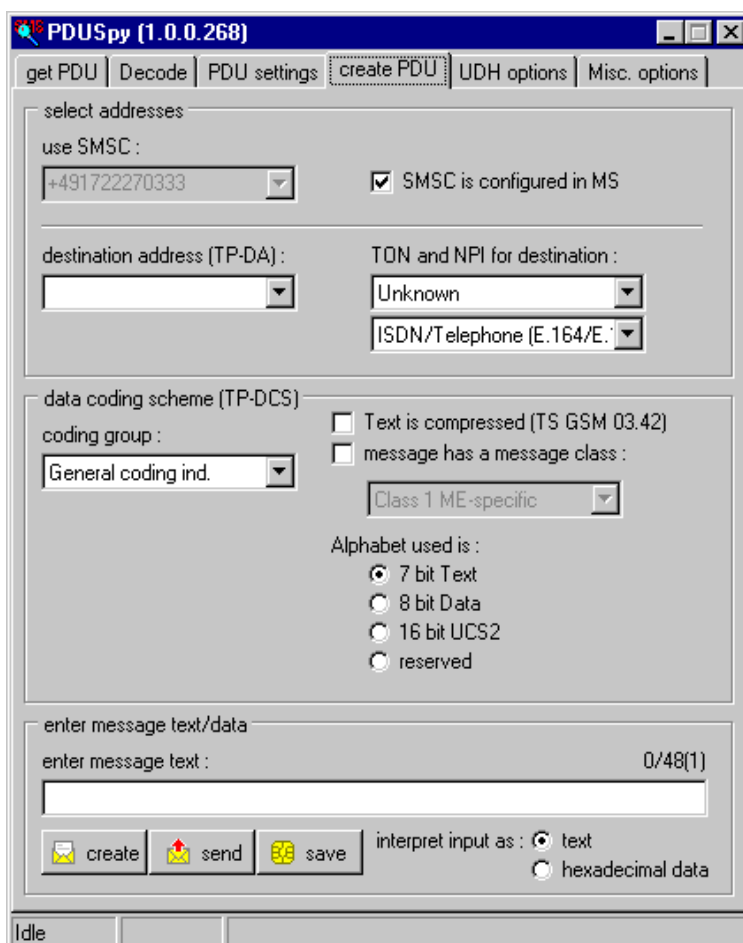
2.6.2. CygWin

CygWin is a set of powerful tools for programmers to develop software and migrate Unix software to Windows.

CygWin is not an emulator. All programs developed with CygWin are real Windows programs but their advantage is that they can use the same commands and functions as Unix programs.

The core part of CygWin is the GNU C Compiler, the cygwin1.dll library and a set of programs that belong to Unix standard. The package is maintained by RedHat. You can download it for free from http://www.cygwin.com.

I use CygWin as a base for all self-written Windows programs. This make it easy to use the usual Unix commands in the same way under Windows - the programs can easily run on both operating systems.

After the installation of cygwin you will get a "CygWin Bash Shell" window that is similar to the "MS DOS Command line" but it accepts Unix commands.

A typical installation needs about 150 MB disk space. Installing it is very easy. Download the "CygWin installer" and start it. This program offers you a list of all available packages. After selecting them the installer downloads them from the internet and installs them automatically.

2.6.3. Apache Webserver

The Apache webserver is the very most used webserver of the world. 80% of all Websites are running on Apache. It supports CGI scripts in Perl, PHP, TCL and more languages. It also support SSL encryption and password protection for web-pages.

You can download the source code and binaries on the website http://www.apache.org.

There is an extensive installation manual - also for quick-start - on the Apache website. For the first steps the program comes with a manual installed as the default example websites.

2.6.4. War FTP Daemon

The War FTP Daemon is a useful expansion to Windows computers with internet connection.

The Apache webserver supports only HTTP protocol but the War FTP Daemon support FTP protocol. Both together are necessary for a full usable webserver.

The file access permissions are easily configurable by mouse klicks and you can create as much user profiles as you want.

Install the War FTP Daemon if you want to offer FTP downloads and uploads to your customers.

You can download the program for free from http://www.jgaa.com, the authors private homepage.

2.6.5. MySQL Database

MySQL is an SQL Database - as the name already said.

I use MySQL always when an application has to store medium or much data and read them back. Using an SQL database is mostly useful for good performance and it simplifies application development. In most cases it is better to use MySQL than developing an own database.

MySQL is fast. You can access the data with graphical tools (e.g. Microsoft Access, StarOffice, MySQL Navigator) and modify them.

There is also a command line utility for the MS-Dos command prompt or Unix shell. This SQL Client is part of the MySQL package. It is useful to access the database with shell scripts or batch files. This book will use the client later in the examples.

If you plan to use a MySQL database you should read the book "MySQL: The definite Guide to Using, Programming and Administering MySQL4" from Paul Dubois, ISBN 0735712123.

MySQL stores data in tables like any SQL database. You can create as many tables as you like. Each table can have as many rows and columns as you like. The concept is similar to Microsoft Access, but SQL does not combine client and server applications into one single program. That makes MySQL much faster and more flexible to use.

MySQL databases are usable with all popular programming languages. In this book I will show you how to use MySQL with shell scripts and PHP scripts.

MySQL offers commercial support and trainings. You can get it for free on http://www.mysql.com.

2.6.6. PHP Programming Language

PHP is a script programming language for websites with dynamical content.

For example, if you want to create an online shop, you can use Apache as the webserver and MySQL as the database. With PHP you connect both together.

PHP is similar to the C++ programming language. It is very useful that you can mix HTML code and PHP code into one single file, you may know this from JavaScript.

The advantage of PHP is that the scripts are executed by the webserver and not by the webbrowser. This avoids compatibility problems and simplifies access to SQL databases. In many cases you can replace JavaScript by PHP.

There are a lot of PHP trainings available on the whole world. The extensive manual was translated into many languages. You can get PHP and the manual for free on http://www.php.net.

2.6.7. nnCron Lite

The program nnCron Lite allows a timed execution of programs. nnCron Lite is only available for Windows because a similar program called cron is part of the Unix standard and included in all Unix operating systems.

After installation, you modify the configuration file cron.tab by entering the times and program names that you like to start. The file format is identical to Unix.

If you like to do more than simply start programs, for example control them by simulating keyboard and mouse inputs, you should take a look at the larger version nnCron (without Lite).

You can download the program for free on http://nncron.ru.

If you think now that there is a cron packge in CygWin and wonder why I do not recommend to use it, read this: As I wrote this book the cron program from CygWin did not work. Therefore I searched for this alternative.

2.6.8. SMS Server Tools

The SMS Server Tools are a collection of programs that send and receive short messages.

To send a message you have to create a text file with destination number and the message text and you have to store this file in a special directory. The main program checks for new files in this directory and sends the messages using one or more GSM modems.

Received messages are stored as text files in another directory.

You can expand the programs and connect it to other programs and the internet by using shell scripts and a webserver.

This book is based on the original SMS Server Tools. This package will be a core part of your SMS applications.

You can download the original SMS Server Tools for free from my homepage http://www.meinemullemaus.de.

Enhanced versions are available on http://smstools3.kekekasvi.com/

2.6.9. EditPad Lite

EditPad Lite is a text editor for Windows. Compared to Notepad it has the following advantages:

- It can open mor than one file at the same time

- Powerful search/replace function

- Opens files from Unix and Macintosh (has no problems with line-breaks)

- Unlimited file size

- Unlimited undo function

- Can mark (select) rectangular blocks (with Alt-key)

- Better print function

- Convert OEM (Dos) to Ansi (Windows) character set

- Displays line and column number

If you like EditPad Lite you may take a look at EditPad Pro. This larger program has some more features:

- Hex-editor

- File comapare

- Combine many files into projects

- Bookmarks

- Special-character table

- Spell checking

- Sorting

- Regular expressions

- Six clipboards

You can get EditPad on the website http://www.editpadlite.com/editpadlite.html. The program is free for non-commercial use.

2.6.10. Fetchmail, Ssmtp And Ncftp

Fetchmail and ssmtp are two small eMail programs that should be included in every Linux distribution.

With fetchmail you fetch eMails from external Mailservers, e.g. using POP3 ptotocol.

With ssmtp you send eMails through external Mailservers using smtp protocol. Ssmtp simulates the sendmail program that is part of nearly all Unix installations. Sendmail is a powerful Mailserver while Ssmtp uses an external mailserver.

Both programs may be useful for you if you have no own Mailserver. They allow eMail transfer at the command line or within shell scripts and batch files.

Ncftp is an FTP client, useful for shell scripts and batch files. Different to the ftp command it is not interactive. You specifiy the filename and servername at the command line and ncftp uploads or downloads them automatically for you.

3. System Design

This chapter will describe important things to know when it comes to developing a commercial SMS application.

After reading this chapter you know all parts that you will need. You will also learn which details are important in the system concept.

This chapter supports you to select hardware for good stability.

3.1. Stability

Commercial applications need a stable system. Stability may be more important than costs.

Looking to the software you have not many choices. But when you use the programs that I described in the last chapters you do not need to fear hard problems. All these programs had a long development time and run very stable. If you find a bug, the authors help you typically.

For the hardware you can decide between many devices. For SMS applications the performance is not very important. In general you can use every PC or SUN computer. I's up to you to select the best computer for your needs.

SUN computer run very stable, but they run only with the operating system Solaris. You need a trained Solaris administrator and you should check, if all programs that you need are available for Solaris. If both is not sure, you cannot use SUN computers for your application.

For PC's it's easier because PC's run with Windows and Linux. Nearly all software is available for at least one of these operating systems. Therefore you first have to decide what hardware you want to use. There are high quality PC's available but also cheap devices. Cheap does not always meant that they are bad - I think you know that.

You should buy a PC only in a shop that offers enough support and spare parts. It would be quite bad if your SMS application does not work for many days because a spare part is not available.

Experienced computer vendors can immediately tell you what hardware components are stable and what components make often problems.

In case of doubt you may better buy a server from a good and large company, like Dell. With large vendors you can agree contracts about support and financial penalties when the system is down for a longer time that allowed.

Last but not least I like to give you a small tip about hardware construction of servers: Try to create your application with a minimum of hardware. As more parts the server has as more defects may occur. Less parts mean less outages and faster repair. If your server has parts that are not really needed, like TV-card, sound-card, internal Modem and so on, remove them.

The mechanical construction influences the stability of a system. The next chapter is written about that.

3.2. Mechanical Construction

The mechanical construction influences the stability of a system.

Imagine a wild large heap of devices that stand in the middle of a room on the floor. I think you have now a picture in your mind of somebody who falls over the cables and breaks plugs and devices. I'm pretty sure that you will not set up a system in that way.

A professional computer system is installed in a 19 inch rack. This size is the distance between the left and right edge of the rack that hold all the devices. You can install servers, switches, and all other computer hardware stable in such a rack. 19 inch racks are available with transparent doors and in full metal versions. They allow you to mount all parts stable and secured.

Connect the rack electrically to the ground. This ensures that the parts inside are secured against electrostatic loads. Today these loads cause very seldom a hardware defect but they can easily cause software crashes.

Place cables neat into the floor or in cable channels.

Do not put devices loosen in the rack. Everything should be secured with screws, cable binders or other materials. This ensures that earth quakes cannot move devices and cause bad connections at the cable ends or cause broken cables.

Check that the air condition is sufficient. Typically the temperature within a rack should not exceed 40°C. Most 19 inch racks have fans at the top or you can install them later if necessary. In some racks the airflow is only adequate when the doors are closed. That does not sound very logical but it is really sometimes necessary to keep the doors closed.

Do never place antennas inside a rack. This would cause a bad RF signal and would force the GSM modems to operate at maximum transmission power. The reflections inside the rack could affect other hardware to fail.

Put the rack at a location where you can open the front and back door. You should be able to reach all parts without problems. This is specially important when the system needs to be repaired quickly.

If you like to put a monitor and keyboard inside the rack then use a 15 inch TFT monitor and a special keyboard for 19 inch racks. They are a little bit narrow and fit into keyboard drawers. Some of them have also an internal trackball as a replacement for the mouse.

A stable system needs a stable power supply. The next chapter is written about that.

3.3. Power Supply

A stable power supply is necessary for a stable system.

Todays computer have good power converters that do not demand perfect power supply but they can react very sensitive to peak signals coming from the wall socket. Such peaks are often the cause of software crashes. You will find peaks in every building because they are caused by tube lamps, devices with motors, and thunderstorms.

Please install a filter against peaks in the power line. You can get them in nearly every electric store and they are not very expensive. Most filters secure your hardware also against over-voltages. But please note that there are also a lot of filters against over-voltages available that do not filter out peak signals.

Some of the filters are not really secure and some are quite good. You should contact an experienced electrician before you buy such a filter.

UPS

UPS is the short name for uninterrupted power supply (not the people with brown cars). UPS are devices that protect computers against short power failures using a backup battery. These devices are available with different capacities.

When you buy an UPS you should ensure that the power is enough (for example 1000VA) and that the battery capacity is sufficient. The capacity of an UPS battery can be calculated with this formula:

Multiply the voltage of the battery wit its capacity and divide the result by two. Example:

The battery has a label saying 12V 110AH. The calculation is 12V * 110AH /2 = 660VAH. That means that your USV can power a system with 660VA for about one hour. If the system would consume 1320VA, then the capacity would be only enough for 30 minutes.

The capacity is divided by two because the electronic of the UPS that converts the battery voltage to 230V AC (or 115V AC) has much less than 100% efficiency and the capacity of the battery reduces during its life-time. By dividing the capacity by two you get a practical usable value.

Batteries do not live forever. After the normal life-time the capacity reduces fast and therefore the maximum time of power failure that this device bridges reduces fast. You should check the battery at least every year. Normally a battery test means that your system will stop working during the test.

If you need a system that never stops, you need a special UPS with an automatic battery test. You should also check if you can replace a bad battery without switching the device off.

There are UPS devices, that generate not only an audible alarm when the battery power goes down but they can also shut down the computer safely. This function is very useful to protect data on harddisks against power failure when the battery capacity is not large enough to bridge a power failure.

This brings us to the next chapters that was written about data safety.

3.4. Data Safety

Data safety is important to ensure that your system works properly and that you earn continuously money from it.

When you set up a SMS application you spend a lot of time in installing and configuring the software. You do not want to repeat this steps later. In addition your system may store data that you need to print bills, data that you never want to loose.

Therefore you should secure your data by using hardware of good quality. But this is not enough. Also the best computers can crash and loose data. Harddisks do not live forever and you cannot calculate the date when a harddisk will crash. In addition a single mistake by the operator can destroy data.

Make regular backups!

Normally you would connect a tape drive to your server. I recommend 4mm DAT drives with DDS3 format. These drives store 12-24GB on each tape depending on the compression factor. The tapes are not expensive.

If you operate many servers, you may prefer a special backup server. They can be equipped with many tapes that you do not need to load them manually.

Like every tape, DAT tapes are not 100% safe. Dropouts in the material and wear and tear can cause loss of data on them. Therefore you should use at least three tapes in a rotating manner. If one tape fails you can restore from an older tape and minimize the data loss.

Don't use DAT tape to often. I keep notes of every use on the label and I replace an old tape by a new one after 20 usages.

Store tapes in a fire-secure place or in another location. Think about unexpected things, for example the building could burn out in fire together with your backup tapes, if you store them in the same building. Such a tape would not help you anymore to setup up a new system and print bills for the last billing cycle.

To save money, a backup on CD-R or CD-RW may be interesting. Software for CD-backups is available for Windows and is included in many Linux distributions. As I wrote about tapes you should also keep older CD-Backups in case the actual DVD becomes unreadable.

3.5. Redundancy

A redundant system is secured against hardware faults by duplication parts of it.

The weakest parts of every server are the harddisks. They break most times and replacing them is not easy because the data are lost and need to be restored.

You can protect your server against broken harddisks by installing more disks than only a single one.

A classical redundant harddisk system uses RAID-5. Such a system has 5 disks. The data are distributed over all 5 disks combined with CRC checksums. The checksums allow to recreate the original data when a single disk fails. When more than one disk fails at the same time, you will loose all data. RAID-5 systems offer the capacity of 4 disks but need a 5th disk for storing the checksums.

Today the capacity of a typical harddisk is so large that RAID-5 system do not make sense very often. For SMS applications this is surely the case.

I recommend to use the easier harddisk-mirroring. That means that your server has two harddisks that store always the same data. If one disk fails, the system continues working with the remaining disk.

For Windows you need a RAID controller to use such disk storages. On Solaris and Linux you can use a regular SCSI controller and emulate the RAID controller by software. This is cheaper but a little bit slower. Linux has a raid function in it's kernel, Solaris can add this function by separate software that is not part of the Solaris installation CD-Rom.

Please don't forget that a duplicated harddisk protects you only against broken disks. You could still loose all your data when the operator enters a wrong command or when a program crashes.

Duplicated harddisks make sense. It is also a good idea to use duplicated power converters (power supplies) and fans because these parts do also often break.

If you want to make the SMS application more stable you could run two servers with the same software. If one computer stops working, the other continues and you will only notice a reduced performance. To set up such a system you need a single file server that holds the message queues for both SMS servers. I think that you probably have already a good and stable file server.

Do not use the file server for other things than simply storing files. A file server that has no other jobs runs typically very stable and without any problems. Outages are very seldom.

If you want to use two file server for redundancy then you should take a look at http://www.linux-ha.org. This website discusses, how to setup a mirrored fileserver.

3.6. Firewall

As soon as your servers are connected to a large network - let's say the internet, you have to install a firewall.

Firewalls protect your computers against acces by unauthorized persons (hackers).

Firewalls block access through the network and allow only a few configured connections from authorized sources. For example you could use a firewall to allow only the administrator to log into the SMS server from his own desktop PC and no other workstation. And the internet customers can only acces websites from your webserver, nothing else.

A simple typical firewall has at least three network interfaces. One goes via a router to the internet, one goes into the company network and one goes to your servers.

Please do not run Firewall software on Windows, because Windows seams to have a lot of security wholes that affect the security of the firewall functionality.

Most Linux distributions come with firewall software, for Solaris you have to buy it. The german company SuSE offers a special Linux-Firewall distribution that contains only the Linux kernel and software that is necessary to set up a firewall. It has a very comfortable setup program specially written to simplify the setup up of a firewall.

3.7. Concrete Example System

Now I like to show you an example hardware construction that you could use to run every application of this book.
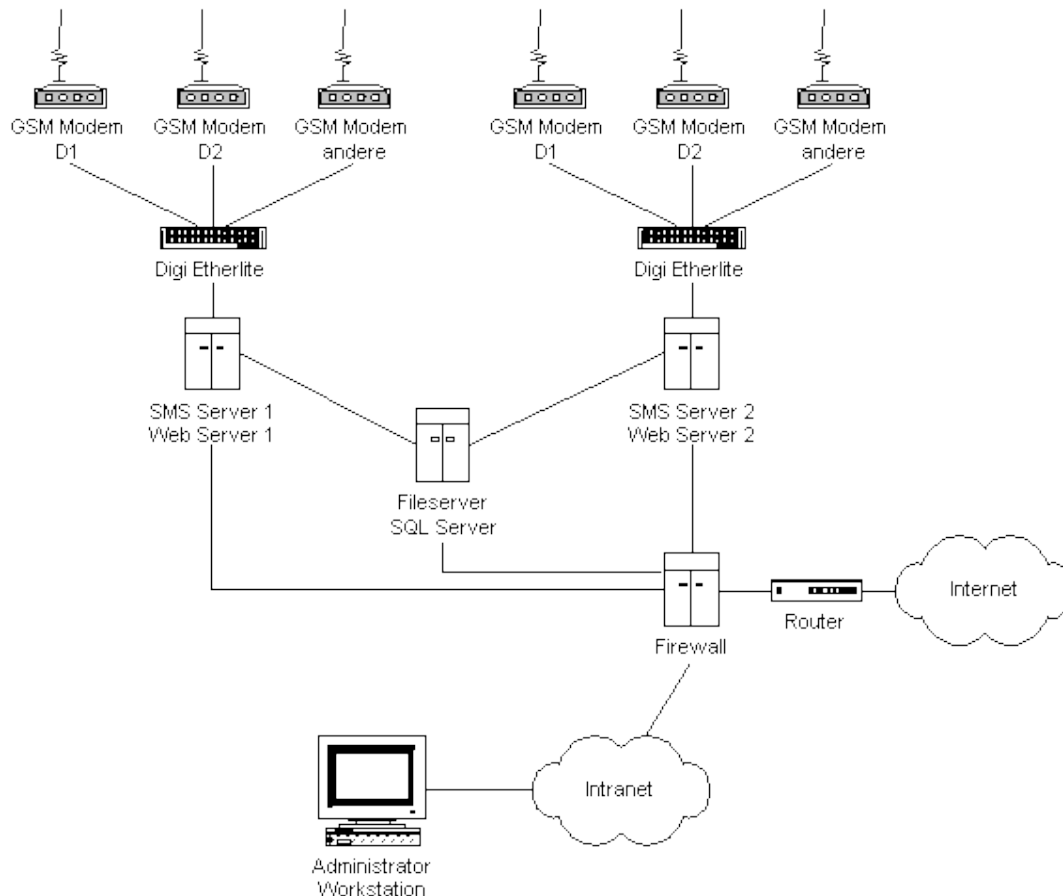
This system is partly redundant. I tried to find a good compromise between costs and stability.

Both SMS Servers are Dell Poweredge 350 with Linux and an additional ethernet card. The Firewall is also a Dell Poweredge 350 but with an additional 4x ethernet card. For data-security all three computers have mirrored harddisks and internal DAT tape drives.

The file server is a SUN V880 with Solaris. It has also an additional ethernet card. Data are stored onto two mirrored harddisks and backuped with a DAT tape drive.

The fileserver holds also the SQL database for dynamic websites and to store billing data.

The router goes to the internet. Your internet provider decides which router you should use.

The system administrator can log on to all Servers from his workstations. Therefore local monitors and keyboards are not necessary, after the operating systems were installed once.

The firewall protects the servers against unauthorized access from the internet and from the company internal intranet.

For redundancy I use two SMS Server. They store statistics and message queues on the file server. Normally they share the load but when one of them stops working, the remaining SMS Server will continue working and sending messages from the queues.

Both SMS servers have at least three modems, two for the major phone network provider (they have 80% of all mobile customers in Germany) and one for the others. Using two modems for the two major network providers helps saving money because sending messages within the own network is cheaper than sending into foreign networks (in germany, maybe also in your country).

The fileserver is a weak point in the system because it exists only once. Therefore I decided to use the very stable Sun V880. You should be careful when administering it and do not use it for testing. As less functions this server has as more stable it will run.

The whole system was set up without any switches or hubs because a defect hub would result in a total loss of service. Instead of them, I use more network interfaces and cables. In this construction every hardware is allowed to break except the fileserver. If the fileserver breaks then you will be out of service. The fileserver has internally redundant hardware, so this should never happen.

Both SMS servers are also used as webserver running Apache and an FTP service. You can give your customers two IP-Addresses. When one of them fails they can switch over to the other Webserver.

There are loadbalancer avaiable that can automatically switch over to a second webserver if the first one fails. When you use a loadbalancer, your customers do not need to know two IP Adresses. My example above does not use loadbalancer.

If you plan to modify your websites or reconfigure a SMS server, you can do this on one SMS Server while the other continues working for the "live" traffic. You could give the "test" server a temporary IP-address and a temporary queue directory to split it clearly from the live traffic.

4. Basic KnowHow

This chapter shows you, how sending and receiving SMS works with GSM modems. You will learn the modem commands for SMS and the data format of the messages. At the end you will learn how all the programs can be installed and how they can be configured.

4.1. SMS Commands For GSM Modems

This chapter shows you how to send and receive SM with GSM modems. It contains the modem commands and the data format of SMS.

If you use the SMS Server Tools to send and receive short messages you do not need to learn this chapter very deeply because the program does all this for you.

But it is useful to understand what is going on, specially when you have to troubleshoot a problem.

4.1.1. Start A Terminal Program

A terminal is a device with a text monitor and a keyboard. It sends all pressed keys out through a serial port and displays all received characters from the serial port on the display.

A terminal program emulates such a device on a computer.

To communicate with a modem you can start any terminal program by setting it to the following parameters: 9600bps, 8 bit, 1 stop bit, no parity, with hardware handshake (RTS/CTS). Some modems work only with 19200bps. In case of doubt you should try 9600 and 19200. At least one of both should work.

On Windows you can use Hyperterminal, on Linux you can use Minicom. Both programs have menus for configuration and are easy to use.

Solaris:

Solaris does not have a terminal program with menus. The Solaris program "tip" reads its configuration from the file /etc/remote. Add the following lines to this file:

```
porta:dv=/dev/cua/a:br#9600
portb:dv=/dev/cua/b:br#9600
```

If you have more than two serial ports in your SUN computer you can add all of them to this configuration file. The name in front of the first colon is free selectable. After the colon the device name is set, and the bitrate.

After modifying the configuration file you can start the program by entering:

tip porta

for the first serial port. You can quit the program by entering the two characters "~.".

4.1.2. GSM Modem Commands

For SMS applications you do not need to establish a connection to the SMSC. GSM modems can send and receive short messages through the signalling channel that is always available when the modem is logged into the network. You will not find any ATD command in this book.

General rules for modem commands:

All modem commands start with AT. After entering a command you have to press the enter (return) key. You can write many commands into one single line. In this case the AT appears only at the beginning of this line.

Multiple commands in a single line are separated by a space character or they are not separated (whatever you prefer).

If you use an used modem you can set it back to the manufacturer default settings by entering **AT &F &W**. This command activates the default settings and stores them into the configuration memory.

For SMS applications you need only some few modem commands. These are:

AT Z

performs a reset. The modem initializes as if it was switched off and on.

AT E0

Switches the echo off. You will not see the commands that you enter any more. This command may seem useless but it can make programming easier. To switch echo back on enter AT E1.

AT +CPIN?

the modem shows if you have to enter the PIN before you can use any other GSM function. Like on mobile phones you need the PIN to use the SIM card.

The modem answers are typically:

- READY means that the pin is already entered.

- SIM PIN means that you have to enter the PIN (see next command)

- SIM PUK means that the PIN is locked because it was entered three times wrong. Now it needs to be unlocked using the PUK (also called super-PIN).

- ERROR indicates a problem with the SIM card. The error message comes with an error code that should be explained in your modems manual.

- There are some other possible answers but I never saw them. If you see another answer to this command take a look into the modem manual.

AT +CPIN=xxxx

Use this command to enter the PIN (instead of xxxx). After entering the PIN the modem tries to log into the network. This command is necessary after every power-on.

AT +CPIN=yyyyyyyy,xxxx

This command changes the PIN. It works also when the PIN is locked. yyyyyyy is the PUK (also called super-PIN) and xxxx is the new PIN that you want to use. You get the PUK always together with your SIM card, typically printed on paper.

AT +CSQ

shows you information about the received signal quality. The answer are two numbers. The first number is the signal strength and should always be between 2 and 31. 0 means that the signal is too low and operation is not possible. 31 is the maximum allowed signal strength. The second number has something to do with the bit error rate. This number is meaningless for SMS applications because it is only measured during voice or data connections. For sending or receiving SM the modem does not establish such a connection.

AT +CREG?

show you if the modem is logged into the network. The answer 0,1 is positive. All other answer indicate a problem and the modem is not logged in. As long as the modem does not answer with 0,1 you cannot send SM.

The next chapter shows you commands to send and receive short messages.

4.1.3. SMS Modem Commands

After you learned some commands to control and check the modem and the connection state you are ready to learn the commands for sending and receiving.

AT +CMGF=0

tells the modem to transfer messages in PDU format from and to the computer. I will describe this format in the next chapter. Alternatively most modem support ASCII mode (AT+CMGF=1) which I do not recommend because this format does not support binary data and is limited to 140 characters. The PDU format supports up to 160 characters.

The next commands are necessary to receive messages:

AT +CPMS="SM"

tells the modem to store received messages into the SIM card. If your modem does not support this command it probably stores all messages into the SIM card by default. The parameter "ME" tells the modem to store all messages into its internal memory.

Some people reported that they cannot read stored messages from the ME memory. My own test were successful. If you have the same problem then simply use the SM memory and it will surely work.

AT +CPMS?

checks how far the memory is used. The answer looks like this:

```
+CPMS: "SM",3,20,"SM",0,20,"SM",0,20
```

The first block is for the received messages. The second block is for sent messages. The last block is for special use that is not part of this book.

The shortcuts "SM" indicate, where these messages are stored (in this example SM=SIM card memory). Then the size of this memory block and the used range follow. The maximum size depends on the modem and SIM card.

For you only the first two numbers "3,20" are interesting. This SIM card can store 20 messages and there are actually 3 messages.

AT +CMGL=x

Lists all stored messages. X can be one of the following values:

0	unread messages. Listing does not count as reading.
1	all received messages
2	unsent stored messages
3	sent and stored messages
4	all messages

AT +CMGR=x

lists one received message. Replace x with the number of memory location that you like to read (for example a value between 1 and 20). When you try to read an empty location some modems answer with ERROR and some answer with an empty text. You cannot read sent messages, there is no command for this.

A message is shown in this format:

```
+CMGR: status,name,length
data
OK
```

Status is a single digit value like in the AT+CMGL command. The name is the senders name if it is in the phone book. The length indicator is not important. The data format is described in the next chapter.

As long as I noticed only Siemens modems show names.,All other modems leave it out.

AT+CMGD=x

deletes a stored received message. Replace x by the memory location.

The next commands are used to send messages:

AT +CSCA="+xxxxxxxxxxx"

sets the number of the SMSC. Write it in international format starting with a plus, for example +491722270000. This setting is meaningless for receiving because every SMSC can always send you messages. If you do not use this command most modems use a default setting from the SIM card.

AT +CMGS=x

This command sends a message. x is the length of the PDU message calculated by this formula:

x=(number of characters/2)-1. If your message has 30 characters then x has to be 14.

After this command the modem shows an ">" prompt. Now enter the PDU message and terminate it with Ctrl-Z.

4.1.4. PDU Message Format (Receive)

This chapter describes modem commands to send and receive short messages. The messages itself have to be coded in PDU format that I will show you now.

PDU messages are strings of hexadecimal numbers. There are always pairs of two digits.

A received message looks like this:

0791947122720000040C919471123254F60000031092516195040AE8329BFD4697D9EC37

07	Length of SMSC number, in this case 7 bytes. The format indicator and number are counted.
91	Format of SMSC number, 91=International, A1=National.
94 71 22 72 00 00	SMSC Number (+)491722270000.
04	Bit 6: User data header indicator. Is 1, if the data contain a header. Only used with 8 bit data.
0C	Length of senders number, in this case 12 digits. The format indicator is not counted.
91	Format of senders number, 91=International, A1=National, D0=Alphanumeric.
94 71 12 32 54 F6	Sender (+)49172123456.
00	Protocol. 00=normal SMS.
00	Bit 3+2: Data coding: 00=7bit text, 01=8bit data, 10=unicode text
	Bit 1+0: 00=Flash-SMS, 01=normal SMS
03 10 92 51 61 95	Sent time in the order "Year Month Day Hour Minute Second" Every second digit is swapped. This date/time means January 29. of 2003 15:16:59 (= 3:16:59 pm)
04	Timezone in quarter hours. Negative values are calculated by adding 80hex. 04 means GMT+1 hour.
0A	Number of characters (7 bit) or number of bytes (8 bit).
E8329BFD4697D9EC37	Text or data. In this case "hellohello"

Phone numbers are filled with F if the number of digits is odd. Some SMSC count the F in the length indicator and some do not count it. The digits of phone numbers are swapped as you can easily see in the example above.

Alphanumeric senders are coded like 7bit text messages. The length indicator counts the number of hex-digits of the senders name.

Text messages consist of character of 7 bit. These bits are concatenated to a long row of bits and then splitted into bytes (each of 8 bit). See this drawing:

I wrote the binary numbers in the wrong direction because that makes it easier to understand how the concatenating and splitting works. So the least significant bit is always at the left side in my drawing. If there are not enough bits at the right end some 0 bits are appended.

If you take a closer look at the drawing you can easily see why 8 bit data messages are limited to 140 bytes while 7 bit text messages can be 160 characters long.

There are more number format, flags, protocol and data coding indicators as I wrote here. Others are not used in Germany and therefore I did never see them myself. For more detailed information you could take a look into the ETSI specification TS 100 900.

4.1.5. PDU Message Format (Status Report)

When somebody sends a short message he can request a status report (also called delivery report). Then the SMSC informs the sender about the status, if the message was delivered or not.

PDU messages are string of hexadecimal numbers. There are always pairs of two digits.

A status report looks like this one:

    07919471227200000201 0C919471123254F60310925161950403109251710 50400...

07	Length of SMSC number, in this case 7 bytes. The format indicator and number are counted.
91	Format of SMSC number, 91=International, A1=National.
94	SMSC Number (+)491722270000
02	Bit 2: Is 0, if there are more messages that belong together
01	ID number of the sent message
0C	Length of receiver number, in this case 12 digits. The format indicator is nor counted.
91	Format of receivers number, 91=International, A1=National, D0=Alphanumeric.
94 71 12 32 54 F6	Receiver of the SMS that belongs to this status report (+)49172123456.
03 10 92 51 61 95	Time when the message arrived the SMSC in the order "Year Month Day Hour Minute Second". Every Second digit is swapped. This date/time value means January 29. of 2003 16:16:59 (=3:16:59 pm)
04	Timezone in quarter hours. Negative values are calculated by adding 80hex. 04 means GMT+1 hour.
03 10 92 51 71 05	Time when the message was delivered in the order "Year Month Day Hour Minute Second". Every second digit is swapped. This date/time value means January 29. of 2003 15:17:50 (=3:17:50 pm). If the message is not delivered this value is filled with zeroes.
04	Timezone in quarter hours. Negative values are calculated by adding 80hex. 04 means GMT+1 hour.
00	Status
...	There may be more data at this point. The meaning depends on the SMSC and is not standarised.

Phone numbers are filled with F if the number of digits is odd. Some SMSC count the F in the length indicator and some do not count it. The digits of phone numbers are swapped as you can easily see in the example above.

Alphanumeric receivers are coded like 7bit text messages. The length indicator counts the number of hex-digits of the receivers name.

The status field can have one of the following values:

0	Ok,short message received by the SME
1	Ok,short message forwarded by the SC to the SME but the SC is unable to confirm delivery
2	Ok,short message replaced by the SC
32	Still trying,congestion
33	Still trying,SME busy
34	Still trying,no response from SME
35	Still trying,service rejected
36	Still trying,quality of service not available
37	Still trying,error in SME
64	Error,remote procedure error
65	Error,incompatible destination
66	Error,connection rejected by SME
67	Error,not obtainable
68	Error,quality of service not available
69	Error,no interworking available
70	Error,SM validity period expired
71	Error,SM deleted by originating SME
72	Error,SM deleted by SC administration

73	Error,SM does not exist
96	Error,congestion
97	Error,SME busy
98	Error,no response from SME
99	Error,service rejected
100	Error,quality of service not available
101	Error,error in SME

SC = Short Message Service Centre (SMSC)
SME = Short Message Equipment (modem or phone)

4.1.6. PDU Message Format (Send)

The message string is composed of hexadecimal numbers. There are always pairs of two digits. A message that will be sent looks like this:

```
0001000C919471123254F600000AE8329BFD4697D9EC37
```

00	Length of SMS number of 00 if the modem should use the already known number.
01	Bit 6: User data header indicator. 1 indicated that the user data contain a header. Only used for 8 bit messages.
00	ID number of the message. In case of 00 the modem insert an automatic incremented number.
0C	Length of receivers number, in this case 12 digits. The format indicator is not counted.
91	Format of the receiver number. 91=International, A1=National,D1=Alphanumeric
94 71 12 32 54 F6	Receiver number or name (+)49172123456.
00	Protocol. 00=regular SMS.
00	Bit 7-2: Data coding: 111100=7bit text, 111101=8bit data, 000110=unicode text
	Bit 1+0: 00=Flash-SMS, 01=normal SMS
(...)	Validity period. Is skipped in this case because of the 00 bin bit 4+3 above.
0A	Number of character (7bit) or number of data bytes (8bit)
E8329BFD4697D9EC37	Text or data. In this case "hellohello".

Phone number are filled with F if the number of digits is odd. The digits of phone numbers are swapped as you can see in the example above.

Text messages consist of character of 7 bit. These bits are concatenated to a long row of bits and then splitted into bytes (each of 8 bit). See this drawing:

I wrote the binary numbers in the wrong direction because that makes it easier to understand how the concatenating and splitting works. So the least significant bit is always at the left side in my drawing. If there are not enough bits at the right end some 0 bits are appended.

If you take a closer look at the drawing you can easily see why 8 bit data messages are limited to 140 bytes while 7 bit text messages can be 160 characters long.

The validity period can be in one of four different formats:

0	use default of the SMSC (normally maximum)
1	extended format, see ETSI TS 100 900 specification
10	absolute format like date/time fields in status report. 7 bytes long.
11	relative format. 1 byte

The relative format is normally used and it is only 1 byte long. Possible values are grouped:

0-143	(value+1)*5 minutes, for 5 minutes to 12 hours.
144-167	12 hours+(value-143)*30 minutes, for 12.5 to 24 hours.
168-196	value-166 days, for 2 to 30 days.
197-255	value-192 weeks, for 5 to 63 weeks.

In Germany all messages are limited to 2 days. Larger validity periods are ignored.

For the number format, flags, protocol and data coding there are more values than shown here. But these are not supported in Germany. For more details see ETSI TS 100 900 specification.

4.1.7. Character Set

Mobile phones use a different character set than computers. The following table lists all characters that mobile phones can display and the corresponding character codes in ISO-8859-1 character set used by computers.

The Euro character and some other consist of two 7bit values! Some countries know additional characters that are not in this table.

Hex	Dez	Name	Zeichen	ISO-8859-1
0x00	0	COMMERCIAL AT	@	64
0x01	1	POUND SIGN	£	163
0x02	2	DOLLAR SIGN	$	36
0x03	3	YEN SIGN	¥	165
0x04	4	LATIN SMALL LETTER E WITH GRAVE	è	232
0x05	5	LATIN SMALL LETTER E WITH ACUTE	é	233
0x06	6	LATIN SMALL LETTER U WITH GRAVE	ù	249
0x07	7	LATIN SMALL LETTER I WITH GRAVE	ì	236
0x08	8	LATIN SMALL LETTER O WITH GRAVE	ò	242
0x09	9	LATIN CAPITAL LETTER C WITH CEDILLA	Ç	199
0x0A	10	LINE FEED		10
0x0B	11	LATIN CAPITAL LETTER O WITH STROKE	Ø	216
0x0C	12	LATIN SMALL LETTER O WITH STROKE	ø	248
0x0D	13	CARRIAGE RETURN		13
0x0E	14	LATIN CAPITAL LETTER A WITH RING ABOVE	Å	197
0x0F	15	LATIN SMALL LETTER A WITH RING ABOVE	å	229
0x10	16	GREEK CAPITAL LETTER DELTA	Δ	
0x11	17	LOW LINE	_	95
0x12	18	GREEK CAPITAL LETTER PHI	Φ	
0x13	19	GREEK CAPITAL LETTER GAMMA	Γ	
0x14	20	GREEK CAPITAL LETTER LAMBDA	Λ	
0x15	21	GREEK CAPITAL LETTER OMEGA	Ω	
0x16	22	GREEK CAPITAL LETTER PI	Π	
0x17	23	GREEK CAPITAL LETTER PSI	Ψ	
0x18	24	GREEK CAPITAL LETTER SIGMA	Σ	
0x19	25	GREEK CAPITAL LETTER THETA	Θ	
0x1A	26	GREEK CAPITAL LETTER XI	Ξ	
0x1B	27	ESCAPE TO EXTENSION TABLE		
0x1B0A	27 10	FORM FEED		12
0x1B14	27 20	CIRCUMFLEX ACCENT	^	94
0x1B28	27 40	LEFT CURLY BRACKET	{	123
0x1B29	27 41	RIGHT CURLY BRACKET	}	125
0x1B2F	27 47	REVERSE SOLIDUS (BACKSLASH)	\	92
0x1B3C	27 60	LEFT SQUARE BRACKET	[	91
0x1B3D	27 61	TILDE	~	126
0x1B3E	27 62	RIGHT SQUARE BRACKET	]	93
0x1B40	27 64	VERTICAL BAR	|	124
0x1B65	27 101	EURO SIGN	€	164
0x1C	28	LATIN CAPITAL LETTER AE	Æ	198
0x1D	29	LATIN SMALL LETTER AE	æ	230
0x1E	30	LATIN SMALL LETTER SHARP S (German)	ss	223
0x1F	31	LATIN CAPITAL LETTER E WITH ACUTE	É	201

Hex	Dez	Name	Zeichen	ISO-8859-1
0x20	32	SPACE		32
0x21	33	EXCLAMATION MARK	!	33
0x22	34	QUOTATION MARK	"	34
0x23	35	NUMBER SIGN	#	35
0x24	36	CURRENCY SIGN	¤	164
0x25	37	PERCENT SIGN	%	37
0x26	38	AMPERSAND	&	38
0x27	39	APOSTROPHE	'	39
0x28	40	LEFT PARENTHESIS	(	40
0x29	41	RIGHT PARENTHESIS	)	41
0x2A	42	ASTERISK	*	42
0x2B	43	PLUS SIGN	+	43
0x2C	44	COMMA	,	44
0x2D	45	HYPHEN-MINUS	-	45
0x2E	46	FULL STOP	.	46
0x2F	47	SOLIDUS (SLASH)	/	47
0x30	48	DIGIT ZERO	0	48
0x31	49	DIGIT ONE	1	49
0x32	50	DIGIT TWO	2	50
0x33	51	DIGIT THREE	3	51
0x34	52	DIGIT FOUR	4	52
0x35	53	DIGIT FIVE	5	53
0x36	54	DIGIT SIX	6	54
0x37	55	DIGIT SEVEN	7	55
0x38	56	DIGIT EIGHT	8	56
0x39	57	DIGIT NINE	9	57
0x3A	58	COLON	:	58
0x3B	59	SEMICOLON	;	59
0x3C	60	LEss-THAN SIGN	<	60
0x3D	61	EQUALS SIGN	=	61
0x3E	62	GREATER-THAN SIGN	>	62
0x3F	63	QUESTION MARK	?	63
0x40	64	INVERTED EXCLAMATION MARK	¡	161
0x41	65	LATIN CAPITAL LETTER A	A	65
0x42	66	LATIN CAPITAL LETTER B	B	66
0x43	67	LATIN CAPITAL LETTER C	C	67
0x44	68	LATIN CAPITAL LETTER D	D	68
0x45	69	LATIN CAPITAL LETTER E	E	69
0x46	70	LATIN CAPITAL LETTER F	F	70
0x47	71	LATIN CAPITAL LETTER G	G	71
0x48	72	LATIN CAPITAL LETTER H	H	72
0x49	73	LATIN CAPITAL LETTER I	I	73
0x4A	74	LATIN CAPITAL LETTER J	J	74
0x4B	75	LATIN CAPITAL LETTER K	K	75
0x4C	76	LATIN CAPITAL LETTER L	L	76
0x4D	77	LATIN CAPITAL LETTER M	M	77
0x4E	78	LATIN CAPITAL LETTER N	N	78
0x4F	79	LATIN CAPITAL LETTER O	O	79
0x50	80	LATIN CAPITAL LETTER P	P	80

Hex	Dez	Name	Zeichen	ISO-8859-1
0x51	81	LATIN CAPITAL LETTER Q	Q	81
0x52	82	LATIN CAPITAL LETTER R	R	82
0x53	83	LATIN CAPITAL LETTER S	S	83
0x54	84	LATIN CAPITAL LETTER T	T	84
0x55	85	LATIN CAPITAL LETTER U	U	85
0x56	86	LATIN CAPITAL LETTER V	V	86
0x57	87	LATIN CAPITAL LETTER W	W	87
0x58	88	LATIN CAPITAL LETTER X	X	88
0x59	89	LATIN CAPITAL LETTER Y	Y	89
0x5A	90	LATIN CAPITAL LETTER Z	Z	90
0x5B	91	LATIN CAPITAL LETTER A WITH DIAERESIS	Ä	196
0x5C	92	LATIN CAPITAL LETTER O WITH DIAERESIS	Ö	214
0x5D	93	LATIN CAPITAL LETTER N WITH TILDE	Ñ	209
0x5E	94	LATIN CAPITAL LETTER U WITH DIAERESIS	Ü	220
0x5F	95	SECTION SIGN	§	167
0x60	96	INVERTED QUESTION MARK	¿	191
0x61	97	LATIN SMALL LETTER A	a	97
0x62	98	LATIN SMALL LETTER B	b	98
0x63	99	LATIN SMALL LETTER C	c	99
0x64	100	LATIN SMALL LETTER D	d	100
0x65	101	LATIN SMALL LETTER E	e	101
0x66	102	LATIN SMALL LETTER F	f	102
0x67	103	LATIN SMALL LETTER G	g	103
0x68	104	LATIN SMALL LETTER H	h	104
0x69	105	LATIN SMALL LETTER I	i	105
0x6A	106	LATIN SMALL LETTER J	j	106
0x6B	107	LATIN SMALL LETTER K	k	107
0x6C	108	LATIN SMALL LETTER L	l	108
0x6D	109	LATIN SMALL LETTER M	m	109
0x6E	110	LATIN SMALL LETTER N	n	110
0x6F	111	LATIN SMALL LETTER O	o	111
0x70	112	LATIN SMALL LETTER P	p	112
0x71	113	LATIN SMALL LETTER Q	q	113
0x72	114	LATIN SMALL LETTER R	r	114
0x73	115	LATIN SMALL LETTER S	s	115
0x74	116	LATIN SMALL LETTER T	t	116
0x75	117	LATIN SMALL LETTER U	u	117
0x76	118	LATIN SMALL LETTER V	v	118
0x77	119	LATIN SMALL LETTER W	w	119
0x78	120	LATIN SMALL LETTER X	x	120
0x79	121	LATIN SMALL LETTER Y	y	121
0x7A	122	LATIN SMALL LETTER Z	z	122
0x7B	123	LATIN SMALL LETTER A WITH DIAERESIS	ä	228
0x7C	124	LATIN SMALL LETTER O WITH DIAERESIS	ö	246
0x7D	125	LATIN SMALL LETTER N WITH TILDE	ñ	241
0x7E	126	LATIN SMALL LETTER U WITH DIAERESIS	ü	252
0x7F	127	LATIN SMALL LETTER A WITH GRAVE	à	224

4.1.8. Practical Examples

At the end of this sub-chapter I like to show you an example how to send and receive one short message. You can try this example with nearly any GSM modem or mobile phone.

I used a Sharp GX-10 mobile phone with infrared interface on Linux. I loaded the infrared driver and checked if the computer detected the phone correctly (with the program irdadump).

Now start any terminal program and enter the commands that you learned from the previous chapter.

atz
OK

at+cpin?
+CPIN: READY
OK

at+csq
+CSQ: 25,0
OK

at+creg?
+CREG: 0,1
OK

The commands above were only playing. I resetted the phone, checked the PIN and the RF signal quality. Then I checked if the phone was logged into the network. All these commands are not necessary on mobile phones because the phone shows all these information permanently on its display. GSM modems have no display so you need these command if you use a GSM modem.

Now I send a short message to myself.

at+cmgf=0
OK

at+csca="+491722270000"
OK

at+cmgs=22
> **0001000C919471123254F600000AE8329BFD4697D9EC37**<Ctrl-Z>
+CMGS: 106
OK

+CMTI: "SM", 1

First I tell the mobile phone that I want to send messages in PDU format. The next command sets the SMSC to be used for sending. If I would leave out this command the phone would decide which SMSC is used.

The third command send the message. I calculated the length of the message with (number of hex digits /2)-1.

The I entered the message data in PDU format and pressed Ctrl-Z. A few seconds later the phone acknowledged the message and showed that this message got the ID number 106.

Some more seconds later the modem informed me that it received a message. +CTMI: "SM",1 means that this message is now stored on the SIM card on memory location 1.

I like to read this received message now:

at+cpms?
+CPMS: "SM",1,10,"SM",1,10,"SM",1,10
OK

at+cmgr=1
+CMGR: 0,,28
0791947122720000040C919471123254F600003010610240510800AE8329BFD4697D9EC37
OK

at+cmgd=1
OK

The first command was not really necessary. I used it to check how many memory location are used. The first part of the result was "SM",1,10 which means that there is space for 10 messages but only one memory location is used at the moment.

The second command reads the message from location 1. The message could be in another location but I knew that this message that I just received is in location one because I saw the notification. If I would not know the memory location I had to try every 10 or I could use the command AT+CMGL command.

After I read the message I deleted it using the last command.

I suppose that you mentioned that the message text is again hellohello as on the previous pages.

Tip: Enter the PDU string in the program PDU-Spy and see how the string is decoded. Compare the output of PDU-Spy with the previous pages.

4.2. Software Installation

Now you know what hardware and software you need for your SMS application. You know how the modem works and sent a short message - manually. This knowledge will help you when a modem does not work as expected.

Now it is time to automatize the short message transfer.

You need to install the software, configure it and use it. This chapter shows you how to install the software that I presented you in beginning of this book.

There is one sub-chapter for each operating system.

After you read this chapter you will be able to install a webserver with dynamic pages (PHP) and database connection (MySQL). You will also be able to install the SMS Server Tools.

The installation instructions are written step for step so you can enter all the commands exactly as written here.

At the end you will test the webserver and database using a small self-written dynamic website.

4.2.1. Software Installation On Solaris

This step for step instruction assumes that you install this software on an empty SUN computer or a new SUN computer with pre-installed Solaris 6, 7, 8,9 or 10.

Read this chapter from the beginning to the end and do not skip paragraphs.

4.2.1.1. Configure Solaris

First install Solaris from the Solaris installation CD-Rom. When it comes to package selection install everything or at least all the developer tools plus the webbrowser Netscape. The installation program has a special select-button for developers.

Log in as root and start a terminal window. Now configure some things:

If your installation includes MySQL or Apache, remove these packages to avoid version conflicts. You will install another version later.

To check if the program are already installed, use the commands **pkginfo | grep apache** and **pkginfo | grep mysql**.

To remove packages, use the command **pkgrm packagename**.

Now please open the file /etc/profile in your preferred editor (maybe xedit or vi) and append the following lines to the end:

```
CC=gcc
PATH=/usr/local/bin:/usr/local/sbin:/usr/ccs/bin:/usr/local/mysql/bin:$PATH
MANPATH=/usr/local/man:$MANPATH
LD_LIBRARY_PATH=/usr/local/lib:$LD_LIBRARY_PATH
export CC PATH MANPATH LD_LIBRARY_PATH
```

These entries add some search paths for programs, manual pages and libraries.

Log off and log in again to activate the changes.

4.2.1.2. Install Packages

Download the following packages from http://www.sunfreeware.com:

- gcc   , the GNU C and C++ compiler
- make   , accessoires for gcc
- autoconf   , accessoires for gcc
- automake   , accessoires for gcc
- tar   , packer/unpacker
- gzip   , packer/unpacker
- sed   , script intepreter for text files
- fetchmail   , download eMails
- procmail   , process eMails
- mysql   , database server
- Libraries for these programs:
  - db
  - expat
  - gdbm
  - libxml2
  - ssl

The website includes installation instructions on the link Downloading/Installing. In short terms:

Download the packages to an empty directory. Gzip is the only uncompressed package, all other need to become uncompressed before installation:

pkgadd -d gzip-1.3.5.10
gunzip *.gz
pkgadd -d bzip-2.1.0.4
pkgadd -d sed-4.1.5

...

4.2.1.3. Configure MySQL

Create a start/stop script named /etc/init.d/mysqld with this content:

```
#! /bin/sh
case "$1" in
    start)
        /usr/local/mysql/bin/mysqld_save &
        ;;
    stop)
        pkill libexec/mysqld
        ;;
    restart|reload)
        $0 stop && $0 start
        ;;
    *)
        echo "Usage: $0 {start|stop|restart}"
        exit 1
esac
```

Enter the next commands:

cd /etc/init.d
chmod a+rx mysqld
cd /etc/rc3.d
ln -s /etc/init.d/mysqld S80mysqld
groupadd mysql
useradd -d /usr/local/mysql -g mysql mysql

Install an empty database. Without this database you will not be able to start the MySQL server:

/usr/local/mysql/bin/mysql_install_db
chown -R mysql /usr/ocal/mysql/var
chgrp -R mysql /usr/ocal/mysql/var

Start the MySQL Server. When you next boot the computer it will start automatically:

/etc/init.d/mysqld start

By default, MySQL allows everybody to use all database without restrictions. You should change this immediately. I recommend to give a password for root@localhost and delete all anonymous accounts. You can add more users later when you need them.

```
mysqladmin -u root password 'My_Password'
mysql -u root -p
mysql> use mysql;
mysql> delete from user where user="" or password="";
mysql> quit
mysqladmin -u root -p reload
```

The username root is not the same as the Solaris user root. MySQL has its own user and password management. In mysql the name root is used for the database administrator by default.

Now you can use this database only from the local computer with a root password. A later chapter shows you how to add more users.

4.2.1.4. Compile Apache and PHP

Do not install Packages of Apache and PHP from any website! They do not work properly together, even if you use packages from SunFreeware. You need to download the original source codes and compile them to get both working together.

Unpack the source code of Apache and compile it:

/usr/local/bin/tar -xvzf httpd-2.2.2.tar.gz

cd httpd-2.2.2

**./configure --prefix=/usr/local/apache2 --enable-mods-shared=all --enable-ssl=shared --enabe-ssl \
--with-ssl=/usr/local/ssl --enable-proxy --enable-proxy-connect --enable-proxy-ftp --enable-proxy-http**

make clean

make

make install

Unpack the source code of PHP and compile it:

/usr/local/bin/tar -xvzf php-5.1.4.tar.gz

cd php-5.1.4

**./configure --prefix=/usr/local/php --with-apxs2=/usr/local/apache2/bin/apxs --with-mysql=/usr/local/mysql \
--with-libxml-dir=/usr/local --with-openssl=/usr/local/ssl**

make clean

make

make install

4.2.1.5. Configure Apache and PHP

Copy the example config file of php:

cp /usr/local/src/php-5.1.4/php.ini-recommended /usr/local/php/lib

Open php.ini with a text editor. Read the comments because some of the possible settings might be interesting for you. I recommend the following changes:

```
display_errors=on
doc_root=/usr/local/apache/htdocs
```

You should disable error messages later when you are sure that your PHP scripts work fine.

Open /usr/local/apache2/conf/httpd.conf and insert two lines where the other LoadModule commands can be found:

```
LoadModule php5_module modules/libphp5.so
AddType application/x-httpd-php .php
```

Ensure that these lines do not appear duplicated in this file.

Enter the following commands:

cd /etc/init.d
ln -s /usr/local/apache/bin/apachectl apache
cd /etc/rc3.d
ln -s /etc/init.d/apache S81apache

Start the Apache webserver manually. It will automatically start when you boot next time.

/etc/init.d/apache start

4.2.1.6. Test Apache, PHP And MySQL

Now please test your webserver.

Start a webbrowser on your webserver and enter the URL **http://localhost**. You should see an example page.

Now try the same on another computer and enter **http://your_webserver_name**. Again you should see the example page from Apache. Now you should test the PHP function and the connection to the MySQL database:

Create the file /usr/local/apache2/htdocs/test.php on your webserver with this content:

```
<html><body>
  This is HTML text.<br>

  <?php
    print("This is PHP text.<br>");
    print("Now a MySQL query follows:<br>");
    mysql_connect("localhost","root","my_password");
    mysql_select_db("mysql");
    $result=mysql_query("select user,host,password from user");
    for ($nr = 1; $row = mysql_fetch_row($result); $nr++)
    {
      print("$row[0] $row[1] $row[2]<br>");
    }
  ?>

</body></html>
```

Use the name of your MySQL server for the mysql_connect command and insert the correct password. If your Apache webserver and MySQL database run on the same computer use "localhost" as the name.

Ensure that the file has enough read permissions:

chmod a+r /usr/local/apache2/htdocs/test.php

Now start your webbrowser and enter **http://localhost/test.php** or **http://your_webserver_name/test.php**. You should see this:

```
This is HTML text.
This is PHP text.
Now a MySQL query follows:
root localhost x4gdsag2354124
```

The first line was created by regular HTML code, the second line tests the PHP integration into the webserver. The third and fourth line test the connection to your MySQL database.

Please review the example program above. You can see how a PHP code was placed into a HTML file. The begin and end are marked with <?php and ?>.

The PHP program connects to the SQL database and queries the list of SQL-users and their passwords. The passwords are encrypted for security therefore you do not see them in clear text.

4.2.1.7. Install Sources

Compile the source codes to create executable programs. The mm library is used by the SMS Server Tools to access shared memory. You can get it on http://www.ossp.org/pkg/lib/mm/ :

/usr/local/bin/tar -xvzf libmm-1.4.0.tar.gz
cd mm-1.4.0
./configure
make
make install

Ssmtp is a small program to send mail through an external mail server, downloadable on http://www.ibiblio.org/pub/Linux/system/mail/mta.

/usr/local/bin/tar -xvzf ssmtp-2.4.8.tar.gz
cd ssmtp-2.4.8
make
make install

The SMS Server Tools send and receive SM through GSM modems, available on

Version 2.x: http://www.meinemullemaus.de/smstools/
Version 3.x: http://smstools3.kekekasvi.com/

/usr/local/bin/tar -xvzf smstools-2.2.4.tar.gz
cd smstools

Open the file /usr/src/smstools/src/Makefile and remove the hash character in front of OPTIONS = -D Solaris. Enable statistics by adding a hash to the line with the NOSTATS option.

make
make install
cd /etc/rc3.d
ln -s /etc/init.d/sms s82sms

If you want to start the SMS Server Tools automatically a boot time you simply need to rename the file in S82sms (capital S). But please wait until you configured and tested the program.

4.2.1.8. Reminder For Solaris

Please remind the following files because they are not identical in every operating system. In the following chapters I will not write about operating system specific things.

SMS Server Tools

start	/etc/init.d/sms start
stop	/etc/init.d/sms stop
config file	/etc/smsd.conf
error logfile	/var/log/smsd.log
sms queues	/var/spool/sms

Apache webserver

start	/etc/init.d/apache start
stop	/etc/init.d/apache stop
config file	/usr/local/apache2/conf/httpd.conf
error logfile	/usr/local/apache2/logs/error.log
web pages	/usr/local/apache2/htdocs

MySQL database

start	/etc/init.d/mysqld start
stop	/etc/init.d/mysqld stop
client	mysql -u root -p

Crontab

modify	EDITOR=dtpad; export EDITOR; crontab -e

4.2.2. Software Installation On Windows

This step for step instruction assumes that you install the software on an empty PC or a new PC with pre-installed Windows of any version.

Read this chapter from the beginning to the end and do not skip paragraphs.

4.2.2.1. Install Windows Programs

Install Windows from the installation CD-Rom. Then install the following programs:

Editpad Lite

http://www.editpadpro.com/editpadlite.html

Copy the executable file editpad.exe into your windows directory (e.g. c:\windows). Start the program. In the Menu options/settings you can assign this program as the default editor for text files.

nnCron Lite

http://www.nncron.ru/download.shtml

Install this program if you want to start other programs by timer. In Windows XP you can also use the timer in the control panel.

The configuration file is in c:\program files\cron\cron.tab. In the subdirectory log you can find protocol files and error messages.

War FTP Daemon

http://www.warftp.org/

Install the War FTP Daemon if you like to share local directories to the internet via FTP. The program is very flexible and easy to use, therefore I will not explain it in this book.

PDU Spy

http://www.nobbi.com/download.htm#pduspy

The program is useful when you have problems with your modem or with the SMS Server Tools. It can decode and encode short messages and is primarily a debug utility for programmers.

MySQL

http://dev.mysql.com/downloads/

MySQL is a database server. Install MySQL if you plan to offer dynamic websites or if you want to store data. The setup program creates a root account for the first steps.

PHP and Apache

http://www.php.net/downloads.php

http://httpd.apache.org/

PHP is a programming language for dynamic websites. Apache is a webserver. Install both together if you like to offer websites.

4.2.2.2. Install CygWin

You will need CygWin on all servers that should run the example programs of this book. So basically install CygWin on all computers that are part of your SMS application.

Download and start the setup program from http://www.cagwin.com and select "Install from Internet". Select c:\cygwin as the destination directory.

To not change other options until this window appears:
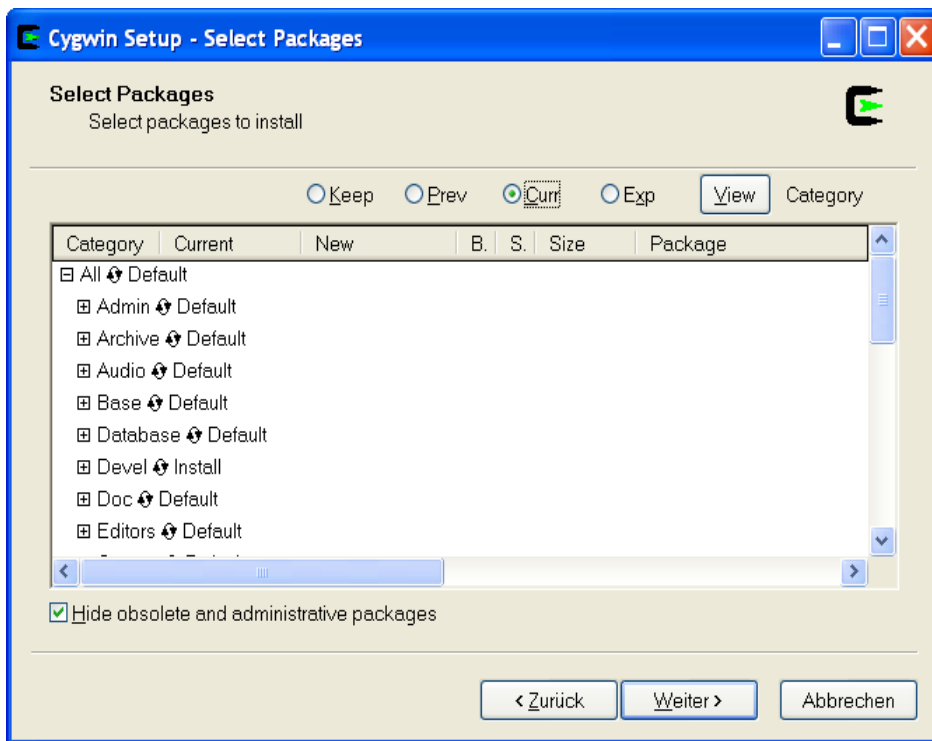
Here you see a lot of grouped programs. By clicking on them you toggle between these options:

- Skip (do not install)

- Install (Versions-Nr. appears)

- Remove

- Re-Install

Do not de-select any pre-selected programs because you need them all. Select the following programs to install:

- ● Devel

    - gcc

    - make

- ● Editors

    - vim

- ● Mail

    - fetchmail

    - procmail

    - ssmtp

- ● Net

    - inetutils

    - ncftp

- ● System

    - procps

4.2.2.3. Install Apache Webserver

The webserver is only needed on the server that should act as your webserver - if you want to have one. Maybe you decided not to use a webserver.

Do not install apache or apache2 package from CygWin because they do not support php!

Install Apache with its setup program. The default settings are ok. Then unpack PHP to the destination directory c:\php. PHP does not have a setup program.

Then update the PATH variable: In case of Windows 95 and 98, you can append this line to c:\autoexec.bat:

PATH=%PATH%;c:\php

In case of Windows XP and 2000 start the system control panel and klick on "System". Then klick on "Enhanced" (or similar, I do not know how this button is called in the english version). At the bottom there is a btton named "Envirroment variables" (or similar) that you please klick on. Search for the variable PATH and append **;c:\php** to it. Unfortunately I do not know how to modify PATH variables on Windows NT.

Restart Windows to activate the changes.

Copy the file c**:\php\php.ini-recommended** to **c:\windows\php.ini**. Open it in a text editor and change four lines so they look like:

```
display_errors=on
doc_root=C:\Program Files\Apache Software Foundation\Apache2.2\htdocs
extension_dir = "c:/php/ext"
extension=php_mysql.dll
```

Save the changes and then open the file C:\Program Files\Apache Software Foundation\Apache2.2\conf\httpd.conf using the submenu of Apache in Windows start-menu. Append one line to the LoadModules section:

```
LoadFile "C:/php/php5ts.dll"
LoadModule php5_module "C:/php/php5apache2_2.dll"
AddType application/x-httpd-php .php
```

Now exit the Apache Monitor (in the taskbar) and restart it. Restart also the Apache server using the Apache Monitor.

4.2.2.4. Test Apache, PHP And MySQL

Now please test your webserver.

Start a webbrowser on your webserver and enter the URL **http://localhost**. You should see an example page.

Now try the same on another computer and enter **http://your_webserver_name**. Again you should see the example page from Apache. Now you should test the PHP function and the connection to the MySQL database:

Create the file c:\Program Files\Apache Software Foundation\Apache2.2\htdocs\test.php on your webserver with this content:

```
<html><body>
  This is HTML text.<br>

  <?php
    print("This is PHP text.<br>");
    print("Now a MySQL query follows:<br>");
    mysql_connect("localhost","root","my_password");
    mysql_select_db("mysql");
    $ergebnis=mysql_query("select user,host,password from user");
    for ($nr = 1; $reihe = mysql_fetch_row($ergebnis); $nr++)
    {
      print("$reihe[0] $reihe[1] $reihe[2]<br>");
    }
  ?>

</body></html>
```

Use the name of your MySQL server for the mysql_connect command and insert the correct password. If your Apache webserver and MySQL database run on the same computer use "localhost" as the name.

Now start your webbrowser and enter **http://localhost/test.php** or **http://your_webserver_name/test.php**. You should see this:

```
This is HTML text.
This is PHP text.
Now a MySQL query follows:
root localhost x4gdsag2354124
```

The first line was created by regular HTML code, the second line tests the PHP integration into the webserver. The third and fourth line test the connection to your MySQL database.

Please review the example program above. You can see how a PHP code was placed into a HTML file. The beginning and end are marked with <?php and ?>.

The PHP program connects to the SQL database and queries the list of SQL-users and their passwords. The passwords are encrypted for security therefore you do not see them in clear text.

4.2.2.5. Install SMS Server Tools

The SMS Server Tools are only necessary on computers that are connected with GSM modems to send and receives short messages.

Download libmm from http://www.ossp.org/pkg/lib/mm/ and the SMS Server Tools from

Version 2.x: http://www.meinemullemaus.de/smstools/
Version 3.x: http://smstools3.kekekasvi.com/

Copy both files into the directory c:\cygwin\usr\src\. Then extract and compile the files in a CygWin Bash Shell using theses commands:

mkdir /etc/init.d

mkdir /usr/local/src
cd /usr/local/src
tar -xvzf libmm-1.4.0.tar.gz
cd /usr//localsrc/mm-1.4.0
./configure
make
make install

cd /usr/local/src
tar -xvzf smstools*.tar.gzcd
cd /usr/local/src/smstools

Open the file c:\cygwin\usr\local\src\smstools\src\Makefile with your text editor and enable statistics by inserting a hash character in front f the NOSTATS option.

mkdir /var/spool
make
make install

4.2.2.6. Reminder For Windows

Please remind the following files because they are not identical in every operating system. In the following chapters I will not write about operating system specific things.

SMS Server Tools

start	smsd &
stop	pkill smsd
config file	/etc/smsd.conf
error logfile	/var/log/smsd.log
sms queues	/var/spool/sms

Apache webserver

start	net start apache (in DOS window, or use the monitor program)
stop	net stop apache (in DOS window, or use the monitor program)
config file	c:\Program Files\Apache Software Foundation\Apache2.2\conf\httpd.conf
web pages	c:\Program Files\Apache Software Foundation\Apache2.2\htdocs

MySQL database

start	net start mysql (in Dos Window)
stop	net stop mysql (in Dos Window)
client	C:\Programme\MySQL\MySQL Server 4.1\bin\mysqlc -u root -p (in Dos Window)

Crontab

change	c:\program files\cron\cron.tab
logfiles	c:\program files\cron\log

4.2.3. Software Installation on Linux

All Linux Distributions contain already all software, that this book describes. Some include also the SMS Server Tools, for example Debian.

But if you prefer the latest version of SMS Server Tools, then you need to install the following Packages:

- gcc
- make
- libmm

Depending on your application, you might also need the following packages:

- procmail
- sendmail (or another mail server)
- fetchmail
- apache
- php
- mysql
- ncftp

Normally, the setup program of your Linux should configure all these programs properly. If this does not work as expected, then you might find help in the following chapters.

4.2.3.1. Preparation

Log in as root to install software. Enter the command

grep :initdefault /ect/inittab

and write down the number between two semicolons (typically a 3).

This is the runlevel that your system boots into. You will create some links later that have to be placed into the correct runlevel directory so they start automatically at boo time (mysql, apache, smstools).

In the next chapters I assume that the runlevel is 3. If your runlevel is a different one you have to change the command to your runlevel.

4.2.3.2. Install SMS Server Tools

The SMS Server Tools (and OSSP mm Shared Memory Library) are needed on the computers that are connected to GSM modems to send and receive short messages.

Install the OSSP mm library from your Linux installation CD's if possible. If they are not included then install the sources.

Download libmm from http://www.ossp.org/pkg/lib/mm/ and the SMS Server Tools from:

Version 2.x: http://www.meinemullemaus.de/smstools/
Version 3.x: http://smstools3.kekekasvi.com/

Copy both files libmm-1.4.0.tar.gz und smstools-x.x.x.tar.gz to the directory /usr/local/src.

cd /usr/local/src
tar -xvzf libmm*.tar.gz
cd mm*
./configure
make
make install

Open the file /etc/ld.so.conf and add the directory /usr/local/lib if it's not already there. After saving the changes run the command **ldconfig** once.

Some Linux version do not have a ldconfig command because they do not need it.

Install the SMS Server Tools:

cd /usr/local/src
tar -xvzf smstools*.tar.gz
cd /usr/local/src/smstools

Open the file /usr/ocal/src/smstools/src/Makefile with a text editor and enable statistics by inserting a hash character in front of the NOSTATS option.

make -s
make -s install

cd /etc/rc3.d or cd /etc/init.d/rc3.d

please change the 3 as described in the previous chapter.

ln -s /etc/init.d/sms s82sms

If you want to start the SMS Server Tools at boot time then simply rename the s82sms to S82sms (capital S). But please wait until you configured and tested the program.

4.2.3.3. Configure MySQL

By default, MySQL allows everybody full access to the database. We should change this immediately. I recommend to set a password for root@localhost and delete all other anonymous users. You can add more users later.

mysqladmin -u root password 'New_Passwort'
mysql -u root -p
mysql> **use mysql;**
mysql> **delete from user where user="" or password="";**
mysql> **quit**
mysqladmin -u root -p reload

The username root is not the same as the Linux user root. MySQL has its own user and password management. In mysql the name root is used for the database administrator by default.

Now you can use the database only as root from the local computer with password. A later chapter will explain how to add more users.

4.2.3.4. Test Apache, PHP and MySQL

Now please test your webserver.

Start a webbrowser on your webserver and enter the URL **http://localhost**. You should see an example page.

Now try the same on another computer and enter **http://your_webserver_name**. Again you should see the example page from Apache. Now you should test the PHP function and the connection to the MySQL database:

Find the htdocs directory:

find / -name htdocs

Create the file test.php inside this directory, with the following content:

```
<html><body>
  This is HTML text.<br>

  <?php
    print("This is PHP text.<br>");
    print("Now a MySQL query follows:<br>");
    mysql_connect("localhost","root","my_password");
    mysql_select_db("mysql");
    $ergebnis=mysql_query("select user,host,password from user");
    for ($nr = 1; $reihe = mysql_fetch_row($ergebnis); $nr++)
    {
      print("$reihe[0] $reihe[1] $reihe[2]<br>");
    }
  ?>

</body></html>
```

Use the name of your MySQL server for the mysql_connect command and insert the correct password. If your Apache webserver and MySQL database run on the same computer use "localhost" as the name.

Ensure that the file has enough read permissions:

chmod a+r /usr/local/apache2/htdocs/test.php

Now start your webbrowser and enter **http://localhost/test.php** or **http://your_webserver_name/test.php**. You should see this:

```
This is HTML text.
This is PHP text.
Now a MySQL query follows:
root localhost x4gdsag2354124
```

The first line was created by regular HTML code, the second line tests the PHP integration into the webserver. The third and fourth line test the connection to your MySQL database.

Please review the example program above. You can see how a PHP code was placed into a HTML file. The beginning and end are marked with <?php and ?>.

The PHP program connects to the SQL database and queries the list of SQL-users and their passwords. The passwords are encrypted for security therefore you do not see them in clear text.

4.2.3.5. Reminder For Linux

Please remind the following files because they are not identical in every operating system. In the following chapters I will not write about operating system specific things.

SMS Server Tools

start	/etc/init.d/sms start
stop	/etc/init.d/sms stop
config file	/etc/smsd.conf
error logfile	/var/log/smsd.log
sms queues	/var/spool/sms

Apache webserver

start	/etc/init.d/apache2 start
stop	/etc/init.d/apache2 stop
config file	/usr/apache2/conf/httpd.conf
error logfile	/usr/apache2/logs/error.log
web pages	/usr/apache2/htdocs

The location of these files is different on some linux distributions.

MySQL database

start	/etc/init.d/mysqld start
stop	/etc/init.d/mysqld stop
client	mysql -u root -p

Crontab

modify	EDITOR=kedit; export EDITOR; crontab -e

If kedit is not installed, you might use xedit or vi instead.

4.3. CygWin Quick Start

Most Windows administrators have nor knowledge about Unix shells.

Therefore I show you the most important differences to the DOS command line in a short course and I show you some other useful hints.

Learn how the CygWin Bash Shell works because you will use it often in future. Nearly all programs of this book are controlled by entering commands in this shell. The programs have no graphical interface that you could youse with your mouse.

4.3.1. The Shell
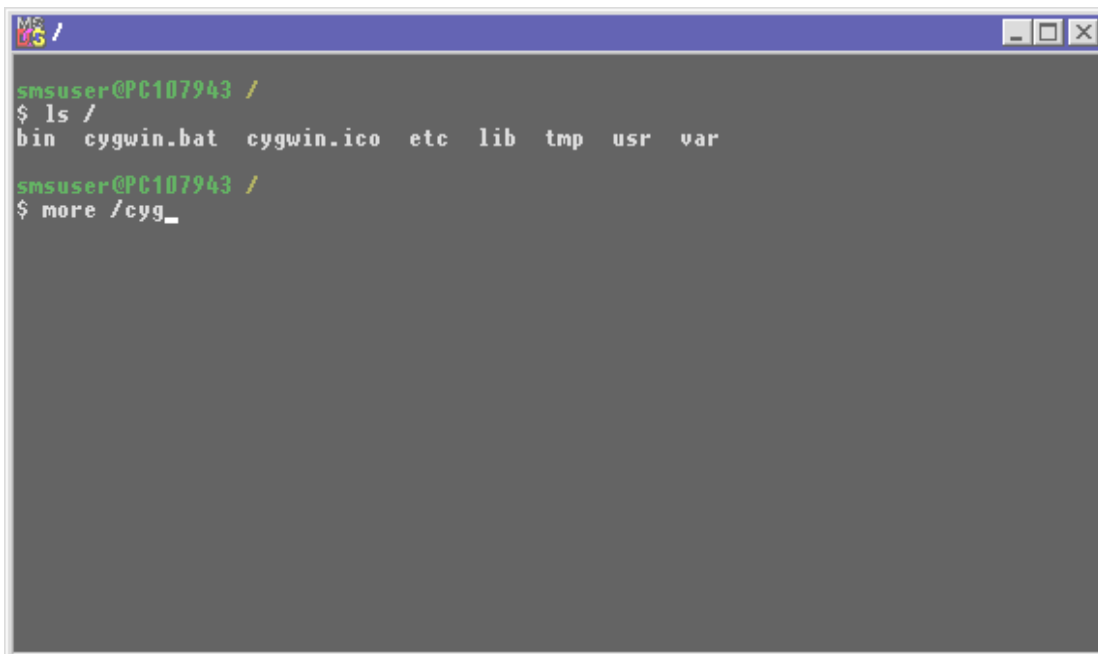
(The following screenshots are a little bit light to avoid paper jam in laser printers and save ink or toner. I hope the contrast is still good enough to read the text within them.)

The Shell is comparable to the MS-DOS command window. Here you enter Unix commands.

If I write in this book that you should enter a command, I mean always that you should enter them in the CygWin Shell window because the MS-DOS command window accepts only DOS and Windows commands.

You can use the cursor keys to repeat and edit the last commands.

The tabulator key can be used to complete incomplete filenames. Please try it:

The first command shows the content of the current directory in the easiest format. It is similar to the command "dir". The second command tests the tabulator key. Enter the half filename as shown above and press the tabulator key. Then the filename is automatically completed. In this case there are two files that begin with "cygwin", therefore the tab key does not complete the filename extension (.bat or .ico) automatically.

If you close a shell window, then the programs that you started within that windows are also normally closed. But some programs remain in memory and continue to run! In case of doubt you can check this using the taskmanager of Windows.

4.3.2. Filenames

All names in Unix are case-sensitive. You have to care capital and small letters.

If you use wildchars (jokers) in filenames you should keep in mind some small differences to the DOS command window. Description of wildchars in Unix:

A star is a placeholder for zero or more characters.

A question mark is a placeholder for exactly one character.

The dot has no special meaning to Unix. It is treated like any other letter. Lets say there are many README files:

```
/usr/doc/Cygwin                                                    _ □ ×

smsuser@PC107943 /
$ cd /usr/doc/Cygwin/

smsuser@PC107943 /usr/doc/Cygwin
$ ls *
apache-1.3.24-5.README        gettext-0.11.5.README     perl-5.6.1.README
binutils-20021117-1.README    guile-1.6.0-1.README      php-4.2.0-1.README
bzip2-1.0.2.README            libxml2-2.4.23.README     postgresql-7.3.1.README
crypt.README                  login.README             procmail-3.22.README
curl-7.10.2-1.README          mc-4.5.55-1.README        readline-4.3.README
cygrunsrv.README              mktemp-1.4.README         rxvt-2.7.9.README
cygutils-1.1.3.README         ncftp-3.1.4.README        ssmtp-2.38.7.README
cygwin-doc-1.3-1.README       ncurses-5.2.README        terminfo-5.2.README
fetchmail-6.2.0.README        openssl-0.9.7.README      zlib-1.1.4.README
gdbm-1.8.0.README             pcre-3.7.README

smsuser@PC107943 /usr/doc/Cygwin
$ _
```

Then the command **ls *** shows all of them. In the DOS command windows you had to enter **dir *.*** because in Dos a star character does not match a dot. In Unix the star also matches the dots, therefore all files appear.

The same happens with the question mark:


```
/usr/doc/Cygwin                                                    _ □ ×

smsuser@PC107943 /
$ cd /usr/doc/Cygwin/

smsuser@PC107943 /usr/doc/Cygwin
$ ls *1?README
binutils-20021117-1.README    guile-1.6.0-1.README      php-4.2.0-1.README
curl-7.10.2-1.README          mc-4.5.55-1.README        postgresql-7.3.1.README
cygwin-doc-1.3-1.README       perl-5.6.1.README

smsuser@PC107943 /usr/doc/Cygwin
$
```

You can combine many wildchars, for example:

This command shows all files that have "READ" somewhere.

In the DOS command window the wildchars are detected and handled by the command itself. You can enter **dir *.txt** because the dir command knows what to do with stars. But not all commands know wilchars - try it with notepad:

Notepad outputs an error message because it does not understand the star and tries to open the file *.txt. This file does not exist.

You will notice that very much programs support stars and question marks but not all. MS-Word 2000 (and maybe older version) for example works with them.

In every Unix shell the wildchars are detected by the shell itself and are replaced by the "real" filenames. Therefore you can start editpad with wildchars (**editpad *.txt**) because the shell replaces *.txt by the filenames that match this pattern before it starts the program editpad.

(This does not work with notepad because notepad cannot open more that one file at the same time).

Executable programs have no special filename extension in Unix, but Windows programs are typically called something with .exe at the end.

4.3.3. Directory Names

In Unix directory names are separated by slashes while Windows uses backslashes instead. Example:

ls /usr/local/

The backslash has another meaning in Unix:

If your command does not fit into one single line you can lengthen it by adding a backslash and writing the rest in the next line, e.g.:

**ls \
"My text.doc"**

This command is similar to **ls "My text.doc"**. This can be useful for very long command lines.

You can leave out double quotes around filenames with space characters if you write a backslash in front of every space, e.g.:

ls My\ text.doc

This tells the shell that the space is part of the filename and not a separator between two filenames.

Unix does not have drive letters. The main directory of the first harddisk is simply called "/". But CygWin is not Unix, therefore it uses c:\cygwin as the main directory within the cygwin shell.

In the CygWin shell the directory c:\cygwin is used as the main directory.

Now you may ask how you can access c:\. This is easy if you know that every windows drive letters are in the directory /cygdrive (emulated drives):

Windows	CygWin Shell
c:\cygwin	/
c:\cygwin\tmp	/tmp
c:\	/cygdrive/c
d:\	/cygdrive/d
e:\	/cygdrive/e
c:\windows	/cygdrive/c/windows

If you enter **ls /cygdrive** you may not see all drive letters that you expected. I was not able to find an explanation for this. But you can use all existing drives even if the ls command does not show their letters!

4.3.4. File Permissions

In MS-DOS any user can access any file without restrictions. You can disable write access by using the attrib command but any user can use the same command to enable write access again.

In Unix this works different because Unix was made for multiple-user systems with network functions from the first beginning. CygWin emulates the file permissions of Unix in Windows to make programs useable in Windows without major modification.

Any file has three sets of permissions

- user

- group

- others

User is the person who created the file, also called the owner.

The group is always "unknown". All users of the Windows PC belong to the group "unknown" because Windows 95 does not know user groups (in Unix style) and CygWin was initially developed for Windows 95.

Others are all other persons, people in the network that this PC does not know. They have normally no access to any file. Therefore the file permissions to these users are meaningless in Windows.

There are three permissions for any of these three groups:

- read (r)

- write (w)

- execute (x)

Read means that the file can be read.

Write means that the file can be written.

Execute means that the file can be executed, this makes only sense for programs.

Directories have also permissions. They use the same letters but their meaning is slightly different:

- show directory table of content (r)

- modify directory table of content (w)

- step into the directory (x)

The w permission allows to modify the table of content. This means adding and deleting files and renaming them because these actions modify the directory content. Changes to existing files do not touch the table of content.

The x permission allows you step into the directory using the **cd** command.

The command **ls -l /etc** shows you the content of the directory /etc in detail:

```
-rw-r--r-- 1 SF unknown 12546 Oct 1 05:53 termcap
```

The first character is a d in case of directories and a dash to indicate a regular file (as above). Then three blocks of rwx permissions follow. Left for the owner, in the middle for the group and right for others.

The file above can be read and written by the owner. The group and others can only read it.

The file was created by the user "SF" who is part of the group "unknown".

The file-size is 12546 bytes and it was created in the 1st october at 5:53.

The filename is termcap.

Nobody can execute the file because it has no "x" permission. This is not an executable program.

4.3.5. Comparision of Windows/Unix commands

If you know DOS/Windows commands you may find the following table useful. It compares some often use DOS/Windows commands with the corresponding Unix version.

Windows Command	Unix Command
dir c:\windows	**ls -l /cygdrive/c/windows**
Show table of content of a directory	
cd \temp	**cd /tmp**
Go to temp directory	
md test	**mkdir test**
Creates a new directory with name test	
rm test	**rmdir test**
Removes the directory temp	
del hello	**rm hello**
Deletes the file hello	
deltree trash	**rm -R trash**
Deletes the whole directory trash with all files within it.	
command /?	**command --help | more** **command -h | more** **command -h | more**
Shows help to a command.	
Windows Taskmanager	**ps -ef** **kill nummer** **kill -9 nummer**
Stops the program without taking care of any risks. This is similar to a multiple klick on "quit task" in the windows Taskmanager for hanging programs.	
attrib -w anyfile	**chmod a-w anyfile**
Removes all users (a) the write (-w) permission. Instead of a you coud also use u,g or o. +w would add the write permission.	
move file destination	**mv file destination**
Moves the file to a destination directory.	
ren old new	**mv old new**

Renames a file

---

type filename   **cat filename**

Shows content of a file

---

type filename | more   **cat filename | more**
                                                                                          more filename
                                                                                          less filename

Shows content of a file page by page. Less can also scroll back.

---

exit   **exit**

Exits a shell

4.3.6. Search Path

Most DOS/Windows user know the PATH variable. The variable specifies where to find executable programs if they are not in the working directory.

Unix has the same variable. The command **echo $PATH** shows it. Please note that Unix uses colons to separate the directory names while DOS/Windows uses semi-colons.

To add a directory enter

PATH=$PATH:/any/directory
export PATH

One thing is really different between Unix and DOS/Windows. Unix never searches programs in the current working directory, except if the current directory is in the PATH variable. Therefore the PATH variable contains mostly a ".". Like in Dos/Windows the dot refers always to the current working directory.

4.4. Working With Cron Jobs

Cron jobs are programs that run every day or week or month at a specified time.

In Windows you need the program nnCron Lite. In Linux and Unix such cron jobs are part of the operating system.

In Windows you simply need to modify the file c:\programme\cron\cron.tab.

In Unix you have to enter the commands **EDITOR="vi";export EDITOR;crontab -e** to modify the cron table. Then Unix starts the editor vi and opens the table in it. You can also use nearly any other editor, for example dtpad or kedit if you like them more.

Please note that every user in a Unix system has its own cron table. Normally you will use the cron table of the root user therefore you need to log in as root before modifying it.

Independently of the editor that you use the file format of the cron table is always the same. One example:

```
file=/tmp/trash.txt
1,31****rm$file
*/10****cp/tmp/logfile.old/tmp/logfile.new
51**1cp/tmp/logfile.old/tmp/logfile.monday
01-3***rm/tmp/textfile.txt
# This is a comment
```

The first command sets an environment variable. All cronjobs can use this variable when they run. You will not use this feature very often. You can set as many variables as you like.

The next four lines start commands at a specified time. Each line has exactly 5 values for the time:

- minute (0-59)

- hour (0-23)

- day (1-31)

- month (1-12)

- weekday (0=sunday, 1=monday, ...)

The first command runs every day and every hour at minute1 and minute 31.

The second command runs daily every 10 seconds (Solaris does not support this syntax, you could enter 0,10,20,30,40,50 instead of */10).

The third cronjob runs every monday at 01:05.

The fourth cronjob runs every day at 01:00, 02:00 and 03:00.

As you can see, the star means every minute, hour, day or month.

*/x means every x minutes, hours, days or month's.

You can enter many values by separating them with a comma. The comma means as much and the word "and".

Ranges can be entered using a dash and they mean from-to.

Solaris does not understand */x.

In Windows all command started as cron jobs write their output into logfiles.

The rest of this chapter refers only to Unix.

Unix sends an eMail to the user who started a cron job whenever the commands generate any output.You can disable this by redirecting the output into a file:

```
* * * * * command >/var/log/logfile
```

This redirects "normal" outputs into a file. Please note that the directory /var/log is only writeable by root user! For other users you need to create an empty logfile with write permissions first.

If you redirect output to /dev/null they simply go to nowhere. /dev/null "eats" everything that you give him and does nothing with it.

You will still get eMails for error messages if you redirect only "normal" output. To disable also error messages use this command:

```
* * * * * command >/var/log/logfile 2>&1
```

You should not use this commadn with /dev/null because you would not be able to see any error message of the command. If you redirect everything including error messages you should do it with a file that you can check later if something goes wrong.

4.5. Working With eMails

Many user like to combine their SMS application with eMail.

The following pages assume that you have already a working mailserver. I will show you how you can use your mailserver with commands at the command line.

You can use these command later so send and receive eMails from shell-scripts.

4.5.1. Configuring Sendmail

This chapter talks about Unix (Solaris and Linux).

Sendmail is a complicated program to send and receive eMails because it can do nearly everything.

Normally I would use another easier program but sendmail is installed on nearly every Unix system and there are not really many alternatives.

A fresh installed sendmail can only deliver local eMails. I will show you how to send eMails out of the local computer (e.g. to the internet) by connecting sendmail to an external Mailserver that does already exist.

If you like to learn more about sendmail you could read the book

sendmail
O'Reilly
ISBN 1-56592-222-0

This book contains 1050 pages about sendmail, really hard stuff. I recommend this book only to very experienced Unix administrators.

Pease note that the config files of every sendmail version are different. Therefore you may not be able to copy the examples from this book to your installed sendmail version without modification.

4.5.1.1. Configuring Sendmail On SuSE Linux

The latest SuSE distributions do not install sendmail as the default mail transfer agent any more. They use postfix instead. This book does not cover postfix, so please uninstall it and install sendmail.

Actual SuSE distributions allow you to configure sendmail with Yast2. Older SuSE distributions and other Linux versions have typically no configuration tool for sendmail. Now I will describe how to configure sendmail manually:

The following steps are tested on SuSE 7.3:

First you need to edit /etc/rc.config.d/sendmail.rc.config according to your network environment. The file has a lot of useful comments.

First enter in the line SENDMAIL_SMARTHOST the hostname of the external working mailserver. All external eMails will be forwarded to this server. After changing this file you need to run /sbin/SuSEconfig to activate the changes.

Try to send an eMail using the command:

/usr/lib/sendmail -v receiver@mail.isis.de
From: sender@mail.isis.de

Hello
<Ctrl-D>

Use a meaningful recipient address. The mail should reach the receiver in some seconds or minutes depending on the speed of the external mailservers.

4.5.1.2. Configuring Sendmail On RedHat Linux

This instruction applies to RedHat 7.3. Newer versions will have a better configuration tool - hopefully. Maybe this instruction works also in newer versions.

RedHat has a template configuration file that you can use.

Edit /etc/mail/sendmail.mc. Some lines are commented out by 'dnl' at the left side. You can actiavte them by removing 'dnl'. Do not remove 'dnl' at the right end of any line.

Remove the left 'dnl' from this line

```
define(`SMART_HOST',`mail.isis.de')dnl
```

and enter your own external mailserver as in the example above. Now enter the following commands to convert this template into a real configuration file.

cd /etc/mail
m4 sendmail.mc > sendmail.cf

Test sending a mail using the command:

/usr/lib/sendmail -v receiver@mail.isis.de
From: sender@mail.isis.de

Hello
<Ctrl-D>

Use a meaningful recipient address. The mail should reach the receiver in some seconds or minutes depending on the speed of the external mailservers.

4.5.1.3. Configuring Sendmail On Solaris

I tested this instruction on Solaris 6 and Solaris 8. If it does not work in your version, read the book "sendmail" from O' Reilly.

Enter the ip-address and hostname of the external Mailserver in /etc/hosts and give him the alias name mailhost.

```
192.168.1.12 mail.isis.de mailhost
```

Copy /etc/mail/subsidiary.cf to /etc/mail/sendmail.cf.

Try to send a mail with the command:

/usr/lib/sendmail -v receiver@mail.isis.de
From: sender@mail.isis.de

Hello
<Ctrl-D>

Use a meaningful recipient address. The mail should reach the receiver in some seconds or minutes depending on the speed of the external mailservers.

4.5.2. Configuring Ssmtp

This chapter applies only to Windows systems.

Ssmtp emulates sendmail, that is the default mail transfer agent for Unix.

The installation program of CygWin has a small bug. The link is missing that allows you to run ssmtp with the command sendmail as described in the manual of ssmtp. Therefore please fix this bug by creating this link yourself:

cd /usr/lib
ln -s /usr/sbin/ssmtp.exe sendmail

Create the configuration file /etc/ssmtp/ssmtp.conf with this content:

```
mailhub=mail.isis.de:25
#hostname=mypc.local
FromLineOverride=YES
```

Mailhub specifies the name and port of your external Mailserver, for example Microsoft Exchange. The Port number is 25 in nearly every case and depends on the configuration of the external Mailserver.

Hostname is the name of your PC and can be commented out normally as shown above. The last line specifies that the From: line of all sent eMails should not be modified by Ssmtp.

Try to send a mail with the command:

/usr/lib/sendmail -v receiver@mail.isis.de
From: sender@mail.isis.de

Hello
<Ctrl-D>

Use a meaningful recipient address. The mail should reach the receiver in some seconds or minutes depending on the speed of the external mailservers.

4.5.3. Sending eMails

Now you have installed sendmail or ssmtp. You are ready to send eMails.

The easies way to send eMails is this:

/usr/lib/sendmail receiver@somewhere.de
Hello!
<Ctrl-D>

Only some few mailserver accept those minimal eMails because it has no header. This mail would appear anonymous on the receivers computer.

You could compare this to a letter that somebody puts into your mailbox without any label, and the text sheet has only one word "Hello!". Most mailserver do not deliver such anonymous mails any more. They require at least the information, who sent this mail.

A complete eMail looks like this:

/usr/lib/sendmail klara.examplewomen@somewhere.de
From: "Michael Exampleman" <michael.exampleman@somewhere.de>
To: "Klara Examplewomen" <klara.examplewoman@somewhere.de>
Subject: Testmail

Hello, this is a test!
<Strg-D>

The empty line between the header and the text is important. For the sender and receiver you can leave out the names or simply write only the eMail addresses without any special characters around them. But the name is useful for a nice display in the eMail reader program:

/usr/lib/sendmail klara.examplewomen@somewhere.de
From: Michael.exampleman@somewhere.de
To: klara.examplewoman@somewhere.de
Subject: Testmail

Hello, this is a test!
<Strg-D>

If you encounter problems while sending eMails then use the option -v with sendmail (before the receipient address). The you will see a lot of information about success or error messages.

4.5.4. Receiving eMails

To receive eMails you can use the program fetchmail.

Create a configuration file /etc/fetchmail.rc with one or more lines like this:

```
poll mail.isis.de protocol POP3 user my_username pass my_password mda "cat >`mktemp
/tmp/mail.XXXXXXXX`"
```

Please write this in one single line. You can add as many lines as you want, one for each external mailserver. This fetches all stored messages in your POP3 mailbox.

In this case mail.isis.de is the external mailserver. The POP3 account has the name my_username and the password my_password.

Enter the command **chmod 0710 /etc/fetchmailrc** because otherwise fetchmail will not read it. Now you can fetch eMails using the command

fetchmail -f /etc/fetchmailrc

You get more status information using the option -v. All received mails will be stored in the directory /tmp/mail/mail.xxxxxxx. The command mktemp replaces the x by random characters and ensures that every mail get its uniqe filename.

You can also start fetchmail as a daemon that runs in background and checks every n seconds for new mails. Use the option -d 300 to fetch mails every 300 seconds. You can quit the background program using the command **fetchmail --quit**.

fetchmail -f /etc/fetchmailrc -d 300

fetchmail --quit

A typical received eMail looks like this example:

```
Return-Path:<root@linux.local>
Date:Wed,12Feb200317:36:42+0100
From:root<root@linux.local>
Message-Id:<200302121636.h1CGagJ16586@linux.local>
To:s.frings@linux.local
Subject:Test3

Hello, my friend!
```

Every program that transfers the mail from the sender to the destination may add more lines to this header. It is normal when the header has much more lines.

4.5.5. Working With eMails

Now you can fetch mails from POP3 accounts and save them into text files. But how can you use this files by programs to trigger actions depending on the file content?

Modify /etc/fetchmail.rc like this example:

```
poll mail.isis.de protocol POP3 user mustermann pass geheim mda "/usr/local/bin/popmail"
```

Please write this in one single line. Now fetchmail runs the script popmail whenever it receives a mail. It gives the content to the standard input channel of popmail. Popmail could be such a scipt:

```
#!/bin/sh
filename=`mktemp /tmp/mail.XXXXXXX`
cat >$filename
echo "You got mail! (in file $filename)"
```

If you want to enter this script then please do not forget to use the command **chmod a+rx /usr/local/bin/popmail**, else the script would be not executable.

Now this script creates a temporary file for the mail and writes the content into it using the cat command. Cat reads everything from the standard input channel and writes it into the file. In addition the script shows a notification on the screen:

```
"You got mail! (in file /tmp/mail.QE4F3AB9)"
```

This is only a very short and easy example. It is not very useful but it shows you the basic concept how to run programs with received eMails. Later chapters will show you better examples.

4.5.6. Using Formail

Formail is useful to extract lines from the whole eMail file or when you want to modify it.

Lets take the eMail from the last chapter as an example:

```
Return-Path:<root@linux.local>
Date:Wed,12Feb200317:36:42+0100
From:root<root@linux.local>
Message-Id:<200302121636.h1CGagJ16586@linux.local>
To:s.frings@linux.local
Subject:Test3

Hello, my friend!
```

If you want to see only the sender, then you can enter

formail -zx From: < /tmp/mail.QE4F3AB9

To get only some header lines and the mail text, enter

formail -k -X From: -X Subject: < /tmp/mail.QE4F3AB9

To get only the mail text without any header, enter

formail -I "" < /tmp/mail.QE4F3AB9

The next command modifies the receiver of this mail

formail -f -I "To: exampleman@somewhere.de" < /tmp/mail.QE4F3AB9

All these command do not modify the source file. They write the new data to the standard output channel that is typically the screen. You can easily redirect this output into a new file or into variables:

formail -zx From: < /tmp/mail.QE4F3AB9 > new_filename

from=`formail -zx From:< /tmp/mail.QE4F3AB9`
echo "Der Sender is $from"

4.6. Working With FTP

FTP interfaces allow the transfer of files between computers. FTP works on all operating systems and on all types of IP networks. It is the most easiest method to transfer files.

If we talk about FTP we have always FTP servers and FTP clients.

The server allows other computers to access directories on his harddisk. The clients are the computer who access the server. FTP is part of every Unix operating system. For Windows you need to install a third party program, for example the War FTP Daemon.
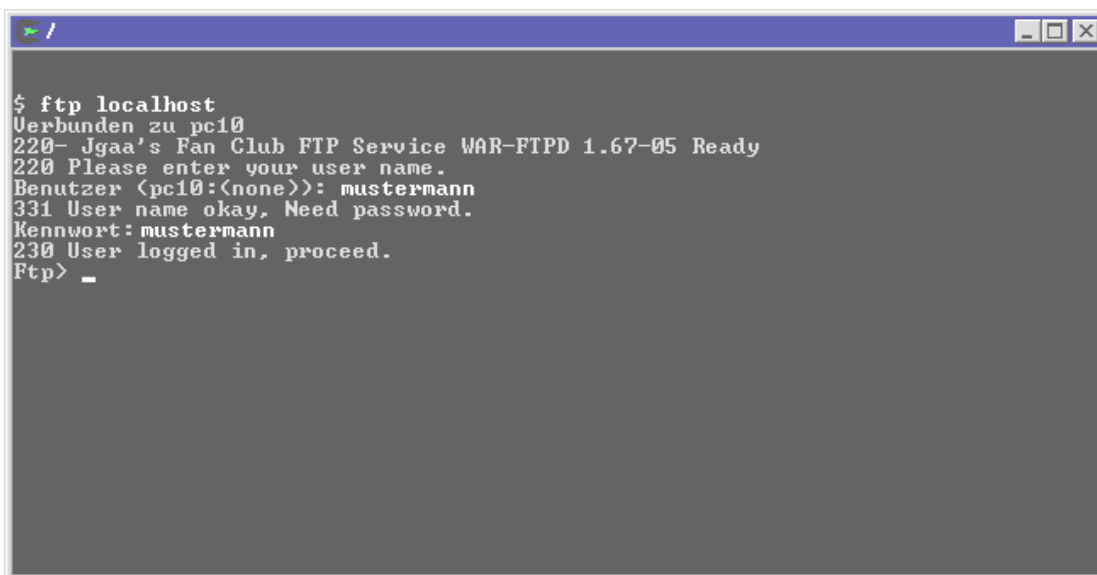
On Unix you need the inet daemon that is surely already installed. Every regular Unix user who can log on to the system can also use FTP and has access to all directories like in a telnet session. That means that there is no special configuration necessary to enable FTP.

The FTP client sends or receives files from the server. Windows contains a client program for the command line that works exactly as the Unix program. The next chapter describes how to use this client.

4.6.1. Using The FTP Client

Every operating system comes with an FTP client. The program is useful for interactive work but it is not good for automatic file transfer.

You start the client in a shell window or in the MS-DOS command window by entering the command **ftp hostname**. You can also use the IP-address instead of the hostname if you like:

After the connection has been established you are prompted for username and password. In this case both "mustermann".

I made this screenshot on a German PC therefore you see a mixture of English and German above.

Verbunden zu = Connected to
Benutzer = User
Kennwort = Password

I recommend to enter the next two commands always at first:

binary Switches from 7bit to 8bit mode. This is necessary for all files except text files. The oppsite mode (text) converts line breaks and removes the highest bit from every byte.

hash Activates a progress bar while transferring files. This is useful for large files to see that something is happening when the transfer takes a long time.

The next commands are used to transfer files. You need some or all of them:
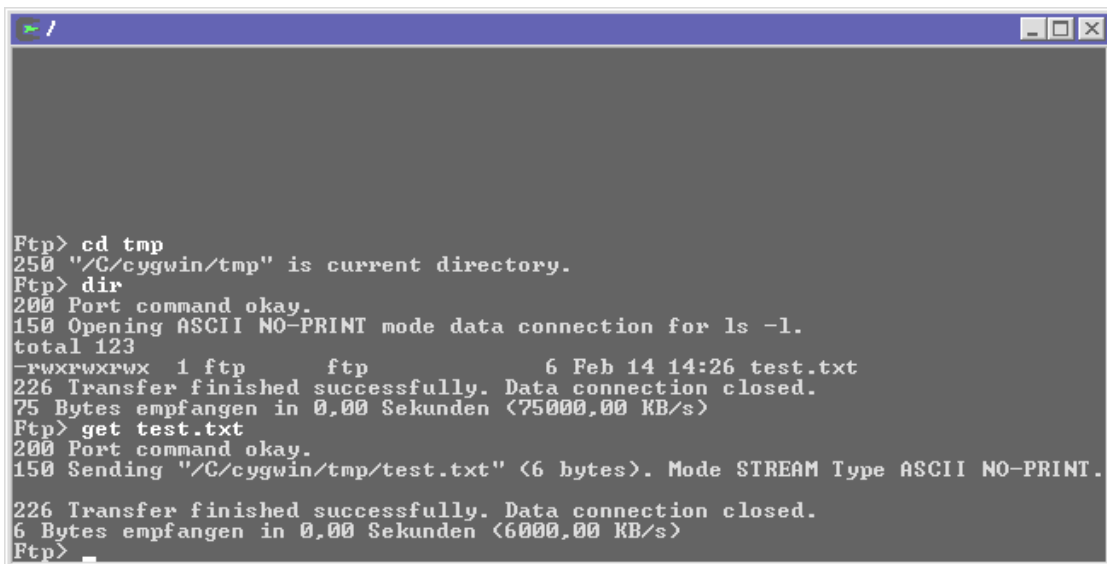
cd directory Changes the working directory on server side

dir Shows content of the directory

rm filename Removes a file

rmdir directory	Removes a directory
mdir directory	Creates a new directory
put filename	Sends a file to the server
get filename	Receives a file from the server
mput filename	Sends many files. The filename may conatain ? and *.
mget filename	Receives many files. The filename may contain ? and *.
prompt	Disables "are you sure" questions for every file in mget and mput commands.
bye	Terminates the connection and quits the ftp client program.

The next screenshot shows how I download the file /tmp/test.txt.


```
Ftp> cd tmp
250 "/C/cygwin/tmp" is current directory.
Ftp> dir
200 Port command okay.
150 Opening ASCII NO-PRINT mode data connection for ls -l.
total 123
-rwxrwxrwx  1 ftp        ftp             6 Feb 14 14:26 test.txt
226 Transfer finished successfully. Data connection closed.
75 Bytes empfangen in 0,00 Sekunden (75000,00 KB/s)
Ftp> get test.txt
200 Port command okay.
150 Sending "/C/cygwin/tmp/test.txt" (6 bytes). Mode STREAM Type ASCII NO-PRINT.

226 Transfer finished successfully. Data connection closed.
6 Bytes empfangen in 0,00 Sekunden (6000,00 KB/s)
Ftp> _
```

Again some german words:

empfangen = received

Sekunden = seconds

4.6.2. Automatic FTP Downloads

To transfer files automatically you can use the program package ncftp. The most interesting commands of this package are ncftpget and ncftpput. On Windows you need to run these commands in a CygWin Bash Shell.

Download files from a server:

ncftpget -u Username -p Password Servername DestinationDirectory Sourcefiles

Source files can be one or more filenames or a filename with wildchars (* and ?). If you use wildchars you have to include the filename in single quotes because otherwise the shell would try to interpret these characters - and that's not what you want.

Two correct examples:

ncftpget -u exampleman -p test MyServer /download /tmp/test.txt
ncftpget -u exampleman -p test MyServer /download '/tmp/test.*'

These commands download the file /tmp/test.txt from the server and store them locally in the /download directory.

If you use the option -DD (ncftp -DD ...) the files will be deleted from the server after successful download.

The option -R allows you to download a whole subdirectory:

ncftpget -R -u eampleman -p test MyServer /download /tmp

After this download you will have the local directory /download/tmp that is a copy of /download on the server.

Ncftpget quits with a return code that informs you about success or failure. You can check this exit code in scripts to react on errors.

You can watch the exit code by entering **echo $?** immediately after the ncftpget command. 0 means that the transfer was completely successful. Other values indicate errors.

The manual page (man ncftpget) contains a list of all possible error codes at its end.

4.6.3. Automatic FTP Uploads

Sending files works similar to receiving. Upload files using the command:

ncftpput -u Username -p Password Servername DestinationDirectory SourceFiles

Source files can be one or more filenames. You can also enter wildchars. If you use wildchars you have to include the filename in single quotes because otherwise the shell would try to interpret these characters - and that's not what you want.

Two correct examples:

ncftpput -u exampleman -p test MyServer /upload /tmp/test.txt
ncftpput -u exampleman -p test MyServer /upload '/tmp/test.*'

These commands upload the file /tmp/test.txt to the server into the directory /upload.

If you use the option -DD (ncftpput -DD ...) the files will be deleted from your computer after succesful transfer.

The option -R allows you to upload a whole subdirectory.

ncftpput -R -u exampleman -p test MyServer /upload /tmp

This uploads the local directory /tmp to the server into /upload and all the files in this directory.

Ncftpput quits with a return code that informs you about success or failure. You can check this exit code in Scripts to react on errors.

You can watch the exit code by entering **echo $?** immediately after the ncftpget command. 0 means that the transfer was completely successful. Other values indicate errors.

The manual page (man ncftpget) contains a list of all possible error codes at its end.

4.7. Using MySQL

SQL databases are complex programs because they are flexible, fast and still easy to use. Therefore they are very often used.

MySQL is a free SQl database server. Its has not very many functions but most applications do not really need more than MySQL can do.

MySQL has a short but complete documentation. It's very hard to read for beginners, therefore I decided to show how to start with MySQL in this chapter.

I will not explain all functions of MySQL because I only want to help you with the first steps. You can learn the details from the original documentation - if you ever need to learn them.

Enter all the examples line by line because they depend on previous commands. Otherwise you will not be able to test the examples.

If you know MySQL already you can skip this chapter.

4.7.1. The Principle Of MySQL

MySQL is a client/server program. That means that you always operate at least one SQL server and one or more clients. Both can physically run on the same computer.

The server stores data while the client reads or modifies them. Any number of clients can access the databases at the same time.

When I write "endless" or "unlimited" this is not 100% true. Every operating system enforces MySQL to limits but you will not reach these limits in "real" life.

A MySQL server can store an unlimited number of databases.

Every database contains an unlimited number of tables.

Every table has unlimited rows and columns. In the table fields the data are stored.

The rows of the tables are called records. For example an address is a record. The parts of the address are the fields, stored in the columns of the table. Every column has a special format, for example text, date or number. You specify the data format of the columns when you create the database.

During creation you specify the number of tables and columns of each table. The number of rows is unspecified and starts with zero. It is possible to add more columns later, whenever necessary.

4.7.2. Data Types

When you create a database you give it a name - nothing more. Database names allow you to group many tables.

In every database you create tables and specify the number and data-types of the columns. Each table has no single row in this moment.

When you create tables you should always use the smallest data-type that matches your need. This saves disk space and ensures the best performance.

The following table shows you some data-types supported by MySQL:

INTEGER	Number in the range -2147483648 to 2147483647
BIGINT	Number in the range -9223372036854775808 to 9223372036854775807
FLOAT	Floating number in the range -3.4E38 to +3.4E38 with 4 byte
DOUBLE	Floating number in the range -1.79E308 to +1.79E308 with 8 byte (more exactly)
DATE	Date with format YYYY-MM-DD
TIME	Time with format HH:MM:SS
DATETIME	Date and timewith format YYYY-MM-DD HH:MM:SS
TIMESTAMP	Timestanp with format YYYYMMDDHHMMSS. Is set automatically on every change-access.
CHAR(n)	String with exactly n characters (max. 255)
VARCHAR	String with 0-255 characters
TEXT	String with 0-65535 characters
BLOB	String with 0-65535 characters
LONGTEXT	String with 0-4294967295 characters
LONGBLOB	String with 0-4294967295 characters

When you store a date you may write the year with only 2 digits. MySQL converts it automatically to four digits. MySQL allows zeroes in date and time, e.g. 0000-00-00 00:00:00 as a dummy value.

Strings of type CHAR have a fixed length, independently how many characters you store. The disk usage is always the same. They are fast. The disadvantage is that they typically reserve unused disk space.

Strings of type TEXT, LONGTEXT, BLOB and LONGBLOB need exactly as much disk space as their length. The access to these types is slower than to CHAR.

I you search or compare strings the types TEXT and LONGTEXT do not care the case while BLOB and LONGBLOB are case sensitive.

If you store strings, trailing spaces at the right side are truncated but heading spaces at the left side are stored.

4.7.3. Options To Data Types

With INTEGER and BIGINTEGER you can specify the length (for displaying) in brackets:

```
INTEGER(n)
BIGINT(n)
```

When you read these fields later they are displayed in the specified width, right oriented. You can still store larger numbers. In combination with the ZEROFILL option small values are filled with zeroes to the specified width:

```
INTEGER(n) ZEROFILL
BIGINT(n) ZEROFILL
```

When a column has no stored data, MySQL shows this using the keyword NULL. You can use the option NOT NULL, to enfores filling this columns when adding or modifying the row:

```
INTEGER NOT NULL
```

You can specify a default value that is used when somebody creates a row and leaves this field empty:

```
CHAR(5) DEFAULT "Hello"
```

The option AUTO_INCREMENT creates a column that is automatically incremented. The first row gets the value 1.

Each table should have one column with the option PRIMARY KEY NOT NULL. This enforces that every row gets a unique value in this column. It cannot be empty (NULL) and it cannot have the same value as another row. Mostly it is used like this:

```
id INTEGER AUTO_INCREMENT PRIMARY KEY NOT NULL
```

This creates a column that is automatically incremented and is protected against duplicate values.

4.7.4. Create A Database With Two Tables

Now please start the mysql client by the command **mysql -u root -p**. Enter the password that you set during installation.

Now create a database using the command **create database mytest;** The new database is called addresses. Tell mysql to use this database for the following commands:

use mytest;

Create a new table and specify the data types of its columns:

```
create table addresses(
    id INTEGER AUTO_INCREMENT PRIMARY KEY NOT NULL,
    name CHAR(30),
    firstname CHAR(30),
    street CHAR(30),
    city_code CHAR(5),
    city CHAR(30)
);
```

Create a second table:

```
create table accounts(
    id INTEGER PRIMARY KEY NOT NULL,
    value FLOAT,
    from TIMESTAMP
);
```

Now you can store addresses and accounts. You could store both in one single table but this is too easy for this exaple. I like to show you how to combine two tables later.

To show the table structure use these commands:

show databases;

use mytest;

show tables;

describe addresses;

describe accounts;

Now you can start to fill these two tables with data. The timestamp will be set automatically. The ID numbers for the addresses will be set automatically while the ID's for the accounts have to be set manually.

The next chapter show you how to do this.

4.7.5. Filling Tables With Data

Enter the first address into the table:

insert into addresses set
        name="Exampleman",
        firstname="Markus",
        street="Long Avenue 3",
        city_code="11111",
        city="Somewhere";

The order of the columns does not matter, the do not need to match the order from the table definition. You can also leave out some columns:

insert into addresses set
        firstname="Lisa",
        name="Examplewoman";

Line break in these commands are only used to make the command more beautiful. They have no meaning to MySQL. You can also write everything in one single line.

Another version of the insert command with the same result is this:

insert into addresses (firstname, name)
        values ("Lisa", "Examplewoman");

Now please take a look ino your table to see the content by entering **select * from addresses;**. You see that the ID column was set and incremented automatically and the empty fields from Lisa are marked with the keyword NULL.

```
mysql> select *from addresses;
+----+-------------+-----------+---------------+----------+-----------+
|id  |name         |firstname  | street        |city_code |city       |
+----+-------------+-----------+---------------+----------+-----------+
| 1  |Examplemann  |Markus     |Long Avenue 3  |11111     |Somewhere  |
| 2  |Examplewoman |Lisa       |NULL           |NULL      |NULL       |
+----+-------------+-----------+---------------+----------+-----------+
2 rows in set(0.00sec)
```

Lets say you want to add the city and city_code of Lisa now. Then you can do it using this command:

update addresses set
        city_code="22222",
        city="Watervalley"
        where id=2;

Now you know why the ID field is so important - it is necessary to specify a table row exactly. The keyword WHERE is used to specify which rows the command should modify. The next chapter explains this keyword in detail.

The result of the last UPDATE command should not surprise you:

```
mysql>select * from addresses;
+----+-------------+-----------+---------------+----------+-------------+
| id |name         |firstname  | street        | city_code | city        |
+----+-------------+-----------+---------------+----------+-------------+
| 1  |Examplemann  |Markus     | Long Avenue 3 | 11111     | Somewhere   |
| 2  |Examplewoman |Lisa       | NULL          | 22222     | Watervalley |
+----+-------------+-----------+---------------+----------+-------------+
2 rows in set(0.00sec)
```

Please create a third address. It is used in the next examples:

insert into addresses set
        name="Exampleman",
        firstname="Sunnyboy",
        street="Long Avenue 3",
        city_code="11111",
        city="Somewhere";

4.7.6. Reading Table Content

Your table should now look like this:

```
mysql>select * from addresses;
+----+-------------+-----------+---------------+-----------+-------------+
| id | name        | firstname | street        | city_code | city        |
+----+-------------+-----------+---------------+-----------+-------------+
| 1  | Examplemann | Markus    | Long Avenue 3 | 11111     | Somewhere   |
| 2  | Examplewoman | Lisa     | NULL          | 22222     | Watervalley |
| 3  | Exampleman  | Sunnyboy  | Long Avenue 3 | 11111     | Somewhere   |
+----+-------------+-----------+---------------+-----------+-------------+
3 rows in set(0.01sec)
```

You did already use the easiest form of the SELECT commandand are familiar with it.

It's also easy to count the number of rows:

select count(*) from addresses;

You learned already about the WHERE keyword that is used to specify the rows within the table that the command should affect. WHERE can also be used with SELECT to filter out only some special lines that match a condition:

select * from addresses where id=2;

This shows you only the row with ID 2. You can also specify other filters:

select * from addresses where city_code="11111";

select * from addresses where name="Exampleman";

select * from addresses where name="Exampleman" and firstname<>"Markus";

The last command shows you all addresses from family Exampleman but not Markus. The remaining person is only one: Sunnyboy.

You can also search for strings that begin with some characters:

select * from adresses where name like "Example%";

You see now all addresses from persons whose name starts with "Example". It is also possible to search for the end of a String:

select * from adresses where name like "%man";

The percent character is a wildchar for one or more characters. Another wildchar is the underscore (_) that matches only exactly one character.

Like you use the WHERE keyword together with SELECT to limit the output to special lines you can use the WHERE keyword also together with the UPDATE command to affect only special lines.

If you do not use the WHERE keyword, the commands affect the whole table.

When having large tables you may not want to see all columns because the width of your screen is limited. In the last SELECT commands you always used the star (*) to select all columns. Now I like to show you howe to select onyl some columns:

```
mysql>select id,name,city from addresses;
+----+-------------+-------------+
| id | name        | city        |
+----+-------------+-------------+
| 1  | Exampleman  | Somewhere   |
| 2  | Examplewoman | Watervalley |
| 3  | Exampleman  | Somewhere   |
+----+-------------+-------------+
3 rows in set(0.00sec)
```

When you enter SQL commands you should always surround strings and Date/Time values with double quotes ("). Numbers should always written without quotes. The examples on the last pages did you show this.

Before you start wondering about the city_code: This is a string and not a number because you created the table columns as CHAR(5). MySQL can convert numbers to strings and vice versa but not always. By using quotes in the correct way you prevent conversion faults and error messages.

4.7.7. Combine Tables

Now I like to show you how to combine two tables. We entered three addresses but the table with the accounts is still empty. So please fill up thie account table. You know already the commands that you need.

- Lisa has got 6.5 Euro

- Markus has got 18.3 Euro

- Sunnyboy has got 23.1 Euro

Now we enter these data into the second table. We have to set the ID numbers manually so that they match the addresses.

insert into accounts set id=2, value=6.5;

insert into accounts set id=1, value=18.3;

insert into accounts set id=3, value=23.1;

Now you have two tables. One for the addresses of our three persons and one for their accounts.

```
mysql>select * from addresses;
+----+-------------+----------+---------------+-----------+-------------+
| id | name        |firstname |street         | city_code | city        |
+----+-------------+----------+---------------+-----------+-------------+
| 1  | Examplemann |Markus    |Long Avenue 3  | 11111     | Somewhere   |
| 2  | Examplewoman |Lisa     |NULL           | 22222     | Watervalley |
| 3  | Exampleman  |Sunnyboy  |Long Avenue 3  | 11111     | Somewhere   |
+----+-------------+----------+---------------+-----------+-------------+
3 rows in set(0.01sec)
```

```
mysql>select * from accounts;
+----+--------+----------------+
| id | value  | from           |
+----+--------+----------------+
| 1  | 18.3   | 20030302131155 |
| 2  | 6.5    | 20030302131206 |
| 3  | 23.1   | 20030302131230 |
+----+--------+----------------+
3 rows in set(0.00sec)
```

As you can see the MySQL server sets the "from" column automatically, as I explained a I wrote about data types. This column shows you always the timestamp when the row was last modified.

Now we can come to the main topic of this chapter, the combination of two tables.

The most important part is already done - giving rows in two tables, that belong together, a column with the same value. In our case we used the ID column. This is the relationship between both tables. The table with the accounts does not include addresses but using the ID numbers MySQL can find the corresponding address in the other table.

The command that queries two tables together by combining them is quite easy:

```
mysql>select addresses.id,name,firstname,value
->from addresses,accounts
->where addresses.id=accounts.id;

+----+-------------+----------+--------+
| id |name         |firstname | value  |
+----+-------------+----------+--------+
| 1  |Exampleman   |Markus    | 18.3   |
| 2  |Examplewoman |Lisa      | 6.5    |
| 3  |Exampleman   |Sunnyboy  | 23.1   |
+----+-------------+----------+--------+
3 rows in set(0.00sec)
```

After the keyword FROM you enter both table names that you want to combine into one single SQL query.

After the keyword SELECT you enter all the columns that you want to see. As there are two ID columns we have to specify this more detailed. Addresses.id means that we want to see the ID column of the addresses table.

After the keyword WHERE, we need to enter a condition that combines both tables. The condition says, which columns have the same value and combine both tables.

If you don't want to see all accounts you can filter the output as you learned before:

```
select addresses.id,name,firstname,value
      from addresses,accounts
      where addresses.id=accounts.id and name="Exampleman";
```

4.7.8. NULL Values

Plese enter this command for preparation in the SQL client:

update addresses set city_code="" where id=3;

Your table lokks now like this:

```
mysql>select * from addresses;
+----+-------------+-----------+---------------+-----------+-------------+
| id | name        | firstname | street        | city_code | city        |
+----+-------------+-----------+---------------+-----------+-------------+
| 1  | Examplemann | Markus    | Long Avenue 3 | 11111     | Somewhere   |
| 2  | Examplewoman| Lisa      | NULL          | 22222     | Watervalley |
| 3  | Exampleman  | Sunnyboy  | Long Avenue 3 |           | Somewhere   |
+----+-------------+-----------+---------------+-----------+-------------+
3 rows in set(0.01sec)
```

Please note that Lisa has no street and Sunnyboy has no city_code. The important difference is that the unset street of Lisa is displayed as NULL while the empty city_code of Sunnyboy is displayed as an empty string.

In SQL "nothing" is not the same as "empty string". If a field is not set, the SQL client shows NULL. If the field contains an empty string, it is shown as an empty field. An empty string is a string of zero characters - the shortest possible string, it has no characters but it is still a string.

The difference is more important when you run the SELECT command in this way:

```
mysql>select * from addresses wherestreet="";
Empty set(0.00sec)
```

```
mysql>select * from addresses where street is NULL;
+----+-------------+-----------+---------------+-----------+-------------+
| id | name        | firstname | street        | city_code | city        |
+----+-------------+-----------+---------------+-----------+-------------+
| 2  | Examplewoman| Lisa      | NULL          | 22222     | Watervalley |
+----+-------------+-----------+---------------+-----------+-------------+
1 row in set(0.00sec)
```

There is no row where the street is an empty string ("") but there is one row where the street is not set (NULL).

Please note that the keyword NULL is slightly different used than in comparision expressions with "=" character.

select * from addresses where street is NULL;

select * from addresses where street is not NULL;

A small note: If you want tu unset a field that is already set, you need to assign the value NULL to it:

update addresses set city_code=NULL where id=3;

4.7.9. More Keywords For SELECT

If you have large tables the keywords ORDER BY and LIMIT can be interesting.

ORDER sorts the output. Example:

```
mysql>select * from addresses order by city_code;
+----+-------------+-----------+---------------+-----------+-------------+
| id | name        | firstname | street        | city_code | city        |
+----+-------------+-----------+---------------+-----------+-------------+
| 3  | Exampleman  | Sunnyboy  | Long Avenue 3 | NULL      | Somewhere   |
| 1  | Examplemann | Markus    | Long Avenue 3 | 11111     | Somewhere   |
| 2  | Examplewoman| Lisa      | NULL          | 22222     | Watervalley |
+----+-------------+-----------+---------------+-----------+-------------+
3 rows in set(0.01sec)
```

The output is now sorted by city_code. Please use the ORDER function only if you really need it, because it can take a long time and use much memory on large tables.

Sorting backwards is also possible:

select * from addresses order by city desc;

Sometimes you may want to limit the sorted output by some lines. To see the row with the lowest city_code you can enter:

```
mysql>select * from addresses order by city_code limit 1;
+----+-------------+-----------+---------------+-----------+-------------+
| id | name        | firstname | street        | city_code | city        |
+----+-------------+-----------+---------------+-----------+-------------+
| 3  | Exampleman  | Sunnyboy  | Long Avenue 3 | NULL      | Somewhere   |
+----+-------------+-----------+---------------+-----------+-------------+
1 row in set(0.01sec)
```

The keyword LIMIT specifies how many rows you want to see at maximum. In this example we queried only one row.

4.7.10. Modifying Columns

The structure of a SQL table is modify-able. You can change, add and remove columns.

Lets add two columns for gender and birthday.

```
mysql>alter table addresses
->add gender CHAR(1),
->add birthdate DATE;

Query OK,3 rows affected( 0.00sec)
Records:3 Duplicates:0 Warnings:0
```

```
mysql>select * from addresses;
+----+-------------+----------+---------------+----------+-------------+--------+-----------+
| id | name        |firstname | street        | city_code|city         | gender | birthdate |
+----+-------------+----------+---------------+----------+-------------+--------+-----------+
| 1  | Exampleman  | Markus   | Long Avenue 3 | 11111    |Somewhere    | NULL   | NULL      |
| 2  | Examplewoman| Lisa     | NULL          | 22222    |Watervalley  | NULL   | NULL      |
| 3  | Exampleman  | Sunnyboy | Long Avenue 3 | NULL     |Somewhere    | NULL   | NULL      |
+----+-------------+----------+---------------+----------+-------------+--------+-----------+
3 rows ins et(0.00sec)
```

Now you added two columns. They are empty.

Lets say you find out that some street names re longer than 30 characters, then you need to make this columns larger. The advantage of MySQL is that you will not loose the existing street names when modifying the columns (whenever possible):

alter table addresses modify street char(50);

Now you can enter longer street names than before.

Now I like to show you how to delete a column in the table:

```
mysql>alter table addresses drop gender;
Query OK,3 rows affected( 0.00sec)
Records:3 Duplicates:0 Warnings:0
```

```
mysql>select * from addresses;
+----+-------------+----------+---------------+----------+-------------+-----------+
| id | name        | firstname | street       | city_code| city        | birthdate |
+----+-------------+----------+---------------+----------+-------------+-----------+
| 1  | Exampleman  | Markus   | Long Avenue 3 | 11111    | Somewhere   | NULL      |
| 2  | Examplewoman| Lisa     | NULL          | 22222    | Watervalley | NULL      |
| 3  | Exampleman  | Sunnyboy | Long Avenue 3 | NULL     | Somewhere   | NULL      |
+----+-------------+----------+---------------+----------+-------------+-----------+
3 rows in set(0.00sec)
```

4.7.11. Deleting Data

You saw how to delete columns on the previous page. The column gender is now dropped.

To delete a row you enter:

```
mysql>delete from addresses where id=3;
Query OK,1 row affected(0.00sec)


mysql>select * from addresses;
+----+-------------+-----------+--------------+-----------+-------------+-----------+
| id | name        | firstname | street       | city_code | city        | birthdate |
+----+-------------+-----------+--------------+-----------+-------------+-----------+
| 1  | Exampleman  | Markus    | Long Avenue 3 | 11111    | Somewhere   | NULL      |
| 2  | Examplewoman | Lisa     | NULL         |22222      | Watervalley | NULL      |
+----+-------------+-----------+--------------+-----------+-------------+-----------+
2 rows in set(0.00sec)
```

Please do not forget the keyword WHERE because otherwise the command would delete all rows!

At last I show you how to delete the whole tables and the database:

drop table addresses;

drop table accounts;

drop database mytest;

Now all the data and tables that you entered while reading this chapter are gone.

4.7.12. SQL Queries In Shell Scripts

Im sure that you want to use MySQL for more than simply reading the table content on the screen. Sometime you want create lists in text files or simply query a single field.

If you entered the last three commands from the previous page you have no mytest database any more that we can use to test the commands. So please create a new database:

create database demo;

use demo;

create table testtable (
 id INTEGER AUTO_INCREMENT PRIMARY KEY NOT NULL,
 value INTEGER);

insert into testtable set value=10;
insert into testtable set value=13;
insert into testtable set value=19;
insert into testtable set value=21;
insert into testtable set value=57;

Now you have a small demo table with the following content:

```
mysql>select * from testtable;
+----+------+
| id | value|
+----+------+
| 1  | 10   |
| 2  | 13   |
| 3  | 19   |
| 4  | 21   |
| 5  | 57   |
+----+------+
5 rows in set(0.00sec)
```

You can enter the same SQL command in non-interactive mode at the command line or in a shell script. Please enter exit now to exit the MySQL client program and return to the shell. Then enter this command:

mysql -u root -p'xxxxx' -D demo -e 'select * from testtable;'

Please note that there is no space between -p and the password (xxxxx). Replace xxxxx with your password that you set during installation of MySQL. The option -D specifies the database to be used and -e specifies the SQL command to be executed.

The result is the same table as shown above and the MySQL client program exits itself.

By adding the option -B you switch the frames off and -N switches the table header off.

mysql -u root -p'xxxxx'-D demo -N -B -e'select * from testtable;'

```
1          10
2          13
3          19
4          21
5          57
```

You get this simple output that you can easily read in with other programs (e.g. Microsoft Excel and kSpread). What you cannot see in this book is that the fields are separated by tab characters. This is exactly the best format for spreadsheet applications.

You can redirect the output into a file using the ">" character:

mysql -u root -p'xxxxx' -D demo -N -B -e 'select * from testtable;' > /tmp/test.txt

The option -H creates a table in HTML text instead of plain text, e.g.:

```
<TABLE BORDER=1>
    <TR>    <TH>id</TH>  <TH>value</TH>   </TR>
    <TR>    <TD>1</TD>   <TD>10</TD>      </TR>
    <TR>    <TD>2</TD>   <TD>13</TD>      </TR>
    <TR>    <TD>3</TD>   <TD>19</TD>      </TR>
    <TR>    <TD>4</TD>   <TD>21</TD>      </TR>
    <TR>    <TD>5</TD>   <TD>57</TD>      </TR>
</TABLE>
```

You can combine -H and -N if you want a bordered table without column headers.

To query only a single value and store it into a variable you can use this command:

```
thevalue=`mysql -u root -p'xxxxx' -D demo -N -B -e 'select value from testtable where id=1;'`
echo The value is $thevalue
```

Take care about correct quotes because there are three similar characters and each of them has another function. The example above works only with SELECT commands that query a single value. If the command would answer with many rows, you would get all of them in one single variable and that is not useful.

4.7.13. SQL Backup

This is the last chapter about MySQL but it is important.

MySQL offers different methods to backup the data. I like to show you two of them:

1. Cold backup

Cold backups means to backup the data while MySQL is stopped. No program can access the database while you run a cold backup.

- Stop MySQL server.

- Copy the files *.frm, *.MYD and *.MYI onto another media, for example a tape.

- Restart the MYSQL server.

2. Hot Backup

The hot backup allows you to backup tables while they can be accessed by other programs. Unfortunately these programs have only read access while the table is backed up. Other programs that want to write into a table must wait until the backup of this special table has finished.

- Enter the command **backup table demo.testtable to "/tmp"** in the mysql client program. This creates new files in /tmp with the content of the table.

- Copy the files /tmp/*.frm and /tmp/*.MYD onto another media, for example a tape.

To restore the table you need to enter the command **restore table demo.testtable from "/tmp"**. Restoring a table requires that it does not already exist in the database. Maybe you need to delete it first.

Some hints about hot backups:

During a hot backup the table can only be read by other programs. Write attempts are queued until the backup into the temporary file has completed. This can cause performance problems on large tables (of many megabytes).

Imagine you had a service that needs to run 24 hours and the database stores billing data. These data are collected monthly for a bill.

Customer-Nr.	Date	Service	Value
10001	01.03.03	SMS	0.49
10001	01.03.03	SMS	0.49
10001	02.03.03	SMS	0.49
10001	03.03.03	SMS	0.49
10001	05.03.03	SMS	0.49
10002	01.03.03	SMS	0.49
10002	02.03.03	SMS	0.49
10002	07.03.03	SMS	0.49

...

For every single service that you delivered you add one row to this table. That means the table must always be ready for write access.

Now you cannot make backups because that would disable write access for some seconds. And if the table becomes larger it may take minutes or hours. It is not acceptable to wait until the backup has finished.

Therefore you need to keep tables as small as possible to make the backup as fast as possible. Small tables can be backed up faster and a delay of some seconds is surely more acceptable than some minutes.

The solution is to split the table into two.

The first table stores the actual billing data until the bill is printed. The second table stores the old billing data that are already printed.

During some years the second table will grow more and more but the first table does not grow. Therefore you can backup the small table that needs to be accesses as fast as possible in a short time. The backup of the second larger table may take longer but this table is not time-critical.

To copy one table into another use this command:

insert into table2 select * from table1;

or

insert into table2 select * from table1 where date < "2003-04-01" ;

Another version of this command is useful when the destination table does not already exist:

create table table2 select * from table1;

or

create table table2 select * from table1 where date < "2003-04-01" ;

4.8. Dynamic Websites With PHP

There are many script languages to program dynamic websites. One of them is PHP and I like to show it.

You will learn about the most important programming rules and commands of PHP. After reading this chapter you will be able to write your own PHP applications and extend your knowledge by reading the original manual of PHP.

PHP is a powerful and language, therefore I recommend to look into the original manual - specially the function list. It's easy to read if you understand the basics.

4.8.1. The First Website

The reminder lists of the operating systems earlier in this book show you, where the configuration files and websites are stored. On the following pages I write only about htdocs directory and httpd.conf configuration file. If necessary go back to the operation system specific chapter to find out where these files/directories are located.

If you do not want the example htdocs directory for your own websites you need to create an empty directory for them. Then open the file httpd.conf in a text editor and enter the new directory name. The two interesting lines are:

```
DocumentRoot "/usr/local/apache/htdocs"
<Directory "/usr/local/apache/htdocs">
```

You need to restart apache to activate the changes.

Now please create a new text file and save it as test.hml in your website directory:

```
<html><body>
This is my first HTML page.
</body></html>
```

You can also use Netscape Composer, Qanta Plus or whatever editor you like. In this book we use only a simple text editor because this is operating system independent and helps you to try the examples 1:1 exactly as written here.

To learn about HTML I recommend to visit a local book store. The have surely a lot of books about HTML and it does not make sense to include a chapter about HTML in this book.

Now please check if you can load this webseite using any webbrowser. Enter the URL **http://localhost/test.html**. If your webserver is not the same PC than the web browser then replace localhost by the servers name.

If nothing appears, check if the webserver is running. For example by entering the command:

ps -ef | grep httpd

```
root       1058       1  0 20:13 ?          00:00:00 /usr/local/apache/bin/httpd
nobody     1073    1058  0 20:13 ?          00:00:00 /usr/local/apache/bin/httpd
nobody     1074    1058  0 20:13 ?          00:00:00 /usr/local/apache/bin/httpd
nobody     1075    1058  0 20:13 ?          00:00:00 /usr/local/apache/bin/httpd
nobody     1076    1058  0 20:13 ?          00:00:00 /usr/local/apache/bin/httpd
nobody     1077    1058  0 20:13 ?          00:00:00 /usr/local/apache/bin/httpd
stefan     5896    5865  0 20:28 pts/1      00:00:00 grep httpd
```

The last line with the grep command does not count. All the other lines show you that some processes of the Apache webserver are running. If you don't see them then Apache is not running.

In case of problems, a look into the error logfile could also help. I wrote their location in the reminder list.

If you see an error message like this in the webbrowser:

```
Forbidden
You don't have permission to access /test.html on this server.
```

then the directory or file has not enough permissions. All users (including other) need to read the file, the directory and all directories above this one.

4.8.2. The First Dynamic Website

Before you continue please ensure that the example of the previous page works on your computer.

Now create the file test2.php in the htdocs directory with this content:

```
<html><body>
This is my first dynamic website with PHP code.<p>
<?php
            print("Todays date is ".strftime("%d.%m.%Y").
                        " and the time is ".strftime("%H:%M:%S"));
?>
</body></html>
```

Dont waste your time trying to understand the commands above. You will learn them later. At the moment it is only important that this example works and that you understand that the generated content of the website is dynamic.

Open this website in your browser by entering the URL **http://localhost/test2.php**. Again you may replace localhost by the name of your webserver. You should see this:

```
This is my first dynamic website with PHP Code.
Todays date is 10.8.2003 and the time is 20:18:35.
```

PHP programs are embedded in the HTML code. The first line was created by static HTML code like in the example of the previous page.The second line was created by a dynamic PHP program.

PHP programs are marked by two things:

- By the file name extension .php instead of .html

- By <?php and ?> around the program code

You can insert many PHP programs into one single HTML file. They are executed in the order of their occurence.

```
<html><body>
This is my first dynamic website with PHP code.<p>
Todays date is <?php print(strftime("%d.%m.%Y")); ?>
and the time is <?php print(strftime("%H:%M:%S")); ?>
</body></html>
```

Save this file as test3.php. This program creates the same result as the one above.

4.8.3. print(), printf() And sprintf()

In the previous chapter you used the print command to create output.

print("Hello");

creates the output

```
        Hello
```

and

print("Hello\nWorld");

creates the output

```
        Hello
        World
```

Please note that some webbrowsers remove all line breaks when displaying HTML text. You can open the page sourcecode by some menu command in the webbrowser to see the line break.

The \n is a special control characters that means "Line break". Other often used control characters are:

\\ if you want to output a single \ character

\" if you want to output a double colon

Print() outputs simple text. With printf() you can output formatted values, this is useful in combination with numeric values in variables.

$day=1;
$month=4;
$year=2003;

printf("The date is %d.%d.%d ",$day,$month,$year);

```
        The date is 1.4.2003
```

The first argument to printf() is a text that contains one or more placeholders for variable content. The next arguments are the variables. Their number and type must match the placeholders.

Often used placeholders are:

%d	decimal number
%i	also decimal number
%f	floating number
%s	string
%x	hexadecimal number with small letters
%X	hexadecimal number with capital letters

If you want to print the % character you need to write it as double percent (%%) then it is not treated as a placeholder.

If you put a number in front of a placeholder and this number starts with 0 ,then the value is displayed with the specified width, filled with zeroes at the left side. Values that are to large are displayed in their full length.

$day=1;
$month=4;
$year=2003;

printf("The date is %02d.%02d.04%d ",$day,$month,$year);

```
        The date is 01.04.2003
```

For floating numbers you can specify the number of digits before and after the dot.

$value1=138.45865;
$value2=1.2;
**printf("The value2 is %02.2f
",$value1);**
printf("The value2 is %02.2f ",$value2);

```
        The value1 is 138.45
        The value2 is 01.20
```

Printf() displays always all digits before the dot, also if they are to many. The digits after the dot are always truncated and mathematically rounded if they are to many.

The example above insert an <br> to indicate a line break in the webbrowser.

The command sprintf() is similar to printf. The only difference is that sprintf() writes into a variable and printf() writes to the screen.

$value=123;
$result=sprintf("The value is %d",$value);

printf($result);

```
        The value is 123
```

4.8.4. Variables And Data-Types

You saw already some variables in the last chapter. Variables store

- boolean vales
- integer numbers
- floating numbers
- character strings
- arrays
- NULL

Variables can also store objects and references but I will not explain them in this book. Now I explain the data types above.

Boolean

Boolean values can only be TRUE or FALSE. You can write these keyword in capital or small letters – whatever you prefer:

```
$value1=true;
$value2=false;
```

Integer

Integer values can be written as decimal, octal or hexadecimal. Octal numbers start always with a zero. Hexadecimal numbers start with 0x. Decimal numbers start with any other digit. The following examples have all the same value:

```
$value1=123;
$value2=0173;
$value3=0x7B;
```

Floating

Floating numbers are written with a dot or in exponential way:

```
$value1 = 1234.56;
$value2 = 12.3456e2;

$value3 = -0.345;
$value4 = 345e-3;
```

Strings

Strings are written between double or single quotes:

```
$text1 = 'Hello, this is Stefan\'s text';
$text2 = "The first text was : $text1"
```

The difference between single and double quotes is that double quotes allow you to insert variables into the text while single quotes do not allow this.

If you want to write a single quote into the text you need to precede it by a backslash, otherwise the PHP language interpreter would misinterpret this character as the end of string.

Arrays

Arrays are lists of one or more values of the same type. They have an index that allows you access each single element of the array.

```
$name[1] = "Miriam";
$name[2] = "Rolf";
$name[3] = "Mucki";
```

Index values can also be zero and negative. Also strings are allowed as indexes:

```
$MotherOf["Miriam"] = "Petra";
$MotherOf["Rolf"] = "Heidi";
$MotherOf["Mucki"] = "Micki";
```

NULL

Null is a special value for variables that have no value or are cleared with the unset() command. You know NULL already from the MySQL chapter.

As you can see in all the examples above, variable names start with a dollar character followed by at least one letter. After the first letter you can also use digits and the underscore. PHP treats variable names case sensitive. Therefore $Test and $test are two different variables.

4.8.5. Operators

Operators calculate or compare values. If you connect an operand and an operator you get an expression. Every expression gives a string, value or boolean as its result.

Arithmetic operators

+	addition **1+2** returns 3
-	subtraction **5-2** returns 3
*	multiplication **3*4** returns 12
/	division **12/2** returns 6
%	modulus, the rest of a division **12%5** returns 2

Bit operators

&	and, only bits that are set in both operands are set in the result **11 & 7** returns 6
|	or, all bits that are set in one or both operands are set in the result **4|2** returns 6
^	exclusive or, only bits that are set in one but not both operands are in the result **6^4** returns 2
<<	shift to left, all bits from the first operand are shifted left by the number of positions given by the second operand **4<<1** returns 8
>>	shift to right, all bits from the first operand are shifted right by the number of positions given by the second operand **4>>1** returns 2

Compare operators

==	equal **3==3** returns true
!=	not equal **3!=4** returns true
<>	not equal, see above
<	smaller than **3<4** returns true
>	greater than **4>3** returns true
<=	smaller or equal than **3<=3** returns true
>=	larger or equan than **4>=4** returns true

Logical operators

and	and **true and true** returns true
&&	and see above
or	or

true or false returns true

|| or
                          see above

xor   exclusive or
                        true xor true returns false
                        rue xor false returns true

!   negation
                        !true returns false

The assignment operator is =. Its used to assign al value to a variable:

```
$result=3+4;
```

Some arithmetic and bit operators can combined. For example

```
$a = 3;
$a += 4;    returns 7
```

The second line is the same as

```
$a=$a+4;
```

The variable on the left side is destination and operand.

You can run external programs and store their text output into a variable

```
$result = `ls -l` ;
```

The ls command lists the current directory content. The directory is written into the variable $result. Take care to write the correct quotes. These are backquotes, not normal quotes.

Strings can be concatenated by the dort operator:

```
"Hallo "."Maxi"                      returns "Hallo Maxi"
```

4.8.6. Control Structures

Programs are normally executed from the first line until the last line in linear order. Control structures. Control structures are used to change this linear execution order.

Control structures allow you to repeat a part of the code or execute a part only if an expression is true or nor zero.

Program blocks are marked by bracelets "{" and "}". You will find some examples in the next chapters.

4.8.6.1. if...elseif...else

If, elseif and else are used to execute a program block only if an expression is true or not zero. You need to write the condition in parentheses. "(" and ")".

```
if ($a > 3)
{
    Commands;
    Commands;
    Commands;
}
```

You cann add an else block that is executed when the expression is not true.

```
if ($a >3)
{
    Commands;
    Commands;
    Commands;
}
else
{
    Other commands;
    Other commands;
    Other Commands;
}
```

Elseif is used to construct an if/else control structure with more than one expression. If the first expression is not true, then the seconds is calculated. If this is also not true, then the else-block is executed.

```
if ($a < 0)
{
    print("$a is negative.");
}
elseif ($a > 0)
{
    print("$a is positive.");
}
else
{
    print("$a is not negative and not positive. It is null.");
}
```

You do not need to use so many line breaks as I did in the examples above. The next example is also correct.

```
if ($a < 0) {
    print("$a is negative.");
} elseif ($a > 0) {
    print("$a is positive.");
} else {
  print("$a is not negative and not positive. It is null.");
}
```

If you want, you can also write everything into one single line but that will not look clear. However it will surely work.

4.8.6.2. switch...case

If you like to combine a lot of if-expressions to check a variable for values, the program code will quickly become hardly readable. A bad example is this:

```
if ($name=="Maria")
{
    Commands_for_Maria;
}
elseif ($name=="Josef")
{
    Commands_for_Josef;
}
elseif ($name=="Martin")
{
    Commands_for Martin;
}
elseif ($name=="Kerstin")
{
    Commands_for_Kerstin;
}
else
{
    print("Unknown Name!");
}
```

PHP offers another way to construct such large commands. You may prefer the switch...case command:

```
switch ($name):
    case "Maria":
        Commands_for_Maria;
        break;
    case "Maria":
        Commands_for_Josef;
        break;
    case "Maria":
        Commands_for_Martin;
        break;
    case "Maria":
        Commands_for_Kerstin;
        break;
    default:
        printf("Unknown Name!")
endswitch
```

Do not forget the break commands, otherwise all following commands will be executed accidentally. The switch...case command is an alternative to if...then...else. Decide yourself what command you prefer in what situation.

4.8.6.3. while And do...while

While is used to execute a program part as long as an expression is true or does not result 0.

After the keyword while you need to write an expression in parentheses "(" and ")". The expression is checked before every repetition.

```
while ($a < 3)
{
   Commands;
   Commands;
   Commands;
}
```

Another version of the while loop is the do...while loop. Again the program part is executed many times as long an expression is true or not 0. But now the expression is checked **after** any execution and not before.

```
do
{
   Commands;
   Commands;
   Commands;
}
while ($a < 3);
```

4.8.6.4. for

The for loop is used to execute a program part for a specified number of repetitions. It also counts the repetitions.

```
for ($i=1; i<=6; $i=$i+1)
{
    print("i is now $i <br>");
}
```

This loop creates the output

```
i is now 1
i is now 2
i is now 3
i is now 4
i is now 5
i is now 6
```

The loop was executed six times. The counter variable was counted from1 up to 6.

After the for command you need to enter three commands or expressions, enclosed in parantheses and separated by semicolons.

```
for (initialisation; expression; modification)
```

The initialisation command is used to assign a start value to the counter variable.

The second expression is used to check if the end value is reached. This check is performed **before** every execution of the program part. If the end value is reached, the for loop ends. The loop executes as long the expression returns true.

The modification command is used to modify the counter variable **after** any loop repetition. Typically you will add 1 to the variable but you could also increase it in larger steps. It is also possible to decrease the variable until a low limit is reached.

The for loop is another version of the while loop, if you think a little bit about their differences.

```
for ($i=1; i<=6; $i=$i+1)
{
    Commands;
}
```

is the same as

```
$i=1;
while ($i<=6)
{
    Commands;
    $i=$i+1;
}
```

4.8.6.5. foreach

The foreach loop is used to do something with all elements of an array.

```
$names[1]="Mary";
$names[2]="Hans";
$names[3]="Maxi";
$names[4]="Hannes";

foreach ($names as $n) {
    print("The name is $n <br>");
}
```

This produces the output

```
The name is Mary
The name is Hans
The name is Maxi
The name is Hannes
```

There is also an extended version of the foreach loop available that can be used to query the array index.

```
$month["january"]="cold";
$month["april"]="unstable";
$month["july"]="warm";
```

```
foreach ($month as $index => $value) {
   print("The  $index is $value <br>");
}
```

This produces the output

```
        The january is cold
        The april is unstable
        The july is warm
```

4.8.6.6. break, continue And exit

A loop can be escaped by using the break command.

```
$pears=0;

while ($pears < 1000)
{
   print("We add another pear<br>");

  if ($pears=12)
  {
     $bad=true;
  }

  if ($bad == true)
  {
     break;
  }

  $pears=$pears+1;
}
print("We have no either 1000 pears or we stopped because one was bad.");
```

This loop adds one pear until 1000 pears are collected. But if a pear is bad the the loop stops. To make the example as easy as possible the 12th pear is always marked ad bad.

The continue command enforces the next loop repetition even if the actual execution is not finished to its end.

```
$pears=0;
$try=0;

while ($try < 1000)
{
   $try=$try+1;

  if ($bad == true)
  {
     print("The pear no. $try is bad <br>");
     continue;
  }

  print("I take the pear no. $try, hmm smells good.<br>");
   $pears=$pears+1;
}
print("Now I took 1000 pears or I skipped some bad fruits.<br>");
print("$pears fruits were good");
```

This loop tries 1000 pears. When it gets a bad fruit, it skips this one and does not count it. The remaining commands

```
   print("I take the pear no. $try, hmm smells good.<br>");
   $pears=$pears+1;
```

are not executed for bad fruits because the continue command enforces the next loop repetition ignoring that the loop is not executed until its end at this point.

The variable $try is increased for every loop repetition but the variable $pears is only executed for good fruits.

continue versus break

The break commend escapes a loop. The next commands after the loop will be executed.

The continue command enforces the next repetition of a loop and breaks the current repetition.

Break and continue are also usable in do...while loops and for loops.

There is also another command to break loops and programs. The name is **exit**. This command stops the whole PHP program execution.

4.8.7. include

The include command loads another file with PHP code into memory. For example:

The file part1.include has the content:

```
$a=2;
$b=3;
```

And the file test4.php has the content:

```
<html><body>
<?php
    include 'part1.include';
    print("The variable a is $a and the variable b is $b");
?>
</body></html>
```

This is very useful. You can collect variables and functions in one include-file and share them to many other php files.

4.8.8. Functions

Functions are small or large program parts that you can execute later like PHP commands. They are used to split the program into many small parts (functions) to make the program code better readable. Example:

```
function calculate($value1, $value2)
{
    $result = $value1 + $value2 * 24;
    return $result;
}

$super = calculate(3,4);
print("The calculation of 3 and 4 returns $super <br>");

$cool = calculate(9,12);
print("The calculation of 9 and 12 returns $cool <br>");

print("Or another way " . calculate(3,2) . "<br>");

printf("Or like this %i <br>", calculate(1,2));
```

The example above uses the function calculate() four times to calculate different numbers. The function returns its result to the main program using the return command.

```
The calculation of 3 and 4 returns 99
The calculation of 9 and 12 returns 297
Or another way 51
Or like this 49
```

Now another example, this time with a function that does something but does not return a result.

```
function error_message($text)
{
    print("An error occurred: <br>");
    print("$text <br>");
    print("The program will now stop <br>");
    exit;
}

commands;
commands;
commands;
error_message("The harddisk is full");
commands;
commands;
commands;
error_message("Text not found");
commands;
commands;
commands;
error_message("Wrong value, 0 is not allowed.");
```

This uses the function many times to output an error message and stop the program. The main program remains clearly readable and its not necessary to write the code for the error message many times. The output of this PHP code would look like this:

```
An error occurred:
The harddisk is full
The program will now stop
```

4.8.9. Form Variables And File Uploads

Websites can contain forms. When the user fill the field of a form and clicks on the submit button, the values will be sent to the webserver.

In PHP programs you will find the values of the fields in the following arrays:

- $_POST

- $_GET

- $_FILES

- $_COOKIE

The design of the form specifies in which array the values will be stored. Lets say you have this form (test5.html):

```
<html><body>
    <form action="test5.php" method="post">
        Name: <input type="text" name="username"><br>
        <input type="submit">
    </form>
</body></html>
```

The webbrowser will show a simple form with one text field named "username" and one submit button.

The value will be sent to the PHP program test5.php when the user clicks the submit button. The variable $_POST["username"] will contain the field value.

You can read the variable, for example with this script (test5.php):

```
<html><body>
<?php
    printf("You entered %s . Thanks", $_POST["username"]);
?>
</body></html>
```

You can also send the form with get method. The difference is that the URL row of the browser shows all get variables but no put variable, for example:

```
http://localhost/test5.php?username=Stefan
```

File uploads work like this example (test6.html):

```
<html><body>
<form method="post" enctype="multipart/form-data" action="test6.php">
    <INPUT TYPE="hidden" name="MAX_FILE_SIZE" value="1000">
    <input type="file" name="userfile">
        <input type="submit">
    </form>
</body></html>
```

The webbrowser checks the file size using the MAX_FILE_SIZE value and does not allow sending if the file is to large.

$_FILES["userfile"]["name"]	contains the original filename
$_FILES["userfile"]["size"]	contains the file size in bytes
$_FILES["userfile"]["tmp_name"]	contains a temporary filename that the webserver gave to store the file onto the harddisk of the webserver.
$_FILES["userfile"]["error"]	contains an error code. 0=no error

You cannot be sure that the webserver really checks the filesize. This is a recommended standard but broken webbrowsers could send larger files. Your program must detect this and output an error message in this case.

An example, how you can receive uploaded files (test6.php):

```
<html><body>
<?php
```

```
        if (is_uploaded_file($_FILES["userfile"]["tmp_name"]))
        {
            copy($_FILES["userfile"]["tmp_name"],
                "/dokumente/uloaded_file.jpg");
            print("Thanks, the file is saved.");
        }
        else
        {
            print("You did not really send the file!");
        }
    ?>
    </body></html>
```

The function is_uploaded_file checks, if the file was really sent. You can use this check to avoid hacker attacks. Without this check hackers could send wrong filenames that may stop your server.

The function copy copies the temporary file into another new file. You could also do any other action with the temporary file at this point.

If you need the file later you have to copy it to another file because the webserver deletes the temporary file immediately when the PHP program ends.

You can read the original PHP manual to learn how to use cookies. I recommend not to use cookies because the number of Internet users that disable cookies in their webbrowser increases permanently. Therefore you cannot be sure that all visitors of your websites can (and want) use cookies.

4.8.10. SQL queries

SQL Queries are very easy within PHP programs. The following example shows you how to use a select query and displays the result as a HTML table (test7.php).

```
<html><body>
This is the result of a SQL query: <br>
<?php
mysql_connect("localhost","root","my_passwort");
$result=mysql_db_query("database","select * from table_name")
if ($result)
{
  $rows = mysql_num_rows($result);
  $columns = mysql_num_fields($result);
  print("<table border=1>\n");
  print("  <tr>\n");
  for ($i=0; $i<$columns; $i++)
  {
    $name=mysql_field_name($result,$i);
    printf("    <th> %s </th>\n",$name);
  }
  print("  </tr>\n");
  for ($j=0; $j<$rows; $j++)
  {
    print("  <tr>\n");
    $row=mysql_fetch_row($result);
    for ($i=0; $i<$columns; $i++)
    {
      printf("    <td> %s </td>\n",$row[$i]);
    }
    print("  </tr>\n");
  }
  print("</table>\n");
}
else
{
  printf("The following error occured: <br> %s",mysql_error());
}
?>
</body></html>
```

Now I like to explain this program line by line:

```
<html><body>
This is the result of a SQL query: <br>
```

This is HTML code. It outputs a label for the table, followed by a line break.

```
<?php
```

This marks the begin of PHP code.

```
mysql_connect("localhost","root","my_passwort");
```

We connect to a SQL server on the own computer (localhost), login with username and password. Change this to your server name, username and password.

```
$result=mysql_db_query("database","select * from table_name")
```

With mysql_db_query we send a SQL command to the server and check if it was successful. The variable $result contains an identification number that we need later to access the result. If the command failed, the variable is NULL.

```
if ($result)
{
...
}
else
{
  printf("The following error occured: <br> %s",mysql_error());
}
```

This control structure checks if $result contains a value. If it is NULL, then the else part is executed and outputs an error message. The function mysql_err() returns the error text of the last failed SQL command.

Instead of **if ($result)** we could also write **if ($result != 0)**. Both versions do the same.

The program part after **if** is only executed if the expression returns TRUE or is not NULL. Also using simply the variable name is a valid expression.

```
$rows = mysql_num_rows($result);
$columns = mysql_num_fields($result);
```

With both functions we ask for the number of rows and columns. Only select commands return a table. All other SQL commands are simply successful or not.

```
for ($i=0; $i<$columns; $i++)
 {
   $name=mysql_field_name($result,$i);
   printf("    <th> %s </th>\n",$name);
 }
```

This loop outputs the names of all columns. It is important to know that the first collums has always the number 0. Therefore the for loop starts counting with 0. The function mysql_field_name() returns the name of the column and we output this with a printf() command.

```
for ($j=0; $j<$rows; $j++)
 {
   print("  <tr>\n");
   $row=mysql_fetch_row($result);
   for ($i=0; $i<$columns; $i++)
   {
     printf("    <td> %s </td>\n",$row[$i]);
   }
   print("  </tr>\n");
 }
```

Here I combined two loops because we want to read out all rows and all columns within these rows. Also the row numbers start with 0, therefore both loops start counting with 0. The print() and printf() commands output the values.

The HTML output created by this PHP program looks like this:

```
<html><body>
Dies ist das Ergebnis einer SQL Abfrage: <br>
<table border=1>
  <tr>
    <th> a </th>
    <th> b </th>
    <th> c </th>
  </tr>
  <tr>
    <td> 1 </td>
    <td> 2 </td>
    <td> 3 </td>
  </tr>
  <tr>
    <td> 4 </td>
    <td> 5 </td>
    <td> 6 </td>
  </tr>
</table>
</body></html>
```

The table has three columns with the names a, b and c. The rows have the numbers 1 to 6 because this is the content of the table that I read in this example. If you use another table you get other column names and other values.

Read this example carefully and try to understand it. Play a little bit around by changing the output format or show more informations about the SQL table. For example you could show the number of rows and columns. Try to show the result in a different format, not a HTML table. Create a HTML form that allows the user to enter any SQL command and see the result. You will need the variable $_POST["fieldname"] instead of a fixed SQL command.

Another good idea is to create a set of HTML forms and PHP scripts that allows you to view and edit an postal address database.

For creation of dynamic websites it is important to understand, how SQL queries and forms are used with PHP.

4.8.11. Local And Global Variables

Variables do not have only a name, they have only a validity area. There are two such areas:

- local variables

- global variables

All variables that you use within a function are local. Variables used outside any function (in the main program part) are global. One example (test8.php):

```
<html><body>
<?php
    function test1()
    {
        $a = "Two"
        print("The local value in test1 is $a <br>");
    }
    function test2()
    {
        print("The local value in test2 is $a <br>");
    }
    $a = "One";
    print("The global value is $a <br>");
    test1();
    test2();
    print("The global value is still $a <br>");
?>
</body></html>
```

Run this program and test it. The output will look like this:

```
The global value is One
The local value in test1 is Two
The local value in test2 is
The global value is still One
```

The last two lines are interesting. The third line shows you that the function test2 has its own variable compared two function test1. Within test2 it has no value because nobody set it.

The fourth line shows you that the variable in the main program has still its old value and was not changed by the function test1. The assignment $a="Two" changed the local variable in test1 but not the global variable.

As you can see, the local and global variables are completely separated. They have the same name, but they are really different variables. And two local variables in different functions are also different variables, even if they have the same name.

The last chapter showed you how to pass a variable to a function so that it also knows the global content. You need to place the variable name between the parentheses, like in test9.php:

```
<html><body>
<?php
    function test($a)
    {
        print("The value in test is $a <br>");
        $wert="two"
        print("The new value in test is $a <br>");
    }
    $a = "One";
    print("The global value is $a <br>");
    test($a);
    printf("The global value is still $a <br>");
?>
</body></html>
```

This program creates the output:

```
The global value is One
The valie in test is One
The new value in test is Two
The global value is still One.
```

The **value** of variable $a is used as an argument of the function, not the variable itself. Because the function changed $a but the main program saw still its old value. Again $a is a local variable within the function and a global variable in the main program. Both are different variables.

Variables passed to functions can be used there, but changes are not visible in the main program because these are two different variables with copied content.

Now please look at this example (test10.php):

```
<html><body>
<?php
    function test($wert)
    {
        print("The value in test is $wert <br>");
        $wert="Two"
        print("The new value in test is $wert <br>");
    }
    $a = "One";
    print("The global value is $a <br>");
    test($a);
    print("The global value is still $a <br>");
?>
</body></html>
```

This example shows exactly the same output on screen as the previous example. As you can see the variable given as an argument to the function may have a different name within the function.

The cause: Only the content of the variable is given to the function and not the variable itself - do you remember? Keep in mind that variable names used as arguments may have different names within the function and outside it.

Now we need a solution to access global variables within functions. Experienced programmers use this method only seldom, because it makes copying source code from old programs into new ones difficult. The next example has the name test11.php and shows you how to do it:

```
<html><body>
<?php
    function test()
    {
        global $a, $b;
        print("The values test are $a und $b <br>");
        $a="Three"
    }
    $a = "One";
    $b = "Two";
    print("The global value are $a and $b <br>");
    test();
    print("Die globalen values are now $a and $b <br>");
?>
</body></html>
```

The program creates this output:

```
The global values are One and Two
The values in test are One and Two
The global values are now Three and Two
```

The keyword "global" makes global variables visible in functions. They can now be used as global and local variables. The second line of the output proves this.

The third line proves that modifications within the function are also visible in the main program.

4.8.12. Variable Variables

The title of this chapter sounds like a joke but it is not. Variable variables are used very seldom. But sometimes a program can become very easy using them. One example (test12.php):

```
<html><body>
<?php
    $january = "cold";
    $july = "hot";
    print("In january it is $january <br>");
    print("In july it is $july <br>");
    $month="july"
    print("In the month $month it is $$month <br>");
?>
</body></html>
```

The program creates this output on screen:

```
In january it is cold
In july it is hot
In the month july it is how
```

The third print command and its output are interesting. $$month is a variable variable. That means as much as "show me the content of a variable, its name is stored in $month".

Because $month has the value "july", the expression $$month is the same as $july. And $july has the value "hot".

4.8.13. Using PHP Scripts Outside Apache

If you have some fun with PHP you may want to use the programming language also outside the Apache webserver.

This is possible, but only on Unix. It does not work in windows.

The example test2.php was a dynamic website, that outputs the current date and time. With some minor changes it can also be used at the shell command line:

```
#!/usr/local/bin/php
This is.my first PHP-shell-script.
<?php
            print("Todays date is ".strftime("%d.%m.%Y").
                    " and the time is ".strftime("%H:%M:%S")."\n");
?>
```

Now enter the command **chmod a+x test2.php** to make this script executable. Then enter the command **./test2.php** to execute it. You will see the same output as before in the webbrowser but this time the output is written to the terminal window.

4.8.14. How To Proceed?

You learned the basics, how to write dynamic websites with MySQL, PHP and the Apache webserver. You understand, why the usage of HTML forms is useful.

PHP is a powerful script language with many functions. You cannot learn all of them in a short time - they are to many functions.

For all further questions please look into the original PHP documentation. You can read it directly on the PHP website http://php.net. The website has a useful search engine and a lot of additional comments from different PHP users. The comments are often useful.

I recommend to read first the following chapters of the PHP function reference in this order:

- Date and time functions

- Mathematical functions

- String functions

- Filesystem functions

- Array functions

- Error handling and logging functions

- HTTP functions

- Mail functions

- MySQL functions

- Regular expressions

If this is not enough for your knowledge hunger, then take a look at all the other function references and study the comments. They are very good written therefore it does not make sense to write the same again into this or other books.

4.9. Shell Script Programming

In Unix many jobs are solved with shell scripts instead of "real" programs. The disadvantage of shell scripts is that they are slow but this is acceptable for many easy jobs.

DOS users do not now what a shell script is but they now what a batch file is. In general both are the same but shell scripts are much more powerful than batch files. DOS can execute batch files only line by line from the beginning to the end. Shell scripts support control structures like if-commands, while-loops, for-loops and some more.

This chapter explains how to write and use shell scripts.

Windows users can also do everything that I show in this chapter because CygWin includes a shell that can do the same as the Unix shell.

Unix knows a lot of different shells and each of them has its own programming language. This books refers to the Bourne shell because every unix version includes this shell.

Please note that you can enter any command also directly at the command prompt. It's not neccessary to save all examples into files before you execute them - but you can if you like.

4.9.1. The First Shell Script

Save the following script as test1.sh:

```
#!/bin/sh
# This is the first example script test1.sh
echo "Hello"
echo -e "Todays date is \c"
date '+%d.%m.%y'
```

This script shows the actual date. To be able to execute it you need to enter the command **chmod a+x test1.sh** first, otherwise the shell does not detect the file as an executable script. Now enter **./test1.sh** to execute it.

The first line above should be the first line of any script. It tells the actual running shell that gives you the command prompt, that the shell /bin/sh (that's the Bourne shell) should be used to execute your script. If you leave this line out, the shell may try to execute your script with the wrong shell and unexpected error messages may appear.

The second line is a comment. The third and fourth lines use the echo command to output some text.

The date command shows the actual date. An explanation of the argument '+%d.%m.%y' can be found in the manual page of date.

man date

Echo commands append always a line break except you write a \c at the end of the text. Therefore the date appears directly after the word "is" and not in another separate line.

Other useful control characters are \t for a tabulator space, \a for a beep tone and \n for an additional line break.

The option -e is not needed in Solaris. Linux and Window need -e whenever you insert control characters like \t, \a, \n and \c.

Shell scripts can have any name. While reading this book you will see that I append mostly ".sh" to the filenames. This is my personal preffered way of naming shell scripts but is has no technical cause.

Shell scripts are marked as executable by the x-permission, not by the filename.

4.9.2. The Printf Command

You now the printf command already from PHP, if you read this chapter.

Printf outputs a text and inserts additional numbers or texts. Example (test2.sh):

```
#!/bin/sh
printf "The number %i is my %s \n" 13 fortune
```

The command outputs 13 instead of the placeholder %i and it outputs "fortune" instead of the placeholder %s. The correct order of arguments is important.

I wrote a \n at the end because printf dows not automatically terminate the output with a line break like echo does.

The are many placeholders, the following are often used:

%i	decimal number
%s	text
%f	floating number
%e	floating number ix exponential writing
%o	octal number
%x	hexadecimal number with small letters
%X	hexadecimal number with capital letters

You can specify the output width for decimal numbers by entering the width between the percent character and the letter:

printf "%10d" 12

This command outputs the number 12 with 10 characters width. The left end is filled with zeroes if the length value starts with a zero.

Floating numbers can also have a fixed width after the dot:

printf "%.2f" 12.345678

This command outputs "12.35", the value is rounded mathematically. Unfortunately you cannot select the width before the comma. You will always see all digits and the number is not filled with spaces or zeroes at the left end.

Please be sure to give printf exactly as many arguments as you insert placeholders into the text. Nobody knows what happens if you enter to many or to less arguments.

4.9.3. Strings

Strings (or texts) are always written between double quotes. Sometimes commands work also without double quotes (for example echo), but you should write them always. For example (test3.sh):

```
#!/bin/sh
echo "Hello       Stefan"
echo  Hello        Stefan
```

Both commands do not produce the same output. The second command writes only one space character between "Hello" and "Stefan" because the string was not included in double quotes. In real this are two strings and not one. The shell uses spaces and tabulators to separate arguments. Many consecutive space and tabulator characters are treated as one single space character.

If you want to output a control character that normally has a special function, you need to preceed it by a backslash \ (test4.sh):

```
#!/bin/sh
echo "Hello \"Stefan\" "
```

This outputs

```
              Hello "Stefan"
```
Another control character that needs a backslash is the dollar character. The backslash itself can be written as \\.

You can write strings also in single quotes. The difference is that " and $ have no special meaning when they are written between single quotes. Example (test5.sh):

```
#!/bin/sh
echo "Your name is $USER"
echo 'Your name is $USER'
echo 'Hello "Stefan"'
echo "Hello 'Stefan'"
```

The first command shows the actual user name, an environment variable of the shell. The second command shows only the name of the variable. The other commands demonstrate that single quotes within double quotes have no special meaning - and vice versa.

4.9.4. Variables

Variables can be set and then read later. Variables in shell scripts store always text. It is important not to write space characters before or after the assignment operator "=". Example test6.sh:

```
#!/bin/sh
variable1="Stefan"
echo "Hello $variable"
variable2=0123
echo $variable2
```

The second echo command outputs 0123 because this is not a number but a text. Therefore the trailing 0 is still there.

I like to show you a special thing when reading variables (test7.sh):

```
#!/bin/sh
variable3="Hello        Stefan"
echo "$variable3"
echo $variable3
```

This script creates the output:

```
Hello        Stefan
Hello Stefan
```

This example demonstrates how the variable name is replaced by the variable content **before** the text is splitted into its words. The multiple space characters are removed by the shell as explained before. The text "Hello   Stefan" was splitted into two single words and the spaces are lost. If you know that a variable stores more than a single word you need to write the variable name between double quotes, then the multiple space characters are not lost.

Variables that you assign values within a shell script are not visible outside this script. The export commands makes them visible outside the script. Example test8.sh:

```
#!/bin/sh
Name1="Cicelle"
Name2="Monique"
echo "$Name1 $Name2"
export $Name2
```

Execute this script by entering the command **./test9.sh**. Both names are shown. Now enter the command **echo "$Name1 $Name2"**. You will see that only the name Monique was exported by the export command. The name Cicelle is not visible outside the script.

Another way to export variables is this (test9.sh):

```
#!/bin/sh
Name1="Cicelle"
Name2="Monique"
echo "$Name1 $Name2"
```

Execute this script by entering

. ./test9.sh

Check the result entering again **echo "$Name1 $Name2"**. This time both names are visible, because you started the script with the dot command. The dot command is not used often because it has a important disadvantage: You cannot decide any more what variables you want to export and what not. All variables will be exported.

If you execute another shell script within a shell script - a sub-script - then the sub script will not know any variable of the main script. And the main script will not see the variables of the sub-script, except when you export them. This sounds complicated but it's easy. Take a look at this example:

```
#!/bin/sh
# This filename is test10.sh
Name1="Rafael"
echo "The 1st name is $Name1."
./test11.sh
echo "The 2nd name is $Name2."

#!/bin/sh
# This filename is test11.sh
echo "--- Start of sub-script"
echo "The 1st name is $Name1."
Name2="Alexander"
echo "The 2nd name is $Name2."
echo "--- Ende of sub-script"
```

Save both scripts and give them the execute permission using the **chmod a+x test*.sh** command. Then enter **./test10.sh**. You will get

this output:

```
The 1st name is Rafael.
--- Start of sub-script
The 1st name is .
The 2nd name is Alexander.
--- Ende of sub-script
The 2nd name is .
```

As you can imagine, you can export $Name2 with export command or you can run the sub-script with the dot command to solve this problem. The Alexander will also be visible in the main script.

At the end I like to tell you something more about variable names: Do not write variable names completely in capital letters. All variables of the operating system are written in capital letters and your own variable names should never be the same as a reserved operating system variable. If you do not use full capital names you avoid problems and do not change system variables accidentally.

For example, if you write accidentally **PS1="Hello"** because you did not know that this variable is used by the system... try it, it's harmless but surely surprising.

Enter **PS1="Hello"** and look at the result. Your command prompt changes! If you had used small letters, the prompt would not have changed.

If you like to see all shell variables, then enter **env|more**. You may find some variables useful.

4.9.5. In/Out Redirection Into Files

Input/Output redirection means a mechanism to redirect the output or the input of a program into another program, a file or a variable.

Input is normally read from the keyboard, while output is normally written to the screen. This is changeable:

echo "Hello" > test.txt

This command does not show the word "Hello" but it creates a text file with this word. Check it!

echo "engineer" >> test.txt

This command appends another word to the text file. The double arrow does not create a new file (except the file does not exist) but it appends the new text to the existing file.

You can also redirect input. This is done by an arrow pointing to the other direction. The read command normally reads one line from the keyboard and stores the text into a variable.

read input
echo "$input"

You could redirect the input to (from) a file:

read input < test.txt
echo "$input"

Now the read command does not read from the keyboard any more but it reads the first line from the text file.

4.9.6. In/Out Redirection Into Programs

You can redirect input and output also into programs:

env | more

The command "env" shows the environment variable of the shell. The redirection into the more command shows this list page by page. The more command reads from its input and writes it page by page onto the screen.

4.9.7. In/Out Redirection Into Variables

Output can also be redirected into variables:

Name=`hostname`
echo $Name

The command hostname shows the name of your computer. The command above redirects this output into a variable. Please note that the used characters around the hostname command are not quotes. The characters ' and ` are very different in their meaning. The character ` is called backtick or backquote.

You can also insert the output of a command into text, but only if the text is between double quotes:

echo "Hello `hostname`"
echo 'Hello `hostname`'

The first example works but the second does not work.

There is a no direct way to redirect the input of a command to a variable. But you can help out using the echo command:

Help=`man tar`
echo "$Help" | more

The manual page of the tar command is written into the variable Help. Using the echo command you output this variable and redirect it to the more command.

4.9.8. Many Commands In One Line

You can write many command into one single line. Depending on how you write them they are executed in different ways.

echo "Hello" ; echo "Rita" ; echo "have fun"

These three echo commands are executed one after the other. You get this output:

```
Hello
Rita
have fun
```

Another example:

rm /tmp/test.txt && echo "The file is deleted"

First the file will be deleted, then the successful message appears but only if the first command was successful. The echo command is executed if the rm command does not return an error code.

Another example:

rm /tmp/test.txt && echo "Could not delete the file"

First it tries to delete the file and if this rm command failed, then the error message appears.

Now I like to explain what exactly is a successful command and what is a non successful command.

Any program and any command (commands are programs) return an exit code to the shell when they end. This is a number between 0 and 255. 0 means that the command was successful, any other values indicates an error.

The numbers 1-255 have a different meaning in any program. For most programs you can find a list of possible exit codes in their manual page.

4.9.9. Control Structures

Control structure allow you to executes parts of the script in loops or only if a condition is true.

4.9.9.1. if...then...else...fi

The control structure if...then...else...fi allows you to execute a part of the program only if a condition is true or a test command returns 0. The usage is:

if Command ; then
  Program_part1
fi

or:

if Command ; then
  Program_part1
else
  Program_part2
fi

If the command was successful the Program_part1 is executed. If the command was not successful then the Program_part2 is executed.

Only seldom the following enhanced version is used, that allows to test for more than one condition:

if Command1 ; then
  Program_part1
elif Command2 ; then
  Program_part2
else
  Program_part3
fi

If Command1 was successful, then Program_part1 is executed. Otherwise Command2 is executed and if Command2 was successful, then Program_Part2 is executed. Otherwise the Program_part3 is executed. The elif part can repeated as often as you want.

A concrete example is test12.sh:

```
#!/bin/sh
if rm /tmp/test.txt ; then
  echo "Deleting was successful"
  echo "Everything is Ok"
else
  echo "Error:"
  echo "The file could not be deleted"
fi
```

And now you see an example for the enhanced version (test13.sh):

```
#!/bin/sh
if rm /tmp/test.txt ; then
  echo "Deleting of test.txt was successful"
elif  rm /tmp/test2.txt ; then
  echo "Deleting of test2.txt was successful"
elif  rm /tmp/test3.txt ; then
  echo "Deleting of test3.txt was successful"
else
  echo "No of the three files could be deleted."
fi
```

Some people write the keyword then in a separate line, then the semicolon can be left out:

```
#!/bin/sh
if rm /tmp/test.txt
then
  echo "Deletion was successful"
fi
```

4.9.9.2. Comparisons

If you want to execute a part of the program you want often compare two strings or numbers. This can be done using the **test** command or brackets **[ ]**.

There is no manual page for the brackets because its the same as the test command.

["Hello" = "Hello"]
echo $?

This returns 0 because the comparison was successful. This command is the same as:

test "Hello" = "Hello"
echo $?

The echo $? command shows the exit code of the last command. $? is a special variable that I will explain later together with some other special variables.

Most programmers like the brackets more than the test command because they are good visible when quickly browsing through a script file. Take care about the spaces before and after any bracket and any operator! It's really important not to forget any space, because otherwise the comparison will work not as expected.

Compare text

["Text1" = "Text2"]	0, if both texts are the same
["Text1" != "Text2"]	0, if both text are not the same
["Text1" < "Text2"]	0, if text1 comes before text2 in alphabetic order
["Text1" > "Text2"]	0, if text1 comes after text2 in alphabetic order
[-z "Text"]	0, if text is empty
["Text"]	0, if text is not empty

When comparing for alphabetic order A-Z comes before a-z. The computer simply uses the order from the ASCII table.

Compare numbers

[Number1 -eq Number2]	0, if both numbers are the same
[Number1 -ne Number2]	0, if the numbers are not the same
[Number1 -lt Number2]	0, if number1 is less than number2
[Number1 -gt Number2]	0, if number1 is greater than number2
[Number1 -le Number2]	0, if number1 is less or equal than number2
[Number1 -ge Number2]	0, if number1 is greater or equal than number2

You can only compare integer numbers, that are number without a dot.

The brackets or the test command are only useful when used together with control structures, like if...then...else or other. One example (test13.sh):

```
#!/bin/sh
Number = 12
if [ $Number -gt 5 ] ; then
   echo "The number $Number is larger than 5"
else
   echo "The number $Number is not larger than 5"
fi
```

Check the result. If you change Number=12 to Number=3 then the result changes.

Another example is test14.sh:

```
#!/bin/sh
Name = "Manuela"
if [ $Name = "Manuela" ] ; then
   echo "The name $Name is Manuela"
else
   echo "The name $Name is not Manuela"
fi
```

Checking filenames

[-r "Filename"]	0, if the file is readable
[-w "Filename"]	0, if the file is writeable
[-x "Filename"]	0, if the file is executable
[-f "Filename"]	0, if it's a regular file
[-d "Filename"]	0, if it's a directory name
[-L "Filename"]	0, if it's a symbolic link
[-e "Name"]	0, if the name exist as a filename, directory or link

One example (test15.sh):

```
#!/bin/sh
if [ -f "/tmp/test.txt" ] ; then
    echo "The file /tmp/test.txt does exist"
fi
```

4.9.9.3. Combine Many Comparisons

Many comparisons can be combined using "and" and "or" functions. There is also a negotiation available.

["Text1" = "Text2" -a "Text3" = "Text4"]	-a means AND
["Text1" = "Text2" -o "Text3" = "Text4"]	-o means OR
[! "Text1" = "Text2"]	! is the negation. This is seldom used.

Combinations of many comparisons are seldom used. One example (test16.sh):

```
#!/bin/sh
echo "Please neterr a number"
read Number
if [ $Number -lt 10 -o $Number -gt 100 ] ; then
    echo "The number is less than 10 or greater than 100"
else
    echo "The number is between 10 and 100"
fi
```

Another more often used version is the following, although it runs slower:

["Text1" = "Text2"] && ["Text3" = "Text4"]	&& means AND
["Text1" = "Text2"] || ["Text3" = "Text4"]	|| means OR

An example of this (test17.sh):

```
#!/bin/sh
echo "Please neterr a number"
read Number
if [ $Number -lt 10 ] || [ $Number -gt 100 ] ; then
    echo "The number is less than 10 or greater than 100"
else
    echo "The number is between 10 and 100"
fi
```

4.9.9.4. case..esac

If you build a large if and elif command with man comparisions the source code can become badly readable. A n example is test18.sh:

```
#!/bin/sh
echo "Please enter a number"
read Number
if [ $Number -eq 1 ] ; then
    echo "You entered one"
elif [ $Number -eq 2 ] ; then
    echo "You entered two"
elif [ $Number -eq 3 ] ; then
    echo "You entered three"
elif [ $Number -eq 4 ] ; then
    echo "You entered four"
else
    echo "You entered another number"
fi
```

You can write the same script using the case...esac control structure. This looks often more beautiful (test19.sh):

```
#!/bin/sh
echo "Please enter a number"
read Number
case $Number in
    1)   echo "You entered one" ;;
    2)   echo "You entered two" ;;
    3)   echo "You entered three" ;;
    4)   echo "You entered four" ;;
    *)   echo "You entered another number" ;;
esac
```

Do not forget the double semicolons! If you want to execute more than one single command for any condition, the write it as shown in test20.sh:

```
#!/bin/sh
echo "Enter a number"
read Number
case $Number in
    1)   echo "You entered one"
         echo "I like ones"
         ;;
    2)   echo "You entered two"
         echo "Two is also ok for me"
         ;;
esac
```

This example demonstrates, that double semicolons can also be written in another line and that you may leave out the *) part.

You can also combine many comparisons with the OR function (test21.sh):

```
#!/bin/sh
echo "Enter a number"
read Number
case $Number in
    1|2)    echo "You entered one or two" ;;
    3|4|9)  echo "You entered three, four or nine" ;;
esac
```

The case...esac control structure works also with text. Write strings and variables in double quotes to avoid problems with spaces (test22.sh);

```
#!/bin/sh
echo "Please enter a word"
read Word
case "$Word" in
    "red"|"green")  echo "You entered red or green" ;;
    "yellow")       echo "You entered yellow" ;;
    *)              echo "You entered something else" ;;
esac
```

4.9.9.5. while...do...done

The while loop executes a block many times, as long a condition is true or a test command returns 0.

```
#!/bin/sh
while read Text ; do
   echo "You entered $Text"
done
```

As long the read command is successful, the echo command is executed. For this program it's good to know that read returns with a non-successful exit code when you press Ctrl-D.

Also the while loop can use brackets (test24.sh):

```
#!/bin/sh
echo "Enter some text or only Enter to quit"
read Text
while [ "$Text" ] ; do
   echo "Thanks"
   read Text
done
```

You can enter as many lines of text as you like. But when you simply press Enter (that is an empty string), the loop ends.

4.9.9.6. for...in...do...done

The for loop executes a program part many times, one time for any element in a list. The following example test25.sh shows how to use the for loop:

```
#!/bin/sh
for Colour in red yellow green blue ; do
  echo "The colour is $Colour"
done
```

This script creates the output:

```
The colour is red
The colour is yellow
The colour is green
The colour is blue
```

4.9.9.7. break, continue And exit

The break command breaks a running loop. No other commands of that loop are executed anymore (test26.sh):

```
#!/bin/sh
echo "Enter something or press enter to stop"
while true ; do
   read Text
   if [ -z "$Text" ] ; then
      break;
   fi
   echo "Thanks"
done
```

This script does nearly the same as test24.sh but it uses different commands.

The while loop repeats endless because the true command returns always 0 (that means successful). True does nothing else. The opposite is false - what a surprise!

The if command checks if the text is empty. If yes, then the loop breaks.

The continue command enforces the next repetition of the loop. The remaining commands are not executes in the current repetition (test27.sh):

```
#!/bin/sh
echo "Please enter something or Ctrl-D to stop"
while read Text ; do
   if [ -z "$Text" ] ; then
      continue;
   fi
   echo "Thanks"
done
```

If you run this script you can enter as much as you like, also empty lines. But a "Thanks" appears only for lines that are not empty.

The exit command stops the whole shell script immediately. You can apply an exit code to the exit command if you like. If you leave out the exit code, 0 is assumed. Example (test28.sh):

```
#!/bin/sh
echo "Please enter something"
read Input
if [ "$Input" = "stupid" ] ; then
   echo "I'm not stupid!"
   exit 1
fi
echo "Ok, thanks"
```

After running this script you can check the exit code by entering **echo $?**.

while true ; do

4.9.10. Functions

Functions are small sub-programs than can be used many times at different places. One example is test29.sh:

```
#!/bin/sh
function Error()
{
    echo "An error occured:"
    echo "$1"
}

echo "Please enter a three"
read Number
if [ $Number -ne 3 ]; then
    Error "Wrong number!"
else
    echo "Thanks"
fi
```

If you run this script and enter something else than a 3, you see this output:

```
Please enter a three
5
An error occured:
Wrong number!
```

This function does not only show simple fixed text but also a variable (Wrong number!) because it got this variable as an argument from the main program.

Within the function the first argument is read with $1. If there would be more arguments, they would have the following number $2, $3 and so on. test30.sh:

```
#!/bin/sh
function Test()
{
    echo "The arguments are:"
    echo "$1 + $2 + $3 + $4"
}
Test "Hello" "Saskia" "my" "Sister"
```

This script creates the output:

```
The arguments are:
Hello + Saskia + my + Sister
```

4.9.11. Special Variables

We used already some special variables but I like to show you more of them:

$0	Is the name of the script
$1, $2 ... $9	are the arguments to a script or function
$#	is the number of arguments
"$@"	are all arguments as individual strings
"$*"	are all arguments as one string
$*	or $@ are all arguments splitted into words
$?	is the exit code of the last command
$RANDOM	Random number in range 0-32767 (only in bash shell)

You learned already how to use $1 until $9 in the last chapter about functions. You can use these variables also to read script-arguments (test31.sh):

```
#!/bin/sh
echo "You entered $# arguments"
echo "The script name is $0"
echo "The first argument is $1"
echo "All arguments together are $*"
```

Run the script by entering **./test31.sh One Two Three**. The output will be:

```
You entered 3 arguments
The script name is ./test31.sh
The first argument is One
All arguments together are One Two Three
```

Now I like to show you the difference between "$@" and "$*" in test32.sh:

```
#!/bin/sh
for Word in "$@"; do
    echo "-- $Word"
done
echo ""
for Word in "$*"; do
    echo "-- $Word"
done
```

The first for loop outputs every single Word, "$@" created a list of three words. The second for loop outputs only one line, alls three words are combined in one single line:

./test32.sh One Two Three

```
-- One
-- Two
-- Three

-- One Two Three
```

./test32.sh "One 1" "Two 2" "Three 3"

```
-- One 1
-- Two 2
-- Three 3

-- One 1 Two 2 Three 3
```

If you compare both outputs you will notice that the text "One 1" is still kept together as one string.

If you write $@ or $* without double quotes, then you will get always a list of single words. In this case "One 1" will also be split into two words (test33.sh):

```
#!/bin/sh
for Word in $*; do
    echo "-- $Word"
done
```

./test33.sh "One 1" "Two 2" "Three 3"

```
-- One
-- 1
-- Two
-- 2
-- Three
-- 3
```

Alltogether there are three different ways to read all arguments with one single variable.

4.9.12. More Than Nine Arguments

You just learned that the variables $1 to $9 represent the arguments given to a script or function.

The number of arguments is not limited to 9. You can override the limit of 9 by using the shift command (test34.sh):

```
#!/bin/sh
echo "The first argument is $1"
shift
echo "The second argument is $1"
shift
echo "The third argument is $1"
shift
echo "The fourth argument is  $1"
```

The shift commands deletes the first argument and moves all other one position to left. The argument $2 moves to $1 and $3 moves to $2 and so on. You can use the shift command as often as you like and it allows you to access more than 9 arguments.

4.9.13. Arithmetic Expressions With expr

Shell Scripts cannot calculate mathematical expression. The expression

Result=1+2

does not work. Mathematical expressions need to be calculated with a small helper program called expr:

Result=`expr 1 + 2`
echo $Result

Unfortunately you can only calculate numbers without a dot (no float). Expr understands the following expressions:

Number1 + Number2	Plus
Number1 - Number2	Minus
Number1 * Number2	Multiplication
Number1 / Number2	Division
Number1 % Number2	Rest of a Division (Modulus)
substr Text Start Count	Returns count characters from the text beginning at the start position. "substr Hello 2 3" returns "ell"
index Text Characters	Returns the position of the first occurence of one character within a string "index Hello eb" returns "2"
length Text	Returns the length of a string

Do not forget the spaces and take care about the star because it has a special meaning to the shell. You need to write a backslash (\) before the star and also before paranthes.

Expressions can use parentheses to specify the evaluation order:

Result=`expr \( 1 + 2 \) \* 3`
echo $Result

        9

Text="Hello"
Length=`expr length "$Text"`
echo $Length

        5

expr 6 * 5

        30

The last command shows you how to output the result to the screen instead redirecting it into a variable. This is used very seldom because normally you want to use the result for whatever, e.g. in an if...then...else...fi control structure.

4.9.14. Error Messages

You learned already how to redirect output into variables, files or into other programs. This redirection does not affect error messages.

One example:

rm xxxaaabbbccc > /dev/null

The command should delete the file xxxaaabbbccc but the file does not exist. Therefore you will see an error message.

You redirected the output to /dev/null but the error message still appears on the screen.

The cause is that there are two different output channels. Channel 1 is used for regular output and channel 2 is used for error messages. The output redirection of the example above affects only channel 1.

If you want to know what /dev/null for a file is then let me tell you that this is not really a file. It is a device driver that eats everything that you give him and outputs nothing. You can use /dev/null as a "trash can" for output if you do not want to see anything.

We have still the problem that the error message appears on screen. And this is the solution:

rm xxxaaabbbccc > /dev/null 2>&1

The appendix 2>&1 has to be placed at the end of the command. It tells the shell that also the error channel 2 shall be redirected to the same where channel 1 goes to. This time you will not see the error message any more.

It works also for redirection into variables, like this example shows:

Result=`rm xxxaaabbbccc 2>&1`
echo $Result

If you want to create your own error message to channel 2, you do it like test35.sh:

```
#!/bin/sh
echo "This is normal text"
echo "Error: Somethin went wrong" > /dev/stderr
```

Test the script by entering

Result=`./test35.sh`
Echo $Result

The normal text is in the variable but the error message appeared directly on screen and was not redirected to the variable.

4.9.15. Useful Tools

I like to show you some more useful tools that are often used in shell scripts. I do not want to describe any function of them, but only tools that may be useful for SMS applications.

If you like to learn all the details then look into the manual pages of the programs.

4.9.15.1. cat

This is not an animal but a shortcut for catalogue. The cat command shows the content of a text file:

cat test.txt

You can redirect the output into a variable:

Variable=`cat test.txt`

If you leave out the filename, cat reads from the keyboard until you press <Ctrl-D>:

Variable=`cat`

4.9.15.2. cut

The command cut reads from its standard input channel and cuts a part of it for output. You can select between

-c range	The characters in range
-f range	The columns in range
-d ','	Specified the separator character for columns

The range can be written in different ways:

3	Only the third character or column
3-5	The characters or columns at position 3-5
-5	From beginning until 5
1,3,5	The characters or columns at position 1,3, and 5

If you give a filename to the cut command, it reads from the file, otherwise it reads from the input channel (normally the keyboard) until you press <Ctrl-D>. Of course you can redirect the output using arrows > < and the pipe character |. Some examples:

cut -c 3-5
Hallihallo
<Ctrl-D>

Creates the output:

```
        lli
```

lets say you have a text file with this content,

```
        Marius
        Mariechen
        Liesa
        Robert
```

and then you enter

cut -c 1-3 test.txt

you will get this output

```
        Mar
        Mar
        Lie
        Rob
```

The options -f and -d are normally used together:

cut -f 2,4 -d ','
One,Two,Three,Four,Five
<Ctrl-D>

Creates the output:

```
        One,Four
```

You filtered out the fields 2 and 4. Please note that cut uses the same delimiter for input and output.

4.9.15.3. sed

Sed is a powerful program that modifies text. There are whole books written about sed, so many functions has it. I like to show you only some few that I use often:

Like cat and cut also sed reads from a file or from the keyboard until you press <Ctrl-D>. Input and output are redirect-able.

Sed does not modify the source file. It shows the modifications only on screen.

Lets say you have the file test.txt with this content:

```
Hello
the
Weather
is
nice
and
i
have to
learn
```

The next command read this file and gets only some few lines from it ignoring the rest:

sed '2,4p' --silent test.txt

```
the
Weather
is
```

2,4p means: give me oly the lines from2 until 4. If you want to get only one single line, you do not need a comma:

sed '2p' --silent test.txt

The next command outputs the whole text except a given line number:

sed '1d' test.txt

Sed can also search and replace words:

sed 's/nice/wet/g' test.txt

The search word "nice" is a regular expression. A later chapter will show you what that means. Read it - its useful!

4.9.15.4. grep

The grep program reads its input line by line and outputs only lines that match a keyword (exact: regular expression). The input is again a file or the keyboard. Grep can also read from more than one file if you give him mor filenames. Input and output can be redirected to whatever you like. If you enter from keyboard, press <Ctrl-D> to stop.

Lets say you have a text file with this content (test.txt):

```
In winter the snows comes.
In summer its warm instead.
My mother is good to me.
Where are you going?
The man looks different.
```

If you now enter **grep 'the' test.txt** you get this output:

```
In winter the snow comes.
My mother is good to me.
```

The second line appears because "Mother" includes "the". The last line does not appear because grep is case sensitive.

The option -i lets grep to forget that it is case sensitive:

grep -i 'the' test.txt

```
In winter the snows comes.
My mother is good to me.
The man looks different.
```

The option -c is used to count the number of matching lines (test36.sh):

```
#!/bin/sh
Count=`grep -c 'der' test.txt`
if [ $Count -gt 2 ]; then
   echo "More than 2 lines were counted, exactly $Count"
else
  echo "Less or exactly 2 lines were counted."
fi
```

The option -l is used to list only the filenames that contain at least one matching line. This makes only sense if you give many filenames to grep.

The option -v makes grep to show all lines that do **not contain** the searched word.

The next chapter explains regular expressions used by grep and sed.

4.9.15.5. Regular Expressions

Regular expressions are search-words for searching in text. This cannot only be words. Regular expressions are often used in shell scripts with sed and grep.

I like to show you some useful regular expressions for the grep command, but the work also with sed:

a-z, A-Z, 0-9	Searches for a letter or digit
^	is the beginning of a line
$	is the end of a line
\<	is the bgin of a word
\>	is the end of a word
.	is any character
[abc]	searches for one of the listed characters
[a-z]	searches for one characters in the given range
a*	searches for zero or more a's
a+	searches for one or more a's (only egrep)
|	is an OR (only egrep)
()	is used to group expressions (only egrep)

Caution: The last three expressions are only known to egrep, an enhanced version of grep!

If you want to search for a character that has a special meaning to grep or egrep, you need to preceed it by a backslash \. For example \. searches for a dot and not any single character.

Create a text file with this content:

```
In winter the snows comes.
In summer it's warm instead.
My mother is good to me.
Where are you going?
The man looks different.
My name is Dr. Irene Meier.
Your name is Ms. Dr.Dr. Liebig.
Brrrrrrr, it's cold here.
The next beep is 08:23:56.
Berries are sweet.
Bumm, that was a crash.
Brrrrumm, and another car.
```

We will use this text file to test some regular expressions.

grep 'i' test.txt

Shows all lines that contain an i. But

grep '^I' test.txt

shows only lines that start with a capital I. The difference is that "Ms. Dr. Irene Meier" is only found by the first command.

grep '\.$' test.txt

Shows all lines that end with a dot. The line that ends with a question mark is not one of them.

grep '..:..:..' test.txt

Shows all lines with a time value.

grep 're\>' test.txt

Shows all lines with a word that ends with "re".

grep '\<....\>' test.txt

Shows all lines that have a word with exactly four characters.

grep 'Br*umm' test.txt

Shows all lines where a word starts with B, continues with any number of r (alows no r) and then ends with umm. The output is:

```
Bumm, that was a crash.
```

```
        Brrrrumm, and another car.
```

egrep 'Br+umm' test.txt

This command shows only the line with "Brrrrumm" because the r+ want at least one r.

egrep '(Dr\.)+' test.txt

This finds "Dr." and "Dr.Dr.". This example demonstrates how to use parentheses. The + does not mean a single character any more because there are parentheses before the +.

egrep 'Meier|Liebig' test.txt

Searches for Ms. Meier or Ms. Liebig. You will get both lines.

grep '[Mi][ys]' test.txt

Shows all lines with a M or i followed by y or s.

4.9.15.6. How To Cut Spaces

Sometimes you need to cut spaces from a value in a variable. This is very easy because the shell has such a function:

a="   Hello"
echo "$a"

b=`echo $a`
echo "$b"

This creates the output:

```
            Hello
        Hello
```

The first echo command demonstrates that the variable content includes unwanted spaces. After that we use the echo command again together with a redirection to variable b to cut off all spaces. The last echo command demonstrates that the spaces are now gone.

Why does that work?

To explain this We split the third line into two parts:

b=`...`

and

echo $a

The interesting part is the second one. This time I did not write $a between double quotes - like we normally do. This lets the shell split the text into words removing all spaces that we do not need. And exactly this is what we want.

4.10. Awk Programming

The homepage of gawk offers a book for free to download or read online. It is in HTML format. See
http://www.gnu.org/software/gawk/gawk.html

4.10.1. How to start awk

Awk needs always a script and one or more text files. The following three commands do the same:

awk -f scriptfile textfile

awk -f scriptfile < textfile

cat textfile | awk -f scriptfile

Many textfiles can be used in this way:

awk -f scriptfile textfile1 textfile2 textfile3 ...

If the script is very small, you may also give it as a command line argument instead of storing it into a separate script-file:

awk 'commands' textfile

4.10.2. First awk examples

Before I explain the awk script language, I like to show you some easy examples. The first example reads a text file and outputs the second word of each line:

awk '{print $2}' textfile

If you play around with different textfiles, you will notice that awk treats multiple spaces and tab-characters as single ones. In addition awk removes leading and trailing spaces from each line. Lets assume, your textfile looks like this one:

```
    Stefan          Frings
Michaela Meier
    Gundula Geier
```

then the result would be:

```
    Frings
Meier
Geier
```

The second example finds all lines with the keyword "Hello" and prints them (to the screen).

```
    awk '/Hello/ {print $0}' textfile
```

The third example prints only the third line of the text file, suppressing all other lines.

```
    awk 'NR==3 {print $0}' textfile
```

The jobs from these simple examples can also be done with other command. But specially more complex jobs can often be solved with awk better than with other shell commands.

4.10.3. Basic syntax of awk commands

Awk reads text files line by line and executes a list of commands for each line. The whole awk script is repeated for each textline. Each command has two parts, the condition and the command.

```
condition {action}
```

The condition controls what textfiles are used by that command. I like to re-use the examples from the previous chapter:

The first example does not have a condition. Therefore the action **print $2** affects all lines of the textfile.

The second example has the condition **/hello/**. This is a regular expresseion (like grep command) that means as much as: use only lines that contain hello.

The third example has the condition N**R==3**. Only the line is used that has the Number==3. NR is a variable that stands for the line number.

Now please create an awk script with your preferred text editor. The name shall be test1.awk and the content shall be:

```
NR==1 { print "The first line is:" $0 }

NR>1 { print "Followed by:" $0;
    count=count+1;
  }

/hello/ { print "hello again.";
        hello=hello+1;
      }

END { print "After line 1 we have " count " lines.";
    print "Hello was counted " hello " times";
    }
```

This script has four conditions. You can see here, how to write many commands into one script file. And you can also see that each action can be contain more than one single command.

The first action affects only the first text line. It prints a comment and after that the whole text line.

The second action affects all lines except the first one. NR means record-number or in easier words line-number. This action prints also a comment. In addition the number of occurrences gets counted.

The third action affects all lines with the keyword hello. This action does not print the lines again. It simply prints "hello again." and counts the occurrences.

The fourth action is executed only once after the textfile has been read until its end. The condition END controls this behaviour. At the end, the scripts outputs the counter results.

If you like to perform an action at the very first beginning before reading the first textline, you need the condition BEGIN.

Now please create a textfile with many lines. At least one line shall contain the word hello. Then execute your script to test it:

awk -f test1.awk textfile

You can also read many textfiles one after the other. In this case, the condition END happens only one time at the end of the last textfile. In the same manner, the condition BEGIN happens only one time at the beginning of the first textfile.

4.10.4. Field separators

Awk splits text lines into fields, normally by using spaces. Many consecutive spaces are treated as single ones. Tab-characters are treated as space characters. Example:

Lets say you have the textfile test1.txt with this content:

```
Stefan Frings 12 10 74 ,Programmer of SMS Server Tools
Bill Gates 23 05 52 ,Was the most richest man of the world
Golden Gate 01 24 65 ,Is a bridge
```

Now lets assume that you want to filter out only the names, then write this script:

```
{print $1,$2}
```

Awk reads each line and splits it into words. The print command prints the first and the second word. The comma tells print, to print a delimiter between $1 and $2. The default delimiter is a space character, so the output looks like:

```
Stefan Frings
Bill Gates
Golden Gate

{print $1 $2}
```

Without the delimiter, the names would appear without spaces:

```
StefanFrings
BillGates
GoldenGate
```

You could also insert text between $1 and $2:

```
{print "Name: " $1 " Lastname: " $2}


Name: Stefan Lastname: Frings
Name: Bill Lastname: Gates
Name: Golden Lastname: Gate
```

The comma inserts a space normally but you can also change it to another separator:

```
BEGIN {OFS="++"}
{print $1,$2}
```

In the BEGIN condition, the script changes the separator to "++". OFS means output-field-separator. The outpull will change to:

```
Stefan++Frings
Bill++Gates
Golden++Gate
```

You have learned, how to change the output filed separator. But you can also change the input field separator. Awk can use more than only spaces and tab-characters.

If you use the textfile test1.txt from above with this script:

```
BEGIN {FS=","}
{print $2}
```

Then you get the second field after the comma:

```
Programmer of SMS Server Tools
Was the most richest man of the world
Is a bridge
```

Spaces are now no separators anymore, only commas. Input field separators can also be regular expressions, like in grep.

```
BEGIN {FS="[ ,]"}
{print $1,$2,$7}
```

Here we used spaces and commas to separate fields. The characters " ," is treated a two separators because it matches the regular expression two times. Result:

```
        Stefan Frings Programmer
        Bill Gates Waa
        Golden Gate Is
```

If you want that multiple separators are treated as one single, you simply need to modify the regular expressen a little bit:

```
        BEGIN {FS="[ ,]+"}
        {print $1,$2,$6}
```

The comment begins now in field $6, because " ," is not treated as multiple separators anymore. Take a look into the grep chapter to learn more about regular expressions.

4.10.5. Internal Variables

Awk knows some internal variables that you can use. You used already OFS and FS.

OFS	Output Field Separator. Print inserts this characters instead of the comma. The default is one space character.
FS	Field Separator. Lines are splitted using this regular expression. The default are one or multiple spaces or tab-characters.
ORS	Output Record Separator. print ends each line with this character. It is normally a line-feed.
RS	Record Separator. Input lines are separated with this character, that is normally a line-feed.
OFMT	Output format for numbers.
NF	Number of Fields in the current line.
NR	Number of Record (lines) counted from the beginning of the textfile.
FNR	Number of Records (lines) counted from the beginning of current textfile.
FILENAME	Name of the current textfile.
RSTART	Start of a search result, used by the command match().
RLENGTH	Length of a search result, used by the command match().
$0	The current line.
$1	The first field of the current line.
$2	The second field of the current line.
$3	The third field of the current line.

4.10.6. Variables at the command line

You might want to give variables to awk scripts at command line level sometimes. This is mostly done within shell scripts. A simple example:

```
#!/bin/sh
echo "We have"
awk -v fruits=36 -v persons=12 'BEGIN {print fruits/persons}'
echo "apples for each child."
```

or

```
#!/bin/sh
apples=36
childs=12
echo "We have"
awk -v fruits=$apples -v persons=$childs 'BEGIN {print fruits/persons}'
echo "apples for each child."
```

or

```
#!/bin/sh
apples=36
childs=12
result=`awk -v fruits=$apples -v persons=$childs 'BEGIN {print fruits/persons}'`
echo "We have $result apples for each child"
```

All three examples produce (nearly) the same output. The last example prints everything in one single row because it has only one echo command.

You saw, how to set variables from outside using the option -v. These variables can be used within awk scripts.

You have surely noticed, that the examples above do not read input files. After the BEGIN condition is only the mathematical calculation. There is no condition that matches a text line and therefore, we do not need to use a textfile.

4.10.7. Operators

The previous chapters show you how do perform mathematical additions and divisions. Awk supports the following operators:

+	addition
-	subtraction
*	multiplication
/	division
%	modulus
^	raise to the power
++	increase by 1
--	decrease by 1

Awk knows the following assignments:

=	assign
+=	add to
-=	subtract from
*=	multiply by
/=	divide by
%=	modulus by
^=	raise to the power

Examples:

a=3   result is 3

a+=5   result is 8, same as a=a+5

b=a+6   result is 14

b++   result is 15, same as b=b+1

b/=3   result is 5, same as b=b/3

Awk knows the following comparision operators:

==	equal
!=	not ewual
>	greater than
<	less than
>=	greater or equal
<=	less or equal
~	contains regular expression /RE/
!~	does not contain regular expression /RE/

Examples:

if ($1=="Stefan")   If the first word is Stefan, then ...

if ($1>5)   If the first number is greater than 5, then ...

if ($1~/fan/)   If the first word contains fan, then ...

Logical operators are used to combine multiple comparisions:

&&	and

||	or
!	not

Example:

if (a==3 && b==4)   if a==3 and b==4, then ...

4.10.8. Conditions with if

With the keyword "if" you can execute one or many commands with a condition. "If" can be used alternatively to the condition of chapter 4.10.3 and is more flexible.

```
if (condition)
  command;
else
  command;
```
or:
```
if (condition) {
  commands;
  commands;
  commands;
} else {
  commands;
  commands;
  commands;
}
```

Use brackets to execute more than one single command. Brackets are used to make blocks of commands. The else-part is optional.

Example:

```
if (a==3)
  print "The variable a is now 3";
```

4.10.9. Loops with while

Loops are used to execute a command or a block of commands multiple times. The while loop can be used in two different ways:

```
while (condition)
  command;


do
  command;
while (condition)
```

The difference is only that the condition is checked either before the command or after the command. Example:

```
a=0
while (a<10) {
  print "The variable a is now" a;
  a++;
}
```

This loop counts from 0 up to 9.

4.10.10. Loops with for

Awk imitates the "for" loop from the programming language C. This loop is made for cases where a value is counted up or down.

```
for (preparation; condition; step)
  command;
```

The example from previous chapter could be rewritten in this way::

```
for (a=0; a<10; a++)
  print "The variable a is now" a;
```

There is another form of for-loops in combination with arrays:

```
for (indexvariable in arrayname)
  command;
```

I will explain this form in a later chapter.

4.10.11. Break and continue

```
Use a break command to cancel a loop:
a=0;
while (a<10) {
  print "The variable a is now" a;
  a++;
  if (a>5)
    break;
}
```

This loop does not count up to 9 anymore. It stops at value 5 because then the break command gets executed. Break cancels a loop.

Continue cancels the current loop repetition and executes the next repetition.

```
a=0;
while (a<10) {
  print "The variable a is now" a;
  a++;
  if (a>5)
    continue;
  print "a is not greater than 5";
}
```

4.10.12. Next and exit

Apart from break and continue, those two commands affect the whole Awk script.

Next stops the execution of the current text line. Awk reads the next text line and executes the script for this line from the top to the bottom.

Exit cancels execution of the whole script immediately. You can enter a number that is used as a return value for the invoking shell. Shell scripts can use the value if you like.

Exit oder exit 0 is normally used for successful program termination. Other values typically indicate and error. There no force to use 0 for successful exits but most programmers do so.

4.10.13. Special characters in print

You used print already to output text and variables. Print knows two control characters:

\n	new line
\t	tab-character
\0xxx	Any character gives as octal number

Print can also write its output to a file instead to the screen.

```
print "Hello" > "filename";
print "Hello" >> "filename";
```

The second version appends to an already existing file. The first version creates a new file and eventually overwrites files that already exist.

Print can also execute external command and send text to them:

```
print "Stefan\nMarkus\nJessica\nOlivia" | "sort";
```

The command above outputs four names and sends them to the external sort command. This command sorts the names alphabetically.

4.10.14. Printf und sprintf

Printf is used to output text without trailing newline character:

```
printf("text");
```

You can use printf also to insert variables into the text:

```
printf("text",variable,variable,variable,...);
```

The text must contain placeholders where printf inserts the variables. There are those placeholders:

%d	an integer number
%f	a floating pint number
%c	a single character
%s	a string (text)

Example:

```
printf("%f Euro",12.5);
```

You can specify the field length of the spaces that are reserved for the variables. The values appear right-aligned:

```
printf("%6f Euro",12.5);
```

Left-aligned is also possible:

```
printf("%-6f Euro",12.5);
```

One more complex example:

```
printf("%-20s %4.2f Euro\n","Inkjet paper",12.5);
```

This outputs the title "Inkjet paper" left-aligned and filled to 20 characters length. The the price follows right-aligned into a field with 4 digits before the dot and 2 digits after the dot. The a newline character follows.

Sprintf works like printf, but the output goes into a variable instead to the screen:

```
variable=sprintf("%f Euro",12.5);
```

4.10.15. Arrays

Arrays are variables that can store many values. Each value has a name or a number plus the value itself.

Examples:

```
fruits["apples"]=36;
fruits["pears"]=20;
fruits["cherries"]=340;
...
print "We have " fruits["apples"] " Apples";
print "We have " fruits["pears"] " Pears";
print "We have " fruits["cherries"] " Cherries";
...
```

The for loop makes this easier:

```
fruits["apples"]=36;
fruits["pears"]=20;
fruits["cherries"]=340;
...
for (name in fruits)
  print "We have " fruits[name] " name;
```

Arrays are very useful to count the occurrences of error codes in a logfile. Lets say you have a logfile with this format:

```
05.04.2005 12:38 Error 12
05.04.2005 12:39 Trying to read configuration file
05.04.2005 12:55 Error 18
05.04.2005 12:58 Starting Internet communication
05.04.2005 12:59 Error 17
```

Then you could count the error codes with this script:

```
/Error/ {errors[$4]++}

END {for (code in errors)
        print "Errro-Code " code " was counted " errors[code] " times.";
     }
```

4.10.16. Internal functions

Awk has a lot of internal functions. The most common functions are:

int(x)	returns integer value of x
log(x)	natural logarithm of x
exp(x)	natural exponent x
sin(x)	sine of x in radians
cos(x)	cosine of x in radians
rand()	Random number between 0 und 1
srand()	Initializes the random number generator. You should use this command one time in the BEGIN condition of each program that uses rand().

More often you will use the string functions::

tolower(s)	returns the string (text) s in lower case
toupper(s)	returns the string (text) s in upper case
length(s)	returns length of s
substr(s,start,length)	Returns a part of s. The part begins with start and has a given length. If you omit the length, then you will gte the whole rest of the string.
match(s,r)	Compares the string s with a regular expression r. It stores the result in two variables, RSTART and RLENGTH. If r was not found in s, then RSTART is 0. Match returns the found position or 0.
sub(r,s,dest)	Searches for the regular expression r in the destination string and replaces the first occurrence by the string s. If you omit the destination, then $0 is used.
gsub(r,s,dest)	Like sub but it replaces all occurrences.
index(s,word)	Searches for a word in the string s and returns the position. The result is 0 if the word was not found in s.
split(s,array,separator)	Splits a string s into fields. Each field becomes an element in an array. If you omit the separator, FS is used.

Information: Regular expressions can be written between slashes instead of double-colons for better visual impression.

Functions for Input/Output:

close(filename)	closes a file
getline variable < filename	Reads one line from a text file. If you omit the filename, the the current input file is used. If you omit the variable name, then $0 is used.
command | getline variable	Executes a command and write the output of that command into a vriable. If you omit the variable, then $0 is used.
system	Executes an external unix command and returns the status.

Other functions:

delete array[element]	deletes one element of an array

Examples:

```
text="Hello bruther";
gsub(/bruther/,"brother",text);
```

Results: "Hello Brother"

```
text="WRitteEN MIxeD";
print tolower(text);
```

Results: "written mixed";

```
text="Errocode [45]";
```

```
match(text,/\[[0-9]+\]/);
if (RSTART>0)
  print substr(text,RSTART+1,RLENGTH-2);
```

Results: 45

Match searches for a number with at least one digit between square brackets. The position is stored in RSTART and RLENGTH.

The regular expression [0-9] searches for a digit. \[searches for a square bracket. The + means that the previous digit must appear at least 1 time.

Substr is used to extract only the number without the surrounding brackets. Printf prints the result.

4.10.17. Self written functions

Self written functions can be declared at the beginning of an Awk script:

```
function duplicate(a) {
  return a*2;
}


{print duplicate($1)}
```

This script reads in a textfile, each line must begin with a number. These numbers are multiplied by 2 and then displayed.

4.10.18. Awk example for SMS applications

You see that Awk can do many things that the shell cannot do. I assume that you will use only a few of Awk's functions. The question is: Why do I explain Awk here?

You can do many easy jobs very well with Awk but only very complicated with the shell.

At the moment you might not be familiar with the SMS Server Tools because I describe this program in a later chapter. But anyway, I like to show you one very concrete example.

Lets assume, your customers can order jokes via SMS by sending the keyword "joke" and an item-number to your SMS server. You will receive a lot of SMS messages, some of them will not contain valid orders. Your program shall fulfill these requirements:

• Ignore all invalid orders

• Extract the 5-digit item-number

• Send the joke via SMS to the customer

You could use grep to check whether the received SMS contains the keyword "joke" and whether a valid item-number is included. But grep cannot extract the number. Also the cut command is not useful because you do not now the exact position of the item-number. Your customer could write the item-number as the very first word in his order or he could place it somewhere else. He could write space characters between and he could include other meaningless words in his SMS. Example:

```
"Hello, I like to order the joke number 12345"
```

This is a valid order because it contains the keyword joke and also the item-number. I like to show you now, how to combine a shell script with awk to process such orders.

Your items (the jokes) are stores as text files:

/var/spool/sms/jokes/12345.txt
/var/spool/sms/jokes/23456.txt
/var/spool/sms/jokes/98765.txt

When the SMS Server Tools receive SMS, they are also stored in text files in this format:

```
From: +491721234567

Hello, I like to order the joke number 12345
```

Whenever the SMS Server Tools receive a message, they store it into a text file and then run an eventhandler script. $1 is always the keyword "RECEIVED" and $2 is the filename of the stored message.

Your eventhandler script could be this one:

```
#!/bin/sh

if [ "$1" != "RECEIVED" ]; then
  exit
fi

from=`formail -zx From: < $2`
text=`formail -I "" <$2 | sed -e"1d"`

if echo "$text" | grep -i "joke"; then
  smsfile=`mktemp /var/spool/sms/outgoing/send_XXXXXX`
  echo "To: $from" >> $smsfile
  echo "" >> $smsfile
  number=`echo "$text" | awk '

    {
      if (match($0,/[0-9][0-9][0-9][0-9][0-9]/))
        print substr($0,RSTART,RLENGTH)
    }

  '`
  if [ -f "/var/spool/sms/jokes/$number.txt" ]; then
    cat "/var/spool/sms/jokes/$number.txt" >> $smsfile
  else
    echo "Sorry, wrong article number" >> $smsfile
  fi
fi
```

Please remember to replace awk by nawk if you use Solaris.

This script extracts the phone number of the sender and also the message text using formail. Then it checks with grep if the keyword "joke" is included. The parameter -i tells grep to ignore upper/lower case. Only if the keyword "joke" was found, the message gets processed.

The command mktemp creates a new SMS file. Mktemp replaces xxxxxx by a random number. The script uses echo commands to

write the destination phone number of your customer and the then an empty line into the SMS file.

After that awk extracts the item-number from the order. The next if command checks if such an article (textfile) really exists. Then the ordered text gets written into the SMS file.

Finally the SMS server tools send this SMS file automatically.

I like to explain the awk line in detail:

```
number=`echo "$text" | awk '{if (match($0,/[0-9][0-9][0-9][0-9][0-9]/) print substr($0,RSTART,RLENGTH)}'`
```

```
number=`...`
```

Writes the result of the next command into the variable $number. We can use this variable to insert it into a filename later.

```
echo "$text" | awk ...
```

The echo command sends the text of the order to awk. We give awk only this next through a pipe and no input filename. I showed you this principle in chapter 4.10.1.

```
if (match($0,/[0-9][0-9][0-9][0-9][0-9]/) ...
```

The match command searches for a regular expression. In this case we search for a number of 5 digits in range 0-9. Only if this number was found, then the next print command gets executed.

```
print substr($0,RSTART,RLENGTH)
```

Here we output the found item-number. The match() command stored the position in RSTART and RLENGTH, so we can use substr() to output only the important part from the order text.

5. The SMS Server Tools

The SMS Server Tools are a program package to send and receive SMS.

As the name says the packages is not primarily made to give you a colourful user interface. The design is concentrated in a stable program made for servers that run automatically.

The programs are developed by the author of this book. He gets a lot of support from different hobby-programmers and commercial users.

The development continues more quickly than most other programs. If you always want to have the latest version then visit my homepage:

http://www.isis.de/~s.frings

I announce new version on

http://www.freshmeat.net.

On this website you can put your eMail address into a list. Then you get notification eMails whenever a new version appears. I cannot read this list because the web administrator of www.freshmeat.net keep it safe. I can only see how many entries are in this list.

Since some time the program is part of any Debian Linux distribution.

This chapter is based on the information on my homepage but with some additional comments.

5.1. Overview

This chapter is an introduction to the SMS Server Tools. You will learn how they work and what they can do.

A note about abbreviations:

SMS means Short Message Service and means the whole service. Therefore you cannot send SMS, but the abbreviation is still often used in the wrong way by users of mobile phones - also the advertising uses it often wrong.

SM means Short Message. You do not send SMS but SM. many people avoid this abbreviation because SM means also Sado-Masochism.

SMSC means Short Message Service Centre. This is the centre for SMS that queues messages and delivers them to the mobile phones. ANY phone network operator has at least one SMSC. Like any phone call goes through the public switch any SM goes through the SMSC.

The SIM card is the small removable chip-card that every GSM modem and mobile phone need to identify the device in the phone network. Without a SIM card no GSM device can operate. Mobile phones and GSM modems do not only store the phone book onto this card, they use it also to store their own phone number (MSISDN).

The next picture shows the most important parts of SMS Server Tools and how they are connected together.
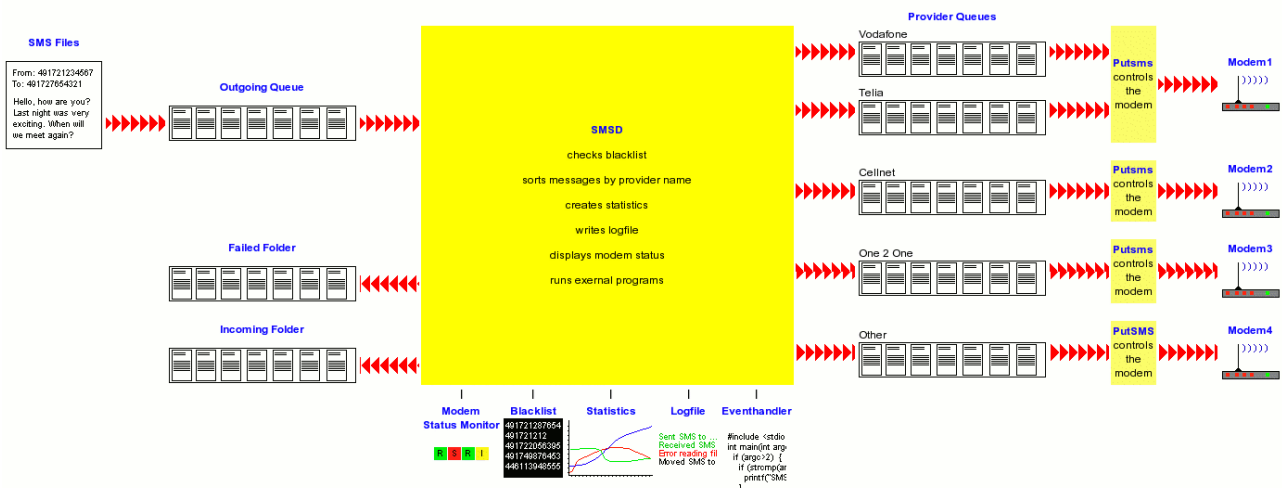
The picture shows how the SMS Server Tools send SM. In this case we have five british phone network provider and we use four modems to send messages.

The explanation follows on the next page.

To send a message you need to create a SMS file.

This text file has to contain at least the phone number of the receiver and the message text.

You can also write the senders name or number into the file, it will appear in logfiles later.

After creating the SMS file you move it to the outgoing queue directory. This is a simple directory on the harddisk.

You can also create the file directly in the outgoing queue directory if this goes quickly. The main program smsd checks this directory any few seconds and if there is a file that does not grow within one second. If smsd finds such a file, it thinks that the file creation is finished and assumes that the file can now be sent.

Therefore, if you create a SMS file directly in the queue directory, do it quickly. This ensures that smsd does not attempt to send the file before it is completely created.

The file format is simple. It is similar to eMails in Unix. First there is a header with sender, receiver and maybe some more information. The header may contain as much lines as you like. Smsd ignores all unknown lines.

Then an empty line follows. After that the text or binary content follows. In case of binay SM, for example ringtones or logos, the same rule is used. The binary content follows immediately after the empty line.

The picture above shows a valid SMS file.

The outgoing queue allows you to create many SMS files that are send one after the other. You do not need to wait for sending after you create each file.

The next step is, to check if the SMS file is valid. Smsd does this for you.
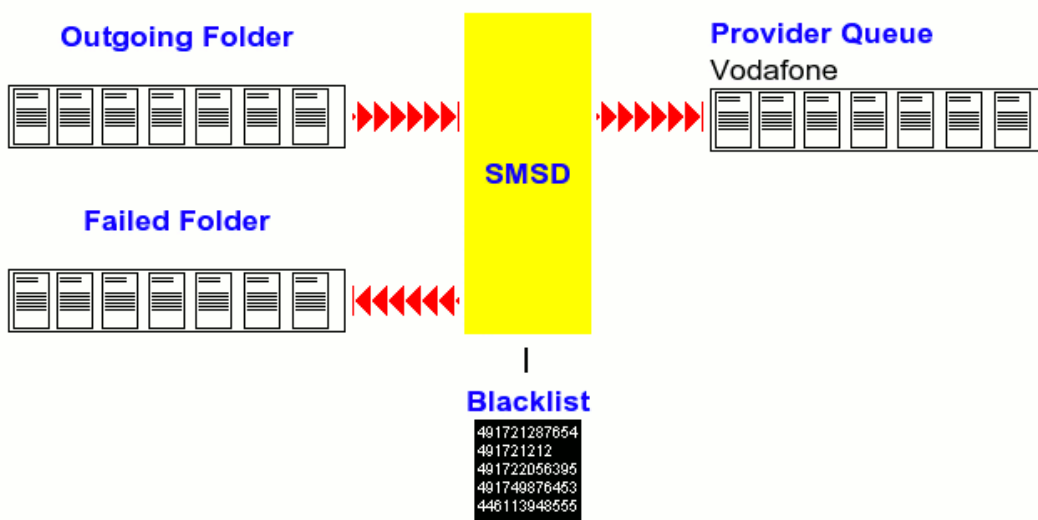
It reads all files in the outgoing queue and compares the recipient numbers with the blacklist. This is a text file that contains all phone numbers that never get a SM. The blacklist can also contain area codes to blacklist a whole area.

Instead of a blacklist you can also use a whitelist. It contains all numbers that are allowed to get SM. If there is a whitelist and a blacklist, then all number in the white list can receive SM but not those who are also in the blacklist. The blacklist is more important.

If the SMS file specifies a provider name, then the files goes into the corresponding provider queue. If the provider name is not specified, then Smsd compares the recipients phone number with the configuration file smsd.conf and assigns the provider by its area code.

Note: After I developed this part the PPRTU project started. It allows mobile customers to change their provider and keep the old phone number. The area code is not a good indicator for the provider name any more. This is not critical for smsd itself but a wrong assignment to provider queues can increase your bill and affects your statistics.

If the SMS file does not include a provider name and the area code is also unknown, the message file is invalid and moved to the failed queue. Such messages will not be sent.

If you do not want to keep failed files you can remove the failed queue directory form the configuration file.

Now the message can be sent.

Any provider queue as at least one attached modem. One program part of smsd (modem task) reads the SMS files from the provider queue and sends it using the sub-program putsms.

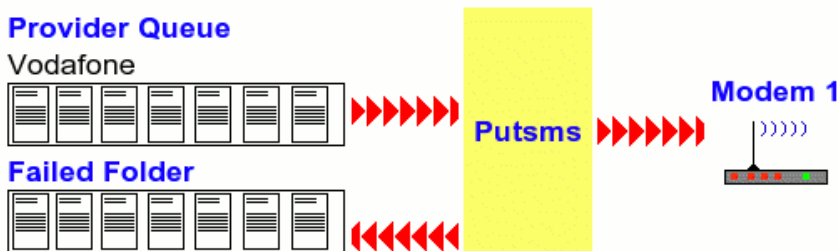You can assign many modems to each provider queue, an option that the pictures of this chapter do not show. This increases the performance when sending many SM. In addition your system will not stop working if a single modems hangs because the other modems continue their work.

To prevent that more than one modem send the same message at a time, smsd uses lockfiles. A lockfile is a file with the same name as the message file but it has the ending ".LOCK". The existence of this file signals the other processes that the message file is already in use.

Smsd writes an information into this file: The process number of the process that created this lockfile. The kernel of any operating system gives any process a number, visible with the ps command. This number has no function for smsd, its only an information for you.

Lockfiles allow you to connect many servers to the same queue directory. If you do so then a single failed computer will not stop your service because the other computers continue to work.

If sending of a message fails also after a second try, the file will be moved into the failed queue directory.

If sending through the same modem fails three times, smsd thinks that the modem is out of order and blocks this modem for a configurable time (for example 1 hour). This ensures that the remaining messages are sent through the other modems - if more modems exist. Therefore it's good to have at least two modems for each provider.

As you can see in the big picture at the beginning of this chapter, it's also possible to use one modem for many provider queues. The configuration file assigns modems to queues. The queue directories are handled with priority. One example with two modems:

Modem1 is for Telekom D1 and Vodafone D2. The modem has a SIM card from Telekom D1.

Modem2 is for Vodafone D2 and Telekom D1. The modem has a SIM card from Vodafone D2.

The Modem2 is normally used for sending messages to the D2 network. Only when there are no messages for D2 it send messages for the secondary queue D1.

The same happens with the Modem1. It sends primarily messages for D1 network and secondarily messages for D2 network if the D1 queue is empty.

What is the advantage of this configuration?

There are many answers:

  1. If the Modem1 goes out of order, then Modem2 is used to send the messages into D1 network. This ensures that your system continues working even if a modem is broken. The same applies to Modem2.

  2. If you send a lot of messages into D2 network and only a few to D1 network the Modem1 helps Modem2 to send messages. It increases the speed. The same happens with Modem2.

  3. If the network of one provider fails, the other modem continues to send the message.

4. You save some money because sending message to the "own" provider network is typically cheaper than sending messages to "foreign" networks. Some contracts give a special low price when you send more than 100 messages within the "own" network. Therefore at least one modem for each provider helps you to save money.

Because of answer 4. the program sorts messages by provider. Without this sorting you would not be able to use the low price in optimum way.

Now I like to show how messages are received.

The program part of smsd that controls the modem (modem task), checks the modem every few minutes or seconds for incoming messages. It uses the sub-program getsms to do this. The received messages are kept in the incoming queue directory.

You can configure which modems are checked for received messages and if sending or receiving has higher priority.

Most modems need higher priority on receiving because if the input buffer is full, they cannot send outgoing messages. In case of doubt you should test if you modem has this problem.

Other functions:
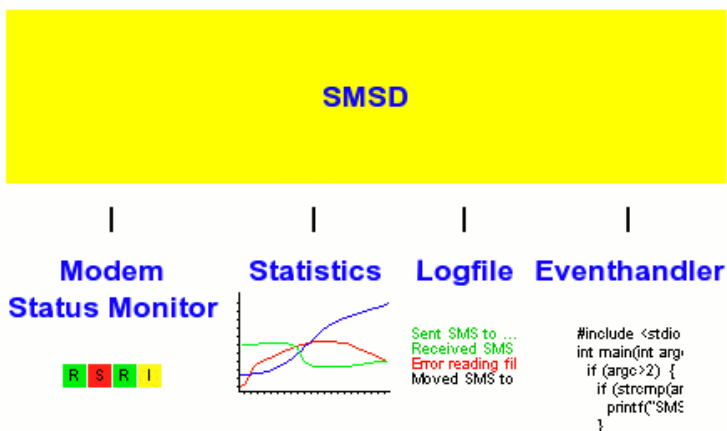
The following functions are secondarily to smsd but still important for many users.

Smsd can run a status monitor. The status monitor shows the state of each modem - if it is sending, receiving, idle or blocked.

Smsd collects statistics about the number of sent and received SM and how much percent the modems are loaded. This statistic helps you to decide if adding more modems increases performance or not. A table calculation program like Microsoft Excel or KSpread can read these statistic files and create beautiful graphs from them.

When you stop smsd or kill it, you do not loose statistic data. This is special to the SMS Server Tools. Most other programs loose all statistic counters of the actual time slice (for example one hour) when you stop the program. But smsd saves these values into a temporary file and loads them back into memory later.

There are logfiles that record any action of the SMS Server Tools. You can configure the detail level. Maybe you want to see only error messages or any smallest detail like the modem commands.

Smsd can write the logfile to screen (standard output channel), into a file or it can use the good old syslog daemon of Unix to record logfiles. If free configurable.

Eventhandler can be executed on special events, for example when a message was sent or received. Eventhandler allow you to combine the SMS Server Tools with other external programs and enhance the function of smsd without much work. A typical usage of this is a cost-counter with SQL database.

Since version 1.10 can also check the validity of SMS files using an external program before it send the message. As you can see in later chapters this is a very useful advantage.

Alarm handler are also eventhandler. Alarm handler are executed whenever an error occurs. The can be used to send alarm messages via eMail or to switch a flash-light on.

5.2. Installation And Configuration

I explained in chapter 4.2, how to install the SMS Server Tools on different operation systems. For reminder, here are the most important commands:

make -s

make -s install

The first command translates the sources into executable programs for your operating system. The second command install the program. The following list shows you, what files are installed (for version 1.15, newer versions may install more files):

/usr/local/bin/smsd	The main program
/usr/local/bin/putsms	Sub-program for sending
/usr/local/bin/getsms	Sub-program for receiving
/usr/local/bin/sendsms	Example shell script, can be used to send a SM from the command line.
/usr/local/bin/sms2html, sms2unicode, unicode2sms	Example eventhandler. This shell script logs sent and received messages and status reports in a MySQL database.
/usr/local/bin/email2sms	Example shell script. Used to storeconvert eMail to a short message. Can be used together with a mailserver to build an eMail to SMS gateway.
/etc/smsd.conf	Template for a configuation file, has to be modified for your system.
/etc/init.d/sms	Startup-script, can be used to start and stop smsd, specially when booting.
/var/spool/sms/incoming	Queue for incoming SM
/var/spool/sms/outgoing	Queue for outgoing SM
/var/spool/sms/checked	Queue for checked outgoing SM

If you update the SMS Server Tools later by a newer version, your configuration files and scripts will be left untouched by the installation command.

Smsd has only one configuration file, and it is normally /etc/smsd.conf. You can use another file using the option -c. Example:

smsd -c /etc/smsd.conf.test

The example configuration file shows you how to set up the SMS Server Tools with one modem. The file was made for a quick start and it's small.

There is another example configuration file in smstools/examples that contains all possible options.

I like to show you all the options now. Let's start with a small and easy configuration file.

5.2.1. Easy Configuration File

An easy configuration file looks like this:

```
devices = GSM1

[GSM1]
device = /dev/ttyS0
incoming = yes
pin = 1111
```

If you compare this file with the large example smsd.conf.full, you will notice that I left out most lines. This is Ok, because smsd uses default values for all missing lines. I will tell you later what values are default.

For the first start it's surely better to start with such an easy configuration file, as shown above.

You can insert as many comments as you like. Mark every comment with a hash character.

With "devices" you specify how many GSM modems you want to use and give them names - delimited by comma.

Then there comes one part for each modem, in this case only one modem.

After "device" you specify the name of the serial port. In Linux the first serial port is named /dev/ttyS0 and the second is named /dev/ttyS1. In Windows their names are /dev/com1 and /dev/com2. Solaris names them /dev/cuaa and /dev/cuab.

The setting "incoming=yes" tells smsd, that this modem is also used for incoming short messages.

SIM cards are normally secured with a PIN. The option "pin=xxxx" tells the SMS Server Tools what PIN should be sent to the modem. Mobile phones do not need this option because they require to enter the PIN via their keypad.

If you deactivate the PIN security, then the program will run a little bit faster. This may be good if you want best performance. To deactivate the PIN, put the SIM card into a mobile phone and use the phones menu to deactivate it. The manual of your phone should explain this.

5.2.2. Many Modems And Provider

Now I show you how to use the SMS Server Tools with many modems and many phone network provider. Take this chapter as an example, you do not need to use it 100% as written here. Play around with other modem-to-provider assignments to find the best configuration for you.

```
devices = GSM1, GSM2, GSM3, GSM4, GSM5
[queues]
D1 = /var/spool/sms/D1
D2 = /var/spool/sms/D2
O2 = /var/spool/sms/O2
EPLUS = /var/spool/sms/EPLUS
QUAM = /var/spool/sms/QUAM
MOBILCOM = /var/spool/sms/MOBILCOM

[provider]
D1 = 49160, 49170, 49171, 49175, 49151
D2 = 491520, 49162, 49172, 49173, 49174
O2 = 49176, 49179, 49159
EPLUS = 49163, 49177, 49178, 49157
QUAM = 49150
MOBILCOM = 49156

[GSM1]
# with D1 Card
device = /dev/ttyN000
incoming = yes
queues = D1

[GSM2]
# with D1 Card
device = /dev/ttyN001
incoming = yes
queues = D1

[GSM3]
# with D2 Card
device = /dev/ttyN002
incoming = yes
queues = D2

[GSM4]
# with D2 Card
device = /dev/ttyN003
incoming = yes
queues = D2

[GSM5]
# with E-Plus Card
device = /dev/ttyN004
incoming = yes
queues = O2, EPLUS, QUAM, MOBILCOM
```

This configuration file does not include PIN numbers, therefore they need to be deactivated on all 5 SIM cards.

The five modems are connected to a Digi Etherlite, because no regular computer has so many serial interfaces. I found the names of the serial ports in the manual of the Digi Etherlite.

The first two modems serve the provider queue for D1. The third and fourth modems serve the queue for D2. The last modem serves all other provider queues.

This configuration doubles the throughput for D1 and D2 and protects the system against a single modem fault. I use only one modem for the other provider to safe some money, because they do not have so many customers.

If you prefer you could also use two modems for the other providers. I like to show you in this example that the number of modems for each provider can be different.

This example does not send messages into foreign countries. If you want that, you need to add an OTHER queue at the end and attach at least one modem to it:

```
[queues]
...
OTHER = /var/spool/sms/OTHER

[provider]
...
```

```
OTHER = 0,1,2,3,4,5,6,7,8,9
```

If smsd find a message, that does not mahch any provider, then it will match to OTHER and is sent by the modem(s) that is attached to this queue.

5.2.3. All settings

Now you learned the basics of configuring the SMS Server Tools and you are ready to learn all the details. The following pages show all settings for smsd.conf. Read this chapter carefully because here are some useful settings that you should not miss.

- The configuration file has four parts:

- Main settings

- Queues

- Providers

- One or more modem settings

The syntax of the file is similar to most Windows configuration files.

Parts begin with a [Part Name], within these parts Values are assigned to variables using the = character:

> Variable = Value

Lists are many values, separated by comma:

> Variable = Value1, Value2, Value3

The whole configuration file is case sensitive, except for boolean values. These are:

> yes, on, true, 1
> no, off, false, 0

You can use whatever boolean you prefer.

5.2.3.1. Main Settings

The main part has no name. It begins at the first line of the configuration file.

devices = list of modem names

Specifies the number and names of your modems. Use only letters, digits and the underscore character. You do not need to name the modems as their really names are. The names have to be unique. They appear in statistics, filenames, logfiles and in event-handler programs.

spool = directory name

Here you specify where to store the outgoing queue. The default value is /var/spool/sms/outgoing.

checked = directory name

Alle outgoing messages are checked and moved into this directory after successful checking. This variable is only used when you do not configure provider queues (see below). The default is /var/spool/sms/checked.

failed = directory name

Here you specify where smsd should stores files that could not be sent. If there is no such entry, smsd deletes such files.

incoming = directory name

Specifies where received messages should be stored. The default is /var/spool/sms/incoming.

sent = directory name

Enter here the directory name where sent messages should be kept. If this entry does not exist, sent messages are not kept anywhere.

mypath = directory name

This setting specified the path where getsms and putsms are installed. The default value is /usr/local/bin.

logfile = filename or number

Name of the logfile. If you do not specify this, then the Unix syslog daemon or Windows eventlog is used.

If you enter a filename, then all informations are written together with a timestamp.

You can also enter a number, it's interpreted as a file handle number. You can use such handles together with shell input/output redirection. The 1 is the standard output channel that is normally the screen. Using redirections you can transfer all logging information to another program or into a file.

loglevel = number

This specifies how many details the logfile should include. The default is 4 if you use a logfile or it is 7 if you use syslog. 0 means that nothing is logged. The log function writes all information into the logfile of which level is smaller or equals the loglevel.

7	debug	All details, including modem commands.
6	info	Informations. The logfile contains many details but not the modem commands that you need for debugging.
5	notice	Informations when a message was sent or received and informations when something not normal happened.
4	warning	Warnings, when a program has a problem, for example sending failed.
3	error	Error messages, when a program has a temporary problem, for example a modem answers with ERROR or a file cannot be read.
2	critical	Critical errors, when a program has a permanent problem, for example sending failed many times or permissions to the queue directories are insufficient.
1		This level is reserved for the operating system kernel and not used by the SMS Server Tools.

The numbers in this table are taken from Linux. I think that all operating system use the same numbers, but I'm not 100% sure.

If you use the syslog daemon or the Windows event log, then you can reduce the detail leve here or in the configuration of your operating system.

alarmhandler = program name

Specifies a program that should run whenever an alarm occurs. This may also be a shell script. Enter the full filename with path. Alarmhandler will be explained later. If you do not need an alarmhandler, leave this out.

alarmlevel = number

Specifies for what alarm levels an alarmhandler should start. You can enter values between 2 and 5.

5	notice	Informations when a message was sent or received and informations when something not normal happened.
4	warning	Warnings, when a program has a problem, for example sending failed.
3	error	Error messages, when a program has a temporary problem, for example a modem answers with ERROR or a file cannot be read.
2	critical	Critical errors, when a program has a permanent problem, for example sending failed many times or permissions to the queue directories are insufficient.

delaytime = number

This specifies the number of seconds that smsd waits if a queue is empty. After this time has elapsed, smsd checks again if there are new SMS files in that queue.

Larger values save CPU and harddisk load. Smaller values like 1 second are good if you want best performance but then the computer should not be used for other programs than SMS Server Tools.

The default is 10 seconds.

errorsleeptime = number

If a modem answers with ERROR, the program can insert an additional delay. Some modems need this before they can accept new commands. The default is 10 seconds.

blocktime = number

When sending failed three times on the same modem, smsd thinks that the modem is broken. Then it block the modem for so many seconds as you specify here. This ensures that the computer does not waste his time talking to a broken modem and that sending a lot of messages fails. The default value is 3600 seconds, that is one hour. Only after one hour delay, smsd tried again to use this modem.

eventhandler = program name

Whenever a short messages was sent or received, or sending failed, then smsd can run an external program. Enter the program name with full path. Eventhandlers are discussed later in this book.

stats = directory

If you write this line then smsd writes statistics into this directory. A later chapter describes how the statistc files look like.

stats_interval = number

This setting tells smsd for how many seconds it should collect statistic data before it writes them into a file. The default is 3600 seconds, that is one hour. Smsd ensures that the timepoints are always the same, it does not matter at what time you start the program. For hourly statistics it write every full hour at 00 minutes into a file and not 60 minutes after program start.

blacklist = filename

This tells smsd the name of the blacklist file. If it is not set then it does not use a blacklist.

whitelist = filename

This tells smsd the name of the whitelist file. If it is not set then it does not use a blacklist.

checkhandler = program name

In addition to the black and white list you can also run external program to check if a SMS file is allowed to be sent or not. Before smsd sorts the file into the provider queue directory, it runs the checkhandler. The checkhandler gets the filename as the first and only argument. If the checkhandler exits with exit-code 0, then smsd send this message. If not, it does not send the message.

If you do not need a checkhandler, then leave this line out.

Autosplit = 0, 1, 2 oder 3

0	Automatic splitting of text messages that are to long is disabled
1	Automatic splitting is enabled, without numbering
2	Automatic splitting is enabled, with numbering
3	Long messages are sent in concatenated format, the receiving phone combines all parts to one message. Not all devices support this format.

Binary and unicode messages (ringtnes, logos, and so on) are never splitted and must not exceed the size of 140 bytes.

receive_before_send = boolean

If you set this to yes then smsd ensures that ate least one memory space is free on the SIM card before it sends a message. Some modems need this because they cannot send messages when the memory is full.

This option lets smsd check for received messages and send one after that. Without this option smsd prefers to send messages before it receives something.

5.2.3.2. Queues

The SMS Server Tools sort outgoing messages into queues for each phone network provider. This allows you to save money using different SIM cards from different provider.

This function is optional. If you do not want it, then do not write [queues] and [provider] sections into the config file.

After the main settings you define the names of the queue directories in the configuration file smsd.conf.

[queues]
Name = Directory Name
Name = Directory Name
...

Use only letters, digits and underscore and do not use special characters. One valid example:

```
[queues]
D1 = /var/spool/sms/D1
D2 = /var/spool/sms/D2
O2 = /var/spool/sms/O2
EPLUS = /var/spool/sms/EPLUS
QUAM = /var/spool/sms/QUAM
MOBILCOM = /var/spool/sms/MOBILCOM
```

If you do not want to sort messages, you need to create only one queue directory as shown below:

```
[queues]
ALL = /var/spool/sms/ALL
```

5.2.3.3. Provider

This section of configuration file lists the area codes of any phone network provider. The software uses this setting to sort outgoing messages into queues.

There must be exactly the same number of rows in the sections [queues] and [provider] and the names must match.

Write area codes always in international format but without heading +.

[provider]
Name = List of area codes
Name = List of area codes
...

Example:

```
[provider]
D1 = 49160, 49170, 49171, 49175, 49151
D2 = 491520, 49162, 49172, 49173, 49174
O2 = 49176, 49179, 49159
EPLUS = 49163, 49177, 49178, 49157
QUAM = 49150
MOBILCOM = 49156
```

5.2.3.4. Modem settings

After the provider area codes you specify some settings for your modems. Each modem gets one section in the configuration file.

Take a look at the line **devices = List of modem names** from the main section. Any modem that is listet heres needs a modem settings section.

You could write more modem settings than the devices line lists. For example if you want to take one modem out of service for a while you can remove it from the devices line and you do not need to remove the modem settings section. Smsd ignores modem settings that are not needed.

The following settings can be written for each modem:

[Modem name]

Each modem has a name. It must be the same than in the devices line. After this section name one or more of the following lines can be written:

init = String

Here you can specify additional modem initialisation commands that the modem executes before it sends or receives messages. You can leave this out in easiest case.

If you write "ATE0" the modem does not echo commands and this makes the logfile a little bit shorter (in loglevel 7) and easier to read. The program will run a little bit faster.

Some GSM modems, specially mobile phones, store messages in the internal memory (RAM) but try to read them from the SIM card. This is bad because you will not be able to read received messages in this case.

If you have this problem, you can initialize your modem with

init = ATE0+CPMS="SM" or

init = ATE0+CPMS="ME"

This tells the modem to store AND read received messages from the SIM card (SM) or internal memeory (ME). Some devices work only correcly with ME, some work only with SM and some other work with both versions.

To receives status reports on Siemens modems you need to initialize them with **AT+CNMI=2,0,0,2,1**.

The default is no init string.

device = Name of serial port

Here you specify the name of the serial port where the modem is connected to.

Linux names the first two serial ports /dev/ttyS0 and /dev/ttyS1. Windows names them /dev/com1 and /dev/com2. Solaris names them /dev/cuaa and /dev/cuab.

incoming = boolean (or number)

If you write here "yes" or "1", smsd reads received messages from the modems memory. The default is "no".

Please note that some modems need at least one memory space free before they can send a message. If you do not want to receive messages and have this problem you could ask your local phone network provider to add an entry to their blacklist, then you will never receive any SM again with this SIM card.

Reading of received messages has lower priority than sending, when you set this to "yes" or "1". Smsd reads the received messages only if there is nothing to send.

If you write a "2" then receiving has higher priority and smsd tries only to send messages when there is nothing to read.

queues = List of queue names

This line attaches the modem to one or more queue directories. Use the same names as in the [queues] and [provider] section.

pin = number

Enter here the secret PIN code of your SIM card that sticks in this modem. You can make the program slightly faster by disabling the PIN because smsd does not need to check if the modem is asking for a PIN or not. Mobile phones do not need a PIN in the configuration files because they require always to enter it via the keypad after power-on. You can disable the PIN of a SIM card by inserting it into a mobile phone and use the security setting menu of the phone. After disabling it you can put the SIM card back into the modem and you can now use it in the modem without entering a PIN after power-on.

The deault is no PIN.

mode = mode

Nearly everx GSM modem runs in mode "new", therefore this is the default setting. Some modems need another setting:

old	Falcom A1 and other very old GSM modems from GSM phase 1
new	All GSM modems from phase 2 or newer
ascii	Onyl for testing. You can only send 140 characters in ascii mode. The program uses clear text to submit a message to the modem instead of PDU coding. Some modems do not support ascii mode

smsc = number

Any GSM modem and any mobile phone use automatically the default SMS centre that the network provider wants. This default setting is stored in the SIM card. You can specify another SMSC by entering its number here. If you use tis setting the program runs slightly slower because it has to send the number to the modem. The default is empty, so the modem uses its default from the SIM card.

Write the number in international format but without the heading +, for example 491722270000.

baudrate = number

Here you specify the speed of the serial communication between modem and computer. All modems that I know work fine with 19200 bits per second. If you modem does not answer to any command you could try 9600. Infrared ports ignore this setting, because the operating system controls the speed. The default is 19200.

rtscts = boolean

If you write here "no", the the SMS Server Tools do not use hardware handshake lines. Some mobile phones and some cheap cables do not support hardware handshake because they have a very cheap hardware.

Whenever possible you should use the default setting that is "yes". Hardware handshake ensures that the computer does not try to talk to fast to the modem.

cs_convert = boolean

The default is "yes" and that means that the program converts the charaters of all short messages to ISO-8859-15 and vice versa. Windows names this character set "Ansi".

You need this conversion the display the message text correctly on Unix and Windows and to send text files correctly as SMS.

If you set this to "no", the text files are transmitted in original form with one exception: The @ character of received messages is translated to ascii code 183, because in the GSM alphabet it has the code 0 and 0 marks always the end of a string in nearly any Windows and Unix program.

report = boolean

If you set this to "yes" the program asks for a status report for any sent message. Not all SMSC aupport this function.

If it does not work then the modem is mostly the weak point. Many modems of Siemens, Digicom and Wavecom cannot receive status reports. And as far as I know all mobile phones can display status reports only on the phones display but you cannot read them through the serial port using a computer.

Falcom modems work absolutely fine with status reports.

eventhandler = program name

Whenever a short message was sent or received or when sending failed, smsd can run an external program. Enter the name of the executable file with full path. I explain eventhandler later.

If you specify an eventhandler in the main section and here the this one here is more important. The SMS Server Tools run always only one eventhandler, and the modem specific eventhandler is preferred.

5.3. start/stop SMS Server Tools

This chapter teaches you how to start and stop the SMS Server Tools

The main program of the SMS Server Tools is smsd. All other programs are run by smsd as sub-programs. You will never need to use the sub-programs directly.

The SMS Server Tools include a basic start/stop script that you can use to start and stop the program. During installation it is copied to /etc/init.d/sms or /sbin/init.d/sms - depending on your operating system version.

Enter **/etc/init.d/sms start** to start the program. The script deletes old lockfiles and runs then smsd in background. You can check this using the command **ps -ef | grep sms**.

Enter **/etc/init.d/sms stop** to stop the program. The script uses ps to find the process id of smsd and then kills it.

The following lines are only for Unix, that means Solaris and Linux:

If you want to start the SMS Server Tools automatically during bootup, then please create a link to the start/stop script in the correct runlevel directory. Their names are typically /etc/init.d/rc0.d until rc6.d. Some Linux versions name them /sbin/init.d/rc0.d until rc6.d.

You need to find out what runlevel is the correct one, because there are 7 directories. I explained in chapter 4.2, how to find the correct runlevel directory. But I repeat it here for convenience:

Enter **grep :initdefault /etc/inittab** and keep the number between two colons in mind, in most cases this is a 3. This number tells you in what directory the link needs to be created, for example /etc/init.d/rc3.d.

Now go into this directory and enter **ln -s /etc/init.d/sms S82sms**. Now the SMS Server Tools are started automaticlly when the server boots.

You do not need to think about system shutdown. The operating system kills all programs during shutdown shortly before the power is switched off. This is acceptable for the SMS Server Tools. A "clean" shutdown is normally not necessary.

Windows NT, 2000 and XP can start the SMS Server Tools automatically as a service during boot process. You might install smsd as a service by entering this command:

cygrunsrv --install smsd --path /usr/local/bin/smsd.exe --type auto --neverexits --shutdown --env "CYGWIN=server" --env "PATH=/usr/local/bin:/usr/bin:/bin" --desc "SMS Server Tools"

Start the service by entering the windows command **net start smsd** or using the control panel.

The program smsd can show a status monitor while it is running in a window. To activate this you need to start it with the option -s. Chapter 5.8 explains, how to use this option.

5.4. Sending Short Messages

The easiest way to send short messages is to use the script sendsms. It is used in this way:

sendsms +491721234567 "Hello, How are you?"

or

sendsms +491721234567
"Hello, How are you?"

The script can also be used with input redirection, for example:

echo "The date ist `date` " | sendsms +491721234567

The echo command creates a text and inserts the output of the date command. Then the text is redirected to the sendsms script.

If you want to take a look into the sendsms script you will see that it's very easy. It creates only a text file with destination number and text in the outgoing queue directory:

```
FILE=`mktemp /var/spool/sms/outgoing/send_XXXXXX`
echo "To: $DEST" >> $FILE
echo "" >> $FILE
echo "$TEXT" >> $FILE
```

The first line creates a temporary file and stores its name in the variable FILE. Mktemp is a standard Unix command that replaces XXXXXX by random characters ensuring that any file gets an unique name.

The next echo commands write the phone number, an empty line and the text into the temporary file.

You can also do the same yourself with any self written script or with a normal text editor. A correct file looks like this:

```
To: 491721234567

Hello
```

Do not forget the empty line and write the phone number always in international format. Apart from the configuration file you may write phone numbers with + or without +. Smsd removes it when necessary.

Store such SMS files with any filename in the outgoing queue directory /var/spool/sms/outgoing. Smsd does not check the filename, only the content is important, therefore you may use any filename that you like.

You should ensure that any message has its own filename. Otherwise you may accidentally overwrite old files that are not sent yet.

5.5. Receiving Short Messages

Receiving short messages is a passive process. The SMS Center (SMSC) send you messages as soon as you modem is ready to receive. You simply need to read out the received messages.

The SMS Server Tools do this job for you, if you enabled this feature in the configuration file:

```
incoming = yes
```

The received messages are stored into text files in the incoming spool directory, that is normally /var/spool/sms/incoming.

If you need to to know what modem the message received or when it was received, then take a look into the file. All these informations are in the files:

```
From: 491721234567
From_SMSC: 491722270333
Sent: 00-02-21 22:26:23
Received: 00-02-21 22:26:29
Alphabet: ISO8859-15
Subject: GSM1

Hello, how are you?
```

The first line tells you who the sender was. The second line is the SMS centre that delivered this message to your modem. Then timestamps follow, when the messages was sent by the sender and when it was received by your computer.

The subject line contains the modem name that received the message. The author of the program decided to store this information in the subject line because this makes it easy to forward the message as an eMail.

5.6. SMS File Format

SMS files are similar to eMails. They begin with a header and end with the message text. Between both parts is an empty line, that is the delimiter:

```
Header

Text
```

As I wrote already, the filename does not matter. The only important thing with filenames is that they have to be unique.

The header may contain different informations depending on what you want to send or what you received. The next pages will show the details.

The order of lines in the header does not matter. Some headers use boolean values. In this case, the values true, yes, on and 1 mean all the same. Other keywords are treated as false (off).

Smsd ignores all unknow informations in the header. Therefore you may add notes like this:

```
Note: only a test
```

Smsd does not know the keyword "Note:" and therefore it ignores the whole line.

Line breaks have to be either in Unix or DOS format. Received messages use always line breaks in Unix format.

The different between both formats is:

Unix stores line breaks with one control character, code 10 (decimal).

Dos stores line breaks with two control characters, code 13 and 10.

If you try to read a message file with Notepad in Windows you will not succeed. Notepad cannot read text files in Unix format. But nearly every other text editor can do it, for example Wordpad, Word and Editpad.

5.6.1. Sending Text

To send a SM, you need to write at least the receivers number and the message text into the SMS file. A simple file looks like this:

```
To: +491721234567

Hello, how are you?
```

As I wrote earlier the destination phone number has to be written in international format, with or without the plus.

If you like you can add more informations to the header of the SMS file, for example:

```
From: Stefan
To: 491721234567
Flash: yes

Hello, how are you?
```

From: Senders name or phone number. This field has currently no function to the software.

To: Receivers phone number in international format without the leading +. When you like to send a message to a short number (for example to order a ringtone), then preceed it with an "s".

Flash: Boolean value. If yes, then the message appear directly on the phones display. Most phones support this feature, but not all. Such messages are also often called Alarm message or Class-0 message.

Alphabet: Tells the program what character set is used in this sms file. Possible values are

ISO or **Latin** or **ANSI**: Normal 8 bit character set, also called Ansi or Latin-1.All three keywords do the same. If the text is longer than 160 characters, then the program can split it automatically.

GSM: 7 bit character set, as described in the GSM specification, with one exception: The @ character is represented by the character code 0xB7. If the text is longer than 160 characters, the program can split it automatically.

UCS or **Chinese** or **Unicode**: UCS2 character set, maximum 70 characters. All three values do the same. The header must be written with an 8 bit character set but the message text part must be written with the 16 bit Unicode (big endian) character set. Please checkout the scripts directory, it contains some useful scripts for file format conversion. More details see below.

The program checks only the first 3 characters of this field, therefore keywords like ISO-8859-15 or UCS-2 will also work fine.

You may use the following header lines, to override settings from the configuration file:

SMSC: Phone number of the SMSC

Provider: or **Queue:** Name of the provider, can be used to override the normal sorting algorithm configured by [provider] and [queues] in the config file. Both keywords do the same.

Report: Boolean value. Controls if a status report is requested for this message.

Autosplit: Boolean value. Controls if the program shall split the message automatically in case it is larger than 160 characters.

5.6.2. Sending Binary Data

Binary data are messages with 8-Bit Data. I dependa highly on the receivers phone how they display such messages. Many phones know a special binary file format for ringtones and operator logos.

The SMS Server Tools have no special functions for ringtones and operator logos. You can send binary file 1:1 to the phone, the file content does not matter.

If you like to compose ringtones yourself or like to edit images and create operator logos from them you need additional software. The SMS Server Tools do not assist you in creating such files.

If you like to learn more about this topic you should read the specifications about binary messages for each phone that you like to support. Any manufacturer has its own specification.

Binary short messages can contain maximum 140 bytes user data. Smsd does never split binary messages if they are to large!

A binary SMS file looks like this:

```
To: 491721234567
Alphabet: binary
UDH: true

gs2389gnsakj92fs2ujtiegbhewqu2ir9jsdgufh3
gjeruqgh87zt243htgerugsqh
3gert3245
```

Alphabet:binary tells smsd that this short message contains 8-Bit data.

UDH: true or "UDH: false" specifies how the UDH flag in the message should be set. This flag has no meaning to the SMS Server Tools and to the SMS centre but some mobile phones require that this bit is set or not set.

If you are not sure, then set this bit to "yes" because most phones require it (for example Nokia). If you leave this line out, smsd uses the default that is "yes".

Like in sent test messages you can add more informations to the header, these are sender, provider and the flash-flag.

The binary data begins immediately after the empty line and ends at the file end.

5.6.3. Sending Unicode Text

Unicode is a 16-bit character set that inclused latin characters and also chinese, arabic and much more characters. It's also called UCS2. There are two different versions, a "big endian" and a "little endian". The GSM network uses the "big endian" version. Unicode messages are limited to 70 characters.

The structure of Unicode messages is basically the same as for binary messages. The header is written in latin characters (iso8859-1), followed by an empty line, followed by the message text in Unicode "big endian" format:

```
To: 491721234567
Alphabet: UCS2

gs2389gnsakj92fs2ujtiegbhewqu2ir9jsdgufh3
gjeruqgh87zt243htgerugsqh
3gert3245
```

I developed two small shell scripts that simplify reading such Unicode messages. The script sms2html converts a received SMS file into HTML. The script sms2unicode converts a received message into a pure Unicode textfile:

```
sms2html received.sms > new.html
sms2unicode received.sms > new.html
```

The script unicode2sms was written for the other direction. It converts a pure Unicode textfile into a proper SMS file:

```
unicode2sms message.txt > send.sms
```

For Windows XP users, I can recommend to use Notepad for editing Unicode textfiles. It can load and create such files properly.

5.6.4. Receiving Text

Received text messages look like this example:

```
From: 491721234567
From_SMSC: 491722270333
Sent: 00-02-21 22:26:23
Received: 00-02-21 22:26:29
Alphabet: ISO8859-15
Subject: GSM1

This is the Text that I sent with my mobile phone.
```

Newer program versions may create larger headers with more informations. Keep that in mind if you develop programs that read the header.

The **From:** line tells you the senders name or number. Mobile phones and GSM modems can only send numbers but companies that use large accounts can also send names.

The **From_SMSC:**"line contains the number of the SMS centre that delivered this message.

Now timestamps follow. **Sent:** is the time when the message was sent. This filed is filled by the SMS centre. Do not trust that this value is very exactly because the system clock of the SMSC may be wrong.

After **Received:** the SMS Server Tools write the date and time, when your computer read this message from the modem. This value depends on the system clock of your computer.

The **Subject:** line contains the name of the modem that received the message. Depending on your applications this information can be very important or completely meaningless.

5.6.5. Receiving Binary Files

Binary files look similar to text messages. Instead of the text you will find the binary data. The header contains additional lines with the settings "Alphabet: binary" and "UDH: true" or "UDH: false".

```
From: 491721234567
From_SMSC: 491722270333
Sent: 00-02-21 22:26:23
Received: 00-02-21 22:26:29
Subject: GSM1
UDH: true
Alphabet: binary

gs2389gnsakj92fs2ujtiegbhewqu2ir9jsdgufh3
gjeruqgh87zt243htgerugsqh
3gert3245
```

Like in sent messages the binary data begins immediately after the empty line and end at the file end.

5.6.6. Received Status Reports

If you request a status report you get at least one status report back from the SMSC for each sent message. The configuration file smsd.conf has one line for this in the modem setting "report=true".

Status reports are messages with binary content. The SMS Server Tools translate this into readable text to simplify your application.

```
From: 491721234567
From_SMSC: 491722270333
Sent: 00-02-21 22:26:23
Received: 00-02-21 22:26:29
Subject: GSM1

SMS STATUS REPORT
Message_id: 117
Discharge_timestamp: 00-02-21 22:27:01
Status: 0,Ok,short message received by the SME
```

The header hat the well known content. It is identical to received text messages. The text part informs in the first line that this is a status report.

The **Message_Id** is an increasing number that any GSM modems gives to sent messages. Status reports use this number to refer to a sent message.

But this number has a small problem. It starts with 0 and increases until 255. Then the next message gets again a 0. The small range of 0-255 makes an exact assignment of sent and received messages impossible.

The **Discharge_Timestamp** tells you when the message was deleted from the queues in the SMSC. This happens whenever a messages was delivered or deleted after a permanent error. If a message cannot be delivered but remains in the queue of the SMSC, then this timestamp contains a lot of zeroes.

The last line reports the status, first a number and then a text-translation of this number:

0	Ok,short message received by the SME
1	Ok,short message forwarded by the SC to the SME but the SC is unable to confirm delivery
2	Ok,short message replaced by the SC
32	Still trying,congestion
33	Still trying,SME busy
34	Still trying,no response from SME
35	Still trying,service rejected
36	Still trying,quality of service not available
37	Still trying,error in SME
64	Error,remote procedure error
65	Error,incompatible destination
66	Error,connection rejected by SME
67	Error,not obtainable
68	Error,quality of service not available
69	Error,no interworking available
70	Error,SM validity period expired
71	Error,SM deleted by originating SME
72	Error,SM deleted by SC administration
73	Error,SM does not exist
96	Error,congestion
97	Error,SME busy
98	Error,no response from SME
99	Error,service rejected
100	Error,quality of service not available
101	Error,error in SME

All other codes are translated to "unknown". Some text translations may appear useless but they appear exactly as written here in the ETSI TS 100 900 specification.

In most cases you will probably see the values 0 and 70.

5.6.7. Timestamps

On the last pages you saw some timestamps. They have always the format:

YY-MM-DD HH:MM:SS

YY	is the year
MM	is the month
DD	is the day
HH	is the hour
MM	are the minutes
SS	are the seconds

All numbers have always two digits. Timestamps may contain a lot of zeroes in case this information is missing in the short messages:

00-00-00 00:00:00

The SMS Server Tools use this way of displaying unknown timestamps because MySQL does it in the same way for date/time fields.

5.7. Modem Selection

This chapter shall help you to select the correct GSM modem for your application.

The most asked question to me is "What modem do you recommend?". The answer is easy because there are not many modems available on the market.

All devices from Falcom with serial interface worked always fine in the past. As I started development of the SMS Server Tools there was only one Falcom modem available, the A1. Falcom sold a lot of different devices until now, the current device is Samba. They all worked fine, therefore I assume that also future Devices will work fine.

The Falcom A2D and A3D can be connected to an external antenna. The Alarm Firmware allows you to use this modem standalone without a computer to send Alarms and GPS positions via SMS.

Please do not use mobile phones in commercial SMS applications. The first bad thing with mobile phones is that you cannot control them remotely (for example power on/off). They are not made for permanently usage 24h each day. Many phones do not run stable enough. They are often fine for mobile usage but as soon as connected to a computer a lot of problems occur. Most problem reports that I get refer to mobile phones!

For selection of GSM modems the following criteria are important:

- Does the modem have a serial interface?

- Can you connect an external antenna (FME connector)?

- Does the modem support SMS commands according to GSM 07.05 or GSM 07.07?

- Does the modem support status reports?

- Can the modem send and receive binary (8 bit) messages?

- Does the modem run stable?

5.8. Status Monitor

The SMS Server Tools can inform you about the status of each GSM Modem. You simply need to start smsd with the option -s.

After start of smsd it shows the modem status:

```
iiiriisssisss---
iiiriisssisss---
rriiiiissisis---
rriiiiiiiiiiii---
iiiiiiiiiiiir---
```

Whenever something changes, a new line appears. SO the last line contains the actual state. There is one character for each modem. The are shown in the same order as the "devices=" line in the configuration file. The example above has 13 modems.

s   sending

r   receiving

i   idle

b   blocked (after many failed retries)

-               no modem

You can start smsd at the command line:

smsd -s

After start you see the status report as shown above. You can stop smsd by pressing <Ctrl>-C.

Many users prefer to start the program automatically during boot. They prefer to use the start/stop script. But now the program is not allowed to write text onto the screen because it runs as a background program.

If you want you can still see the status by using a small trick. You simply need to modify your start/stop scrpt a little bit:

```
...
    /usr/local/bin/smsd -s >/var/log/smsd.status &
...
```

Now smsd writes status informations into a text file with the name /var/log/smsd.status. You can read the file later using the command:

tail -f -n1 /var/log/smsd.status

This command shows you always the last line of the logfile and waits endless that the file grows. The key <Ctrl>-C can be used to stop the tail command.

Now you should not forget that the logfile grown permanently and some time your harddisk will be full. A cronjob can help out, that deletes the logfile every night. Enter this into your cron table:

```
0 0 * * * cat /dev/log /var/log/smsd.status
```

Now the status protocol will be deleted every midnight. This should be enough.

How to work with cronjobs was explained in a previous chapter.

5.9. Eventhandler

Eventhandler are programs that are run whenever

- a short messages was RECEIVED

- a status REPORT was received

- a SM was SENT

- sending FAILED.

You can use eventhandler to react on events. The examples at the end of this book will show you how eventhandlers can be used meaningful.

Smsd installs an example script (smsevent) that does the following:

- It shows the arguments on screen

- It changes the file permissions of received messages so anybody can read them.

- It forwards received messages via eMail to the administrator (root)

- It forwards received messages via SM to the phone number 491721234567

You can use this script as a template for your own scripts.

When the SMS Server Tools run an eventhandler, the give always two or three arguments:

$1	Type of event (RECEIVED, REPORT, SENT or FAILED)
$2	Name of the SMS file
$3	ID Number of the message (only for SENT messages)

Sent messages are either deleted or moved into the sent queue or moved into the failed queue directory, depending on the success and the settins in the configuration file.

Eventhandler are always run before the files will be moved or deleted. The seconds argument is always a readable SMS file in a provider queue directory or incoming queue.

A very simple eventhandler that forwards each received messages via eMails looks like this:

```
#!/bin/sh
if [ $1 = RECEIVED ]; then
   cat $2 | /usr/sbin/sendmail root
fi
```

5.10. Alarm Programs

Alarm programs (alarmhandler) are external programs that are started by the SMS Server Tools to indicate a problem. Use them as an addition to the logfile.

You could send eMails or switch a flash-light on when the SMS Server Tools have a problem.

Many large companies have their own central tools that collects alarms from all machines. The SMS Server Tools can easily be connected to such systems using alarm programs.

The severity level that triggers an alarm program is configurable with the line "alarmlevel=..." in the configuration file smsd.conf.

The alarm program gets the following arguments:

$1	The keyword ALARM
$2	The date in format YYYY-MM-DD
$3	The time in format hh:mm:ss
$4	The severity (one digit)
$5	The name of the modem or SMSD
$6	The alarm text (is the same as in the logfile)

5.11. Black And White Lists

Blacklists are text files that list all phone numbers that should never receive SM. You activate a blacklist by entering the filename into the configuration file:

```
blacklist = /etc/smsd.black
```

If you do not have a blacklist, then anybody can get SM.

Whitelists are text files that list all phone numbers that are allowed to get SM. Number that are not listed do not get SM.

```
whitelist = /etc/smsd.white
```

If you have no whitelist, anybody can receive messages, except those that are listed in the blacklist.

What happens if a phone number is in both lists? The the blacklist is more important, the number gets no SM.

The lists may contain comments. Comments start wih hash. Example:

```
#This is the blacklist
491721234567 # Is not allowed to receive SM
491721111111 # Does not want SM. 7.4.2003
```

If your black or white list becomes very large it may be better to use a checkhandler. This is a program that allows or disallows sending messages by comparing the phone number with a SQL table. The examples at the end of this book show you how to write a checkhandler program.

5.12. Statistic Files

Statistic files will be written any few minutes, hours or days by smsd, when you enable this feature. These files inform you about the modem load and the number of sent and received messages. Example:

```
runtime,rejected
1200,1

name,succeeded,failed,received,multiple_failed,usage_s,usage_r
GSM1,20,0,1,0,80,0
GSM2,5,0,1,0,40,900
```

This examples was taken from a system with two GSM modems. The file format allows you to read it with any spreadsheet application like Microsoft Excel or KSpread.

There is always one line with global counters and then one line for each modem with modem-specific counters.

- Global counters:

- Runtime of the program in seconds since last statistic file.

Number of rejected SM, e.g. blacklisted number

- Counter per modem:

- Name of the modem

- Number of successfully sent messages

- Number of failed sent messages

- Number of received messages

- Number of multiple errors (modem blocked)

- Number of seconds that the modem was used to send

- Number of seconds that the modem was used to receive

All counters are reset to zero as soon as they are written into a file. The filename is composed of the date and time in the format YYMMDD.hhmmss.

YY	year
MM	month
DD	day
hh	hour
mm	minute
ss	second

The SMS Server Tools ensure that statistics are written every day at the same times. For example if you write that smsd should write every hour a file (3600 seconds), then this happens every full hour when the minutes and seconds are zero.

Smsd handles the statistics carefully. Different to most other programs does smsd not forget the values in memory when you stop or restart the program. Also when you kill the program, the counters are not lost als long as you do not use the "hard" kill -9.

When the program exist it stores the counter values from memory into a temporary file. When the program starts next time, it reads the values back into memory, so they will appear in the next statistic file. You will not loose counters.

Please note that newer program versions may add more counters. They will be added at the right end of the rows. If you write programs that read these files you should keep this in mind.

5.13. Logging Into SQL Database

Logging into a MySQL database is specially useful for commercial SMS Services. This allows you to collect billing data and to check how many SM were successfully sent.

The log database stores a list of any sent and received message. Sent messages contains also the status code returned as status reports by the SMSC.

The logging into a SQL database is not direct function of the SMS Server Tools. It was added by using a shell script that has to be run as an eventhandler. If you want to write your own script that connects the SMS Server Tools to a SQL Database this script is a perfect template because it is not very complex.

The name of that script is mysmsd and the installation program copies it to /usr/local/bin. Open the script with your preferred text editor and modify the variables at the beginning to your environment:

```
SQL_HOST=localhost
SQL_USER=root
SQL_PASSWORD=""
SQL_DATABASE=smsd
SQL_TABLE=sms_log
```

SQL_HOST ist the sevrer name that runs the SQL database.

SQL_USER ist the username that the script need to log on to the database.

SQL_PASSWORD is the password for the username. Write it between quotes.

SQL_DATABASE is the name of the database that the SMS Server Tools should use.

Normally you need to change only the password and hostname. Edit the configuration file /etc/smsd.conf and insert the name of this eventhandler:

```
eventhandler = /usr/local/bin/mysmsd
```

Check if the following programs are installed, because the script needs them:

mysql client
formail
sed
cut

Now you need to create a database with an empty table that will be filles with logging data. The table structure has to be this one:

```
+-------------+----------+------+-----+---------+----------------+
| Field       | Type     | Null | Key | Default | Extra          |
+-------------+----------+------+-----+---------+----------------+
| id          | int(11)  |      | PRI | NULL    | auto_increment |
| type        | char(16) | YES  |     | NULL    |                |
| sent        | datetime | YES  |     | NULL    |                |
| received    | datetime | YES  |     | NULL    |                |
| sender      | char(32) | YES  |     | NULL    |                |
| receiver    | char(32) | YES  |     | NULL    |                |
| status      | char(3)  | YES  |     | NULL    |                |
| msgid       | char(3)  | YES  |     | NULL    |                |
+-------------+----------+------+-----+---------+----------------+
```

Enter the following commands to create this table:

```
stefan@server> mysql -u root -p
mysql> create database smsd;
Query Ok...
mysql> use smsd;
Database changed.
mysql> create table sms_log (
    -> id int auto_increment not null,
    -> primary key(id),
    -> type char(16),
    -> sent datetime,
    -> received datetime,
    -> sender char(32),
    -> receiver char(32),
    -> status char(3),
    -> msgid char(3)
    -> );
Query Ok...
```

Now you prepared everything and can restart the SMS Server Tools to activate the changes. Now smsd creates database entries using the script mysmsd for any sent and received SM.

After three actions the table could look like this example:

```
+---+--------+-------------------+-------------------+------------+------------+-------+------+
|id |type    |sent               |received           |sender      |receiver    |status |msgid |
+---+--------+-------------------+-------------------+------------+------------+-------+------+
| 1 |RECEIVED|2000-02-21 22:26:23 |2002-06-06 12:16:23|491721234567 |MODEM1      |  NULL | NULL |
| 2 |SENT    |2002-06-06 12:16:34 |2002-06-06 12:16:59|somebody     |491721234567 |    0 |  117 |
| 3 |FAILED  |2002-06-06 12:16:48 |NULL               |somebody     |491721234567 |  NULL | NULL |
+---+--------+-------------------+-------------------+------------+------------+-------+------+
```

The first row tells you that the MODEM1 received a message at 22:26. It was sent at 12:16 by the sender 01721234567.

The second row tells you that you sent a message at 12:16 to 01721234567 and that this message was delivered successfully at 12:16:59 (status=0). The senders name "somebody" comes from the From: line in the SMS file.

The third row tells you that you tried to send a message at 12:16 to 01721234567 but sending failed.

Now I show you some example SQL commands that you can use to query this table:

To query the whole table, simply enter in the mysql client

select * from sms_log

To count all messages to a special destination number, enter

select count(*) from sms_log where type="SENT" AND receiver="491722056395";

The command

select * from sms_log where type="SENT" AND receiver="491721234567";

creates a list of all sent messages to a special destination number. The next command lists all received messages:

select * from sms_log where type="RECEIVED";

```
+---+--------+-------------------+-------------------+------------+------------+-------+------+
|id |type    |sent               |received           |sender      |receiver    |status |msgid |
+---+--------+-------------------+-------------------+------------+------------+-------+------+
| 2 |RECEIVED|2000-02-21 22:26:23 |2002-06-06 12:16:23|491721234567 |MODEM1      |  NULL | NULL |
+---+--------+-------------------+-------------------+------------+------------+-------+------+
```

5.14. Explanation Of Mysmsd

This chapter explains, how the script mysmsd works.

Smsd runs this script whenever a messages was sent or received or when sending failed. The commands are:

/usr/local/bin/mysmsd SENT Filename ID-Number
/usr/local/bin/mysmsd FAILED Filename
/usr/local/bin/mysmsd RECEIVED Filename
/usr/local/bin/mysmsd FAILED Filename

The filename is the name of theSMS file.

```
#!/bin/sh
SQL_HOST=localhost
SQL_USER=root
SQL_PASSWORD=""
SQL_DATABASE=smsd
SQL_TABLE=sms_log

DATE=`date +"%Y-%m-%d %H:%M:%S"`
FROM=`formail -zx From: < $2 | sed 's/"//g'`
TO=`formail -zx To: < $2`
SUBJECT=`formail -zx Subject: < $2`
SENT=`formail -zx Sent: < $2`

if [ "$SQL_PASSWORD" != "" ]; then
  SQL_ARGS="-p$SQL_PASSWORD";
else
  SQL_ARGS="";
fi

SQL_ARGS="-h $SQL_HOST -u $SQL_USER $SQL_ARGS -D $SQL_DATABASE -s -e"

if [ "$1" = "FAILED" ] || [ "$1" = "SENT" ]; then

   mysql $SQL_ARGS "insert into $SQL_TABLE (type,sent,sender,receiver,msgid) values
(\"$1\",\"$DATE\",\"$FROM\",\"$TO\",\"$3\");";

elif [ "$1" = "RECEIVED" ]; then

   mysql $SQL_ARGS "insert into $SQL_TABLE (type,sent,received,sender,receiver) values
(\"RECEIVED\",\"$SENT\",\"$DATE\",\"$FROM\",\"$SUBJECT\");";

elif [ "$1" = "REPORT" ]; then

   DISCHARGE=`sed -e 1,7d < $2 | formail -zx Discharge_timestamp:`
   MSGID=`sed -e 1,7d < $2 | formail -zx Message_id:`
   STATUS=`sed -e 1,7d < $2 | formail -zx Status: | cut -f1 -d,`

   ID=`mysql $SQL_ARGS "select id from $SQL_TABLE where receiver=\"$FROM\" and
type=\"SENT\" and msgid=\"$MSGID\" order by id desc limit 1;"`

   mysql $SQL_ARGS "update $SQL_TABLE set received=\"$DISCHARGE\",status=\"$STATUS\" where
id=\"$ID\";"

fi
```

```
DATE=`date +"%Y-%m-%d %H:%M:%S"`
```

The date command show the actual date in the given format. An output redirection put the result into the variable DATE.

```
FROM=`formail -zx From: < $2 | sed 's/"//g'`
```

%2 is the name of the SMS file. Formail extracts the senders name or number. The input of formail is redirected to the SMS file $2 using the character "<" because formail can only read from its standard input channel.

The senders name is after the keyword "From:" in the SMS file header. We use sed to remove unwanted double quotes, if there are some of them. The result is written to the variable FROM using output redirection.

```
TO=`formail -zx To: < $2`
```

We use the same method to get the phone number of the receiver from the SMS file.

```
SUBJECT=`formail -zx Subject: < $2`
```

And again we use the same method to extract the subject line from the file. In case of received messages this line contains the modem

name.

```
        SENT=`formail -zx Sent: < $2`
```

At last we extract the timestamp when the message was sent.

We have now the following informations:

DATE = Actual date and time
FROM = Sender
TO = Receiver
SUBJET = Name of the modem
SENT = Date and time, when the message was sent

```
        if [ "$SQL_PASSWORD" != "" ]; then
          SQL_ARGS="-p$SQL_PASSWORD";
        else
          SQL_ARGS="";
        fi
```

The if-control structure checks if a password is set. If yes, then it is written into the variable SQL_ARGS that we use later to give it to the SQL client.

If there is no password, then SQL_ARGS remains empty for this moment.

```
        SQL_ARGS="-h $SQL_HOST -u $SQL_USER $SQL_ARGS -D $SQL_DATABASE -s -e"
```

Now more options for the SQL clients are collected, these are: hostname, username, password (the old content of $SQL_ARGS) and the name of the database.

At this point we have all options for the MySQL client and we can send SQL queries in this way:

```
        mysql $SQL_ARGS "..."
```

Instead of the three dots we can insert any SQL command.

Now it is important what type of event occured. Different SQL commands are necessary for sending, receiving and status reports.

```
        if [ "$1" = "FAILED" ] || [ "$1" = "SENT" ]; then
            ...;
        elif [ "$1" = "RECEIVED" ]; then
            ...;
        elif [ "$1" = "REPORT" ]; then
            ...;
        fi
```

For sent and failed sendings we can use the same SQL command. Another second command is applied for received messages, and a third command for status reports. The control structure uses $1 to detect the type of event.

Successful and failed sent messages are inserted into the database using this command:

```
        insert into $SQL_TABLE (type,sent,sender,receiver,msgid)
                values ("$1","$DATE","$FROM","$TO","$3");
```

The source code of the script contains a backslash before any double quote because quotes within the same quotes are not allowed.

This command creates a new table row with these values:

type   SENT or FAILED
sent   actual date
sender   senders name or number
receiver   receivers number
msgid   ID-number of the message

Received messages are entered into the database by a slightly different SQL command:

```
        insert into $SQL_TABLE (type,sent,received,sender,receiver)
                values ("RECEIVED","$SENT","$DATE","$FROM","$SUBJECT");
```

This creates a new table row with these data:

type	RECEIVED
sent	date and time of sending

received	actual date
sender	Senders name or number
receiver	Name of the modem

Status reports contain some more informations that we did not extract from the message file at this point. This is done at this point where it is clear that we have actually a status report:

```
DISCHARGE=`sed -e 1,7d < $2 | formail -zx Discharge_timestamp:`
MSGID=`sed -e 1,7d < $2 | formail -zx Message_id:`
STATUS=`sed -e 1,7d < $2 | formail -zx Status: | cut -f1 -d,`
```

The first command is the timestamp when the SMSC delivered the message. The second line extracts the message ID number that refers to the original message for this status report. The third line extracts the status code.

A new command is sed. We use it because formail reads only the message header, not the message text field. But the status information is in the text field.

The command

```
sed -e 1,7d < $2
```

reads the file and cuts the first seven lines (1,7d) from it using sed. The rest is the text part of the file and now we can use formail to extract fields as done before with the header.

The output of sed is redirected to the input channel of formail using the pipe character "|".

The third command contains an additional redirection:

```
cut -f1 -d,
```

The status code is in one single row together with the status text, for example:

```
0,Ok,short message received by the SME
```

The cut command splits the line into three fields and returns only the first field (-f1). The delimiter between the fields is the comma (-d,).

Now we have all status information, these are:

DISCHARGE	when the message was sent
STATUS	status code is 0 on successful delivery
MSGID	The id number of the message that belongs to this report

Status reports are not simply added to the table. For the previously sent message is already a row in the table. Our job is now to find out which row it is and add the status code to exactly this already existing row.

We do not need to search the whole table manually, we simply ask MySQL, what row matches our criteria. The criteria is the ID number of the sent message. Therefore the SQL command is:

```
select id from $SQL_TABLE where receiver="$FROM"
          and type="SENT"
          and msgid="$MSGID"
          order by id desc limit 1;
```

That means in words: Give me the ID from the table row in table sms_log, where the receiver matches the status report and the message was previously sent and the Message-ID number matches the status report. If there are many matching rows, then give me only the last one.

The ID-Number has nothing to do with the Message-ID number of the SM. It is only a number for the row.

ID   The number of the table row in the database

Now we know what row number in the table contains the sent message for the actual status report. We use this knowledge to add the status code to this row:

```
update $SQL_TABLE set received="$DISCHARGE",
          status="$STATUS"
          where id="$ID";
```

6. Useful Enhancements

You learned what programs you can use to sent and received short messages.

You can install and use these programs.

This chapter shows how to enhance the functions of the SMS Server Tools. The content of this chapter is based on frequently asked questions of users. They ask often for the same enhancements.

I offer the program for free but only the owners of this book get the following hints to enhance the program.

6.1. Maintenance-free Operation

Most software manufacturer forget that the customer expects a maintenance-free running software 24 hours each day. The customer wants to work at the system only when hardware is broken or another unexpected problem occurs. Therefore the following demand appears:

As long as the system works fine it has to run endless without any maintenance work.

This is no problem and it does not cost much but it is still often ignored by software developers.

In most cases the maintenance-free operation fails because some temporary files or logfile fill up a harddisk until it is full. A full disk lets the program stop.

Some programs are written badly. They crash spontaneously or need slightly more and more memory as longer they are running. This makes the server slow or crashing because there is no memory free after several days or month runtime.

Sometimes I noticed also that some program cannot run endless because counters overrun or the program does not like the daylight saving changes to the system time. Its very interesting how many programs get major problems when the system time is adjusted while they are running.

Also the SMS Server Tools had such problems in the past but this is long ago. They were all fixed except one: The harddisk becomes full.

6.1.1. Clean-up logfiles

The harddisk becomes full for many reasons. We have to take care each of them separately:

- Logfile becomes permanently larger

- Sent messages are stored endless

- Received messages are stroed endless

- Failed messages are stroed endless

- The SQL database increases permanently

The following pages will show you, what you can do against these problems.

You do not need to care the operating system itself. All operating system of this book, that means SuSE Linux, RedHat Linux, Solaris and Windows look after their own logfiles and limit their disk usage.

The program that you install on top of the operating system should be checked deeply, specially the Apache webserver. Its logfiles caused many servers to crash.

Logfiles are typically written permanently. This is also for the Apache webserver and the SMS Server Tools the case. The first solution is surely to rename these files daily and start logging into new files.

But this does not work!

Running programs open their logfiles only once when they start and then they write into the same file as long as they are running. Windows does not allow you to rename or delete a file that is currently in use.

Unix handles files a little bit different but not better. If you rename a used file then the running program does not notice this and continues writing into the same file. The renaming does not affect the connection between the file and the program. So your renamed file will still grow. And if you delete the file it will disappear from the directory but it will still exist on the disk and grow.

You can verify this:

Open a two shell windows. In the first window enter this:

while true; do sleep 1; echo "Hello"; done >>test.txt

This is an endless loop that writes every seconds the word "Hello" into the file test.txt. The file remains open while the loop is running.

See that the file grows permanently by entering this command in the second window:

ls -l test.txt

If you use windows, then try to delete the file now using the windows explorer or try to rename it. Windows will not allow it.

If you use Unix, then try to rename the file:

mv test.txt test2.txt

Check the file size with the ls command. You will see that it is still growing. Delete the file using the command

rm test2.txt

The file does not appear in the directory content any more but the endless loop continues writing into the file. Some time the harddisk will be full. There is no command to check the file content. And also the command df -k does not show that the disk gets full because it uses the information from the directories and no directory contains the deleted file anymore. But you could watch the harddisk LED and you can see that is flashing permanently.

Therefore another solution is necessary, and I will tell it to you:

- first copy the logfile to another file

- then write the content of /dev/null into it

The device /dev/null delivers nothing when you read from it, therefore its name. By copying nothing into the logfile you delete its content. The file still exists but now it is empty. This works in Unix and Windows.

Normally this is done automatically by a cron job like this:

```
10 0 * * * cp /var/log/smsd.log /var/log/smsd.log.old; cp /dev/null /var/log/smsd.log
```

In Windows you need to write the cronjob slightly different (all in one row):

```
10 0 * * * c:\cygwin\bin\bash -c "cp /var/log/smsd.log /var/log/smsd.log.old; cp
/dev/null /var/log/smsd.log"
```

In Windows you need to write **bash -c "..."** because these are CygWin commands. Without running the shell nnCron Lite can only execute DOS and Windows commands.

This truncates the logfile every night shortly past midnight. The old logfile is kept in the backup file smsd.log.old.

You may wonder why I recommend to run this cronjob shortly past midnight and not exactly midnight. I avoid running scripts at exactly full hours, specially midnight because the operating system itself maintains logfile at this time. As more actions are done at the same time as slower they become.

In addition this would increase the fragmentation of the harddisk more than necessary. Fragmentation occurs always when many files grow or become smaller at the same time. Fragmentation reduces the performance of the harddisk. You do not need think much about fragmentation. This was a large problem 10 years ago because the disks were much slower than today. Also the filesystem of MS-DOS (FAT) did not handle the fragmentation well and forced you to defragment it sooner or later.

Today's harddisks are much faster and also the operating system became better. Fragmentation is only seldom a problem.

The cronjob above ensures that the logfile of the SMS Server Tools does not grow endless. You can always access the last logfile from the previous day. But what to do if you want more?

Copy the following script and run it once every night (logfiles.sh):

```
#!/bin/sh

rename()
{
  # $1 is a filename
  # $2 is the number of days to keep
  rm $1.$2 >/dev/null 2>&1
  num=$2
  while [ $num -gt 1 ]; do
    before=`expr $num - 1`
    mv $1.$before $1.$num >/dev/null 2>&1
    num=$before
  done
  cp $1 $1.1
  cp /dev/null $1
}

rename /var/log/smsd.log 7
rename /usr/local/apache/logs/access_log 2
rename /usr/local/apache/logs/error_log 7
```

The cronjob entry for Unix is this:

```
10 0 * * * /usr/local/bin/logfiles.sh
```

The cronjob entry for windows is this:

```
10 0 * * * c:\cygwin\bin\bash /usr/local/bin/logfiles.sh
```

This script renames the three files at the end of the script. It keeps 7 or 2 old files. You can insert more filenames if you like.

The old logfiles are kept with numbers in the filename extension.

The file "smsd.log" of yesterday is "smsd.log.1". The file two days ago is "smsd.log.2" and so on.

Now I like to explain how the script works. You can read it if you are interested in a detailed explanation.

At the beginning the script declares the function rename(), that can be used many times in the main part of the script.

```
rename()
{
...
}

rename /var/log/smsd.log 7
rename /usr/local/apache/logs/access_log 2
rename /usr/local/apache/logs/error_log 7
```

The function rename() deletes the logfile and keeps so many old versions als you gives as second argument.

The filename within the function is $2 because it's the first argument. The number of days is $2.

First the function deletes the oldest logfile, because it has expired:

```
rm $1.$2 >/dev/null 2>&1
```

Via output redirection the error message is suppressed. An error message soccurs when the file does not exist. In this case we do not want to see the error message.

```
num=$2
 while [ $num -gt 1 ]; do
   before=`expr $num - 1`
   mv $1.$before $1.$num >/dev/null 2>&1
   num=$before
 done
```

Now the existing logfile names are changed by adding one to each filename.

The variable "num" starts with the highest number and is decreased in each loop until it is not larger than 1 anymore. The expr command and an output redirection set the variable "before" to the number that preceeds the actual filename number.

Within the loop $num is always the new number and $before is the old number. The mv command renames the file. The filename is a combination of $1 and the number. Again an error message is suppressed.

After the loop the actual filename is copied to something.1 and then overwritten by "nothing". This emties the current logfile.

```
cp $1 $1.1
 cp /dev/null $1
```

- If 7 logfiles are kept the script does these steps:

- it deletes filename.7

- it renames filename.6 to filename.7

- it renames filename.5 to filename.6

- it renames filename.4 to filename.5

- it renames filename.3 to filename.4

- it renames filename.2 to filename.3

- it renames filename.1 to filename.2

- filename if overwritten by "nothing" (truncated)

6.1.2. Delete Old SMS Files

The logfiles do not bother us anymore. You can keep backups as long as you like but now the harddisk does not fill up without control anymore.

Now you should care the SMS files. Depending on your configuration here are more or less files to care. Check /etc/smsd.conf:

```
sent=/var/spool/smsd/sent
failed=/var/spool/smsd/failed
received=/var/spool/smsd/received
```

If there are one or more of these lines then collected SMS files will fill your harddisk. Old files have to be deleted.

Use the command

find /var/spool/sms -ctime +100 -exec rm {} \;

in a cronjob. In Unix the cronjob looks like this:

```
13 0 * * * find /var/spool/sms -ctime +100 -exec rm {} \;
```

And in Windows the cronjob looks like this:

```
13 0 * * * c:\cygwin\bin\bash -c "find /var/spool/sms -ctime +100 -exec rm {} \;"
```

The find command searches all old files in /var/spool/sms and deletes them via the rm command.

6.1.3. Delete Old SQL Data

If you use the script mysmsd, all sent and received messages are logged in a table. These data needs to be removed regulary, for example by this SQL command:

use smsd;
delete from sms_log where sent < now()- interval 100 day;

This command deletes table rows that are older than 100 days. The where part uses two MySQL functions that I did not explain in this book. You will find an explanation in the MySQL Manual in chapter 6.3.4.

The Function now() returns the actual date and time. The function "interval 100 day" returns a numeric value for 100 days. This is subtracted from the current date/time and the result is a date 100 days ago.

If you use this command in a shell script it could look like this (delete_old_sql.sh):

```
#!/bin/sh
SQL_HOST=localhost
SQL_USER=root
SQL_PASSWORD=""
SQL_DATABASE=smsd
SQL_TABLE=sms_log

if [ "$SQL_PASSWORD" != "" ]; then
  SQL_ARGS="-p$SQL_PASSWORD";
else
  SQL_ARGS="";
fi
SQL_ARGS="-h $SQL_HOST -u $SQL_USER $SQL_ARGS -D $SQL_DATABASE -s -e"

mysql $SQL_ARGS "delete from sms_log where sent < now()- interval 100 day;"
```

Installed as an Unix cronjob it deletes old SQL data every night:

```
14 0 * * * /usr/local/bin/delete_old_sql.sh >/dev/null 2>&1;
```

In windows the cronjob looks like this:

```
14 0 * * * c:\cygwin\bin\bash -c "/usr/local/bin/delete_old_sql.sh >/dev/null 2>&1"
```

6.1.4. Self-Test

To ensure that the SMS Server Tools run correctly, you can write a script that performs a self-test. Send yourself messages and check it they come in.

The following lines assume that you have three modems named GSM1, GSM2 and GSM3. You probably need to change this to your configuration.

Create the following directories:

```
/var/spool/sms/self-test/GSM1.in
/var/spool/sms/self-test/GSM2.in
/var/spool/sms/self-test/GSM3.in

/var/spool/sms/self-test/GSM1.out
/var/spool/sms/self-test/GSM2.out
/var/spool/sms/self-test/GSM3.out
```

Enter the outgoing queues into the configuration file /etc/smsd.conf. Every modem shall serve this queue as the last one. The new lines are:

```
[queues]
GSM1 = /var/spool/sms/GSM1.out
GSM2 = /var/spool/sms/GSM2.out
GSM3 = /var/spool/sms/GSM3.out

[GSM1]
queues = ..., GSM1

[GSM2]
queues = ..., GSM2

[GSM3]
queues = ..., GSM3
```

Instead of the dots (...) insert the already existing queues that the mode serves. It is important to add the selftest queues as the last queue.

Now you can send messages via special selected modem by giving the mode name in the header:

```
To: 491721234567
Provider: GSM1

SELF-TEST-MESSAGE
```

The received test messages should not remain in the normal incoming queue. We want them in the modem specific incoming queue directories that we just created. Add the following eventhandler to the configuration file /etc/smsd.conf:

```
eventhandler = /usr/local/bin/self-test-eventhandler.sh
```

The script content is this (self-test-eventhandler.sh):

```
#!/bin/sh
if [ "$1" = "RECEIVED" ]; then
   if grep "SELF-TEST-MESSAGE" $2 >/dev/null; then
      modem=`cat $2 | formail -zx Subject:`
      mv $2 /var/spool/sms/self-test/$modem.in
   fi
fi
```

If moves all received messages with the text "SELF-TEST-MESSAGE" into the test directory with ending .in. Tets messages from Modem GSM1 go into the directory GSM1.in and so on.

You have no the possibility to send messages via selected modem and you receive these messages in modem specific incoming queue directories.

Now you need a script that sends these test messages regulary and checks if they come back. The following script does this for one single modem, so you need to run it three times in parallel if you have three modems. Name the script self-test.sh:

```
#!/bin/sh
out=/var/spool/sms/outgoing/$1.test
in=/var/spool/sms/self-test/$1.in
while true; do
    echo "To: $2" > $out
    echo "Provider: $1" >> $out
    echo "" >> $out
    echo "SELF-TEST-MESSAGE" >> $out
    sleep 3600
    found=`ls -A $in`
    if [ "$found" ]; then
        rm $in/*
        echo "$1 test ok"
    else
        echo "$1 test failed"
    fi
done
```

Run the script in this way: **self-test.sh GSM1 491721234567 &**

Replace the phone number by the number of your modem GSM1. The script send a test message via the named modem to itself. Then it waits 1 hour (3600 seconds) and checks if the message came back in this time.

If yes then the test messages are deleted and an success message is displayed. If no then an error message appears.

The whole process is repeated in an endless loop until you stop it with

pkill self-test.sh

Start the script for any modem by giving him the modem name and phone number. Then you have an automatic self-test for the SMS Server Tools:

self-test.sh GSM1 491721234001 &
self-test.sh GSM2 491721234002 &
self-test.sh GSM3 491721234003 &

The & character at the line end tells the shell to run the command in background. This allows you to monitor each modem in one single window.

Instead of the error message you could use any command that informs you about the problem. How about an eMail?:

```
echo "Subject: $1 test failed"  | sendmail myname@mydomain
```

If you want to run the self test when the computer boots, you need to create the start script /etc/init.d/self-test:

```
#!/bin/sh
case "$1" in
  start)
     /usr/local/bin/self-test.sh GSM1 401721234001 &
     /usr/local/bin/self-test.sh GSM2 401721234002 &
     /usr/local/bin/self-test.sh GSM3 401721234003 &
     ;;
  stop)
     pkill self-test.sh
     ;;
esac
```

Do not forget to make the script executable:

chmod a+x /etc/init.d/self-test
chmod a+x /usr/local/bin/self-test.sh

Create the link:

cd /etc/etc/rc3.d
ln -s /etc/init.d/self-test S83self-test

As I explained already in the chapter about installation you may need to change the 3 in rc3.d to the correct value that refers to your runlevel directory that is used during boot.

Programs that start during boot must not write anything to the screen. Therefore replace any screen output by eMails or whatever informs you about the problem. (self-test.sh):

```
#!/bin/sh
out=/var/spool/sms/outgoing/$1.test
in=/var/spool/sms/self-test/$1.in
while true; do
    echo "To: $2" > $out
    echo "Provider: $1" >> $out
    echo "" >> $out
    echo "SELF-TEST-MESSAGE" >> $out
    sleep 3600
    found=`ls -A $in`
    if [ "$found" ]; then
       rm $in/*
    else
       echo "Subject: $1 test failed" | sendmail myname@mydomain
    fi
done
```

Insert your own eMail address instead of myname@mydomain.

You extended the SMS Server Tools by a program that detects failures after maximum one hour and alarms you via eMail.

The program detects also a system overload, if you have not enough modems to send the messages in an acceptable time.

If sending goes to slow, more and more entries are queued ind the provider queues. The queues for the test messages are the last queues in the order, that means that they are only sent if the provider queues are empty.

A full loaded system will never send the test messages. Therefore the program self-test.sh will alarm this.

An alarm from the self test can be triggered by many causes:

- a modem does not sent any more

- a modem does not received any more

- a modem is full loaded = capacity problem

For the last access there are three possible solutions available:

- send less messages per hour

- add more modems

- allow longer delays by changing the sleep time from 3600 seconds to more.

Of course you can also reduce the sleep time if you want to allow only shorter delays.

6.2. Better Statusmonitor

Im sure that you already tried the statusmonitor of the SMS Server Tools. You learned already that this is done by using the option -s:

smsd -s

The screen output is useful but very easy styled and ugly.

```
iiiriissssisss---
iiiriissssisss---
rriiiiiissisis---
rriiiiiiiiiii---
iiiiiiiiiiiir---
```

I wrote in chapter 5.8 that the status monitor is not usable when you start the SMS Server Tools during boot.

The enhancements that you may need are these:

- redirect the output to a helper program
- create a more beautiful style

How to do the first should be clear. Change the start script of the SMS Server Tools (/etc/init.d/sms):

```
...
    /usr/local/bin/smsd -s | /usr/local/bin/statusmonitor.sh &
...
```

Now write the following script and save it under the name /usr/local/bin/statusmonitor.sh:

```
#!/bin/sh
show()
{
  case "$2" in
    r) echo "Modem $1 is receiving" ;;
    s) echo "Modem $1 is sending" ;;
    i) echo "Modem $1 is idle" ;;
    b) echo "Modem $1 is blocked" ;;
  esac
}

while read status; do
  for num in 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16; do
    show $num `echo $status | cut -c$num`
  done > /var/log/smsd.status
done
```

The read command reads from the input channel one single line and stores the result in the variable $status. The while loop repeats this as long as smsd is running. When smsd exits, read returns an error code that causes the while loop to stop.

In a for loop any single character of the status line is shown. It uses the cut command to extract the characters at position 1 to 16 one after the other.

The show function changes the single character to readable text but only for the characters r, w, i and b. Other characters are not displayed.

The output of the loop is written into the file /var/log/smsd.status. The actual modem state is always visible by looking into this file.

If you have three modem the file looks similar to this:

```
Modem 1 is sending
Modem 2 is idle
Modem 3 is receiving
```

Now it's not very comfortable to open the same file again and again to see always the actual status. You can enter the following command to automate this:

while true; do cat /var/log/smsd.status; sleep 1; done

The while loop runs as long as you do not press Ctrl-C. It shows the actual state every second.

6.2.1. Statusmonitor As Website

If you prefer a coloured status monitor you can change the statusmonitor.sh script to create a HTML file instead of the text file.

The HTL file should contain a Java script that refreshes the screen regulary. The HTML version of the statusmonitor could be realized with this script (statusmonitor2.sh):

```
#!/bin/sh
show()
{
  case "$2" in
    r) echo "<td bgcolor=green>Modem $1<br>receiving</td>" ;;
    s) echo "<td bgcolor=red>Modem $1<br>sending</td>" ;;
    i) echo "<td bgcolor=white>Modem $1<br>idle</td>" ;;
    b) echo "<td bgcolor=lightgray>Modem $1<br>blocked</td>" ;;
  esac
}

while read status; do
  echo "<html><body " \
       "onLoad=\"window.setTimeout('location.reload()',3000)\">" \
       "<table border=1><tr>" \
       > /var/log/smsd.status.html
  for num in 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16; do
    show $num `echo $status | cut -c$num`
  done >> /var/log/smsd.status.html
  echo "</tr></table></body></html>" >> /var/log/smsd.status.html
done
```

If you use this script to enhance the status monitor, you get a coloured HTML table with one cell per modem. The website refreshes itself every 3 seconds. For other refreshment intervals you need to change the 3000 to another value, for example 5000 for 5 seconds.

As already known the start script needs to be modified to use the new statusmonitor:

```
...
   /usr/local/bin/smsd -s | /usr/local/bin/statusmonitor2.sh &
...
```

6.2.2. Alarms As Website

You can watch the status in a webbrowser and now you may want to see also the alarm in a webbrowser.

Create an alarmhandler that writes them into a logfile in HTML format (alarmhandler.sh):

```
#!/bin/sh
file=/var/log/smsd.alarm.html
touch $file

mv $file $file.old
echo "<html><body bgcolor=lightgray" \
     "onLoad=\"window.setTimeout('location.reload()',60000)\">" \
     > $file

case "$4" in
  1|2) echo "<font color=red>$2 $3 ($5) $6</font><br>" ;;
   3) echo "<font color=orange>$2 $3 ($5) $6</font><br>" ;;
   4) echo "<font color=yellow>$2 $3 ($5) $6</font><br>" ;;
   *) echo "<font color=black>$2 $3 ($5) $6</font><br>" ;;
esac >> $file

sed -e'2,100p' --silent $file.old >> $file
```

The script above creates a logfile if no one exist. The it renames the existing logfile into .old.

Then it creates a new logfile and writes a HTML header into it using an echo command. The header contains a Java script that refreshes the screen every minute.

The case control structure writes coloured alarm text into the file.

After that the script copies the first (99) lines of the old file into the new one. The very first line is skipped because it contains a header and we do not what a duplicated header in out new HTML file.

The new alarm file contains a header, the actual alarm and the latest 99 alarm from the old file. The messages are sorted in backward order, so the latest messages is always at the top.

If you look into the resulting HTML file you will notice that the end mark "</body></html>" is missing. I did this because if there would be this end we had to cut also the last line from the old file and this is difficult. All webbrowsers that I know show the page correctly anyway.

mv $file $file.old

6.3. More than 32 Modems

As you maybe noticed in the status monitor, the SMS Server Tools support up to 32 modems.

If you really need more modems, you need to modify the source code of the SMS Server Tools, recompile and reinstall the program.

The good news is that only one file file needs to be modified, and this is src/smsd_cfg.h:

```
#define PROVIDER 16
#define DEVICES 32
#define NUMS 16
```

Here are the limits for providers, modems and area codes for each provider.

Change the devices to 50, then you can attach up to 50 modems.

The provider value limits the number of provider and queues!

6.4. One Message To Many Recipients

Many users ask me if it is possible to send one message to multiple receivers. This is not possible because the transfer protocol of short messages allows only one receiver per message.

But you can tell SMS to send many messages from one file to many receivers. You need a checkhandler that splits the message file to many files.

A SMS file with many receivers looks like this:

```
From: Stefan
To: 491720000001
To: 491720000002
To: 491720000003

Hello, Test!
```

If you would send the file like any other only the last receiver would get it because smsd reads only the last To: line.

The following script needs to be used as an checkhandler. It creates one single SMS file for each receiver. The sending of the original file is disallowed by returning the exit code 1. Name the script split_multi_sms.sh:

```
#!/bin/sh
if [ `cat $1 | formail -zx To: | wc -l` -gt 1 ]; then
    cat $1 | formail -zx To: | \
    while read num; do
      file=`mktemp /var/spool/sms/outgoing/send_XXXXXX`
      cat $1 | formail -f -I "To: $num" > $file
    done
    exit 1
else
  exit 0
fi
```

This script uses formail three times. The first and second usage is to create a list with all receivers. The wc command is used to count the number of lines (the number of recipients). Only in case of more than one single recipient, the file gets splittet.

In the second case, the output is redirected to the following while-loop. The loop is repeated for every receiver and it reads the receivers number into the variable $num. The it creates a new SMS file using the mktemp command. The second formail command is used to replace the list of receivers by one single receiver from the variable $num. The new SMS file contains only one single receiver.

Please note that this method does not work with binary files. Formail was made for text files, it cannot read binary files correctly. If you need the same function also for binary files you need to write a "real" C program yourself. This is not part of this book.

7. Practical Example Applications

This chapter shows you some practical example applications. Use then for stimulation of your own ideas.

7.1. Central Alarm System

The central alarm system collects alarms from many sources in one central point.

Burglary alarm system can send alarms wireless to the security personal. Machines can send alarms to the support engineer. Messages from any sources are collected with one computer and shown on the screen.

At the receivers end you need a computer with SMS Server Tools and any GSM modem. It receives the alarms of all senders and displays them.

At the senders end you need one or more Falcom A3D modems. These modems have a special firmware that was specially made for such alarm applications.

7.1.1. Sender Side

You can use up to three alarm inputs on each modem. They are located at the 15 pin D-Type connector. When you put 4,5V-12V via a 47kOhm resistor to an input the modem sends an alarm SM. Please check the connector description notes in the modems manual.

Pin   1: Input 1
Pin   2: Input 2
Pin   3: Input 3
Pin   9: Output 8-36V prom power supply
Pin 15: Gnd

Case: HL   Case: LH

VDD   VDD

PullUp

Input KEY 1 .. 3   Switch
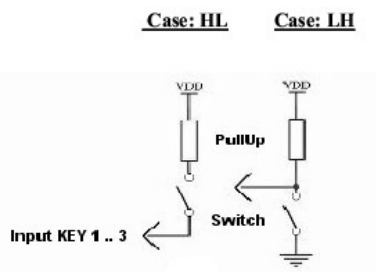
The alarm inputs work only, when the modem gets power from an external power supply. The internal battery does not source this part of the modem with energy.

Configure the modem to enter the PIN automatically after poweron, because you want to use it later without a computer standalone.

AT+CPIN=1234
&CNF PIN=1

Set the SMSC number:

AT+CSCA="+49172270000"
AT+CSAS

The next commands specify the destination numbers for the alarm messages and the text content:

&CNF KEY1,s01721234567,"Alarm Signal 1",H
&CNF KEY2,s01721234567,"Alarm Signal 2",H
&CNF KEY3,s01721234567,"Alarm Signal 3",H

H means that an alarm should be raised when the input signal changes to High. L would send an alarm when the input signal changes to Low and E sends an alarm in both cases.

Now you can disconnect the modem from the computer and use it a a standalone alarm sender. On activation of one input the text "Alarm Signal x" is sent via SMS.

If you need you can also tell the modem to send regular heathcheck messages that mean as much as "Im still living". These messages can be used at the receivers end to see that the modem is still working and has power.

Enter

&CNF MSG,s01721234567,"Test",60

Then the modem send the given text messages every 60 minutes. To disable this enter 0 minutes.

7.1.2. Receiver Side

On the receiver side you install the SMS Server Tools as written in an earlier chapter. Now you need an Eventhandler that shows all alarms.

The following eventhandler creates an HTML file that shows the last 1000 alarms. They are logged in backward order so you can see the latest alarm at the top. The list refreshes itself every minute.

Save the script as /usr/local/bin/alarmevent.sh:

```
#!/bin/sh
if [ "$1" != "RECEIVED" ]; then
  exit
fi

file=/var/log/smsd.alarm.html
touch $file
mv $file $file.old
echo "<html><body bgcolor=lightgray" \
     "onLoad=\"window.setTimeout('location.reload()',60000)\">" \
     > $file

SENT=`formail -zx Sent: < $2`
TEXT=`formail -I "" < $2`
echo "<b>$SENT</b> $TEXT<br>" >> $file

sed -e'2,100p' --silent $file.old >> $file
```

Then install it in the configuration file smsd.conf. The important line is:

```
eventhandler=/usr/local/bin/alarmevent.sh
```

Do not forget to mark the script as executable by entering

chmod a+x /usr/local/bin/alarmevent.sh

That's all. Now you can receive alarms with a central computer and watch them in a webbrowser. If you install also the Apache Webserver in this computer or export the HTML file using local network functions you can see this list also on other PC's.

7.2. Car Tracking

You can track car movements using the SMS Server Tools. A Falcom GSM modem with GPS option sends the geographical position of the car to your computer. With the help of a map you can always track the position and movements of that car.

A MySQL database is used to store the positions. You can read the table entries using the MySQL client or a PHP website.

7.2.1. Sender Side

Use a Falcom A3D modem with GPS and GSM antenna. Connect the ignition line of the modem to the same line of the car's key to allow the modem to switch on and off automatically.

Configure the modem to enter the PIN automatically because it will run later standalone withou a computer.

AT+CPIN=1234
&CNF PIN=1

Now enter the SMSC number:

AT+CSCA="+49172270000"
AT+CSAS

The next commands tell the modem to send the cars position immediately on poweron and then every 5 minutes:

&CNF BOOT,s01721234567,"Auto Ein",HP
&CNF MSG,s01721234567,"Auto Timer",5

If you want to monitor many cars the give every car its own text or use the phone number to separate them.

You can configure the modem to remain some minutes powerded on after the car is switched off. The following commands sets a 20 minutes delay for poweroff:

&CNF IGN,20

You can use values between 0 and 60 where 0 means that the modem switches off together with the car's engine.

Now the modem sends the car's position when the engine starts and after that every 5 minutes.

7.2.2. Receiver Side

At the reciever side install the SMS Server Tools as shown in an earlier chapter. Now you need an eventhandler that puts all received messages into a SQL database.

Write the following script and save it as /usr/local/bin/gpsevent.sh:

```
#!/bin/sh
if [ "$1" != "RECEIVED" ]; then
  exit 1
fi
SENT=`formail -zx Sent: < $2`
TEXT=`formail -I "" < $2`

SQL_ARGS="-h localhost -u root -ppassword -D smsd -s -e"
mysql $SQL_ARGS "insert into gps (sent,text) values (\"$SENT\",\"$TEXT\");"
```

Change the line SQL_ARGS=... so that it matches the settings of your database.

Then install it as an eventhandler in the configuration file smsd.conf. The important line is:

```
eventhandler=/usr/local/bin/gpsevent.sh
```

Do not forget to make the script executable by entering

chmod a+x /usr/local/bin/alarmevent.sh

Create a database for the GPS positions:

```
stefan@server> mysql -u root -p
mysql> create database smsd;
Query Ok...
mysql> use smsd;
Database changed.
mysql> create table gps (
    -> id int auto_increment not null,
    -> primary key(id),
    -> sent datetime,
    -> text char(160)
    -> );
Query Ok...
```

That's all. No you can receive the car's positions with your computer. To show the table entries you cound enter this command in an SQL Client:

select * from gps where text like "%Auto%";

A list appears with all entries of the car with the name "Auto". To limit the output to a special time range enter:

select * from gps where text like "%Auto%"
     and sent>="2003-02-22" and sent<="2003-02-24";

This shows all positions between 22.2.2003 0:00 and 24.2.2003 0:00.

The modem supports many different message formats. If you use the defaults, you receive messages in GGA format, that looks like:

```
$GPGGA,161229.487,3723.2475,N,12158.3416,W,1,07,1.0,9.0,M,,,,0000*18
```

The following numbers are the most interesting:

161229487	16:12:29 time UTC
3723.2475 N	Latitude 37 degrees 23 minutes 24 seconds north
12158.3416 W	Longitude 121 degrees 58 minutes 34 seconds west
7	7 satelites are reachable
9.0 M	Height 9 meter above normal null

7.3. eMail To SMS Gateway

You can send SMS using an eMail Client if you connect your Mailserver to the SMS Server Tools. I like to show you two different solutions in this chapter:

- eMail to SMS gateway with local mailserver
- eMail to SMS gateway with external mailserver

The first version works only in Unix because only Unix mailserver support the method that I will show.

The second version is also useable in Windows.

After installation of such a gateway you can send SMS with any eMail program. The receiver is a fixed address. The subject line contains the phone number of the receiver.

You need to configure the eMail program to send the message in Ascii format and not HTML format!

A useable eMail looks like this:

```
From: Frank Mustermann <mustermann@irgendwo.de>
To: "+491721234567" <sms@localhost>
Subject: not used

Hello, this is a Test-SMS!
```

7.3.1. eMail To SMS Gateway With Local Mailserver

You need a Server with SMS Server Tools and Sendmail. You can run both programs on different computers but then they need a disk directory that they can both read and write.

The connection point between the mailserver and the SMS Server Tools is the directory for outgoing messages, normally /var/spool/sms/outgoing.

Create a Unix user with the name "sms". Ensure that this user can send eMails via sendmail.

If sendmail does not work correctly, then read chapter 4.5.

Test sendmail by entering:

/usr/lib/sendmail sms
Hello!
<Strg-D>

Now log in as username "sms" and check the eMails or look into the file /var/spool/mail/sms. You should find the text "Hello!".

Sendmail delivers local eMails using the sub-program procmail. Any user can configure procmail to save received eMails somewhere else than in the local eMail file. We use this feature.

Log in as user "sms". Create the file .procmailrc in the home directory. It should have this content:

```
VERBOSE=off
MAILDIR=/var/spool/mail
DEFAULT=/var/spool/mail/sms
LOGFILE=/var/log/procmail

:0
* ^TOsms
| /usr/local/bin/email2sms
```

If you want to send an eMail to the sms user, this will not be put into its eMail box. The mail will be redirected to the script email2sms. This script converts the mail into a SMS file that is sent by the SMS Server Tools.

Now send an eMail to the local sms user. Ensure that the phone number is included in the To: field, for example:

To: "Herbert +491721234567" <sms@localhost>

The script email2sms is part of the SMS Server Tools. You can take a look into it to learn from it. The script is small and simple.

7.3.2. eMail To SMS Gateway With External Mailserver

You need a Server with SMS Server Tools, Fetchmail and Formail. And you need an external Mailserver and an POP3 account on it.

This version works also on Windows because it does not depend on Sendmail.

Before you start to search the installation files of Formail read this: Formail is a helper program of Procmail and it is part of the Procmail package.

Chapter 4.5.4 showed you, how to receive eMail from a POP3 account.

Create the file /etc/fetchmailrc with this content:

```
poll mail.isis.de protocol POP3 user sms pass passwort mda "/usr/local/bin/email2sms"
```

Insert the name of your mailserver, the correct username and password. Set the file permissions in this way:

chmod 0710 /ect/fetchmailrc

Start the fetchmail daemon by entering

fetchmail -f /etc/fetchmailrc -d 300

Fetchmail checks every 300 seconds for new incoming eMails in the POP3 box and sends them via the script email2sms. This script converts the eMails to SMS files and saves them in the outgoing queue directory. The SMS Server Tools send these messages.

As you see, the solution is as easy as the other solution in the previous chapter. The disadvantage is that fetchmail needs to check every n seconds for new incoming mail. This makes the process slower than using a local mailserver.

The advantage is that you can use any existing mailserver and that there is no special configuration necessary in the mailserver.

The script email2sms is part of the SMS Server Tools. You can take a look into it to learn from it. The script is small and simple.

7.4. Verifying Sender With SQL

If you set up an eMail to SMS gateway or an online form to offer public SMS services (for example in the internet) you typically want to allow the service only to authorized persons.

For this purpose you create a SQL table that contains a list of all authorized sender names and passwords. Then you use a checkhandler that checks for any outgoing SMS file if the sender is allowed to send SM.

Create the table by entering the following commands in a MySQL client:

```
stefan@server> mysql -u root -p
mysql> create database smsd;
Query Ok...
mysql> use smsd;
Database changed.
mysql> create table smsuser (
    -> id int auto_increment not null,
    -> primary key(id),
    -> name char(40),
    -> password char(16)
    -> );
Query Ok...
```

Fill the table with some usernames and passwords:

mysql> **insert into smsuser set name="Michaela Meier", password="abxdf";**
mysql> **insert into smsuser set name="Richard Wagner", password="blabla";**

And so on.

The checkhandler shall be written in a way that it can also handle messages coming from an eMail gateway (see previous chapter). The username can be extracted from the senders eMail address. The password shall be written into the subject field:

```
Sender:      "Michaela Meier" <michaela.meier@mail.isis.de>
Receiver:    "+491721234567" <sms@mailserver.de>
Subject:     password
Text:        Hello World!
```

The script email2sms that you saw in the previous chapter creates an SMS file from the eMail that looks like this:

```
From: Michaela Meier
To: +491721234567
Subject: password

Hello World!
```

Write the following script and save it as /usr/local/checkhandler.sh:

```
#!/bin/sh
name=`formail -zx From: < $1`
to=`formail -zx To: < $1`
password=`formail -zx Subject: < $1`

SQL_ARGS="-h localhost -u root -ppasswort -D smsd -s -N -B -e"
result=`mysql $SQL_ARGS select name from smsuser where name=\"$name\" and password=\"$password\";`

if [ -z "$result" ]; then
  exit 1
fi
```

The script checks the senders name and the password. If there is a matching entry in the database, the script exist without an error code. If no matching table row exists, the script exits with exit code 1.

The name and password are extracted from the From: and Subject: lines with Formail.

The SMS Server Tools send the short message only when the checkhandler exits without an error code.

Install the script as a checkhandler in the configuration file /etc/smsd.conf:

```
checkhandler=/usr/local/bin/checkhandler.sh
```

7.5. Ovulation Reminder

This example is not really my own idea. While searching for interesting websites, my wife found this idea. I like to show you how to realize such a service with the SMS Server Tools.

The Ovulation Reminder informs registered women shortly before the ovulation happens and shortly before the menstruation starts.

There is a web form where the users enter the date of the last bleeding and how long a period between two menstruations is.

These values are used to calculate the date of the next ovulation and menstruation. The calculated days are used to send reminder SM.

A period is typically 28 days but the exact length differs from women to women. The ovulation starts about the 13th day after the beginning of the last menstruation.

7.5.1. The Registration Form

The registration form is register.html and looks like this:

```
<html><body>
<b>Welcome to the Ovulation Reminder service.</b>
<p>
After registration you get one year short messages on your
mobile phone that remind before ovulation and menstruation starts.
<p>
<form action="register.php" method="post">
  My mobile phone number is
  <input type="text" name="phonenumber"
    size=15 maxlength=15 value="+491721234567">  .<p>
  My settings shall be saved with the password
  <input type="text" name="password"
    size=10 maxlength=15 value="secret">
  .<p>
  The last menstruation started at
  <input type="text" name="menstruation"
    size=10 maxlength=10 value="23.3.2003">.<p>
  A period is
  <input type="text" name="period"
    size=2 maxlength=2 value="28"> days long.<p>
  <input type="submit" name="action" value="submit">
  <input type="submit" name="action" value="remove">
</form>
<p>
To remove from the service you need to enter only your
phone number and the password. Then klick on "remove".
</body></html>
```

The website looks like this screenshot:
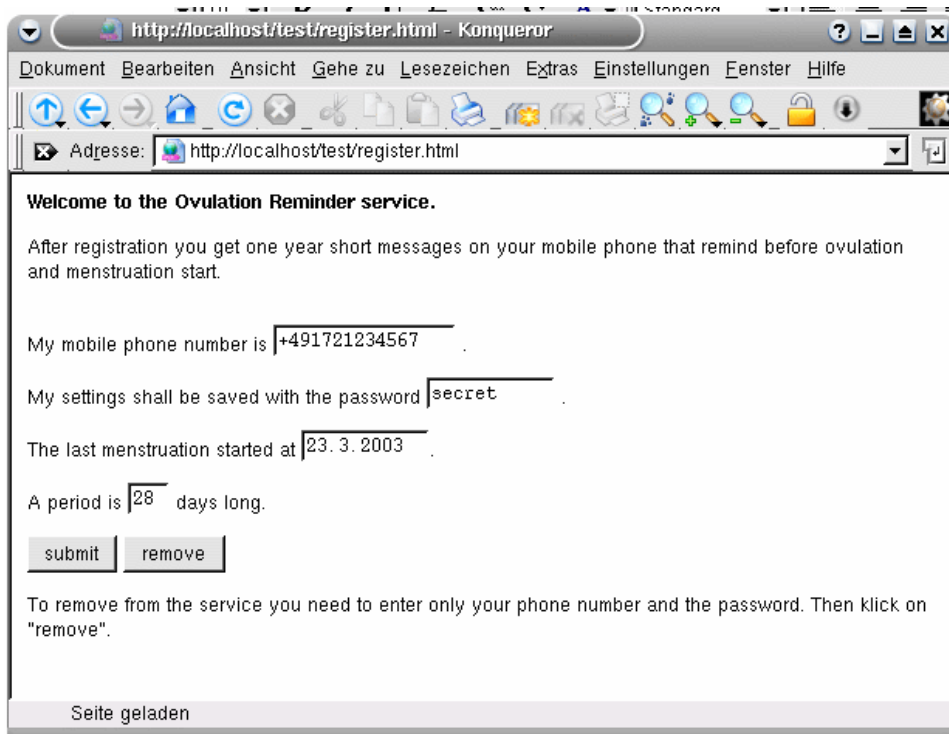
After registration a simple success message appears:
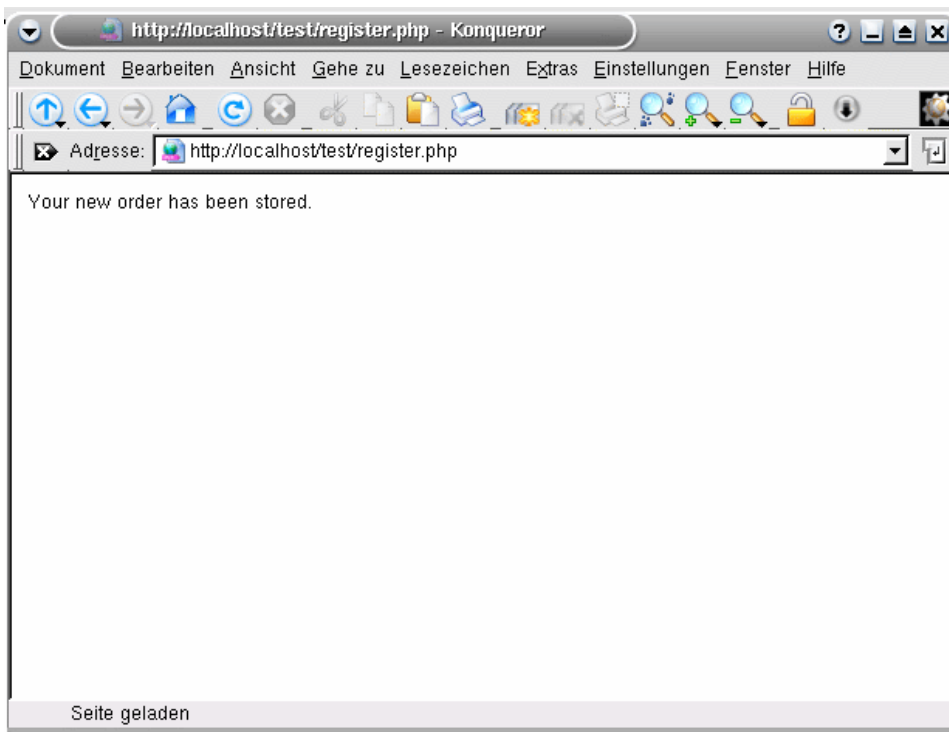
7.5.2. The Registration Script

When the user clicks on [submit] or [remove], the script register.php is executed:

```
<html><body>
<?php

function inputerror($message)
{
```

```
 print("$message <br>");
 print("Please correct your input.");
 exit;
}

function sqlerror()
{
 print("An error occured in the database: <br>");
 print(mysql_error());
 exit;
}

$phonenumber=$_POST["phonenumber"];
$password=$_POST["password"];
$menstruation=$_POST["menstruation"];
$period=$_POST["period"];
$action=$_POST["action"];

if (($phonenumber=="") || ($password=="") || ($menstruation=="") || ($period==""))
 inputerror("You need to fill all fields.");

if (($phonenumber=="+491721234567") || ($password=="secret"))
 inputerror("You need to enter your phone number and the password.");

if (($period<20) || ($period>36))
 inputerror("The length of the period must be wrong.");

$date=explode(".",$menstruation);
if (checkdate($date[1],$date[0],$date[2])==false)
 inputerror("The date is invalid");

$now=time();
$timestamp=mktime(0,0,0,$date[1],$date[0],$date[2]);
if ($timestamp>$now)
 inputerror("The date needs to be in the past.");

mysql_connect("localhost","root","password");

if ($result=mysql_db_query("smsd","select password from ovulation ".
   "where phonenumber=\"$phonenumber\" limit 1;"))
{
 $row=mysql_fetch_row($result);
 $old_password=$row[0];
}
else
 sqlerror();

if ($old_password!="")
{
 if ($old_password!=$password)
   inputerror("You entered a wrong password!");
 else
 {
   if (mysql_db_query("smsd","delete from ovulation ".
     "where phonenumber=\"$phonenumber\";"))
   {
     print("Your current order has been deleted.<br>");
   }
   else
     sqlerror();
 }
}

if ($action=="submit")
{
 $anzahl=0;
 while ($anzahl<24)
 {
   $timestamp=$timestamp+13*60*60*24;
   if ($timestamp>$now)
   {
     $newdate=strftime("%Y-%m-%d",$timestamp);
     if (mysql_db_query("smsd","insert into ovulation (phonenumber,password,date,type) ".
       "values (\"$phonenumber\",\"$password\",\"$newdate\",\"E\");")==false)
       sqlerror();
     $anzahl++;
   }
   $timestamp=$timestamp+($period-13)*60*60*24;
   if ($timestamp>$now)
   {
     $newdate=strftime("%Y-%m-%d",$timestamp);
     if (mysql_db_query("smsd","insert into ovulation (phonenumber,password,date,type) ".
       "values (\"$phonenumber\",\"$password\",\"$newdate\",\"B\");")==false)
       sqlerror();
     $anzahl++;
```

```
        }
     }
     print("Your new order has been stored.");
   }

   ?>
   </body></html>
```
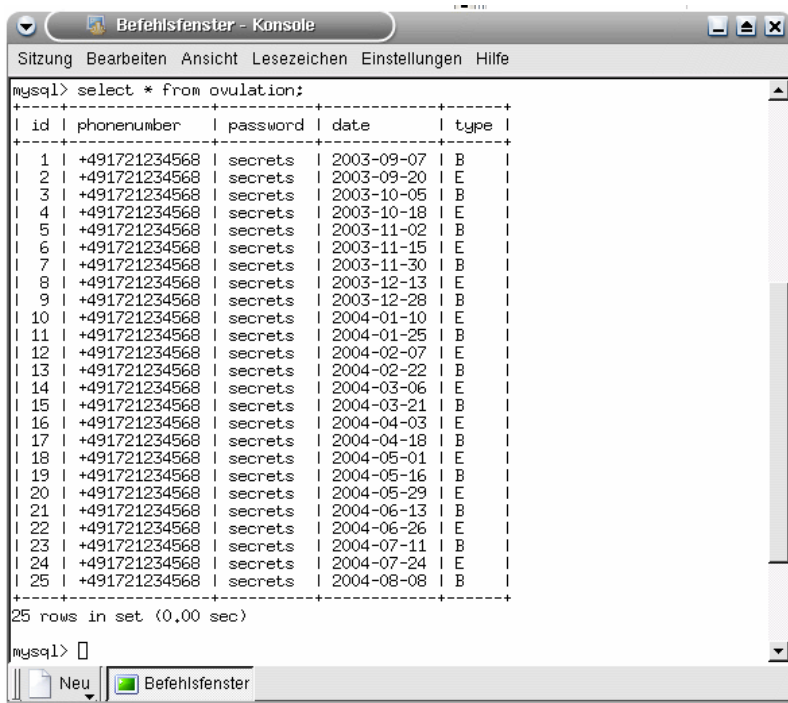
The script does some checks of the entered data. Then it deletes all existing entries in the SQL table and creates new orders in the SQL table.

The arguments to the mysql_connect command need to be changed to match your database.

7.5.3. The Table With Dates

The script creates a timetable in the SQL database that contains all days when the user should get an SM. The table looks liek this:

This table needs to be created before you can access it. Otherwise the script will not work:

```
stefan@server> mysql -u root -p
mysql> create database smsd;
Query Ok...
mysql> use smsd;
Database changed.
mysql> create table ovulation (
    -> id int auto_increment not null,
    -> primary key(id),
    -> phonenumber char(15),
    -> password char(15)
   -> date date,
    -> type char(1)
    -> );
Query Ok...
```

7.5.4. The Cronjob

The users can now submit an order and delete it. Now we need a cronjob that checks the database every day and sends shot messages.

Write the script /usr/local/bin/ovulation.sh:

```
#!/bin/sh
list=/tmp/ovulation.list
today=`date +%Y-%m-%d`

SQL_ARGS="-h localhost -u root -ppassword -D smsd -N -B -e"
mysql $SQL_ARGS "select type,phonenumber from ovulation where date=\"$today\";" > $list

while read line; do
  type=`echo $line | cut -f1 -d' '`
  number=`echo $line | cut -f2 -d' '`
  case $type in
    E) sendsms $number "You have an ovulation!" ;;
    B) sendsms $number "Your menstruation will start today!" ;;
  esac
  echo $type $number
done < $list
mysql $SQL_ARGS "delete from ovulation where date<=\"$today\";"
```

The SQL arguments in the SQL_ARGS need to be changed to your SQL database settings.

Executed as a cronjob this script sends every morning SMS according to the ovulation table and after that it deletes old table entries.

The cronjob could look like this:

```
30 6 * * * /usr/local/bin/ovulation.sh
```

And in Windows it looks like this:

```
30 6 * * * c:\cygwin\bin\bash -c "/usr/local/bin/ovulation.sh"
```

7.6. Ringtones, Logos And Jokes

One of the most popular SMS Service are the ringtones and logos. Therefore I like to show you how to realize such a service.

First you need ringtones, logos and jokes or whatever files you want to send. In the internet are many programs that can be used to create such files.

Most of these program run only on Windows. Typically you cannot save the files in binary format onto the harddisk. To get the correct file format you need a computer with SMS Server Tools and a Windows PC that runs the creator program.

Now create for example a logo and sent it as an SM from the Windows PC to the other computer with the SMS Server Tools. This saves the data into a file in the directory /var/spool/sms/incoming. Now you have an SMS file in the correct format and you can resend the file to any other phone number.

You should now that operator logos contains the name (more exact: the network identification code) somewhere in the binary file. If you create Logo for the provider D2 and sent it to an D1 customer he will never see the logo on his display. The mobile phone would show the logo only when the customer would change to the D2 network.

To send binary files to another destination you need to insert a To: line into the file header. Unfortunately the program formail cannot be used to do this because it does not work with binary files. Formail works only with text files.

Insert the To: line using the echo command and a cat command, for example:

echo "To: 491721234567" > new.sms
cat original.sms >>new.sms

The original file must not have a To: line, otherwise the resulting file would contain two To: lines. Smsd detects always the last To: line if there are more than one and that is not what you want.

7.6.1. Make Content Avaiable

Now you know how to create binary files for provider logos that you can send to other receivers. You know how to insert the receivers number into the SMS file.

Now create a new directory that will store all files that you want to offer your customers. To keep it easy use numbers as filenames.

Ensure that the name of the provider is part of the filename in case of operator logos because any provider needs its own file for the same logo.

You could save the files in /var/sms/content/ directory. Example:

/var/sms/content/0001	Ringtone 1
/var/sms/content/0002	Ringtone 2
/var/sms/content/0003	Ringtone 3
/var/sms/content/1001D1	Operator Logo 1 für D1
/var/sms/content/1001D2	Operator Logo 1 für D2
/var/sms/content/1001EPLUS	Operator Logo 1 für EPLUS
/var/sms/content/1002D1	Operator Logo 2 für D1
/var/sms/content/1002D2	Operator Logo 2 für D2
/var/sms/content/1002EPLUS	Operator Logo 2 für EPLUS
/var/sms/content/1003D1	Operator Logo 3 für D1
/var/sms/content/1003D2	Operator Logo 3 für D2
/var/sms/content/1003EPLUS	Operator Logo 3 für EPLUS
/var/sms/content/2001	Joke 1
/var/sms/content/2002	Joke 2
/var/sms/content/2003	Joke 3

All these files are SMS files with a header without a To: line.

If your customers order a file, they shall simply send them an SM with the filename. Your customers pay a higher price for sending messages to you. The phone network provider gives you a large part of this money.

7.6.2. The Script

You need a computer with the SMS Server Tools with the SMS files that have the logos, ringtones or whatever you want to send. And you need the following script that answers all orders.

Install an SQL log-database as show in chapter 5.13 to be able to check bills and reclamations. Activate the status reports (report=yes).

Install the following script as an eventhandler, (order.sh):

```
#!/bin/sh
if [ $1 != RECEIVED ]; then
   exit
fi
from=`formail -zx From: < $2`
text=`formail -I "" < $2`
text=`echo $text`
if [ -f "/var/sms/content/$text" ]; then
  file=`mktemp /var/spool/sms/outgoing/send_XXXXXX`
  echo "To: $from" > $file
  cat "/var/sms/content/$text" >>$file
else
  file=`mktemp /var/spool/sms/outgoing/send_XXXXXX`
  echo "To: $from" > $file
  echo "" >> $file
  echo "Sorry, wrong order number ($text) !" >> $file
fi
```
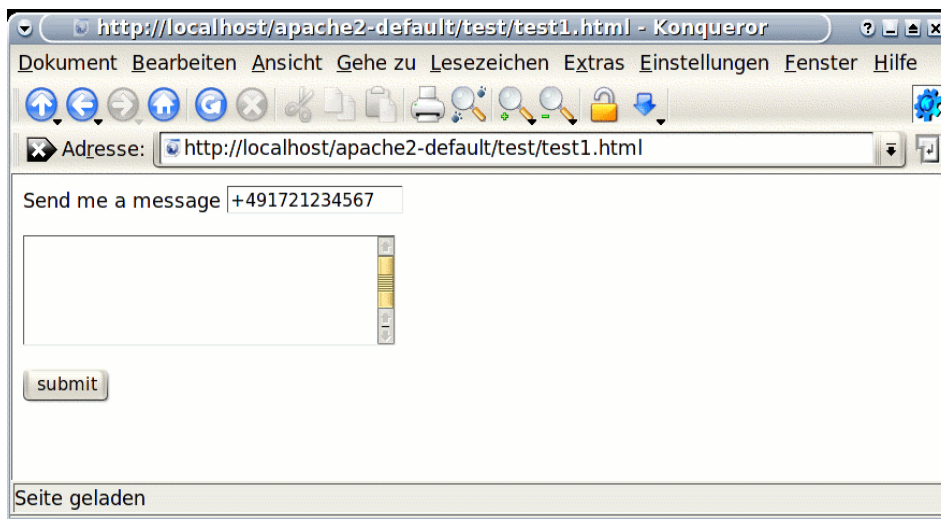
The line

```
text=`echo $text`
```

may seem useless for you. It is used to remove unwanted spaces around the order number (filename). The if command can be used after this to check if a file with the same name exists.

If a customer wants to order the file 1001D2, he or she needs to send a message with the text "1001D2" to your server. Customers who enter a wrong order number get an error messages as an reply.

7.7. Web form to send a message

Sending a message from a web form is very easy. Basically you need to store the message into a file in the ougoing spool directory.

7.7.1. The web form

This is the source code of the website test1.html:

```
<html><body>
<form action="send.php" method="post">
  Send me a message
  <input type="text" name="destination"
    size=15 maxlength=15 value="+491721234567">
  <p>
  <textarea name="message" cols="30" rows="4"></textarea>
  <p>
  <input type="submit" name="action" value="submit">
</form>
</body></html>
```

7.7.2. The send script

This is the source code of the send-script.

```
<?php
$filename = tempnam("","");
$file=fopen($filename,"w");

fwrite($file, "To: ". $_POST["destination"] ."\n");
fwrite($file, "\n");
fwrite($file, $_POST["message"] ."\n");

fclose($file);
copy($filename,"/var/spool/sms/outgoing/". basename($filename));
unlink($filename);
?>

<html><body>
  Your message will be sent in a few seconds.
</body></html>
```

At first the scripts creates a temporary file. The function tempnam is operating system dependent and creates a file (not only a filename) with a random name in the directory that is normally made for temporary files.

Then the script writes the phone number, an empty line and the message text into that temporary file.

Then it copies the file to the outgoing spool directory of the SMS Server and deletes the temporary file.

You could also directly create a file in the spool directory but then you have to make sure yourself that the filename is unique. I prefer to use the operating system function for this work.

If the webserver do not have direct access to the spool directory of the SMS Server, then you can also upload the file via ftp:

```
copy($filename,"ftp://servername/directory/". basename($filename));
```

Please read the manual of PHP to check if your version supports remote files and how to enable this feature.

- END -