Kannel svn-r5336M User's Guide

Open Source WAP and SMS gateway

Andreas Fink
Chairman & CTO
Global Networks Inc.

andreas@fink.org
http://www.smsrelay.com/
http://www.gni.ch/

Bruno Rodrigues

bruno.rodrigues@litux.org
http://litux.org/bruno

Stipe Tolj
System Architect
Tolj System Architecture

stolj@kannel.org

http://www.tolj.org/

Aarno Syvänen
Chief MMS Developer
Global Networks Inc.

as@gni.ch
http://www.gni.ch/

Alexander Malysh
Senior Software Engineer
GTX Messaging GmbH.

amalysh at kannel.org
http://www.gtx-messaging.com/

Alejandro Guerrieri
Senior System Architect
3Cinteractive, L.L.C.

aguerrieri@kannel.org
http://www.3cinteractive.com/

Lars Wirzenius
Gateway architect
former Wapit Ltd

Kalle Marjola
Senior Software Engineer
Enpocket

marjola@enpocket.com
http://www.enpocket.com

Kannel svn-r5336M User's Guide: Open Source WAP and SMS gateway

by Andreas Fink, Bruno Rodrigues, Stipe Tolj, Aarno Syvänen, Alexander Malysh, Alejandro Guerrieri, Lars Wirzenius, and Kalle Marjola

Abstract

This document describes how to install and use Kannel, the Open Source WAP and SMS Gateway originally developed by Wapit Ltd (now out of business) and now being developed further by the open source community, namely the Kannel Group.

Revision History

Revision svn-r5336M 2026.11.09

Table of Contents

1. Introduction 1

Overview of WAP 1
Overview of WAP Push 2
Overview of SMS 3
Features 4
WAP 4
WAP Push 4
SMS 4
Requirements 5

2. Installing the gateway 6

Getting the source code 6
Finding the documentation 6
Compiling the gateway 6
Installing the gateway 8
Using pre-compiled binary packages 9
Installing Kannel from RPM packages 9
Installing Kannel from DEB packages 10

3. Using the gateway 13

Configuring the gateway 13
Configuration file syntax 13
Inclusion of configuration files 14
Core configuration 15
Running Kannel 26
Starting the gateway 26
Command line options 27
Kannel statuses 28
HTTP administration 29

4. Setting up a WAP gateway 32

WAP gateway configuration 32
Wapbox group 32
Running WAP gateway 35
Checking whether the WAP gateway is alive 35

5. Setting up wtls security 36

Wtls configuration 37

6. Setting up MSISDN provisioning for WAP gateway 38

RADIUS accounting proxy configuration 38
RADIUS-Acct configuration 38
Using the MSISDN provisioning on HTTP server side 40

7. Setting up a SMS Gateway 41

Required components 41
SMS gateway configuration 41
SMS centers 41
Nokia CIMD 1.37 and 2.0 48

CMG UCP/EMI 4.0 and 3.5 49
SMPP 3.3, 3.4 and 5.0 54
Sema Group SMS2000 OIS 4.0, 5.0 and 5.8 61
SM/ASI (for CriticalPath InVoke SMS Center 4.x) 62
GSM modem 63
Modem initialization 69
GSMA OneAPI v1.0 ParlayX SMS SOAP 70
Fake SMSC 71
HTTP-based relay and content gateways 71
The "generic" system-type 73
Loopback SMSC (MT to MO direction switching) 76
Using multiple SMS centers 76
Feature checklist 77
External delivery report (DLR) storage 79
Internal DLR storage 79
MySQL DLR storage 79
MySQL connection configuration 80
LibSDB DLR storage 81
Oracle 8i/9i DLR storage 82
PostgreSQL DLR storage 82
MS-SQL/Sybase DLR storage 83
SQLite3 DLR storage 84
Redis DLR storage 84
Redis connection configuration 85
Cassandra DLR storage 86
DLR database field configuration 87
SMSBox configuration 89
Smsbox group 89
Smsbox routing inside bearerbox 94
SMS-service configurations 96
How sms-service interprets the HTTP response 106
Extended headers 107
Kannel POST 108
XML Post 109
SendSMS-user configurations 110
Over-The-Air configurations 113
Setting up more complex services 115
Redirected replies 115
Setting up operator specific services 115
Setting up multi-operator Kannel 116
Running SMS gateway 117
Using the HTTP interface to send SMS messages 117
Using the HTTP interface to send OTA configuration messages 121
OTA settings and bookmark documents 122
OTA syncsettings documents 122
OMA provisioning content 123
GET method for the OTA HTTP interface 123

8. Setting up Push Proxy Gateway 126
Configuring ppg core group, for push initiator (PI) interface 126
Configuring PPG user group variables 128
Finishing ppg configuration 130
Running a push proxy gateway 131
An example using HTTP SMSC 131
An example of minimum SI document 131
An example push (tokenized SI) document 132
Default network and bearer used by push proxy gateway 132
Push related Kannel headers 133
Requesting SMS level delivery reports for WAP pushes 133
Routing WAP pushes to a specific smsc 135
9. Using SSL for HTTP 136
Using SSL client support 136
Using SSL server support for the administration HTTP interface 136
Using SSL server support for the sendsms HTTP interface 136
Using SSL server support for PPG HTTPS interface 137
10. SMS Delivery Reports 138
11. Getting help and reporting bugs 140
A. Upgrading notes 141
From 1.2.x or 1.3.1 to 1.3.2 and later 141
B. Certificate generation 142
Self-signed 1024-bit RSA SSL certificates using openssl 142
C. Using the fake WAP sender 143
D. Using the fake SMS center 144
Setting up fakesmsc 144
Compiling fakesmsc 144
Configuring Kannel 144
Running Kannel with fakesmsc connections 144
Starting fake SMS center 144
Fake messages 145
Interactive mode 145
Fakesmsc command line options 145
E. Setting up a test environment for Push Proxy Gateway 147
Creating push content and control document for testing 147
Starting necessary programs 148
Using Nokia Toolkit as a part of a developing environment 150
Testing PAP protocol over HTTPS 151
F. Setting up a dial-up line 153
Analog modem 153
ISDN terminal 154

G. Regular Expressions **155**
Syntax and semantics of the regex configuration parameter 155
How-to setup the regex-parameters 155
Regex and non-regex-parameters 156
Performance issues 157
Examples 157
Example 1: core-configuration 157
Example 3: smsc-configuration 157
Example 4: sms-service-configuration 158
H. Meta Data **159**
Overview 159
SMPP Implementation 159
MO Messages 160
Example 160
MT Messages 160
Examples 161
I. Log files **162**
Bearerbox Access Log 162
Log rotation 162
Glossary **164**
Bibliography **165**

List of Tables

3-1. Core Group Variables 15
3-2. Kannel Command Line Options 27
3-3. Kannel HTTP Administration Commands 29
4-1. Wapbox Group Variables 32
5-1. Wtls Group Variables 37
6-1. RADIUS-Acct Group Variables 38
7-1. SMSC Group Variables 41
7-2. SMSC driver features 77
7-3. SMSC driver internal features 78
7-4. MySQL Connection Group Variables 80
7-5. Redis Connection Group Variables 85
7-6. DLR Database Field Configuration Group Variables 87
7-7. Smsbox Group Variables 89
7-8. Smsbox-route Group Variables 95
7-9. SMS-Service Group Variables 97
7-10. Parameters (Escape Codes) 104
7-11. X-Kannel Headers 107
7-12. X-Kannel Post Headers 108
7-13. SendSMS-User Group Variables 110
7-14. OTA Setting Group Variables 113
7-15. OTA Bookmark Group Variables 114
7-16. SMS Push reply codes 117
7-17. SMS Push (send-sms) CGI Variables 118
7-18. OTA CGI Variables 124
8-1. PPG core group configuration variables 126
8-2. PPG user group configuration variables 128
8-3. Kannel headers used by PPG 133
D-1. Fakesmsc command line options 145
E-1. Test_ppg's command line options 149
E-2. Test_ppg's configuration file directives 151

Chapter 1. Introduction

This chapter introduces WAP and SMS in general terms, and explains the role of the gateway in WAP and SMS, outlining their duties and features. It also explains why the Kannel project was started in the first place, and why it is open source.

With hundreds of millions of mobile phones in use all over the world, the market for services targeted at mobile users is mind-bogglingly immense. Even simple services find plenty of users, as long as they're useful or fun. Being able to get news, send e-mail or just be entertained wherever you are is extremely attractive to many.

The hottest technology for implementing mobile services is WAP, short for Wireless Application Protocol. It lets the phone act as a simple web browser, but optimizes the markup language, scripting language, and the transmission protocols for wireless use. The optimized protocols are translated to plain old HTTP by a *WAP gateway*.

Kannel is an open source WAP gateway. It attempts to provide this essential part of the WAP infrastructure freely to everyone so that the market potential for WAP services, both from wireless operators and specialized service providers, will be realized as efficiently as possible.

Kannel also works as an SMS gateway for GSM networks. Almost all GSM phones can send and receive SMS messages, so this is a way to serve many more clients than just those using a new WAP phone.

In addition, Kannel operates as *Push Proxy Gateway* , or *PPG*, making possible for content servers to send data to the phones. This is a new type of WAP service, and have many interesting applications. Usually servers know whether some data is new, not the users.

Kannel's quality has been recognized on March 7, 2001 when it was certified (http://www.kannel.org/oldnews.shtml#wapcert) by WAP Forum (http://www.wapforum.org) as the first WAP 1.1 gateway in the world.

Greater quality recognition are the quantity of companies using Kannel to successful connect to a variety of SMSC protocols in lots of countries.

Open Source (http://www.opensource.org) is a way to formalize the principle of openness by placing the source code of a product under a Open Source compliant software license. The BSD license was chosen over other Open Source licenses by the merit of placing the least amount of limitations on what a third party is able to do with the source code. In practice this means that Kannel is going to be a fully-featured WAP implementation and compatible with the maximum number of bearers with special emphasis on SMSC compatibility. The Kannel project was founded by Wapit Ltd in June, 1999.

Overview of WAP

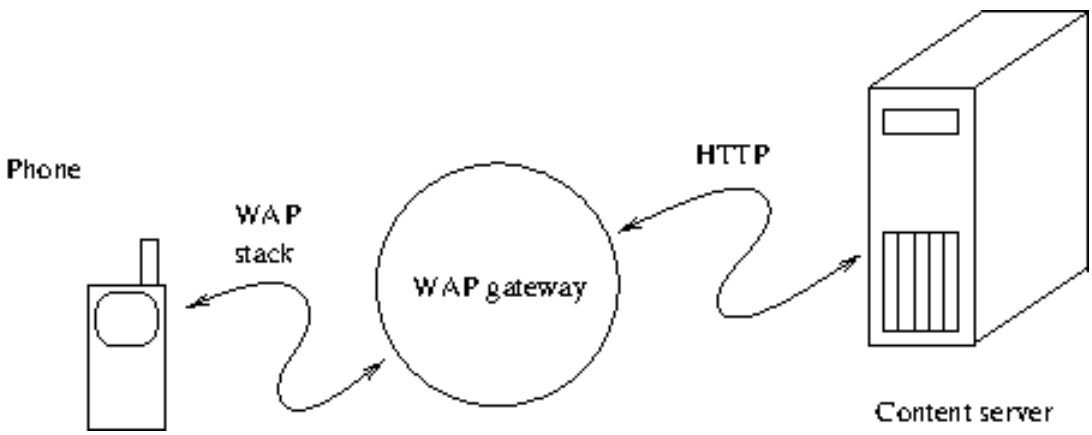
WAP, short for Wireless Application Protocol, is a collection of various languages and tools and an infrastructure for implementing services for mobile phones. Traditionally such services have worked via normal phone calls or short textual messages (e.g., SMS messages in GSM networks). Neither are very efficient to use, nor very user friendly. WAP makes it possible to implement services similar to the World Wide Web.

Unlike marketers claim, WAP does not bring the existing content of the Internet directly to the phone. There are too many technical and other problems for this to ever work properly. The main problem is that Internet content is mainly in the form of HTML pages, and they are written in such way that they require fast connections, fast processors, large memories, big screens, audio output and often also fairly efficient input mechanisms. That's OK, since they hopefully work better for traditional computers and networks that way. However, portable phones have very slow processors, very little memory, abysmal and intermittent bandwidth, and extremely awkward input mechanisms. Most existing HTML pages do not work on mobiles phones, and never will.

WAP defines a completely new markup language, the Wireless Markup Language (WML), which is simpler and much more strictly defined than HTML. It also defines a scripting language, WMLScript, which all browsers are required to support. To make things even simpler for the phones, it even defines its own bitmap format (Wireless Bitmap, or WBMP).

HTTP is also too inefficient for wireless use. However, by using a semantically similar binary and compressed format it is possible to reduce the protocol overhead to a few bytes per request, instead of the usual hundreds of bytes. Thus, WAP defines a new protocol stack to be used. However, to make things simpler also for the people actually implementing the services, WAP introduces a gateway between the phones and the servers providing content to the phones.

Figure 1-1. Logical position of WAP gateway (and PPG) between a phone and a content server.



The WAP gateway talks to the phone using the WAP protocol stack, and translates the requests it receives to normal HTTP. Thus content providers can use any HTTP servers and utilize existing know-how about HTTP service implementation and administration.

In addition to protocol translations, the gateway also compresses the WML pages into a more compact form, to save on-the-air bandwidth and to further reduce the phone's processing requirements. It also compiles WMLScript programs into a byte-code format. Latest WAP specifications defines some additional conversions that Kannel is starting to implement, like multipart/form-data, multipart/mixed or MMS content conversion.

Kannel is not just a WAP gateway. It also works as an SMS gateway. Although WAP is the hot and technically superior technology, SMS phones exist in huge numbers and SMS services are thus quite

useful. Therefore, Kannel functions simultaneously as both a WAP and an SMS gateway.

Overview of WAP Push

Previous chapter explained pull mode of operation: the phone initiates the transaction. There is, however, situations when the server (called in this context a push initiator) should be the initiator, for instance, when it must send a mail notification or a stock quote. For this purpose WAP Forum defined WAP Push.

Push is an application level service, sitting on the top of existing WAP stack. It defines two protocols, OTA and PAP. OTA is a ligthweigth protocol speaking with WAP stack (to be more specific, with WSP), PAP speaks with the push initiator. It defines three kind of XML documents, one for the push data itself and another for protocol purposes (these are called pap document or push control documents).

The server does not simply send push content to the phone, the user would surely not accept, for instance, interrupting of a voice call. Instead it sends a specific XML document, either Service Indication or Service Loading. These inform the user about the content become available, and it is displayed only when it is not interrupting anything. It contains an URL specifying the service and a text for user describing the content. Then the user can decide does he accept push or not.

The push content is send ed to the phones over SMS, but the content is fetched by the phone over IP bearer, for instance CSD or GPRS. Because Push Proxy Gateway tokenises SI and SL documents, it may fit one SMS message (if not, it is segmented for transfer).

Using two bearers seems to be an unnecessary complication. But quite simply, phones currently operate this way. Push over GPRS can only simplify matters.

Overview of SMS

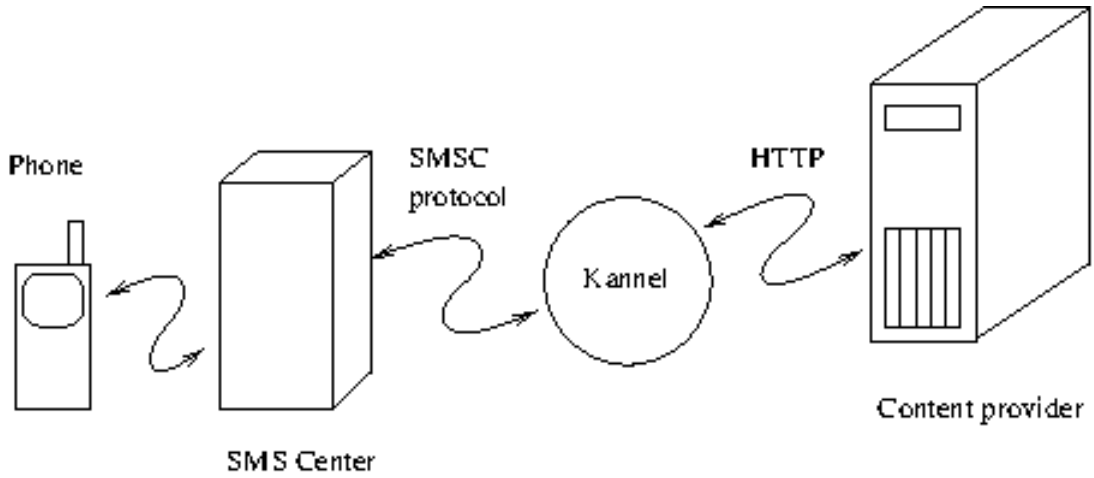
SMS, short messaging service, is a way to send short (160 character) messages from one GSM phone to another. It can also be used to send regular text as well as advanced content like operator logos, ringing tones, business cards and phone configurations.

SMS services are content services initiated by SMS message to certain (usually short) phone number, which then answers with requested content, if available.

When SMS services are used, the client (mobile terminal) sends an SMS message to certain number, usually a very short specialized number, which points to specific SMS center responsible for that number (plus possibly many others). This SMS center then sends the message onward to specified receiver in intra- or Internet, using an SMS center specific protocol. For example, a Nokia SMS center uses CIMD protocol.

As practically every different kind of SMS center uses different protocol, an *SMS gateway* is used to handle connections with SMS centers and to relay them onward in an unified form. Kannel's biggest feature is to abstract each SMSC protocol to a well-known HTTP protocol, simplifying services deployment.

Figure 1-2. Logical position of SMS gateway between a phone and a content server.



An SMS gateway can also be used to relay SMS messages from one GSM network to another, if the networks do not roam messages normally.

Kannel works as an SMS gateway, talking with many different kind of SMS centers, and relaying the messages onward to content providers, as HTTP queries. Content providers then answer to this HTTP query and the answer is sent back to mobile terminal, with appropriate SMS center connection using SMS center specific protocol.

In addition to serving mobile originated (MO) SMS messages Kannel also works as an SMS push gateway - content providers can request Kannel to send SMS messages to terminals. Kannel then determines the correct SMS center to relay the SMS message and sends the SMS message to that SMS center, again using SMS center specific protocol. This way the content provider does not need to know any SMS center specific protocol, just unified Kannel SMS sending interface.

Features

WAP

- Fully compliant with WAP 1.1 specification
- Already implements some WAP 1.2 and even WAP 2.0 features.

WAP Push

- Supports WAP Push SI and SL
- ...

SMS

- Supports a variety of SMSC protocols, namely:
  - CMG's UCP/EMI 4.0 and 3.5
  - ...
- Full support for MO and MT messages

Requirements

Kannel is being developed on Linux systems, and should be fairly easy to export to other Unix-like systems. However, we don't yet support other platforms, due to lack of time, although it should be working without major problems on Windows (through Cygwin), Solaris and FreeBSD.

Kannel requires the following software environment:

- C compiler and libraries for ANSI C, with normal Unix extensions such as BSD sockets and related tools. (GNU's GCC tool-chain is recommended)
- The Gnome XML library (known as gnome-xml and libxml), version 2.2.5 or newer. See http://xmlsoft.org/xml.html.

  If you are installing it from your distribution's packages, you'll need `libxml2-dev` in addition to run-time `libxml2` package libraries.
- GNU Make.
- An implementation of POSIX threads (`pthread.h`).
- GNU Bison 1.28, if you want to modify the WMLScript compiler (a pre-generated parser is included for those who just want to compile Kannel).
- DocBook processing tools: DocBook style-sheets, jade, jadetex, etc; see `README`, section 'Documentation', for more information (pre-formatted versions of the documentation are available, and you can compile Kannel itself even without the documentation tools).
- GNU autoconf, if you want to modify the configuration script.

Hardware requirements are fluffier. Some informal benchmarking have shown that with a reasonably fast PC architecture (e.g. 400MHz Pentium II with 128MB RAM), SMS performance's bottleneck is always on the SMSC side, even for example with multiple connections summing a pipeline with 400 msg/sec. We haven't benchmarked Kannel yet on WAP side, so there are no hard numbers.

Chapter 2. Installing the gateway

This chapter explains how the gateway can be installed, either from a source code package or by using a pre-compiled binary version. The goal of this chapter is to get the gateway compiled and all the files in the correct places; the next chapter will explain how the gateway is configured.

> **Note:** If you are upgrading from a previous version, please look at Appendix A for any important information.

Getting the source code

The source code to Kannel is available for download at http://www.kannel.org/download.shtml. It is available in various formats and you can choose to download either the latest release version or the daily snapshot of the development source tree for the next release version, depending on whether you want to use Kannel for production use or to participate in the development.

If you're serious about development, you probably want to use CVS, the version control system used by the Kannel project. This allows you to participate in Kannel development much more easily than by downloading the current daily snapshot and integrating any changes you've made every day. CVS does that for you. (See the Kannel web site for more information on how to use CVS.)

Finding the documentation

The documentation for Kannel consists of three parts:

1. *User's Guide*, i.e., the one you're reading at the moment.
2. *Architecture and Design*, in `doc/arch` or at http://www.kannel.org/arch.shtml (http://www.kannel.org/arch.shtml)
3. The `README` and various other text files in the source tree.

You can also find general information on Kannel's website (http://www.kannel.org) and information about existing problems at our bug tracker (http://bugs.kannel.org).

We intend to cover everything you need to install and use Kannel is in *User's Guide*, but the guide is still incomplete in this respect. Similarly, the *Architecture and Design* document should tell you everything you need to know to dive into the sources and quickly make your own modifications. It's not a replacement for actually reading the source code, but it should work as a map to the source code. The `README` is not supposed to be very important, nor contain much information. Instead, it will just point at the other documentation.

Compiling the gateway

If you are using Kannel on a supported platform, or one that is similar enough to one, compiling Kannel should be trivial. After you have unpacked the source package of your choose, or after you have checked out the source code from CVS, enter the following commands:

```
./configure
make
```

The `configure` script investigates various things on your computer for the Kannel compilation needs, and writes out the `Makefile` used to compile Kannel. **make** then runs the commands to actually compile Kannel.

If either command writes out an error message and stops before it finishes its job, you have a problem, and you either need to fix it yourself, if you can, or report the problem to the Kannel project. See Chapter 11 for details.

For detailed instruction on using the configuration script, see file `INSTALL`. That file is a generic documentation for **configure**. Kannel defines a few additional options:

- `--with-defaults=`*`type`* Set defaults for the other options. *`type`* is either `speed` or `debug`. The default is `speed`. `speed` options is equivalent to `--with-malloc=native --disable-assertions`, while `debug` is `--with-malloc=checking --enable-assertions`.
- `--disable-docs (default is --enable-docs)` Use this option if you don't have DocBook installed and/or you want to save some time and CPU cycles. Pre-generated documentation is available on Kannel's site. Default behavior is to build documentation, b.e., converting the User Guide and the Architecture Guide from the DocBook markup language to PostScript and HTML if DocBook is available.
- `--enable-drafts (default is --disable-drafts)` When building documentation, include the sections marked as `draft`.
- `--enable-debug (default is --disable-debug)` Enable non-reentrant development time debugging of WMLScript compiler.
- `--disable-localtime (default is --enable-localtime)` Write log file time stamps in GMT, not in local time.
- `--enable-assertions / --disable-assertions` Turn on or off runtime assertion checking. `enable` makes Kannel faster, but gives less information if it crashes. Default value is dependent on `--with-defaults`.
- `--with-malloc=`*`type`* Select memory allocation module to use: *`type`* is `native`, `checking`, or `slow`. For production use you probably want `native`. The `slow` module is more thorough than `checking`, but much slower. Default value is dependent on `--with-defaults`.
- `--enable-mutex-stats` Produce information about lock contention.
- `--enable-start-stop-daemon` Compile the start-stop-daemon program.
- `--enable-pam` Enable using PAM for authentication of sendsms users for smsbox.

- `--with-mssql[=DIR]` Enable using FreeTDS libraries for DBPool and DLR support on MS-SQL and Sybase. Optional DIR specifies where to look for FreeTDS libraries and include files (defaults to /usr/local).
- `--with-mysql` Enable using MySQL libraries for DBPool and DLR support.
- `--with-mysql-dir=DIR` Where to look for MySQL libs and header files. DIR points to the installation of MySQL.
- `--with-sdb` Enable using LibSDB libraries for dlr support.
- `--with-oracle` Enable using Oracle OCI-Libraries for Oracle 8i/9i DBPool and DLR support.
- `--with-oracle-includes=DIR` Where to look for Oracle OCI-Header files.
- `--with-oracle-libs=DIR` Where to look for Oracle OCI-Library files.
- `--with-redis` Enable using Redis for DBPool and DLR support. Requires the hiredis library.
- `--with-redis-dir=DIR` Where to look for the hiredis library and header files.
- `--with-cassandra` Enable using Cassandra for DBPool and DLR support. Requires the DataStax C/C++ driver for Apache Cassandra.
- `--with-cassandra-dir=DIR` Where to look for the DataStax C/C++ Cassandra library and header files.

You may need to add compilations flags to configure:

```
CFLAGS='-pthread' ./configure
```

The above, for instance, seems to be required on FreeBSD. If you want to develop Kannel, you probably want to add CFLAGS that make your compiler use warning messages. For example, for GCC:

```
CFLAGS='-Wall -O2 -g' ./configure
```

(You may, at your preference, use even stricter checking options.)

Installing the gateway

After you have compiled Kannel, you need to install certain programs in a suitable place. This is most easily done by using **make** again:

```
make bindir=/path/to/directory install
```

Replace */path/to/directory* with the pathname of the actual directory where the programs should be installed. The programs that are installed are (as filenames from the root of the source directory):

```
gw/bearerbox
gw/smsbox
```

`gw/wapbox`

The version number of the gateway is added to the file names during installation. This makes it easier to have several versions installed, and makes it easy to go back to an older version if the new version proves problematic.

Kannel consists of three programs called boxes: the bearer box is the interface towards the phones. It accepts WAP and SMS messages from the phones and sends them to the other boxes. The SMS box handles SMS gateway functionality, and the WAP box handles WAP gateway functionality. There can be several SMS boxes and several WAP boxes running and they don't have to run on the same host. This makes it possible to handle much larger loads.

Using pre-compiled binary packages

Installing Kannel from RPM packages

This chapter explains how to install, upgrade and remove Kannel binary RPM packages.

Before you install Kannel, check that you have libxml2 installed on your system:

```
rpm -q libxml2
```

Installing Kannel

1. Download the binary RPM packet from the Kannel web site.

2. Install the RPM package:

```
rpm -ivh kannel-VERSION.i386.rpm
```

Upgrading Kannel

1. Download the binary RPM packet from the Kannel web site.

2. Upgrade the RPM package:

```
rpm -Uvh kannel-VERSION.i386.rpm
```

Removing Kannel

1. Remove the RPM package:

```
rpm -e kannel
```

After you have installed Kannel from the RPM packages you should now be able to run the Kannel init.d script that will start Kannel as a WAP gateway. Run the script as root.

```
/etc/rc.d/init.d/kannel start
```

To stop the gateway just run the same script with the stop parameter.

```
/etc/rc.d/init.d/kannel stop
```

If Kannel is already running and you just want to quickly stop and start the gateway,e.g.to set a new configuration option, run the script with the restart parameter.

```
/etc/rc.d/init.d/kannel restart
```

If you want Kannel to run as a daemon, you need to add a symbolic link to the Kannel script from the runlevel you want Kannel to run in. E.g. to run Kannel in runlevel 5 add symbolic links to /etc/rc.d/rc5.d/.

```
cd /etc/rc.d/rc5.d/
ln -s ../init.d/kannel S91kannel
ln -s ../init.d/kannel K91kannel
```

To run Kannel as a SMS gateway you need to edit the configuration file which is at /etc/kannel/kannel.conf. In Kannel's documentation directory (`/usr/share/doc/kannel`) there is an example file called examples/smskannel.conf. It has some basic examples of the configuration groups needed to run Kannel as a SMS gateway. For more detailed information please see `examples/kannel.conf` in the same directory for a commented complete example, and read the section "SMS gateway configuration" later in this same document.

The logging is disabled by default and you can enable it from the kannel.conf file. Just add the log-file option to the group of which box you want to log.

The documentation will be installed at /usr/doc/kannel-VERSION/ or /usr/share/doc/kannel-VERSION/ depending on if you used the RedHat 6.x or a 7.x or newer package.

In the Kannel documentation directory there is a HTML file called control.html. It is an example file that shows how to use the Kannel HTTP administration interface. It also has a template for sending SMS messages.

Installing Kannel from DEB packages

This chapter explains how to install, upgrade and remove Kannel binary DEB packages.

Kannel's packages are available on main Debian repository (http://packages.debian.org/kannel) although that version may be out-of-sync with latest Kannel version.

Before you install Kannel, check that you have libxml2 installed on your system:

```
dpkg -l libxml2
```

Installing or upgrading Kannel from Debian or Kannel repository

1. Install or upgrade the package:

```
apt-get install kannel
```

See http://kannel.org/download.shtml#debian_repository for information about kannel repository sources.list

Installing or upgrading Kannel from a file

1. Download the binary DEB packet from the Kannel web site.

2. Log in as root:

```
su -
```

3. Install or upgrade the DEB package:

```
dpkg -i kannel-VERSION.deb
```

Removing Kannel

1. Log in as root:

2. Remove the package keeping configuration files:

```
dpkg --remove kannel
```

3. Remove the package completely:

```
dpkg --purge kannel
```

After you have installed Kannel from the DEB packages you should now be able to run the Kannel init.d script that will start Kannel as a WAP gateway. Run the script as root.

```
/etc/init.d/kannel start
```

To stop the gateway just run the same script with the stop parameter.

```
/etc/init.d/kannel stop
```

If Kannel is already running and you just want to quickly stop and start the gateway,e.g.to set a new configuration option, run the script with the restart parameter.

```
/etc/init.d/kannel restart
```

If you don't want Kannel to run as a daemon, run:

```
update-rc.d -f kannel remove
```

If you want to restore Kannel running as a daemon, you need to add a symbolic link to the Kannel script from the runlevel you want Kannel to run in. E.g. to run Kannel in default runlevel, just run:

```
update-rc.d kannel defaults
```

Kannel package starts by default with a wapbox daemon. To activate smsbox or select which box you want to start, edit /etc/default/kannel and comment/uncomment START_xxxBOX.

To run Kannel as a SMS gateway you need to edit the configuration file which is at /etc/kannel/kannel.conf. In /usr/share/docs/kannel/examples/ there are example files. They have some basic examples of the configuration groups needed to run Kannel as a SMS gateway. For more detailed information please read the section "SMS gateway configuration" later in this same document.

The documentation will be installed at /usr/share/doc/kannel/.

In the Kannel documentation directory there is a html file called control.html. It is an example file that shows how to use the Kannel HTTP administration interface. It also has a template for sending SMS messages.

Additionally to kannel-VERSION.deb, there's now an optional kannel-docs-VERSION.deb with documentation (userguide et al) and a kannel-extras-VERSION.deb with contrib and test stuff.

If you want to test development version, use the packages called kannel-devel-*.deb.

Chapter 3. Using the gateway

This chapter explains how the gateway core, bearerbox, is configured and used. It covers the configuration file, keeping an eye on the gateway while it is running, and using the HTTP interface to control the gateway.

After this chapter there is distinct chapter for each kind of gateway use: WAP gateway, SMS gateway and WAP Push proxy. These chapters explain the configuration and other aspects of gateway of that type.

You can configure your Kannel to be only a WAP or a SMS gateway, or to be both at the same time. You just need to read each chapter and combine all configurations.

There is only one configuration file for all parts of Kannel, although when Kannel is distributed to several hosts some lines from the configuration file can be removed in some hosts.

Configuring the gateway

The configuration file can be divided into three parts: bearerbox configurations, smsbox configurations and wapbox configurations. Bearerbox part has one 'core' group and any used SMS center groups, while wapbox part has only one wapbox group. In smsbox part there is one smsbox group and then number of sms-service and sendsms-user groups.

Details of each part are in an appropriate section of this documentation. The 'core' group used by the bearerbox is explained in this chapter, while 'wapbox' part is in the next chapter and 'smsbox', 'smsc' (SMS center), 'sms-service' and 'sendsms-user' groups are in the SMS Kannel chapter.

Configuration file syntax

A configuration file consists of groups of configuration variables. Groups are separated by empty lines, and each variable is defined on its own line. Each group in Kannel configuration is distinguished with a group variable. Comments are lines that begin with a number sign (#) and are ignored (they don't, for example, separate groups of variables).

A variable definition line has the name of the variable, and equals sign (=) and the value of the variable. The name of the variable can contain any characters except white space and equals. The value of the variable is a string, with or without quotation marks () around it. Quotation marks are needed if the variable needs to begin or end with white space or contain special characters. Normal C escape character syntax works inside quotation marks.

Perhaps an example will make things easier to comprehend:

```
1    # A do-nothing service.
2    group = sms-service
3    keyword = nop
4    text = "You asked nothing and I did it!"
5
6    # Default service.
7    group = sms-service
8    keyword = default
9    text = "No services defined"
```

The above snippet defines the keyword `nop` for an SMS service, and a default action for situation when the keyword in the SMS message does not match any defined service.

Lines 1 and 6 are comment lines. Line 5 separates the two groups. The remaining lines define variables. The group type is defined by the group variable value.

The various variables that are understood in each type of configuration group are explained below.

Some variable values are marked as `'bool'`. The value for variable can be like true, false, yes, no, on, off, 0 or 1. Other values are treated as 'true' while if the variable is not present at all, it is treated as being 'false'.

In order to make some configuration lines more readable you may use the delimiter '\' at the end of a line to wrap and concatenate the next line up to the current line. Here is an example:

```
1    # A group with a wrapped alias line
2    group = sms-service
3    keyword = hello
4    aliases = hallo;haalloo;\
5      heelloo;haelloo;healloo
5    text = "Hello world!"
```

The above example shows how a list for various alias keywords is wrapped to two lines using the line wrap delimiter. In order to use the delimiter '\' itself, you need to escape it via a prefixed '\' itself. So this is '\\' to escape the wrapping function and use the character in the string.

Inclusion of configuration files

A configuration file may contain a special directive called `include` to include other file or a directory with files to the configuration processing.

This allows to segment the specific configuration groups required for several services and boxes to different files and hence to have more control in larger setups.

Here is an example that illustrates the `include` statement :

```
group = core
admin-port = 13000
wapbox-port = 13002
admin-password = bar
wdp-interface-name = "*"
log-file = "/var/log/bearerbox.log"
log-level = 1
box-deny-ip = "*.*.*.*"
box-allow-ip = "127.0.0.1"

include = "wapbox.conf"

include = "configurations"
```

Above is the main `kannel.conf` configuration file that includes the following `wapbox.conf` file with all required directives for the specific box, and a `configurations` directory which may include more files to include.

```
group = wapbox
bearerbox-host = localhost
log-file = "/var/log/wapbox.log"
log-level = 0
syslog-level = none
```

The above `include` statement may be defined at any point in the configuration file and at any inclusion depth. Hence you can cascade numerous inclusions if necessary.

At process start time inclusion of configuration files breaks if either the included file can not be opened and processed or the included file has been processed already in the stack and a recursive cycling has been detected.

Core configuration

Configuration for Kannel *MUST* always include a group for general bearerbox configuration. This group is named as 'core' in configuration file, and should be the first group in the configuration file.

As its simplest form, 'core' group looks like this:

```
group = core
admin-port = 13000
admin-password = f00bar
```

Naturally this is not sufficient for any real use, as you want to use Kannel as an SMS gateway, or WAP gateway, or both. Thus, one or more of the optional configuration variables are used. In following list (as in any other similar lists), all mandatory variables are marked with `(m)`, while conditionally mandatory (variables which must be set in certain cases) are marked with `(c)`.

Table 3-1. Core Group Variables

Variable	Value	Description
`group (m)`	`core`	This is a mandatory variable
`admin-port (m)`	port-number	The port number in which the bearerbox listens to HTTP administration commands. It is NOT the same as the HTTP port of the local HTTP server, just invent any port, but it must be over 1023 unless you are running Kannel as a root process (not recommended)

Variable	Value	Description
admin-port-ssl (o)	bool	If set to true a SSL-enabled administration HTTP server will be used instead of the default insecure plain HTTP server. To access the administration pages you will have to use a HTTP client that is capable of talking to such a server. Use the "https://" scheme to access the secured HTTP server. Defaults to "no".
admin-password (m)	string	Password for HTTP administration commands (see below)
status-password	string	Password to request Kannel status. If not set, no password is required, and if set, either this or admin-password can be used
admin-deny-ip admin-allow-ip	IP-list	These lists can be used to prevent connection from given IP addresses. Each list can have several addresses, separated with semicolons (';'). An asterisk ('*') can be used as a wild-card in a place of any ONE number, so *.*.*.* matches any IP.
smsbox-interface (o)	string	If this is set, the smsbox listener will only bind to a specified address. For example: "127.0.0.1".
smsbox-port (c)	port-number	This is the port number to which the smsboxes, if any, connect. As with admin-port, this can be anything you want. Must be set if you want to handle any SMS traffic.

Variable	Value	Description
`smsbox-port-ssl (o)`	bool	If set to true, the smsbox connection module will be SSL-enabled. Your smsboxes will have to connect using SSL to the bearerbox then. This is used to secure communication between bearerbox and smsboxes in case they are in separate networks operated and the TCP communication is not secured on a lower network layer. Defaults to "no".
`wapbox-port (c)`	port-number	Like smsbox-port, but for wapbox-connections. If not set, Kannel cannot handle WAP traffic
`wapbox-port-ssl (o)`	bool	If set to true, the wapbox connection module will be SSL-enabled. Your wapboxes will have to connect using SSL to the bearerbox then. This is used to secure communication between bearerbox and wapboxes in case they are in separate networks operated and the TCP communication is not secured on a lower network layer. Defaults to "no".
`box-deny-ip` `box-allow-ip`	IP-list	These lists can be used to prevent box connections from given IP addresses. Each list can have several addresses, separated with semicolons (';'). An asterisk ('*') can be used as a wild-card in place of any ONE number, so *.*.*.* matches any IP.
`udp-deny-ip`	IP-list	These lists can be used to prevent UDP packets from given IP addresses, thus preventing unwanted use of the WAP gateway. Used the same way as `box-deny-ip` and `box-allow-ip`.

Variable	Value	Description
`udp-allow-ip`		If this is set, Kannel listens to WAP UDP packets incoming to ports 9200-9208, bound to given IP. If no specific IP is needed, use just an asterisk ('*'). If UDP messages are listened to, wapbox-port variable MUST be set.
`wdp-interface-name (c)`	IP or '*'	A file in which to write a log. This in addition to `stdout` and any log file defined in command line. Log-file in 'core' group is only used by the bearerbox.
`log-file`	filename	Minimum level of log-file events logged. 0 is for 'debug', 1 'info', 2 'warning, 3 'error' and 4 'panic' (see Command Line Options)
`log-level`	number 0..5	A file in which information about received/sent SMS messages is stored. Access-log in 'core' group is only used by the bearerbox.
`access-log`	filename	Indicates if `access-log` will contain standard 'markers', which means the 'Log begins', 'Log ends' markers and the prefixed timestamp. This config directive should be set to 'true' if a custom logging format is desired without a prefixed default timestamp.
`access-log-clean`	boolean	

Variable	Value	Description
`access-log-format`	string	String defining a custom log file line format. May use escape codes as defined later on to substitute values of the messages into the log entry. If no custom log format is used the standard format will be: `"%t %l [SMSC:%i] [SVC:%n] [ACT:%A] [BINF:%B] [FID:%F] [META:%D] [from:%p] [to:%P] [flags:%m:%c:%M:%C:%d] [msg:%L:%b] [udh:%U:%u]"`
`syslog-level`	number	Messages of this log level or higher will also be sent to syslog, the UNIX system log daemon. If not explicitly set with `syslog-facility`, it logs under the 'daemon' category. The default is not to use syslog, and you can set that explicitly by setting `syslog-level` to 'none'.
`syslog-facility`	string	The syslog facility to send the syslog entries to. The default is 'daemon'.

Variable	Value	Description
		String to unify received phone numbers, for SMSC routing and to ensure that SMS centers can handle them properly. This is applied to 'sender' number when receiving SMS messages from SMS Center and for 'receiver' number when receiving messages from smsbox (either sendsms message or reply to original message). Format is that first comes the unified prefix, then all prefixes which are replaced by the unified prefix, separated with comma (','). For example, for Finland an unified-prefix "+358,00358,0;+,00" should do the trick. If there are several unified prefixes, separate their rules with semicolon (';'), like "+35850,050;+35840,040". *Note that prefix routing is next to useless now that there are SMSC ID entries. To remove prefixes, use like "-,+35850,050;-,+35840,040".*
`unified-prefix`	prefix-list	
`white-list`	URL	(Deprecated, see `white-list-sender`).
`white-list-regex`	POSIX regular expression	(Deprecated, see `white-list-sender-regex`).
`black-list`	URL	(Deprecated, see `black-list-sender`).
`black-list-regex`	POSIX regular expression	(Deprecated, see `black-list-sender-regex`).

Variable	Value	Description
white-list-sender	URL	Load a list of accepted senders of SMS messages. If a sender of an SMS message is not in this list, any message received from the SMS Center, or tried to be delivered to the SMS Center is rejected. See notes of phone number format from numhash.h header file. NOTE: the system has only a precision of last 9 or 18 digits of phone numbers, so beware!
white-list-sender-regex	POSIX regular expression	A regular expression defining the set of accepted senders. See section on Regular Expressions for details.
black-list-sender	URL	As white-list-sender, but SMS messages from these sender numbers are automatically rejected.
black-list-sender-regex	POSIX regular expression	A regular expression defining the set of rejected senders. See section on Regular Expressions for details.
white-list-receiver	URL	Load a list of accepted receiver of SMS messages. If a receiver of an SMS message is not in this list, any message received from the SMS Center, or tried to be delivered to the SMS Center is rejected. See notes of phone number format from numhash.h header file. NOTE: the system has only a precision of last 9 or 18 digits of phone numbers, so beware!
white-list-receiver-regex	POSIX regular expression	A regular expression defining the set of accepted receivers. See section on Regular Expressions for details.
black-list-receiver	URL	As white-list, but SMS messages from these receiver numbers are automatically rejected.

Variable	Value	Description
black-list-receiver-regex	POSIX regular expression	A regular expression defining the set of rejected receivers. See section on Regular Expressions for details.
store-type	filename	Kannel can use store subsystem that means storing messages on hard disk or in a remote key store until they are successfully handled. By using this subsystem, no SMS messages are lost in Kannel even by crash, but theoretically some messages can duplicate when system is taken down violently. This variable defines a type of backend used for store subsystem. Now two types are supported: a) file: writes store into one single file b) spool: writes store into spool directory (one file for each message) c) redis: writes store into a redis key storage. For redis as message storage you will need a group = redis-connection and a group = store-db configuration group that define the connection to the redis server and the table name to be used. See file doc/examples/store-redis.conf for an example config secttion.
store-location	filename	Depends on store-type option used, it is ether file or spool directory, or none if redis is used as storage subsystem.
store-dump-freq	seconds	Approximated frequency how often the memory dump of current pending messages are stored to store-file, providing something has happened. Defaults to 10 seconds if not set.
http-proxy-host	hostname	Enable the use of an HTTP proxy for all HTTP requests.

Variable	Value	Description
`http-proxy-port`	port-number	
`http-proxy-exceptions`	URL-list	A list of excluded hosts from being used via a proxy. Separate each entry with space.
`http-proxy-exceptions-regex`	UNIX regular expression	Same as http-proxy-exceptions but match against UNIX regular expression.
`http-proxy-username`	username	Username for authenticating proxy use, for proxies that require this.
`http-proxy-password`	URL-list	Password for authenticating proxy use, for proxies that require this.
`ssl-client-certkey-file (c)`	filename	A PEM encoded SSL certificate and private key file to be used with SSL client connections. This certificate is used for the HTTPS client side only, i.e. for SMS service requests to SSL-enabled HTTP servers.
`ssl-server-cert-file (c)`	filename	A PEM encoded SSL certificate file to be used with SSL server connections. This certificate is used for the HTTPS server side only, i.e. for the administration HTTP server and the HTTP interface to send SMS messages.
`ssl-server-key-file (c)`	filename	A PEM encoded SSL private key file to be used with SSL server connections. This key is associated to the specified certificate and is used for the HTTPS server side only.
`ssl-trusted-ca-file`	filename	This file contains the certificates Kannel is willing to trust when working as a HTTPS client. If this option is not set, certificates are not validated and those the identity of the server is not proven.

Variable	Value	Description
`ssl-client-cipher-list`	filename	Defines the list of encryption suites and ciphers to be used for client side connections. For further details please see https://www.openssl.org/docs/manmaster/man1/ciph
`ssl-server-cipher-list`	filename	Defines the list of encryption suites and ciphers to be used for server side connections. For further details please see https://www.openssl.org/docs/manmaster/man1/ciph
`dlr-storage`	type	Defines the way DLRs are stored. Natively in-memory storage is supported, which is very fast, but has no persistancy. This means you will loose all your temporary DLR information on bearebox re-start. In addition a spool directory approach is natively supported, which garantees persistancy with a reasonable performance, depending on your disk IO performance. If you have build-in external DLR storage support, i.e. using MySQL you may define here the alternative storage type like `mysql`. Supported types are: `internal`, `spool`, `mysql`, `pgsql`, `sdb`, `mssql`, `sqlite3`, `oracle`, `redis` and `cassandra`. By default this is set to `internal`.
`dlr-spool`	filename	Depends on `dlr-storage = spool` option used, it is the spool directory to use for DLR storage data.
`maximum-queue-length`	number of messages	(Deprecated, see `sms-incoming-queue-limit`).

Variable	Value	Description
`sms-incoming-queue-limit`	number of messages	Set maximum size of incoming message queue. After number of messages has hit this value, Kannel began to discard them. Value 0 means giving strict priority to outgoing messages. -1, default, means that the queue of infinite length is accepted. (This works with any normal input, use this variable only when Kannel message queues grow very long).
`sms-outgoing-queue-limit`	number of messages	Set maximum size of outgoing message queue. After number of messages has hit this value, Kannel began to discard them. The default value, 1 million, works for most installations.
`smsbox-max-pending`	number of messages	Maximum number of pending messages on the line to smsbox compatible boxes.
`sms-resend-freq`	seconds	Frequency for the SMS resend thread in which temporarily failed or queued messages will be resent. Defaults to 60 seconds.
`sms-resend-retry`	number	Maximum retry attempts for the temporarily failed messages. Defaults to -1, means: unlimited.
`sms-combine-concatenated-mo`	boolean	Whether Kannel should attempt to combine concatenated MO SMS prior to passing them over to smsbox. Default is true
`sms-combine-concatenated-mo-timeout`	seconds	How long to wait for all concatenated message parts to arrive before timeouting. Default 1800 seconds.
`http-timeout`	seconds	Sets socket timeout in seconds for outgoing client http connections. Optional. Defaults to 240 seconds.

A sample more complex 'core' group could be something like this:

```
group = core
admin-port = 13000
admin-password = f00bar
status-password = sTat
admin-deny-ip = "*.*.*.*"
admin-allow-ip = "127.0.0.1;200.100.0.*"
smsbox-port = 13003
wapbox-port = 13004
box-deny-ip = "*.*.*.*"
box-allow-ip = "127.0.0.1;200.100.0.*"
wdp-interface-name = "*"
log-file = "kannel.log"
log-level = 1
access-log = "kannel.access"
unified-prefix = "+358,00358,0;+,00"
white-list = "http://localhost/whitelist.txt"
```

Running Kannel

To start the gateway, you need to start each box you need. You always need the bearer box, and depending on whether you want WAP and SMS gateways you need to start the WAP and SMS boxes. If you want, you can run several of them, but we'll explain the simple case of only running one each.

Starting the gateway

After you have compiled Kannel and edited configuration file for your taste, you can either run Kannel from command line or use supplied `start-stop-daemon` and `run_kannel_box` programs to use it as a daemon service (more documentation about that later).

If you cannot or do not know how to set up daemon systems or just want to test Kannel, you probably want to start it from command line. This means that you probably want to have one terminal window for each box you want to start (xterm or screen will do fine). To start the bearerbox, give the following command:

```
./bearerbox -v 1 [config-file]
```

The `-v 1` sets the logging level to `INFO`. This way, you won't see a large amount of debugging output (the default is `DEBUG`). Full explanation of Kannel command line arguments is below.

[config-file] is the name of the configuration file you are using with Kannel. The basic distribution packet comes with two sample configuration files, `smskannel.conf` and `wapkannel.conf` (in `gw` sub directory), of which the first one is for testing out SMS Kannel and the second one for setting up a WAP Kannel. Feel free to edit those configuration files to set up your own specialized system.

After the bearer box, you can start the WAP box:

```
./wapbox -v 1 [config-file]
```

or the SMS box:

```
./smsbox -v 1 [config-file]
```

or both, of course. The order does not matter, except that you need to start the bearer box before the other boxes. Without the bearer box, the other boxes won't even start.

Command line options

Bearerbox, smsbox and wapbox each accept certain command line options and arguments when they are launched. These arguments are:

Table 3-2. Kannel Command Line Options

`--version`	Print the version of the Kannel binary.
`-v <level>` `--verbosity <level>`	Set verbosity level for stdout (screen) logging. Default is 0, which means 'debug'. 1 is 'info, 2 'warning', 3 'error' and 4 'panic'
`-D <places>` `--debug <places>`	Set debug-places for 'debug' level output.
`-F <file-name>` `--logfile <file-name>`	Log to file named file-name, too. Does not overrun or affect any log-file defined in configuration file.
`-V <level>` `--fileverbosity <level>`	Set verbosity level for that extra log-file (default 0, which means 'debug'). Does not affect verbosity level of the log-file defined in configuration file, not verbosity level of the `stdout` output.
`-S` `--suspended`	Start the system initially at SUSPENDED state (see below, bearerbox only)
`-I` `--isolated`	Start the system initially at ISOLATED state (see below, bearerbox only)

`-H` `--tryhttp`	Only try to open HTTP sendsms interface; if it fails, only warn about that, do not exit. (smsbox only)
`-g` `--generate`	Dump all known config groups and config keys to stdout and exit.
`-u <username>` `--user <username>`	Change process user-id to the given.
`-p <filename>` `--pid-file <filename>`	Write process PID to the given file.
`-d` `--daemonize`	Start process as daemon (detached from a current shell session). Note: Process will change CWD (Current working directory) to `/`, therefore you should ensure that all paths to binary/config/config-includes are absolute instead of relative.
`-P` `--parachute`	Start watcher process. This process watch a child process and if child process crashed will restart them automatically.
`-X <scriptname>` `--panic-script <scriptname>`	Execute a given shell script or binary when child process crash detected. This option is usable only with `--parachute/-P`. Script will be executed with 2 arguments: scriptname 'processname' 'respawn-count'.

Kannel statuses

In Kannel, there are four states for the program (which currently directly only apply to bearerbox):

a. Running. The gateway accepts, proceeds and relies messages normally. This is the default state for the bearerbox.

b. Suspended. The gateway does not accept any new messages from SMS centers nor from UDP ports. Neither does it accept new sms and wapbox connections nor sends any messages already in the system onward.

c. Isolated. In this state, the gateway does not accept any messages from external message providers, which means SMS Centers and UDP ports. It still processes any messages in the system and can accept new messages from sendsms interface in smsbox.

d. Full. Gateway does not accept any messages from SMS centers, because `maximum-queue-length` is achieved.

e. Shutdown. When the gateway is brought down, it does not accept any new messages from SMS centers and UDP ports, but processes all systems already in the system. As soon as any queues are emptied, the system exits

The state can be changed via HTTP administration interface (see below), and shutdown can also be initiated via TERM or INT signal from terminal. In addition, the bearerbox can be started already in suspended or isolated state with -S or -I command line option, see above.

HTTP administration

Kannel can be controlled via an HTTP administration interface. All commands are done as normal HTTP queries, so they can be easily done from command line like this:

```
lynx -dump "http://localhost:12345/shutdown?password=bar"
```

...in which the '12345' is the configured admin-port in Kannel configuration file (see above). For most commands, admin-password is required as a argument as shown above. In addition, HTTP administration can be denied from certain IP addresses, as explained in configuration chapter.

Note that you can use these commands with WAP terminal, too, but if you use it through the same Kannel, replies to various suspend commands never arrive nor can you restart it via WAP anymore.

Table 3-3. Kannel HTTP Administration Commands

`status or status.txt`	Get the current status of the gateway in a text version. Tells the current state (see above) and total number of messages relied and queuing in the system right now. Also lists the total number of smsbox and wapbox connections. No password required, unless `status-password` set, in which case either that or main admin password must be supplied.
`status.html`	HTML version of status
`status.xml`	XML version of status
`status.wml`	WML version of status
`store-status or store-status.txt`	Get the current content of the store queue of the gateway in a text version. No password required, unless `status-password` set, in which case either that or main admin password must be supplied.
`store-status.html`	HTML version of store-status
`store-status.xml`	XML version of store-status
`suspend`	Set Kannel state as 'suspended' (see above). Password required.

`isolate`	Set Kannel state as 'isolated' (see above). Password required.
`resume`	Set Kannel state as 'running' if it is suspended or isolated. Password required.
`shutdown`	Bring down the gateway, by setting state to 'shutdown'. After a shutdown is initiated, there is no other chance to resume normal operation. However, 'status' command still works. Password required. If shutdown is sent for a second time, the gateway is forced down, even if it has still messages in queue.
`flush-dlr`	If Kannel state is 'suspended' this will flush all queued DLR messages in the current storage space. Password required.
`start-smsc`	Re-start a single SMSC link. Password required. Additionally the `smsc` parameter must be given to identify which `smsc-admin-id` should be re-started. The configuration for the SMSC link is re-read from disk before the action is performed.
`stop-smsc`	Shutdown a single SMSC link. Password required. Additionally the `smsc` parameter must be given (see above).
`add-smsc`	Adds an SMSC link previously removed or created after the service was started. Password required. Additionally the `smsc` parameter must be given (see above).
`remove-smsc`	Removes an existing SMSC link. Password required. Additionally the `smsc` parameter must be given (see above). If you want a permanent removal, you should also remove the entry from the configuration file or it will be recreated after a service restart.
`restart`	This is the hard restarting version, where we fully stop the bearerbox and then initiate from ground up. Beware that you loose the smsbox connections in such a case.

`graceful-restart`	Re-start bearerbox gracefully, hence only the changes in the configuration for the SMSC connections are performed. I.e. an added SMSC group will be start, and a removed SMSC group will be shutdown and destroyed. Any unchanged SMSC connection keeps on running without interruption. Password required. This is the soft restarting version, where we keep all internal parts of the bearebox running, including the smsbox connections. The same graceful restart can be initiated by sending a SIGHUP signal to the bearerbox daemon process.
`log-level`	Set Kannel log-level of log-files while running. This allows you to change the current log-level of the log-files on the fly. This includes the main event log and the SMSC specific logs, if any exist.
`reload-lists`	Re-loads the 'white-list' and 'black-list' URLs provided in the core group. This allows Kannel to keep running while the remote lists change and signal bearerbox to re-load them on the fly.
`remove-message`	Removes the message with the give `id` (an UUID) from the local storage subsystem.

Chapter 4. Setting up a WAP gateway

This chapter tells you how to set Kannel up as a WAP gateway.

WAP gateway configuration

To set up a WAP Kannel, you have to edit the 'core' group in the configuration file, and define the 'wapbox' group.

You must set following variables for the 'core' group: `wapbox-port` and `wdp-interface-name`. See previous chapter about details of these variables.

With standard distribution, a sample configuration file `wapkannel.conf` is supplied. You may want to take a look at that when setting up a WAP Kannel.

Wapbox group

If you have set `wapbox-port` variable in the 'core' configuration group, you *MUST* supply a 'wapbox' group.

The simplest working 'wapbox' group looks like this:

```
group = wapbox
bearerbox-host = localhost
```

There is, however, multiple optional variables for the 'wapbox' group.

Table 4-1. Wapbox Group Variables

Variable	Value	Description
`group (m)`	`wapbox`	This is mandatory variable
`bearerbox-host (m)`	`hostname`	The machine in which the bearerbox is.
`timer-freq`	value-in-seconds	The frequency of how often timers are checked out. Default is 1
`http-interface-name`	IP address	If this is set then Kannel do outgoing HTTP requests using this interface.

Variable	Value	Description
map-url	URL-pair	The pair is separated with space. Adds a single mapping for the left side URL to the given destination. If you append an asterisk '*' to the left side URL, its prefix Is matched against the incoming URL. Whenever the prefix matches, the URL will be replaced completely by the right side. In addition, if if you append an asterisk to the right side URL, the part of the incoming URL coming after the prefix, will be appended to the right side URL. Thus, for a line: map-url = "http://source/* http://destination/*" and an incoming URL of "http://source/some/path", the result will be "http://destination/some/path"
map-url-max	number	If you need more than one mapping, set this to the highest number mapping you need. The default gives you 10 mappings, numbered from 0 to 9. Default: 9
map-url-0	URL-pair	Adds a mapping for the left side URL to the given destination URL. Repeat these lines, with 0 replaced by a number up to map-url-max, if you need several mappings.
device-home	URL	Adds a mapping for the URL DEVICE:home (as sent by Phone.com browsers) to the given destination URL. There is no default mapping. NOTE: the mapping is added with both asterisks, as described above for the "map-url" setting. Thus, the above example line is equivalent to writing map-url = "DEVICE:home* http://some.where/*"

Variable	Value	Description
		As with bearerbox 'core' group.
log-file	filename	
log-level	number 0..5	
access-log	filename	A file containing all requested URLs from clients while wapbox operation, including client IP, HTTP server User-Agent string, HTTP response code, size of reply.
access-log-clean	boolean	Indicates if access-log will contain standard 'markers', which means the 'Log begins', 'Log ends' markers and the prefixed timestamp. This config directive should be set to 'true' if a custom logging format is desired without a prefixed default timestamp.
access-log-time	string	Determine which timezone we use for access logging. Use 'gmt' for GMT time, anything else will use local time. Default is local time.
syslog-level	number	Messages of this log level or higher will also be sent to syslog, the UNIX system log daemon. If not explicitly set with syslog-facility, it logs under the 'daemon' category. The default is not to use syslog, and you can set that explicitly by setting syslog-level to 'none'.
syslog-facility	string	The syslog facility to send the syslog entries to. The default is 'daemon'.
smart-errors	bool	If set wapbox will return a valid WML deck describing the error that occurred while processing an WSP request. This may be used to have a smarter gateway and let the user know what happened actually.

Variable	Value	Description
`wml-strict`	bool	If set wapbox will use a strict policy in XML parsing the WML deck. If not set it will be more relaxing and let the XML parser recover from parsing errors. This has mainly impacts in how smart the WML deck and it's character set encoding can be adopted, even while you used an encoding that does not fit the XML preamble. BEWARE: This may be a vulnerability in your wapbox for large bogus WML input. Therefore this defaults to 'yes', strict parsing. is assumed.
`http-timeout`	seconds	Sets socket timeout in seconds for outgoing client http connections. Optional. Defaults to 240 seconds.

Running WAP gateway

WAP Gateway is ran as explained in previous chapter.

Checking whether the WAP gateway is alive

You can check whether the WAP gateway (both the bearerbox and the wapbox) is alive by fetching the URL `kannel:alive`.

Chapter 5. Setting up wtls security

This chapter tells you how to set Kannel up to handle wtls traffic.

`wtls` group is optional and single. The prerequisites for this group are to have defined a wapbox group, and a pair of SSL certificates available. Instructions on how to create self-signed 1024-bit RSA certificates are in Appendix B.

Current imlementation provides for the following functionality:

A. Supported MACs:

- SHA_0
- SHA_40
- SHA_80
- SHA_NOLIMIT
- MD5_40
- MD5_80
- MD5_NOLIMIT

Missing:

- SHA_XOR_40

B. Supported Ciphers:

- RC5_CBC_40
- RC5_CBC_56
- RC5_CBC
- DES_CBC
- DES_CBC_40

Missing:

- NULL_bulk
- TRIPLE_DES_CBC_EDE
- IDEA_CBC_40
- IDEA_CBC_56
- IDEA_CBC

C. Supported Keys:

- RSA_anon

Missing:

- RSA_anon_512
- RSA_anon_768
- RSA_NOLIMIT
- RSA_512
- RSA_768
- ECDH_anon
- ECDH_anon_113
- ECDH_anon_131
- ECDH_ECDSA_NOLIMIT

Keys might seem a shortcoming, but all mobiles support RSA_anon. Some of the other RSA_anon keys (i.e. RSA_anon_512, RSA_anon_768) are propably supported as well, just haven't been tested yet.

D. All wtls states except:

- Suspend/Resume wtls session
- Cipher change when already connected. In practice this is handled through another client hello, while already connected to the same client

The simplest working `wtls` group looks like this:

```
group = wtls
certificate-file = /etc/kannel/server.crt
privatekey-file = /etc/kannel/server.key
```

Can also be the same single combined pem file with both certificate and privatekey parts. The complete variable list for the `wtls` group is:

Wtls configuration

Table 5-1. Wtls Group Variables

Variable	Value	Description
`group (m)`	`wtls`	This is mandatory variable
`certificate-file (m)`	`filename`	Public key SSL certificate.
`privatekey-file (m)`	`filename`	Private key SSL certificate.
`privatekey-password (o)`	`Phrase`	Optional. Needed only if private key was created with a passphrase.

Chapter 6. Setting up MSISDN provisioning for WAP gateway

This chapter tells you how to set Kannel up to deliver the MSISDN number of the WAP device using the WAP gateway.

Mostly this feature is interested because the WAP protocol stack itself does not provide such a protocol layer bridging of information. In case you want to know which unique MSISDN is used by the WAP device your HTTP application is currently interacting, then you can pick this information from a RADIUS server that is used by a NAS (network access server) for authentication and accounting purposes.

Kannel provides a RADIUS accounting proxy thread inside wapbox which holds a mapping table for the dynamically assigned (DHCP) IP addresses of the WAP clients and their MSISDN numbers provided to the NAS device.

Beware that you *HAVE TO* to be in control of the NAS to configure it to use Kannel's wapbox as RADIUS accounting server. You can not use MSISDN provisioning via Kannel's RADIUS accounting proxy if you can not forward the accounting packets to Kannel's accounting proxy thread. So obviously this feature is for operators and people who have dedicated dial-in servers (NAS).

RADIUS accounting proxy configuration

To set up the RADIUS accounting proxy thread inside Kannel you have to define a 'radius-acct' group.

RADIUS-Acct configuration

The simplest working 'radius-acct' group looks like this:

```
group = radius-acct
our-port = 1646
secret-nas = radius
```

This example does not forward any accounting packets to a remote RADIUS server. There is, however, multiple optional variables for the 'radius-acct' group.

Table 6-1. RADIUS-Acct Group Variables

Variable	Value	Description
`group (m)`	`radius-acct`	This is mandatory variable if you want to have Kannel's RADIUS accounting proxy thread to be active inside wapbox.

Variable	Value	Description
secret-nas (m)	string	Specifies the shared secret between NAS and our RADIUS accounting proxy.
secret-radius	string	Specifies the shared secret between our RADIUS accounting proxy and the remote RADIUS server itself in case we are forwarding packets.
our-host	IP address	Specifies the local interface where our-port value will bind to. (defaults to 0.0.0.0)
remote-host	FQDN or IP address	Specifies the remote RADIUS server to which we forward the accounting packets. If none is given, then no forwarding of accounting packets is performed.
our-port	number	Specifies the port to which our RADIUS accounting proxy thread will listen to. (defaults according to RFC2866 to 1813)
remote-port	number	Specifies the port to which our RADIUS accounting proxy thread will forward any packets to the remote-host. (defaults according to RFC2866 to 1813)
remote-timeout	number	Specifies the timeout value in milliseconds for waiting on a response from the remote RADIUS server. (defaults to 40 msecs.)
nas-ports	number	Specifies how many possible clients can connect simultaneously to the NAS. This value is used to initialize the mapping hash table. If does not require to be exact, because the table grows automatically if required. (defaults to 30)

Variable	Value	Description
`unified-prefix`	prefix-list	String to unify provided phone numbers from NAS. Format is that first comes the unified prefix, then all prefixes which are replaced by the unified prefix, separated with comma (’,’). For example, for Finland an unified-prefix "+358,00358,0;+,00" should do the trick. If there are several unified prefixes, separate their rules with semicolon (’;’), like "+35850,050;+35840,040".

Using the MSISDN provisioning on HTTP server side

Kannel’s wapbox will include a specific HTTP header in the HTTP request issued to the URL provided by the WAP client. That HTTP header is `X-WAP-Network-Client-MSISDN`. It will carry as value the MSISDN of the WAP client if the RADIUS accounting proxy has received a valid accounting packet from the NAS that provided the client IP.

Chapter 7. Setting up a SMS Gateway

This chapter is a more detailed guide on how to set up Kannel as an SMS gateway.

Required components

To set up an SMS gateway, you need, in addition to a machine running Kannel, access to (an operator's) SMS center, or possibly to multiple ones. The list of supported SMS centers and their configuration variables is below.

If you do not have such access, you can still use Kannel as an SMS gateway via *phone-as-SMSC* feature, by using a GSM phone as a virtual SMS center.

In addition to an SMS center (real or virtual), you need some server to handle any SMS requests received. This server then has simple or more complex cgi-bins, programs or scripts to serve HTTP requests generated by Kannel in response to received SMS messages. These services can also initiate SMS push via Kannel smsbox HTTP sendsms interface.

SMS gateway configuration

To set up a SMS Kannel, you have to edit the 'core' group in the configuration file, and define an 'smsbox' group plus one or more 'sms-service' groups, plus possibly one or more 'sendsms-user' groups.

For the 'core' group, you must set the following variable: `smsbox-port`. In addition, you may be interested to set `unified-prefix`, `white-list` and/or `black-list` variables. See above for details of these variables.

A sample configuration file `smskannel.conf` is supplied with the standard distribution. You may want to take a look at that when setting up an SMS Kannel.

SMS centers

To set up the SMS center at Kannel, you have to add a 'smsc' group into configuration file. This group must include all the data needed to connect that SMS center. You may also want to define an ID (identification) name for the SMSC, for logging and routing purposes.

SMSC ID is an abstract name for the connection. It can be anything you like, but you should avoid any special characters. You do not need to use ID, but rely on SMS center IP address and other information. However, if you use the ID, you do not need to re-define sms-services nor routing systems if the IP of the SMS Center is changed, for example.

Common 'smsc' group variables are defined in the following table. The first two (`group` and `smsc`) are mandatory, but rest can be used if needed.

Table 7-1. SMSC Group Variables

Variable	Value	Description
`group (m)`	`smsc`	This is a mandatory variable
`smsc (m)`	`string`	Identifies the SMS center type. See below for a complete list.
`smsc-id`	`string`	An optional name or id for the smsc. Any string is acceptable, but semicolon ';' may cause problems, so avoid it and any other special non-alphabet characters. This 'id' is written into log files and can be used to route SMS messages, and to specify the used SMS-service. Several SMSCs can have the same id. The name is case-insensitive. Note that if SMS Center connection has an assigned SMSC ID, it does NOT automatically mean that messages with identical SMSC ID are routed to it; instead configuration variables `denied-smsc-id`, `allowed-smsc-id` and `preferred-smsc-id` is used for that. If you want to use Delivery Reports, you must define this.
`smsc-admin-id`	`string`	An optional id for the smsc to be used on administrative commands. This allows commands targeted to individual binds even if they share the same `smsc-id` (for load balancing scenarios, for example). Any string is acceptable, but semicolon ';' may cause problems, so avoid it and any other special non-alphabet characters. Several SMSCs can have the same admin-id, though it's not recommended. The name is case-insensitive.

Variable	Value	Description
throughput	float (messages/sec)	If SMSC requires that Kannel limits the number of messages per second, use this variable. This is considered as active throttling. (optional)
denied-smsc-id	id-list	SMS messages with SMSC ID equal to any of the IDs in this list are never routed to this SMSC. Multiple entries are separated with semicolons (';')
allowed-smsc-id	id-list	This list is opposite to previous: only SMS messages with SMSC ID in this list are ever routed to this SMSC. Multiple entries are separated with semicolons (';')
preferred-smsc-id	id-list	SMS messages with SMSC ID from this list are sent to this SMSC instead than to SMSC without that ID as preferred. Multiple entries are separated with semicolons (';')
allowed-prefix	prefix-list	A list of phone number prefixes which are accepted to be sent through this SMSC. Multiple entries are separated with semicolon (';'). For example, "040;050" prevents sending of any SMS message with prefix of 040 or 050 through this SMSC. If denied-prefix is unset, only these numbers are allowed. If set, number are allowed if present in allowed or not in denied list.
denied-prefix	prefix-list	A list of phone number prefixes which are NOT accepted to be sent through this SMSC.
preferred-prefix	prefix-list	As denied-prefix, but SMS messages with receiver starting with any of these prefixes are preferably sent through this SMSC. In a case of multiple preferences, one is selected at random (also if there are preferences, SMSC is selected randomly)

Variable	Value	Description
unified-prefix	prefix-list	String to unify received phone numbers, for SMSC routing and to ensure that SMS centers can handle them properly. This is applied to 'sender' number when receiving SMS messages from SMS Center and for 'receiver' number when receiving messages from smsbox (either sendsms message or reply to original message). Format is that first comes the unified prefix, then all prefixes which are replaced by the unified prefix, separated with comma (','). For example, for Finland an unified-prefix "+358,00358,0;+,00" should do the trick. If there are several unified prefixes, separate their rules with semicolon (';'), like "+35850,050;+35840,040". *Note that prefix routing is next to useless now that there are SMSC ID entries. To remove prefixes, use like "-,+35850,050;-,+35840,040".*
alt-charset	number	As some SMS Centers do not follow the standards in character coding, an alt-charset character conversion is presented. This directive acts different for specific SMSC types. Please see your SMSC module type you want to use for more details.
alt-dcs	boolean	Optional data coding scheme value usage. Sets the 'alt-dcs' value for the sendsms HTTP interface to a fixed value for each SMS sent via this SMSC connection. If equals to yes, uses FX. If equals to no (default), uses 0X.

Variable	Value	Description
our-host	hostname	Optional hostname or IP address in which to bind the connection in our end. TCP/IP connection only.
log-file	filename	A file in which to write a log of the given smsc output. Hence this allows to log smsc specific entries to a separate file.
log-level	number 0..5	Minimum level of log-file events logged. 0 is for 'debug', 1 'info', 2 'warning, 3 'error' and 4 'panic' (see Command Line Options)
reconnect-delay	number	Number of seconds to wait between single re-connection attempts. Hence all SMSC modules should use this value for their re-connect behavior. (Defaults to '10' seconds).
reroute	boolean	If set for a smsc group, all inbound messages coming from this smsc connection are passed internally to the outbound routing functions. Hence this messages is not delivered to any connected box for processing. It is passed to the bearerbox routing as if it would have originated from an externally connected smsbox. (Defaults to 'no').
reroute-smsc-id	string	Similar to 'reroute'. Defines the explicit smsc-id the MO message should be passed to for MT traffic. Hence all messages coming from the the configuration group smsc are passed to the outbound queue of the specified smsc-id. This allows direct proxying of messages between 2 smsc connections without injecting them to the general routing procedure in bearerbox.

Variable	Value	Description
reroute-receiver	string	Similar to 'reroute'. Defines the explicit smsc-id routes for specific receiver numbers of messages that are coming from this smsc connection. Format is that first comes the smsc-id to route the message to, then all receiver numbers that should be routed, separated with comma (','). For example, route receivers 111 and 222 to smsc-id A and 333 and 444 to smsc-id B would look like: "A,111,222; B,333,444".
reroute-receiver-regex	string	Similar to 'reroute-receiver'. Defines the explicit smsc-id routes for specific receiver number regex patterns of messages that are coming from this smsc connection. Format is that first comes the smsc-id to route the message to, then all receiver numbers as regular expressions that should be routed, separated with comma (','). For example, route receivers that start with 1 and 2 to smsc-id A and those that start with 3 and 4 to smsc-id B would look like: "A,^1.*,^2.*; B,^3.*,^4.*". (Please ensure the reserved characters ';' and ',' are not used in the regular expression to avoid parsing errors).
reroute-dlr	boolean	Indicate whether DLR's should be re-routed too, if one of above reroute rules are enabled. Please note, that SMSC-Module should support DLR sending. At time of writing none of SMSC-Module supports DLR sending.
allowed-smsc-id-regex	POSIX regular expression	SMS messages with SMSC ID equal to any of the IDs in this set of SMSC IDs are always routed to this SMSC. See section on Regular Expressions for details.

Variable	Value	Description
denied-smsc-id-regex	POSIX regular expression	SMS messages with SMSC ID equal to any of the IDs in this set of SMSC IDs are never routed to this SMSC. See section on Regular Expressions for details.
preferred-smsc-id-regex	POSIX regular expression	SMS messages with SMSC ID in this set of SMSC IDs are sent to this SMSC as preferred. See section on Regular Expressions for details.
allowed-prefix-regex	POSIX regular expression	A set of phone number prefixes which are accepted to be sent through this SMSC. See section on Regular Expressions for details.
denied-prefix-regex	POSIX regular expression	A set of phone number prefixes which may not be sent through this SMSC. See section on Regular Expressions for details.
preferred-prefix-regex	POSIX regular expression	As denied-prefix-regex, but SMS messages with receiver matching any of these prefixes are preferably sent through this SMSC. In a case of multiple preferences, one is selected at random. See section on Regular Expressions for details.
max-sms-octets	number	Maximum allowed octets for this SMSC. If this maximum exceeds Kannel will split SMS into multiparts. Default: 140
dead-start	boolean	Optional, defines if the SMSC connection type is started in disconnected mode (dead). This allows SMSC to be connected at a later time with the add-smsc HTTP administration command. If set to 'true' the connection will start as 'dead' and will require a start-smsc command to activate it. Defaults to 'false' if not present.

Variable	Value	Description
instances	number	The number of same instances of this group to be created. This allows to declare the config group one time in the configuration and to multiply it by this value for the numner of runtime instances. (default: 1)

In addition to these common variables there are several variables used by certain SMS center connections. Each currently supported SMS center type is explained below, with configuration group for each. Note that many of them use variables with same name, but most also have some specific variables.

NOTE: SMS center configuration variables are a bit incomplete, and will be updated as soon as people responsible for the protocols are contacted. Meanwhile, please have patience.

Nokia CIMD 1.37 and 2.0

Support for CIMD 1.37 is quite old and will be removed in a future version of Kannel. Please let us know if you still need it.

```
group = smsc
smsc = cimd
host = 100.101.102.103
port = 600
smsc-username = foo
smsc-password = bar
```

The driver for CIMD2 is a "receiving SME" and expects the SMSC to be configured for that. It also expects the SMSC to automatically send stored messages as soon as Kannel logs in (this is the normal configuration).

```
group = smsc
smsc = cimd2
host = 100.101.102.103
port = 600
smsc-username = foo
smsc-password = bar
keepalive = 5
my-number = "12345"
```

Variable	Value	Description
host (m)	hostname	Machine that runs the SMSC. As IP (100.100.100.100) or hostname (their.machine.here)
port (m)	port-number	Port number in the smsc host machine
smsc-username (m)	string	Username in the SMSC machine/connection account

Variable	Value	Description
smsc-password (m)	string	Password in the SMSC machine needed to contact SMSC
keepalive	number	SMSC connection will not be left idle for longer than this many seconds. The right value to use depends on how eager the SMSC is to close idle connections. If you see many unexplained reconnects, try lowering this value. Set it to 0 to disable this feature.
no-dlr	boolean	Optional. If defined, status delivery report requests (DLR) won't be requested at all. Some CIMD2 SMSC have prohibited these reports so if you are getting error like "Incorrect status report request parameter usage", this option is for you. Defaults to "false".
my-number	string	The number that the SMSC will add in front of the sender number of all messages sent from Kannel. If Kannel is asked to send a message, it will remove this prefix from the sender number so that the SMSC will add it again. If the prefix was not present, Kannel will log a warning and will not send the sender number. If my-number is not set, or is set to "never", then Kannel will not send the sender number to the SMSC at all. If you want Kannel to pass all sender numbers to the SMSC unchanged, then just set sender-prefix to the empty string "".
our-port	port-number	Optional port number in which to bind the connection in our end.

CMG UCP/EMI 4.0 and 3.5

Warning

See Appendix A for changes.

Kannel supports two types of connections with CMG SMS centers: direct TCP/IP connections (`emi`) and ISDN/modem (X.25) connection (`emi_x25`).

> **Note:** `emi_x25` is an old implementation and supports less features than it's IP counterpart. If you still need this module, please tell us to `devel@kannel.org` so we can fix it.

Sample configurations for these are:

```
group = smsc
smsc = emi
host = 103.102.101.100
port = 6000
smsc-username = foo
smsc-password = bar
keepalive = 55
our-port = 600 (optional bind in our end)
receive-port = 700 (the port in which the SMSC will contact)
idle-timeout = 30

group = smsc
smsc = emi_x25
phone = ...
device = /dev/tty0
smsc-username = foo
smsc-password = bar
```

Variable	Value	Description
`host (c)`	`hostname`	Machine that runs SMSC. As IP (100.100.100.100) or hostname (their.machine.here)
`port (c)`	`port-number`	Port number in the SMSC host machine
`alt-host`	`hostname`	Optional alternate Machine that runs SMSC. As IP (100.100.100.100) or hostname (their.machine.here) (If undef but exists alt-port, emi2 would try host:alt-port)

Variable	Value	Description
alt-port	port-number	Optional alternate Port number in the SMSC host machine (If undef but exists alt-host, emi2 would try alt-host:port)
smsc-username	string	Username in the SMSC machine/connection account
smsc-password	string	Password in the SMSC machine needed to contact SMSC
device (c)	device-name	The device the modem is connected to, like /dev/ttyS0. ISDN connection only.
phone (c)	string	Phone number to dial to, when connecting over a modem to an SMS center.
our-port	port-number	Optional port number in which to bind the connection in our end. TCP/IP connection only.
our-receiver-port	port-number	Optional port number allowing to specify our port when connecting an SMSC. TCP/IP connection only.
receive-port	port-number	Optional port number we listen to and to which the SMS center connects when it has messages to send. Required if SMS center needs one connection to send and other to receive. TCP/IP connection only.
appname	string	Name of a "Send only" service. Defaults to send. All outgoing messages are routed through this service.
connect-allow-ip	IP-list	If set, only connections from these IP addresses are accepted to receive-port. TCP/IP connection only.

Variable	Value	Description
idle-timeout	number (seconds)	If this option is set to a value larger than 0, then the connection will be closed after the configured amount of seconds without activity. This option interacts with the keepalive configuration option. If keepalive is smaller than idle-timeout, then the connection will never be idle and those this option has no effect. If keepalive is larger than idle-timeout, than keepalive reopens the connection. This allows one to poll for pending mobile originated Short Messages at the SMSC.
keepalive	number (seconds)	A keepalive command will be sent to the SMSC connection this many seconds after the last message. The right value to use depends on how eager the SMSC is to close idle connections. 50 seconds is a good guess. If you see many unexplained reconnects, try lowering this value. Set it to 0 to disable this feature. Requires username or my-number to be set.
wait-ack	number (seconds)	A message is resent if the acknowledge from SMSC takes more than this time. Defaults to 60 seconds.
wait-ack-expire	number	Defines what kind of action should be taken if the the ack of a message expires. The options for this value are: 0x00 - disconnect/reconnect, (default) 0x01 - as is now, re-queue, but this could potentially result in the msg arriving twice 0x02 - just carry on waiting (given that the wait-ack should never expire this is the must accurate)

Variable	Value	Description
flow-control	number	This SMSC can support two types of flow control. The first type of flow control is a `stop-and-wait` protocol, when this parameter equals to '1'. During the handling of commands, no other commands shall be sent before the a response is received. Any command that is sent before the reception of the response will be discarded. The second type of flow control is `windowing`, when this parameter is unset or equals '0'. In this case a maximum of n commands can be sent before a response is received.
window	number (messages)	When using `flow-control=0`, emi works in windowed flow control mode. This variable defines the size of the window used to send messages. (optional, defaults to the maximum - 100)
my-number	number	If the large account number is different from the short number, assign it with this variable. For example, if short number is 12345 and large account is 0100100100101234 (IP+port), set my-number to 12345 and every message received will have that receiver. In addition, if you are bound to the SMSC without an explicit login, use this configuration directive to enable keep-alive (OP/31) operations.
alt-charset	number	Defines which character conversion kludge may be used for this specific link. Currently implemented alternative charsets are defined in "alt_charsets.h" and new ones can be added.
notification-pid	4 num char.	Notification PID value. See below for a complete list and other notes. Example: 0539 a

Variable	Value	Description
notification-addr	string (max 16)	Notification Address. Example: 0100110120131234. [b]

Notes:

a. If you set notification-pid, you should also set notification-addr.

b. If you set notification-addr, you should also set notification-pid.

You should use notification-pid and notification-addr if you need to inform your exact "address" to your smsc. For example, if you have a Multiple-Address (MA) account with several connections to the same short number, you may need to tell your smsc to send delivery reports to the exact instance that sent the message. This is required because if you send a message with instance 1, your instance 2 wouldn't know about it, unless you use a shared DB store for delivery reports.

Notification PID Value	Description
0100	Mobile Station
0122	Fax Group 3
0131	X.400
0138	Menu over PSTN
0139	PC appl. over PSTN (E.164)
0339	PC appl. over X.25 (X.121)
0439	PC appl. over ISDN (E.164)
0539	PC appl. over TCP/IP

SMPP 3.3, 3.4 and 5.0

This implements Short Message Peer to Peer (SMPP) Protocol 5.0, with most used support for 3.4 and full support for 3.3. Sample configuration:

```
group = smsc
smsc = smpp
host = 123.123.123.123
port = 600
smsc-username = "STT"
smsc-password = foo
system-type = "VMA"
address-range = ""
```

Variable	Value	Description
host (m)	hostname	Machine that runs SMSC. As IP (100.100.100.100) or hostname (their.machine.here)
port (m)	port-number	The port number for the TRANSMITTER connection to the SMSC. If set, receive-port must be ommited or set to 0.

Variable	Value	Description
use-ssl	bool	Defines whether we should try to bind with SSL enabled connection to create a SSMPP (secure SMPP) session.
ssl-client-certkey-file (c)	filename	A PEM encoded SSL certificate and private key file to be used for SSL connections. This option is used together with use-ssl for client certificate validation with SMPP SMSCs requiring it.
transceiver-mode	bool	Use a TRANSCEIVER mode connection to the SMSC. It uses the standard transmit 'port' to define the port to connect to. This is a SMPP 3.4 and 5.0 feature and will not work on an earlier SMSC using SMPP 3.3. This will do a bind_transceiver only and will not attempt to fall back to doing transmit and receive on the same connection.
receive-port	port-number	The port number for the RECEIVER connection to the SMSC. If set, port must be ommited or set to 0.
smsc-username (m)	string	The 'username' of the Messaging Entity connecting to the SMSC. If the SMSC operator reports that the "TELEPATH SYSTEM MANAGER TERMINAL" view "Control.Apps.View" value "Name:" is "SMPP_ZAPVMA_T" for the transmitter and "SMPP_ZAPVMA_R" for the receiver the smsc-username value is accordingly "SMPP_ZAP". Note that this used to be called system-id (the name in SMPP documentation) and has been changed to smsc-username to make all Kannel SMS center drivers use the same name.

Variable	Value	Description
smsc-password (m)	string	The password matching the "smsc-username" your teleoperator provided you with.
system-type (m)	string	Optional; used to categorize the type of ESME that is binding to the SMSC. Examples include "VMS" (voice mail system) and "OTA" (over-the-air activation system). If not set, defaults to "VMA" (voice mail alerting).
service-type	string	Optional; if specified, sets the service type for the SMSC. If unset, the default service type is used. This may be used to influence SMS routing (for example). The SMSC operator may also refer to this as the "profile ID". The maximum length of the service type is 6, according to the SMPP specification v3.4. Defined values are: "CMT" (cellular messaging), "CPT" (cellular paging), "VMN" (voice mail notification), "VMA" (voice mail alerting), "WAP" (wireless application protocol), "USSD" (unstructured supplementary services data), "CBS" (cell broadcast service) and "GUTS" (generic UDP transport service). Other values may be defined mutually between the SMSC and the ESME application.
interface-version	number	Change the "interface version" parameter sent from Kannel to a value other then 0x34 (for SMPP v3.4). The value entered here should be the hexadecimal representation of the interface version parameter. For example, the default (if not set) is "34" which stands for 0x34. For SMPP v3.3 set to "33". For SMPP v5.0 set to "50".

Variable	Value	Description
address-range (m)	string	According to the SMPP 3.4 spec this is supposed to affect which MS's can send messages to this account. Doesn't seem to work, though.
my-number	number	Optional smsc short number. Should be set if smsc sends a different one.
enquire-link-interval	number	Optional the time lapse allowed between operations after which an SMPP entity should interrogate whether it's peer still has an active session. The default is 30 seconds.
max-pending-submits	number	Optional the maximum number of outstanding (i.e. acknowledged) SMPP operations between an ESME and SMSC. This number is not specified explicitly in the SMPP Protocol Specification and will be governed by the SMPP implementation on the SMSC. As a guideline it is recommended that no more than 10 (default) SMPP messages are outstanding at any time.
reconnect-delay	number	Optional the time between attempts to connect an ESME to an SMSC having failed to connect initiating or during an SMPP session. The default is 10 seconds.
source-addr-ton	number	Optional, source address TON setting for the link. (Defaults to 2).
source-addr-npi	number	Optional, source address NPI setting for the link. (Defaults to 1).

Variable	Value	Description
source-addr-autodetect	boolean	Optional, if defined tries to scan the source address and set TON and NPI settings accordingly. If you don't want to autodetect the source address, turn this off, by setting it to no. (Defaults to yes).
dest-addr-ton	number	Optional, destination address TON setting for the link. (Defaults to 2).
dest-addr-npi	number	Optional, destination address NPI setting for the link. (Defaults to 1).
bind-addr-ton	number	Optional, bind address TON setting for the link. (Defaults to 0).
bind-addr-npi	number	Optional, bind address NPI setting for the link. (Defaults to 0).

Variable	Value	Description
msg-id-type	number	Optional, specifies which number base the SMSC is using for the message ID numbers in the corresponding `submit_sm_resp` and `deliver_sm` PDUs. This is required to make delivery reports (DLR) work on SMSC that behave differently. The number is a combined set of bit 1 and bit 2 that indicate as follows: bit 1: type for `submit_sm_resp`, bit 2: type for `deliver_sm`. If the bit is set then the value is in hex otherwise in decimal number base. Which means the following combinations are possible and valid: 0x00 `deliver_sm` decimal, `submit_sm_resp` decimal; 0x01 `deliver_sm` decimal, `submit_sm_resp` hex; 0x02 `deliver_sm` hex, `submit_sm_resp` decimal; 0x03 `deliver_sm` hex, `submit_sm_resp` hex. In accordance to the SMPP v3.4 specs the default will be a C string literal if no of the above values is explicitly indicated using the config directive.
alt-charset	string	Defines which character encoding is used for this specific smsc link. Uses `iconv()` routines to convert from and to that specific character set encoding. See your local iconv_open(3) manual page for the supported character encodings and the type strings that should be presented for this directive.

Variable	Value	Description
alt-addr-charset	string	Defines which character encoding is used for alphanumeric addresses. When set to GSM, addresses are converted into the GSM 03.38 charset (Since @ is translated into 0x00 which will break the SMPP PDU, @ replaced with ?). If set to another value, iconv() is used. (Defaults to windows-1252)
retry	boolean	Defines if SMSC should retry to connect even if SMSC sent error that username or password is incorrect. (Defaults to false)
connection-timeout	number (seconds)	This timer specifies the maximum time lapse allowed between transactions , after which period of inactivity, an SMPP driver may assume that the session is no longer active and does reconnect. Defaults to 300 seconds, to disable set it to 0.
wait-ack	number (seconds)	A message is resent if the acknowledge from SMSC takes more than this time. Defaults to 60 seconds.
wait-ack-expire	number	Defines what kind of action should be taken if the ack of a message expires. The options for this value are: 0x00 - disconnect/reconnect, (default) 0x01 - as is now, re-queue, but this could potentially result in the msg arriving twice 0x02 - just carry on waiting (given that the wait-ack should never expire this is the must accurate)
validityperiod	integer	How long the message will be valid, i.e., how long the SMSC will try try to send the message to the recipient. Defined in minutes.

Variable	Value	Description
esm-class	number	Change the "esm_class" parameter sent from Kannel. Accepted values are 0 (Default SMSC Mode) and 3 (Store and Forward). Defaults to 3.
log-format	number	Selects which SMPP PDU dump format to use: 1 - for multiline (default) 2 - for one line

Sema Group SMS2000 OIS 4.0, 5.0 and 5.8

The 4.0 implementation is over Radio PAD (X.28). Following configuration variables are needed, and if you find out the more exact meaning, please send a report.

The 5.0 implementation uses X.25 access gateway.

The 5.8 implementation uses direct TCP/IP access interface.

```
group = smsc
smsc = sema
device = /dev/tty0
smsc_nua = (X121 smsc address)
home_nua = (x121 radio pad address)
wait_report = 0/1 (0 means false, 1 means true)
```

Variable	Value	Description
device (m)	device	ex: /dev/tty0
smsc_nua (m)	X121 smsc address	The address of an SMSC for SEMA SMS2000 protocols using an X.28 connection.
home_nua (m)	X121 radio pad address	The address of a radio PAD implementing Sema SMS2000 using X.28 connection.
wait_report	0 (false)/1 (true)	Report indicator used by the Sema SMS2000 protocol. Optional.

```
group = smsc
smsc = ois
host = 103.102.101.100
port = 10000
receive-port = 10000
ois-debug-level = 0
```

Variable	Value	Description
host (m)	ip	SMSC Host name or IP

Variable	Value	Description
port (m)	port number	SMSC Port number
receive-port (m)	port number	The port in which the SMSC will contact
ois-debug-level	number 0 to 8	extra debug, optional, see smsc_ois.c

```
group = smsc
smsc = oisd
host = 103.102.101.100
port = 10000
keepalive = 25
my-number = 12345
validityperiod = 30
```

Variable	Value	Description
host (m)	ip	SMSC Host name or IP
port (m)	port number	SMSC Port number
keepalive	number	SMSC connection will not be left idle for longer than this many seconds. The right value to use depends on how eager the SMSC is to close idle connections. If you see many unexplained reconnects, try lowering this value. Set it to 0 to disable this feature.
my-number	string	Any valid SME number acceptable by SMSC. This number is used only in keepalive request.
validityperiod	integer	How long the message will be valid, i.e., how long the SMS center (the real one, not the phone acting as one for Kannel) will try to send the message to the recipient. Defined in minutes.

SM/ASI (for CriticalPath InVoke SMS Center 4.x)

This implements Short Message/Advanced Service Interface (SM/ASI) Protocol for the use of connecting to a CriticalPath InVoke SMS Center. Sample configuration:

```
group = smsc
smsc = smasi
```

```
host = 10.11.12.13
port = 23456
smsc-username = foo
smsc-password = foo
```

Variable	Value	Description
host (m)	hostname	Machine that runs SMSC. As IP (10.11.12.13) or hostname (host.foobar.com)
port (m)	port-number	The port number for the connection to the SMSC.
smsc-username (m)	string	The 'username' of the Messaging Entity connecting to the SMSC.
smsc-password (m)	string	The password matching the "smsc-username" your teleoperator provided you with.
reconnect-delay	number	Optional, the time between attempts to connect to an SMSC having failed to connect initiating or during an session. The default is 10 seconds.
source-addr-ton	number	Optional, source address TON setting for the link. (Defaults to 1).
source-addr-npi	number	Optional, source address NPI setting for the link. (Defaults to 1).
dest-addr-ton	number	Optional, destination address TON setting for the link. (Defaults to 1).
dest-addr-npi	number	Optional, destination address NPI setting for the link. (Defaults to 1).
priority	number	Optional, sets the default priority of messages transmitted over this smsc link. (Defaults to 0, which is the lowest priority).

GSM modem

Warning

See Appendix A for changes.

This driver allows a GSM Modem or Phone to be connected to Kannel and work as a virtual SMSC

```
group = smsc
smsc = at
modemtype = auto
device = /dev/ttyS0
speed = 9600
pin = 2345
```

Variable	Value	Description
`modemtype`	`string`	Modems from different manufacturers have slightly different behavior. We need to know what type of modem is used. Use "auto" or omit parameter to have Kannel detect the modem type automatically. (some types should not be auto-detected like the Nokia Premicell).
`device (m)`	`device-name`	The device the modem is connected to, like `/dev/ttyS0`. When the device name is set to `rawtcp` or `telnet` two other variables are required in the configuration: `host` and `port`. See the note below.
`speed`	`serial speed in bps`	The speed in bits per second. Default value 0 means to try to use speed from modem definition, or if it fails, try to autodetect.
`pin`	`string`	This is the PIN number of the SIM card in the GSM modem. You can specify this option if your SIM has never been used before and needs to have the PIN number entered. The PIN is usually a four digit number.

Variable	Value	Description
validityperiod	integer	How long the message will be valid, i.e., how long the SMS center (the real one, not the phone acting as one for Kannel) will try to send the message to the recipient. Encoded as per the GSM 03.40 standard, section 9.2.3.12. Default is 167, meaning 24 hours.
keepalive	seconds	Kannel would "ping" the modem for this many seconds. If the probe fails, try to reconnect to it.
my-number	number	Optional phone number.
sms-center	number	SMS Center to send messages.
sim-buffering	boolean	Whether to enable the so-called "SIM buffering behavior" of the GSM module. if assigned a true value, the module will query the message storage memory of the modem and will process and delete any messages found there. this does not alter normal behavior, but only add the capability of reading messages that were stored in the memory for some reason. The type of memory to use can be selected using the 'message-storage' parameter of the modem configuration. Polling the memory is done at the same interval as keepalive (if set) or 60 seconds (if not set). NOTE: This behavior is known to cause minor or major hiccups for a few buggy modems. Modems known to work with this setting are Wavecom WM02/M1200 and the Siemens M20.

Variable	Value	Description
max-error-count	integer	Maximal error count for opening modem device or initializing of the modem before reset-string will be executed. This is useful when modem crashed and needs hard reset. Default disabled.
host	IP address	Hostname or IP address to connect in rawtcp mode. Required if device is set to rawtcp.
port	integer	TCP port to connect to on rawtcp-host. Required if device is set to rawtcp.
smsc-username	string	Username to use on a Login: or User: prompt prior to be connected to the modem. Useful if modem is connected to a terminal server and rawtcp mode or telnet mode are used.
smsc-password	string	Username to use on a Password prompt prior to be connected to the modem. Useful if modem is connected to a terminal server and rawtcp mode or telnet mode are used. smsc-password can be used alone without smsc-username for devices only asking for a password.
login-prompt	string	An alternative string to be used instead of Login: or Username:
password-prompt	string	An alternative string to be used instead of Password:

Modem definitions are now multiple groups present in kannel.conf, either directly or, for example, by including the example modems.conf. (See *Inclusion of configuration files*)

Variable	Value	Description
group	modems	This is a mandatory variable
id	string	This is the the id that should be used in modemtype variable from AT2
name	string	The name of this modem configuration. Used in logs

Variable	Value	Description
detect-string	string	String to use when trying to detect the modem. See detect-string2
detect-string2	string	Second string to use to detect the modem. For example, if the modem replies with "SIEMENS MODEM M20", detect-string could be "SIEMENS" and detect-string2 "M20"
init-string	string	Optional initialization string. Defaults to "AT+CNMI=1,2,0,1,0"
speed	number	Serial port hint speed to use. Optional. Defaults to smsc group speed or autodetect
enable-hwhs	string	Optional AT command to enable hardware handshake. Defaults to "AT+IFC=2,2"
hardware-flow-control	boolean	Optionally disable hardware handshake on the computer side by setting it to false. Defaults to true.
need-sleep	boolean	Optional. Defaults to false. Some modems needs to sleep after opening the serial port and before first command
no-pin	boolean	Optional. Defaults to false. If the modem doesn't support the PIN command, enable this
no-smsc	boolean	Optional. Defaults to false. If the modem doesn't support setting the SMSC directly on the PDU, enable this. (Default is to include a "00" at the beginning of the PDU to say it's the default smsc, and remove the "00" when receiving)
sendline-sleep	number (milliseconds)	Optional, defaults to 100 milliseconds. The sleep time after sending a AT command.

Variable	Value	Description
keepalive-cmd	string	Optional, defaults to "AT". If keepalive is activated in AT2 group, this is the command to be sent. If your modem supports it, for example, use "AT+CBC;+CSQ", and see in logs the reply "+CBC: 0,64" (0=On battery, 64% full) and "+CSQ: 14,99" (0-31, 0-7: signal strength and channel bit error rate; 99 for unknown). See 3GPP 27007.
message-storage	string	Message storage memory type to enable for "SIM buffering". Possible values are: "SM" - SIM card memory or "ME" - Mobile equipment memory (may not be supported by your modem). check your modem's manual for more types. By default, if the option is not set, no message storage command will be sent to the modem and the modem's default message storage will be used (usually "SM").
message-start	string	Optional integer, defaults to 1. Specifies starting index in SIM buffer for new SMS. Most modems start numbering from 1, however a few like the UMG181 start numbering from 0. This parameter ensures that all SMS are fetched when using "SIM buffering".
enable-mms	boolean	Optional, defaults to false. If enabled, Kannel would send an AT+CMMS=2 if it have more than one message on queue and hopefully will be quicker sending the messages.
reset-string	string	Which reset string to execute when `max-error-count` reached. Example for Falcom: AT+CFUN=1

A note about delivery reports and GSM modems: while it is possible (and supported) to receive delivery

reports on GSM modems, it may not work for you. if you encounter problems, check that your modem's init string (if not the default) is set to correctly allow the modem to send delivery reports using unsolicited notification (check your modem's manual). If the init-string is not set as si, some modems will store delivery reports to SIM memory, to get at which you will need to enable sim-buffering. finally your GSM network provider may not support delivery reports to mobile units.

About `rawtcp` *mode:* This mode allows you to use a GSM modem connected to a remote terminal server, such as Perle IOLAN DS1 or a Cisco router with reverse telnet. The teminal server should support raw TCP mode. The driver is not tested in telnet mode. It is recommended to use `keepalive` variable, in order to automatically reconnect in case of network connectivity problems.

Modem initialization

This question has been asked frequently in the users mailing list. It usually comes in two forms:

1. Modem doesn't receive or transmit SMS.

   In its current implementation, the at driver will panic if it cannot find the modem device and bearerbox won't start. This can happen either because the wrong modem device is configured or the device has no write access for kannel's user.

   In the first case use system logs to discover where is your modem assigned to:

   grep tty /var/log/messages

   Then verify that this is indeed your GSM modem device by connecting to it with minicom or wvdial and issuing a few AT commands. Wvdial is preferrable, since it can also give you a working `init-string`

   In the second case you need to give write access to kannel user. This is no problem if you run kannel as root, but it is not recommended to do so. In Ubuntu linux you just need to assign kannel user to group "dial". In other systems you can either:

   ```
   chmod 666 /dev/modem
   ```

   or preferably:

   ```
   chmod 664 /dev/modem
   chgrp group /dev/modem
   usermod -G group kannel-user
   ```

2. Modem doesn't receive DLRs or SMS.

   By default the modem doesn't know how to treat incoming messages. To tell it to send it to kannel you have to configure `init-string` with the *+CNMI=x,x,x,x,x* command. X's are decimal numbers and CNMI stands for Command New Message Indication. There is no hard rule about its values. They are model and manufacturer specific. Best to consult modem manual or manufacturer's web site. Failing that, wvdialconf might give a working CNMI. Finally see modems.conf for examples and try out different values. Some Nokia phones do not support this command, but are ready to transmit incoming messages from the start.

   Since this setting is modem specific, kannel cannot understand what it means from its value, and if using unconventional values you should tell kannel through `sim-buffering` and

`message-storage`. In any case it is useful to add +CMEE = 1 or 2 in the `init-string` in order to make the modem more verbose in its responses.

```
group = modems
init-string = "AT+CNMI=2,3,0,1,0;+CMEE=1"
message-storage = ME or SM    Depending on your modem, you may need to adjust it
...

group = smsc
smsc = at
sim-buffering = true          This will tell kannel to search for messages in
...                           the modem (either memory or its SIM)
```

GSMA OneAPI v1.0 ParlayX SMS SOAP

This implements the GSMA OneAPI v1.0 ParlayX SMS SOAP/XML interface in two flavors, the Ericsson SDP (service delivery platform) ParlayX and the GSMA OneAPI v1.0 SOAP interfaces.

Example configuration for the Ericsson SDP variant:

```
group = smsc
smsc = parlayx
system-type = ericsson-sdp
port = 10000
connect-allow-ip = 10.11.12.13
send-url = http://host/location
dlr-url = http://our-host:10000/dlr-location
smsc-username = tester
smsc-password = foobar
```

Variable	Value	Description
`system-type (m)`	`string`	Type of ParlayX connection. Currently supported are: 'oneapi-v1', 'ericsson-sdp'. The variants have the same ParlayX SOAP XML PDUs, but differ in the authentication scheme they use, where 'ericsson-sdp' uses WS-Security via wsse and 'oneapi-v1' uses plain HTTP basic authentication.
`port (m)`	`port-number`	Port number in which Kannel listens to (MO and DLR) messages from other SOAP service side.

Variable	Value	Description
send-url (m)	url	Location of the SOAP service to send MT messages.
dlr-url (m)	url	Location of our side SOAP service for DLR notifications.
smsc-username (m)	string	Username associated for the SOAP service connection.
smsc-password (m)	string	Password associated for the SOAP service username.
use-ssl	boolean	Defines whether listen port should use SSL.
connect-allow-ip	IP-list	IPs allowed to use this SOAP service interface. If not set, "127.0.0.1" (localhost) is the only host allowed to connect. Typically you will allow the IP of the other side SOAP service here.
window	integer	Number of concurrent sending threads. If not set, one thread is used.

Fake SMSC

Fake SMSC is a simple protocol to test out Kannel. It is not a real SMS center, and cannot be used to send or receive SMS messages from real phones. So, it is ONLY used for testing purposes.

```
group = smsc
smsc = fake
port = 10000
connect-allow-ip = 127.0.0.1
```

Variable	Value	Description
host (m)	hostname	Machine that runs the SMSC. As IP (100.100.100.100) or hostname (their.machine.here)
port (m)	port-number	Port number in smsc host machine
connect-allow-ip	IP-list	If set, only connections from these IP addresses are accepted.

HTTP-based relay and content gateways

This special "SMSC" is used for HTTP based connections with other gateways and various other relay services, when direct SMSC is not available.

```
group = smsc
smsc = http
system-type = kannel
smsc-username = nork
smsc-password = z0rK
port = 13015
send-url = "http://localhost:20022"
```

Variable	Value	Description
system-type (m)	string	Type of HTTP connection. Currently supported are: 'kannel', 'brunet', 'xidris', 'clickatell' and 'generic'.
send-url (m)	url	Location to send MT messages. This URL is expanded by used system, if need to.
no-sender	boolean	Do not add variable sender to the send-url.
no-coding	boolean	Do not add variable coding to the send-url.
no-sep	boolean	Represent udh and text as a numeric string containing the hex-dump. For instance, text=%2b123 is represented as text=2b313233.
port (m)	port-number	Port number in which Kannel listens to (MO) messages from other gateway
use-ssl	boolean	Defines whether listen port should use SSL.
connect-allow-ip	IP-list	IPs allowed to use this interface. If not set, "127.0.0.1" (localhost) is the only host allowed to connect.
smsc-username	string	Username associated to connection, if needed. Kannel requires this, and it is the same as send-sms username at other end.
smsc-password	string	Password for username, if needed.

Variable	Value	Description
max-pending-submits	number	Optional the maximum number of outstanding (i.e. acknowledged) requests. As a guideline it is recommended that no more than 10 (default) requests are outstanding at any time.
system-id	string	Clickatell only: API id.
mobile-originated	integer	Clickatell only: 'mo' parameter used for a two-way service (0 or 1).
use-post	boolean	Generic only: Defines if a HTTP POST method with a content-type application/x-www-form-urlencoded should be used rather then the HTTP GET method. If set, both directions will use the HTTP POST method. (Default: no)

The "generic" system-type

For a generic HTTP-based relay, the system-type 'generic' can be used. It can use the escape codes known from sms-service's get-url config directives in the send-url string to indicate the HTTP API calling style of the remote HTTP server, and the 3 HTTP body response parsing regular expressions 'status-success-regex' for successfull acknowledge, 'status-permfail-regex' for permanent failure and 'status-tempfail-regex' for temporary failure. It can also accomodate the incoming parameters and response codes and text. This system-type has a lot of extra parameters that gives it a lot of flexibility:

Variable	Value	Description
status-success-regex (o)	POSIX regular expression	Regular expression to match against HTTP response body content, indicating a successful submission.
status-permfail-regex (o)	POSIX regular expression	Regular expression to match against HTTP response body content, indicating a permanent failure.
status-tempfail-regex (o)	POSIX regular expression	Regular expression to match against HTTP response body content, indicating a temporary failure.

Variable	Value	Description
generic-foreign-id-regex (o)	POSIX regular expression	Regular expression to match against HTTP response body content to get the foreign message id in case of successful submission.
generic-param-username (o)	string	Overrides the default parameter for the 'username' field used on incoming requests.
generic-param-password (o)	string	Overrides the default parameter for the 'password' field used on incoming requests.
generic-param-from (o)	string	Overrides the default parameter for the 'from' field used on incoming requests.
generic-param-to (o)	string	Overrides the default parameter for the 'to' field used on incoming requests.
generic-param-text (o)	string	Overrides the default parameter for the 'text' field used on incoming requests.
generic-param-udh (o)	string	Overrides the default parameter for the 'udh' field used on incoming requests.
generic-param-service (o)	string	Overrides the default parameter for the 'service' field used on incoming requests.
generic-param-account (o)	string	Overrides the default parameter for the 'account' field used on incoming requests.
generic-param-binfo (o)	string	Overrides the default parameter for the 'binfo' field used on incoming requests.
generic-param-dlr-mask (o)	string	Overrides the default parameter for the 'dlr-mask' field used on incoming requests.
generic-param-dlr-err (o)	string	Overrides the default parameter for the 'dlr-err' field used on incoming requests.

Variable	Value	Description
generic-param-dlr-url (o)	string	Overrides the default parameter for the 'dlr-url' field used on incoming requests.
generic-param-dlr-mid (o)	string	Overrides the default parameter for the 'dlr-mid' field used on incoming requests.
generic-param-flash (o)	string	Overrides the default parameter for the 'flash' field used on incoming requests.
generic-param-mclass (o)	string	Overrides the default parameter for the 'mclass' field used on incoming requests.
generic-param-mwi (o)	string	Overrides the default parameter for the 'mwi' field used on incoming requests.
generic-param-coding (o)	string	Overrides the default parameter for the 'coding' field used on incoming requests.
generic-param-validity (o)	string	Overrides the default parameter for the 'validity' field used on incoming requests.
generic-param-deferred (o)	string	Overrides the default parameter for the 'deferred' field used on incoming requests.
generic-param-foreign-id (o)	string	Overrides the default parameter for the 'foreign-id' field used on incoming requests.
generic-param-meta-data (o)	string	Overrides the default parameter for the 'meta-data' field used on incoming requests.
generic-message-sent (o)	string	It allows you to set the text returned when a succesful request is made. If not set, defaults to 'Sent.'. Note that you can use all sms service escapes here, see *Parameters (Escape Codes)* for details.

Variable	Value	Description
generic-status-sent (o)	string	Overrides the HTTP status code returned when a successful request is made. If not set, defaults to 202 (HTTP_ACCEPTED).
generic-status-error (o)	string	Overrides the HTTP status code returned when a request is rejected for any reason. If not set, defaults to 202 (HTTP_ACCEPTED).

An example for a 'generic' group looks like this:

```
group = smsc
smsc = http
system-type = generic
port = 13015
send-url = "http://www.foobar.com/mt.php?from=%P&to=%p&text=%b"
status-success-regex = "ok"
status-permfail-regex = "failure"
status-tempfail-regex = "retry later"
generic-foreign-id-regex = "<id>(.+)</id>"
generic-param-from = "phoneNumber"
generic-param-to = "shortCode"
generic-message-sent = "Message sent with ID: %I"
generic-status-sent = 200
generic-status-error = 404
```

Loopback SMSC (MT to MO direction switching)

This special "SMSC" type can be used to route MT messages in bearerbox internally as MO input messages again. Therefore this tyoe is the MT wise counterpart of the 'reroute' functionality of the smsc group when MOs are re-routed as MTs.

```
group = smsc
smsc = loopback
smsc-id = loop1
reroute-smsc-id = my_id
```

Variable	Value	Description
reroute-smsc-id (o)	string	Tag the rerouted MO with the given value for smsc-id instead of the canonical smsc-id value of the smsc group itself.

Using multiple SMS centers

If you have several SMS center connections (multiple operators or a number of GSM modems) you need to configure one smsc group per SMS center (or GSM modem). When doing this, you might want to use routing systems to rout messages to specific centers - for example, you have 2 operator SMS centers, and the other is much faster and cheaper to use.

To set up routing systems, first give an unique ID for each SMS center - or if you want to treat multiple ones completely identical, give them identical ID. Then use `preferred-smsc-id` and `denied-smsc-id` to set up the routing to your taste. See also SMS PUSH settings ('sendsms-user' groups), below.

Feature checklist

Not all of Kannel's SMSC drivers support the same set of features. This is because they were written at different times, and new features are often only added to drivers that the feature author can test.

The table in this section is an attempt to show exactly what features to expect from a driver, and to help identify areas where drivers need to be updated. Currently most of the entries are marked as "not tested" because the table is still new.

Table 7-2. SMSC driver features

Feature	cimd	cimd2	emi	emi_x25	smpp	sema	ois	oisd	at	http	fake	smasi
Can use DLR												
	n	y?	y	n	y?	n	n	y	n	n	y	y
Can set DCS[a]												
	?	?	y	?	?	?	?	y	y	?	?	?
Can set Alt-DCS												
	n	n	y	n	n	n	n	n	y	n	n	?
Can set Validity												
	?	?	y	?	?	?	?	y	y	?	?	?
Can set Deferred												
	?	?	y	?	?	?	?	n	n	?	?	?
Can set PID												
	n	y	y	n	y	n	n	n	y	n	n	?
Can set RPI												
	n	y	y	n	y	n	n	n	n	n	n	?
Can send Unicode												
	?	?	y	?	?	?	?	y	y	?	?	?
Can send 8 bits												
	?	?	y	?	?	?	?	y	y	?	?	y

Feature	cimd	cimd2	emi	emi_x25	smpp	sema	ois	oisd	at	http	fake	smasi
Correctly send GSM alphabet												
	?	?	y	?	?	?	?	y	?	?	?	?
Can set binfo / tariff class												
	?	y	?	?	?	?	?	n	?	?	?	?
Can use throttling												
	n	n	y	n	y	n	n	n	n	y	y	y
Notes: a. To use `mclass`, `mwi`, `coding` and `compress` fields.												

Table 7-3. SMSC driver internal features

Feature	cimd	cimd2	emi2	emi_x25	smpp	sema	ois	oisd	at	http	fake	smasi
Can keep idle connections alive												
	n	y?	y	n	y?	?	?	y	y	?	?	y
Can send octet data without UDH												
	n	y?	y	y?	n	n	y?	y	y	n	y?[a]	?
Can send octet data with UDH												
	N	y?	y	y?	y?	n	?	y	y	y?	y?[a]	?
Can send text messages with UDH												
	n	y?	y	y?	n	n	?	y	y	n	y?	?
Can receive octet data without UDH												
	n	y?	y	n	n	y?[b]	y?	y	y	n	n	?
Can receive Unicode messages												
	n	n	y	n	n	n	n	y	y	n	n	?
Can receive octet data with UDH												
	n	y?	y	n	n	n	N	y	y	y?	y?	?
Can receive text messages with UDH												
	n	y?	y	n	n	n	N	y	y	n	n	?
Correctly encodes `@` when sending												
	y?	y?	y	?	y?	?	y?	y	y	y?	y?	?
Correctly encodes `ä` when sending												
	y?	y?	y	?	y?	?	y?	y	y	y?	y?	?
Correctly encodes `{` when sending												
	n	y?	y	?	y?	?	n	y	N[c]	y?	y?	?
Can receive `@` in text messages												
	y?	y?	y	?	y?	?	y?	y	y	y?	y?	?

Feature	cimd	cimd2	emi2	emi_x25	smpp	sema	ois	oisd	at	http	fake	smasi
Can receive ä in text messages												
	y?	y?	y	?	y?	?	y?	y	y?	y?	y?	?
Can receive { in text messages												
	n	y?	y	?	y?	?	n	y	y?	y?	y?	?
Can shut down idle connections												
	n	n	y	n	n	?	?	n	?	?	?	?

Notes:
a. Does not mark it as octet data
b. However, it looks like the `sema` driver can't receive *text* data.
c. Miscalculates message length

Symbol	Meaning
?	not yet investigated
y	driver has this feature, and it has been tested
y?	driver probably has this feature, has not been tested
n	driver does not have this feature
N	driver claims to have this feature but it doesn't work
-	feature is not applicable for this driver

External delivery report (DLR) storage

Delivery reports are supported by default internally, which means all DLRs are stored in the memory of the bearerbox process. This is problematic if bearerbox crashes or you take the process down in a controlled way, but there are still DLRs open. Therefore you may use external DLR storage places, i.e. a MySQL database.

Following are the supported DLR storage types and how to use them:

Internal DLR storage

This is the default way in handling DLRs and does not require any special configuration. In order to configure bearerbox to use internal DLR storage use `dlr-storage = internal` in the `core` group.

MySQL DLR storage

To store DLR information into a MySQL database you may use the `dlr-storage = mysql`

configuration directive in the `core` group.

In addition to that you must have a `dlr-db` group defined that specifies the table field names that are used to the DLR attributes and a `mysql-connection` group that defines the connection to the MySQL server itself.

Here is the example configuration from `doc/examples/dlr-mysql.conf`:

```
group = mysql-connection
id = mydlr
host = localhost
username = foo
password = bar
database = dlr
max-connections = 1

group = dlr-db
id = mydlr
table = dlr
field-smsc = smsc
field-timestamp = ts
field-destination = destination
field-source = source
field-service = service
field-url = url
field-mask = mask
field-status = status
field-boxc-id = boxc
```

MySQL connection configuration

For several reasons external storage may be required to handle dynamical issues, i.e. DLRs, sms-service, sendsms-user, ota-setting, ota-bookmark definitions and so on. To define a MySQL database connection you simple need to specify a `mysql-connection` group as follows:

Table 7-4. MySQL Connection Group Variables

Variable	Value	Description
`group`	`mysql-connection`	This is a mandatory variable
`id (m)`	`string`	An optional name or id to identify this MySQL connection for internal reference with other MySQL related configuration groups. Any string is acceptable, but semicolon ';' may cause problems, so avoid it and any other special non-alphabet characters.

Variable	Value	Description
host (m)	hostname or IP	Hostname or IP of a server running a MySQL database to connect to.
username (m)	username	User name for connecting to MySQL database.
password (m)	password	Password for connecting to MySQL database.
database (m)	string	Name of database in MySQL database server to connect to.
max-connections	integer	How many connections should be opened to the given database. This is used for database pool.

A sample 'mysql-connection' group:

```
group = mysql-connection
id = dlr-db
host = localhost
username = foo
password = bar
database = dlr
max-connections = 1
```

In case you use different MySQL connections for several storage issues, i.e. one for DLR and another different one for sms-service you may use the `include` configuration statement to extract the MySQL related configuration groups to a separate `mysql.conf` file.

LibSDB DLR storage

To store DLR information into a LibSDB resource (which is an abstraction of a real database) you may use the `dlr-storage = sdb` configuration directive in the `core` group.

In addition to that you must have a `dlr-db` group defined that specifies the table field names that are used to the DLR attributes and a `sdb-connection` group that defines the LibSDB resource itself.

Here is the example configuration from `doc/examples/dlr-sdb.conf` using a PostgreSQL resource:

```
group = sdb-connection
id = pgdlr
url = "postgres:host=localhost:db=myapp:port=1234"
max-connections = 1

group = dlr-db
id = pgdlr
table = dlr
field-smsc = smsc
```

```
field-timestamp = ts
field-destination = destination
field-source = source
field-service = service
field-url = url
field-mask = mask
field-status = status
field-boxc-id = boxc
```

Beware that you have the DB support build in your LibSDB installation when trying to use a specific DB type within the URL.

Oracle 8i/9i DLR storage

To store DLR information into a Oracle database you may use the `dlr-storage = oracle` configuration directive in the `core` group.

In addition to that you must have a `dlr-db` group defined that specifies the table field names that are used to the DLR attributes and a `oracle-connection` group that defines the connection to the Oracle server itself.

Here is the example configuration from `doc/examples/dlr-oracle.conf`:

```
group = oracle-connection
id = mydlr
username = foo
password = bar
tnsname = dlr
max-connections = 1

group = dlr-db
id = mydlr
table = dlr
field-smsc = smsc
field-timestamp = ts
field-destination = destination
field-source = source
field-service = service
field-url = url
field-mask = mask
field-status = status
field-boxc-id = boxc
```

PostgreSQL DLR storage

To store DLR information into a PostgreSQL database you may use the `dlr-storage = pgsql` configuration directive in the `core` group.

In addition to that you must have a `dlr-db` group defined that specifies the table field names that are used to the DLR attributes and a `pgsql-connection` group that defines the connection to the PostgreSQL server itself.

Here is the example configuration:

```
group = pgsql-connection
id = mydlr
host = myhost.com
username = foo
password = bar
database = dlr
max-connections = 1

group = dlr-db
id = mydlr
table = dlr
field-smsc = smsc
field-timestamp = ts
field-destination = destination
field-source = source
field-service = service
field-url = url
field-mask = mask
field-status = status
field-boxc-id = boxc
```

MS-SQL/Sybase DLR storage

To store DLR information into a MS-SQL or Sybase (via FreeTDS) database you may use the `dlr-storage = mssql` configuration directive in the `core` group.

In addition to that you must have a `dlr-db` group defined that specifies the table field names that are used to the DLR attributes and a `mssql-connection` group that defines the connection to the MS-SQL or Sybase server itself.

Please note that the `server` configuration directive must match the corresponding `[section]` on `freetds.conf`

Here is the example configuration:

```
group = mssql-connection
id = msdlr
server = myservername
username = foo
password = bar
database = dlr
```

```
max-connections = 1

group = dlr-db
id = msdlr
table = dlr
field-smsc = smsc
field-timestamp = ts
field-destination = destination
field-source = source
field-service = service
field-url = url
field-mask = mask
field-status = status
field-boxc-id = boxc
```

SQLite3 DLR storage

To store DLR information into a SQLite3 database you may use the `dlr-storage = sqlite3` configuration directive in the `core` group.

In addition to that you must have a `dlr-db` group defined that specifies the table field names that are used to the DLR attributes and a `sqlite3-connection` group that defines the connection to the SQLite3 database itself.

Here is the example configuration:

```
group = sqlite3-connection
id = mydlr
database = /path/to/file
max-connections = 1

group = dlr-db
id = mydlr
table = dlr
field-smsc = smsc
field-timestamp = ts
field-destination = destination
field-source = source
field-service = service
field-url = url
field-mask = mask
field-status = status
field-boxc-id = boxc
```

Redis DLR storage

To store DLR information using the Redis (http://www.redis.io) keystore, you should set the `dlr-storage = redis` configuration directive in the `core` group.

In addition, you must have a `dlr-db` group defined that specifies the key and hash field names that are used to store the DLR attributes and a `redis-connection` group that defines the connection to the Redis server.

Since Redis is a keystore and not a SQL database, keys and not tables are used to store DLR attributes. For ease of configuration, the same configuration parameters in the `dlr-db` group that are used to define SQL tables in other DLR storage engines are used to configure key names in Redis. DLRs are stored as Redis hash keys with a separate key for each DLR. Key names are in the format `<table>:<smsc>:<timestamp>`. Some SMSCs also append the destination to the DLR keyname resulting DLR keynames in the format `<table>:<smsc>:<timestamp>:<destination>`. You can specify the `<table>` portion of the keyname by specifying the `table` value in the `dlr-db` group.

In addition to one key per DLR, an additional key with a name in the format `<table>:Count` is created to maintain a count of the total number of outstanding DLRs.

Here is the example configuration from `doc/examples/dlr-redis.conf`:

```
group = redis-connection
id = mydlr
host = localhost
port = 6379
#password = foo
database = 2
idle-timeout = 3600
max-connections = 1

group = dlr-db
id = mydlr
ttl = 604800
table = dlr
field-smsc = smsc
field-timestamp = ts
field-destination = destination
field-source = source
field-service = service
field-url = url
field-mask = mask
field-status = status
field-boxc-id = boxc
```

Redis connection configuration

To define a Redis connection you need to specify a `redis-connection` group as follows:

Table 7-5. Redis Connection Group Variables

Variable	Value	Description
`group`	`redis-connection`	This is a mandatory variable
`id (m)`	`string`	An optional name or id to identify this Redis connection for internal reference with other Redis related configuration groups. Any string is acceptable, but semicolon ';' may cause problems, so avoid it and any other special non-alphabet characters.
`host (m)`	hostname or IP	Hostname or IP of a server running Redis to connect to.
`password`	password	If the Redis server is secured with a password in redis.conf, it must be specified here. By default, Redis servers are unsecured and in that case, this parameter may be ommitted.
`database`	integer	The numeric ID (0-15) of the Redis database to use. Ideally this database should be dedicated to storing Redis DLRs since a separate key wil be created for each DLR. If this variable is ommitted, the default database (0) will be used.
`idle-timeout`	integer	The number of seconds that Redis should hold the connection open without any traffic. Some Redis servers are configured to disconnect idle clients very quickly, and if this is the case you should set this value to something more reasonable. If ommitted, the Redis server-wide timeout from redis.conf will apply.
`max-connections`	integer	How many connections should be opened to the Redis server.

Cassandra DLR storage

To store DLR information using Cassandra (http://cassandra.apache.org/) database, you should set the `dlr-storage = cassandra` configuration directive in the `core` group.

In addition, you must have a `dlr-db` group defined that specifies the field names that are used to store the DLR attributes and a `cassandra-connection` group that defines the connection to Cassandra.

Here is the example configuration from `doc/examples/dlr-cassandra.conf`:

```
group = cassandra-connection
id = mydlr
host = localhost
#username = foo
#password = bar
#database = dlr
max-connections = 1

group = dlr-db
id = mydlr
ttl = 604800
table = dlr
field-smsc = smsc
field-timestamp = ts
field-destination = destination
field-source = source
field-service = service
field-url = url
field-mask = mask
field-status = status
field-boxc-id = boxc
```

DLR database field configuration

For external database storage of DLR information in relational database management systems (RDBMS) you will have to specify which table field are used to represent the stored data. This is done via the `dlr-db` group as follows:

Table 7-6. DLR Database Field Configuration Group Variables

Variable	Value	Description
`group`	`dlr-db`	This is a mandatory variable

Variable	Value	Description
id (m)	string	An id to identify which external connection should be used for DLR storage. Any string is acceptable, but semicolon ';' may cause problems, so avoid it and any other special non-alphabet characters.
table (m)	string	The name of the table that is used to store the DLR information.
field-smsc (m)	string	The table field that is used for the smsc data.
field-timestamp (m)	string	The table field that is used for the timestamp data.
field-destination (m)	string	The table field that is used for the destination number data.
field-source (m)	string	The table field that is used for the source number data.
field-service (m)	string	The table field that is used for the service username data.
field-url (m)	string	The table field that is used for the DLR URL which is triggered when the DLR for this message arrives from the SMSC.
field-mask (m)	string	The table field that is used for the DLR mask that has been set for a message.
field-status (m)	string	The table field that is used to reflect the status of the DLR for a specific message.
field-boxc-id (m)	string	The table field that is used to store the smsbox connection id that has passed the message for delivery. This is required in cases you want to guarantee that DLR messages are routed back to the same smsbox conn instance. This is done via the smsbox routing. If you don't use smsbox routing simply add this field to your database table and keep it empty.

A sample 'dlr-db' group:

```
group = dlr-db
id = dlr-db
table = dlr
field-smsc = smsc
field-timestamp = ts
field-source = source
field-destination = destination
field-service = service
field-url = url
field-mask = mask
field-status = status
field-boxc-id = boxc
```

Beware that all variables in this group are mandatory, so you have to specify all fields to enable bearerbox to know how to store and retrieve the DLR information from the external storage spaces.

SMSBox configuration

Smsbox group

You must define an 'smsbox' group into the configuration file to be able to use SMS Kannel. The simplest working 'smsbox' group looks like this:

```
group = smsbox
bearerbox-host = localhost
```

...but you would most probably want to define 'sendsms-port' to be able to use SMS push.

SMSBox inherits from core the following fields:

```
smsbox-port
http-proxy-port
http-proxy-host
http-proxy-username
http-proxy-password
http-proxy-exceptions
http-proxy-exceptions-regex
ssl-certkey-file
```

Table 7-7. Smsbox Group Variables

Variable	Value	Description
`group (m)`	`smsbox`	This is a mandatory variable
`bearerbox-host (m)`	`hostname`	The machine in which the bearerbox is.

Variable	Value	Description
bearerbox-port (o)	port-number	This is the port number to which smsbox will connect bearerbox. If not given smsbox-port from core group used.
bearerbox-port-ssl (o)	bool	If set to true, the smsbox connection will be SSL-enabled. Your smsbox will connect using SSL to the bearerbox then. This is used to secure communication between bearerbox and smsboxes in case they are in separate networks operated and the TCP communication is not secured on a lower network layer. If not given smsbox-port-ssl from core group used.
smsbox-id (o)	string	Optional smsbox instance identifier. This is used to identify an smsbox connected to an bearerbox for the purpose of having smsbox specific routing inside bearerbox. So if you you own boxes that do pass messages into bearerbox for delivery you may want that answers to those are routed back to your specific smsbox instance, i.e. SMPP or EMI proxying boxes.
sendsms-interface (o)	string	If this is set, the sendsms HTTP interface will only bind to a specified address. For example: "127.0.0.1".
sendsms-port (c)	port-number	The port in which any sendsms HTTP requests are done. As with other ports in Kannel, can be set as anything desired.
sendsms-port-ssl (o)	bool	If set to true, the sendsms HTTP interface will use a SSL-enabled HTTP server with the specified ssl-server-cert-file and ssl-server-key-file from the core group. Defaults to "no".
sendsms-url (o)	url	URL locating the sendsms service. Defaults to /cgi-bin/sendsms.

Variable	Value	Description
sendota-url (o)	url	URL locating the sendota service. Defaults to /cgi-bin/sendota.
immediate-sendsms-reply (o)	boolean	This is a backward compatibility flag: when set, Kannel will immediately answer to any sendsms requests, without knowing if the bearerbox will ever accept the message. If set to false (default), smsbox will not reply to HTTP request until the bearerbox has received the message.
sendsms-chars	string	Only these characters are allowed in 'to' field when send-SMS service is requested via HTTP. Naturally, you should allow at least 0123456789. The *space* character (' ') has special meaning: it is used to separate multiple phone numbers from each other in multi-send. To disable this feature, do not have it as an accepted character. If this variable is not set, the default set "0123456789 +-" is used.
global-sender	phone-number	If set, all sendsms originators are set as these before proceeding. Note that in a case of most SMS centers you cannot set the sender number, but it is automatically set as the number of SMSC
log-file	filename	As with the bearerbox 'core' group. Access-log is used to store information about MO and send-sms requests. Can be named same as the 'main' access-log (in 'core' group).
log-level	number 0..5	
access-log	filename	

Variable	Value	Description
		Messages of this log level or higher will also be sent to syslog, the UNIX system log daemon. If not explicitly set with `syslog-facility`, it logs under the 'daemon' category. The default is not to use syslog, and you can set that explicitly by setting `syslog-level` to 'none'.
`syslog-level`	number	
`syslog-facility`	string	The syslog facility to send the syslog entries to. The default is 'daemon'.
`white-list`	URL	Load a list of accepted destinations of SMS messages. If a destination of an SMS message is not in this list, any message received from the HTTP interface is rejected. See notes of phone number format from numhash.h header file.
`black-list`	URL	As white-list, but SMS messages to these numbers are automatically discarded
`reply-couldnotfetch`	string	If set, replaces the SMS message sent back to user when Kannel could not fetch content. Defaults to `Could not fetch content, sorry.`.
`reply-couldnotrepresent`	string	If set, replaces the SMS message sent back when Kannel could not represent the result as a SMS message. Defaults to `Result could not be represented as an SMS message.`.
`reply-requestfailed`	string	If set, replaces the SMS message sent back when Kannel could not contact http service. Defaults to `Request Failed`.
`reply-emptymessage`	string	If set, replaces the SMS message sent back when message is empty. Set to "" to enable empty messages. Defaults to `<Empty reply from service provider>`.

Variable	Value	Description
mo-recode	boolean	If enabled, Kannel will try to convert received messages with UCS-2 charset to WINDOWS-1252 or to UTF-8, simplifying external servers jobs. If Kannel is able to recode message, it will also change coding to 7 bits and charset to windows-1252 or to utf-8.
http-request-retry	integer	If set, specifies how many retries should be performed for failing HTTP requests of sms-services. Defaults to 0, which means no retries should be performed and hence no HTTP request queuing is done.
http-queue-delay	integer	If set, specifies how many seconds should pass within the HTTP queuing thread for retrying a failed HTTP request. Defaults to 10 sec. and is only obeyed if http-request-retry is set to a non-zero value.
white-list-regex	POSIX regular expression	Defines the set of accepted destinations of SMS messages. If a destination of an SMS message is not in this set, any message received from the HTTP interface is rejected. See section on Regular Expressions for details.
black-list-regex	POSIX regular expression	As white-list-regex, but SMS messages to numbers within in this set are automatically discarded. See section on Regular Expressions for details.
max-pending-requests	number of messages	Maximum number of pending MO or DLR messages that are handled in parallel. (Default: 512)
http-timeout	seconds	Sets socket timeout in seconds for outgoing client http connections. Optional. Defaults to 240 seconds.

Variable	Value	Description
sms-length	number	Maximum allowed number of characters for a single SMS in smsbox. If this maximum exceeds Kannel will split SMS into multiparts. Default: 140

A typical 'smsbox' group could be something like this:

```
group = smsbox
bearerbox-host = localhost
sendsms-port = 13131
sendsms-chars = "0123456789 "
global-sender = 123456
access-log = "kannel.access"
log-file = "smsbox.log"
log-level = 0
```

Smsbox routing inside bearerbox

The communication link between bearerbox and smsbox has been designed for the purpose of load-balancing via random assignment. Which means, bearerbox holds all smsc connections and passes inbound message to one of the connected smsboxes. So you have a determined route for outbound messages, but no determined route for inbound messages.

The smsbox routing solves this for the inbound direction. In certain scenarios you want that bearerbox to know to which smsbox instance it should pass messages. I.e. if you implement own boxes that pass messages to bearerbox and expect to receive messages defined on certain rules, like receiver number or smsc-id. This is the case for EMI/UCP and SMPP proxies that can be written easily using smsbox routing facility.

If you box handles the SMPP server specific communication to your EMSEs, and if an client send a submit_sm PDU, the box would transform the message into Kannel message representation and inject the message to bearerbox as if it would be an smsbox. As you want to assign your clients shortcodes for certain networks or route any inbound traffic from a certain smsc link connected to bearerbox, you need to separate in the scope of bearerbox where the inbound message will be going to. An example may look like this:

```
group = smsbox
...
smsbox-id = mysmsc
...

group = smsbox-route
smsbox-id = mysmsc
shortcode = "1111;2222;3333"
```

Which means any inbound message with receiver number 1111, 2222 or 3333 will be delivered to the smsbox instance that has identified itself via the id "mysmsc" to bearerbox. Using this routing the smsbox instance (which may be an EMI/UCP or SMPP proxy) is able to send a deliver_sm PDU. Another example showing the combination use of criteria smsc-id and shortcode looks like this:

```
group = smsbox
...
smsbox-id = mysmsc
...

group = smsbox-route
smsbox-id = mysmsc
smsc-id = "A;B;C"
shortcode = "1111;2222"
```

Which means any inbound message with receiver number 1111 or 2222 that have been originating from the smsc-id connections A, B or C will be delivered to the via the id "mysmsc" to bearerbox.

To define a default smsbox-route when no other conditions match, you may defined a group without smsc-id and shortcode vaues, like this:

```
group = smsbox
...
smsbox-id = mysmsc
...

group = smsbox-route
smsbox-id = mysmsc
```

If no such default smsbox-route is defined, then bearerbox will deliver to any un-identified smsbox connection.

smsbox-route inherits from core the following fields:

Table 7-8. Smsbox-route Group Variables

Variable	Value	Description
`group (m)`	`smsbox-route`	This is a mandatory variable
`smsbox-id (m)`	`string`	Defines for which smsbox instance the routing rules do apply.

Variable	Value	Description
smsc-id	word-list	If set, specifies from which smsc-ids all inbound messages should be routed to this smsbox instance. List contains smsc-ids separated by semicolon (";"). This rule may be used to pull any smsc specific message stream to an smsbox instance. If used in combination with config directive shortcode, then this is another matching criteria for the routing decission.
shortcode	number-list	If set, specifies which receiver numbers for inbound messages should be routed to this smsbox instance. List contains numbers separated by semicolon (";"). This rule may be used to pull receiver number specific message streams to an smsbox instance. If used in combination with config directive smsc-id, then only messages originating from the connections in the smsc-id are matched against the shortcode list.

SMS-service configurations

Now that you have an SMS center connection to send and receive SMS messages you need to define services for incoming messages. This is done via 'sms-service' configuration groups.

These groups define SMS services in the smsbox, so they are only used by the smsbox. Each service is recognized from the first word in an SMS message and by the number of arguments accepted by the service configuration (unless `catch-all` configuration variable is used). By adding a username and password in the URL in the following manner "http://luser:password@host.domain:port/path?query" we can perform HTTP Basic authentication.

The simplest service group looks like this:

```
group = sms-service
keyword = www
get-url = "http://%S"
```

This service grabs any SMS with two words and 'www' as the first word, and then does an HTTP request to an URL which is taken from the rest of the message. Any result is sent back to the phone (or requester), but is truncated to the 160 characters that will fit into an SMS message, naturally.

Service group `default` has a special meaning: if the incoming message is not routed to any other service, `default` 'sms-service' group is used. You should always define `default` service.

Service group `black-list` has a special meaning: if the incoming message is in service's black-list, this service is used to reply to user. If unset, message will be discarded.

Table 7-9. SMS-Service Group Variables

Variable	Value	Description
`group (m)`	`sms-service`	This is a mandatory variable
`keyword (m)`	word	Services are identified by the first word in the SMS. Each '%s' in the URL corresponds to one word in the SMS message. Words are separated with spaces. A keyword is matched only if the number of words in the SMS message is the same as the number of '%s' fields in the URL. This allows you to configure the gateway to use different URLs for the same keyword depending on the number of words the SMS message contains. The keyword is case insensitive, which means you don't have to use aliases to handle different cased versions of your keyword.
`keyword-regex`	POSIX regular expression	This field may be used to enable service-selection based on a regular expression. You can define either keyword or keyword-regex in configuration, but not both in the same sms-service. keyword-regex is case sensitive. See section on Regular Expressions for details.
`aliases`	word-list	If the service has aliases, they are listed as a list with each entry separated with a semicolon (';'). Aliases are case insensitive just like keyword.
`name`	string	Optional name to identify the service in logs. If unset, `keyword` is used.

Variable	Value	Description
`get-url (c)`	URL	Requested URL. The url can include a list of parameters, which are parsed before the url is fetched. See below for these parameters. Also works with plain 'url'
`post-url (c)`	URL	Requested URL. As above, but request is done as POST, not GET. Always matches the keyword, regardless of pattern matching. See notes on POST other where.
`post-xml (c)`	URL	Requested URL. As above, but request is done as XML POST. Always matches the keyword, regardless of pattern matching. See notes on POST other where and *XML Post*
`file (c)`	filename	File read from a local disc. Use this variable only if no `url` is set. All escape codes (parameters) in `url` are supported in filename. The last character of the file (usually linefeed) is removed.
`text (c)`	string	Predefined text answer. Only if there is neither `url` nor `file` set. Escape codes (parameters) are usable here, too.
`exec (c)`	string	Executes the given shell command as the current UID of the running smsbox user and returns the output to `stdout` as reply. Escape codes (parameters) are usable here, too. BEWARE: You may harm your system if you use this sms-service type without serious caution! Make sure anyone who is allowed to use these kind of services is checked using white/black-list mechanisms for security reasons.

Variable	Value	Description
accepted-smsc	id-list	Accept ONLY SMS messages arriving from SMSC with matching ID. [a] Separate multiple entries with ';'. For example, if accepted-smsc is "RL;SON", accept messages which originate from SMSC with ID set as 'RL' or 'SON'
accepted-account	id-list	Accept ONLY SMS messages arriving with a matching ACCOUNT field. [b] Separate multiple entries with ';'. For example, if accepted-account is "FOO;BAR", accept messages which originate with the ACCOUNT field set as 'FOO' or 'BAR'
allowed-prefix	prefix-list	A list of phone number prefixes of the sender number which are accepted to be received by this service. [c] Multiple entries are separated with semicolon (';'). For example, "91;93" selects this service for these prefixes. If denied-prefix is unset, only this numbers are allowed. If denied is set, number are allowed if present in allowed or not in denied list.
denied-prefix	prefix-list	A list of phone number prefixes of the sender number which are NOT accepted to be sent through this SMSC.
allowed-receiver-prefix	prefix-list	A list of phone number prefixes of the receiver number which are accepted to be received by this service. This may be used to allow only inbound SMS to certain shortcut numbers to be allowed to this service.
denied-receiver-prefix	prefix-list	A list of phone number prefixes of the receiver number which are NOT accepted to be sent through this SMSC.

Variable	Value	Description
`catch-all`	bool	Catch keyword regardless of '%s' parameters in pattern.
`send-sender`	bool	Used only with POST. If set to true, number of the handset is set, otherwise not.
`strip-keyword`	bool	Used only with POST. Remove matched keyword from message text before sending it onward.
`faked-sender`	phone-number	This number is set as sender. Most SMS centers ignore this, and use their fixed number instead. This option overrides all other sender setting methods.
`max-messages`	number	If the message to be sent is longer than maximum length of an SMS it will be split into several parts. `max-messages` lets you specify a maximum number of individual SMS messages that can be used. If `max-messages` is set to 0, no reply is sent, except for error messages.
`accept-x-kannel-headers`	bool	Request reply can include special X-Kannel headers but these are only accepted if this variable is set to true. See *Extended headers*.
`assume-plain-text`	bool	If client does not set Content-Type for reply, it is normally application/octet-stream which is then handled as data in Kannel. This can be forced to be plain/text to allow backward compatibility, when data was not expected.

Variable	Value	Description
`concatenation`	bool	Long messages can be sent as independent SMS messages with `concatenation = false` or as concatenated messages with `concatenation = true`. Concatenated messages are reassembled into one long message by the receiving device.
`split-chars`	string	Allowed characters to split the message into several messages. So, with "#!" the message is split from last '#' or '!', which is included in the previous part.
`split-suffix`	string	If the message is split into several ones, this string is appended to each message except the last one.
`omit-empty`	bool	Normally, Kannel sends a warning to the user if there was an empty reply from the service provider. If `omit-empty` is set to 'true', Kannel will send nothing at all in such a case.
`header`	string	If specified, this string is automatically added to each SMS sent with this service. If the message is split, it is added to each part.
`footer`	string	As header, but not inserted into head but appended to end.
`prefix`	string	Stuff in answer that is cut away, only things between prefix and suffix is left. Not case sensitive. Matches the first prefix and then the first suffix. These are only used for `url` type services, and only if both are specified.
`suffix`	string	

Variable	Value	Description
white-list	URL	Load a list of accepted senders of SMS messages. If a sender of an SMS message is not in this list, any message received from the SMSC is rejected, unless a black-list service is defined. See notes of phone number format from numhash.h header file.
black-list	URL	As white-list, but SMS messages from these numbers are automatically discarded
accepted-smsc-regex	POSIX regular expression	Accept only SMS messages arriving from SMSCs with a matching ID. [d] See section on Regular Expressions for details.
accepted-account-regex	POSIX regular expression	Accept ONLY SMS messages arriving with a matching ACCOUNT field. [e] See section on Regular Expressions for details.
allowed-prefix-regex	POSIX regular expression	A set of phone number prefixes of sender-numbers accepted by this service. [f] See section on Regular Expressions for details.
denied-prefix-regex	POSIX regular expression	A set of phone number prefixes of sender-numbers which may not use this service. See section on Regular Expressions for details.
allowed-receiver-prefix-regex	POSIX regular expression	A set of phone number prefixes of receiver-numbers which may receive data sent by this service. This can be used to allow only inbound SMS to certain shortcut numbers to be allowed to this service. See section on Regular Expressions for details.
denied-receiver-prefix-regex	POSIX regular expression	A set of phone number prefixes of receiver-numbers which may not receive data sent by this service. See section on Regular Expressions for details.

Variable	Value	Description
white-list-regex	POSIX regular expression	Defines a set of accepted senders of SMS messages. If a sender of an SMS message is not in this list, the message is rejected. See section on Regular Expressions for details.
black-list-regex	POSIX regular expression	As white-list-regex, but SMS messages from these numbers are automatically discarded. See section on Regular Expressions for details.
alt-charset	string	Defines which character encoding is used for the SMS message when passed to a remote HTTP application. This includes how the SMS message text is send in the HTTP GET parameter list and in the HTTP POST bidy. Uses `iconv()` routines to convert from and to that specific character set encoding. See your local `iconv_open(3)` manual page for the supported character encodings and the type strings that should be presented for this directive.
dlr-url	string (URL)	Optional Defines a default URL which is fetched for DLR event, if no specific X-Kannel HTTP header parameter is returned by the HTTP response.
dlr-mask	number (bit mask)	Optional. Defines a default DLR event bit mask which is used in combination with the `dlr-url` config directive, if no specific X-Kannel HTTP header parameter is returned by the HTTP response.

Variable	Value	Description

Notes:
a. Even if this service is denied, Kannel still searches for other service which accepts the message, or default service.
b. Even if this service is denied, Kannel still searches for other service which accepts the message, or default service.
c. Like in accepted-smsc, Kannel still searches for other service which accepts the message. This way there could be several services with the same keyword and different results.
d. Even if this service is denied, Kannel still searches for other service which accepts the message, or default service.
e. Even if this service is denied, Kannel still searches for other service which accepts the message, or default service.
f. Like in accepted-smsc-regex, Kannel still searches for another service which accepts the message. This way there could be several services with the same keyword and different results.

Table 7-10. Parameters (Escape Codes)

`%k`	the keyword in the SMS request (i.e., the first word in the SMS message)
`%s`	next word from the SMS message, starting with the second one (i.e., the first word, the keyword, is not included); problematic characters for URLs are encoded (e.g., '+' becomes '%2B')
`%S`	same as %s, but '*' is converted to '~' (useful when user enters a URL) and URL encoding isn't done (all others do URL encode)
`%r`	words not yet used by %s; e.g., if the message is "FOO BAR FOOBAR BAZ", and the has been one %s, %r will mean "FOOBAR BAZ"
`%a`	all words of the SMS message, including the first one, with spaces squeezed to one
`%b`	the original SMS message, in a binary form (URL-encoded)
`%e`	the original SMS message, in a binary form (printable hexadecimal byte codes). I.e. "xyz" would be send as byte codes "78797A".
`%t`	the time the message was sent, formatted as "YYYY-MM-DD HH:MM", e.g., "1999-09-21 14:18"
`%T`	the time the message was sent, in UNIX epoch timestamp format
`%p`	the phone number of the sender of the SMS message
`%P`	the phone number of the receiver of the SMS message

%q	like %p, but a leading ‘00’ is replaced with ‘+’
%Q	like %P, but a leading ‘00’ is replaced with ‘+’
%i	the smsc-id of the connection that received the message
%I	the SMS ID of the internal message structure
%d	the delivery report value
%R	the delivery report URL value
%N	the delivery report notification message
%D	meta-data from the SMSC. See Meta Data for more into.
%A	the delivery report SMSC reply, if any
%F	the `foreign` (smsc-provided) message ID. Only relevant on DLR url’s.
%n	the sendsms-user or sms-service name
%c	message coding: 0 (default, 7 bits), 1 (8 bits) or 2 (Unicode)
%m	message class bits of DCS: 0 (directly to display, flash), 1 (to mobile), 2 (to SIM) or 3 (to SIM toolkit).
%M	mwi (message waiting indicator) bits of DCS: 0 (voice), 1, (fax), 2 (email) or 3 (other) for activation and 4, 5, 6, 7 for deactivation respectively.
%y	message priority: 0 (lowest) to 3 (highest).
%C	message charset: for a "normal" message, it will be "GSM" (coding=0), "binary" (coding=1) or "UTF-16BE" (coding=2). If the message was successfully recoded from Unicode, it will be "WINDOWS-1252"
%u	`udh` of incoming message (URL-encoded)
%U	`udh` of incoming message (printable hexadecimal byte codes). I.e. "xyz" would be send as byte codes "78797A".
%B	billing identifier/information of incoming message. The value depends on the SMSC module and the associated billing semantics of the specific SMSC providing the information. For EMI2 the value is the XSer 0c field, for SMPP MO it is the service_type of the deliver_sm PDU, and for SMPP DLR it is the DLR message err component. (Note: This is used for proxying billing information to external applications. There is no semantics associated while processing these.)

%o	account identifier/information of incoming message. The value depends on the SMSC module and has been introduced to allow the forwarding of an operator ID from aggregator SMSCs to the application layer, hence the smsbox HTTP calling instance.
%O	DCS (Data coding schema) value.
%f	Originating SMSC of incoming message. The value is set if the AT driver is used to receive a SMS on a gsm modem. The value of %f will contain the number of the SMSC sending the SMS to the SIM card. Other SMSC types than AT do not set this field so it will be empty.
%x	the smsbox-id of the message, identifying from which smsbox connection the message is originating.
%v	Validity period in minutes. Available only if SMSC supports and sent this value.
%V	Deferred in minutes. Available only if SMSC supports and sent this value.
%#	This allows to pass meta-data individual parameters into urls. The syntax is as follows: `%#group#parameter#` For example: `%#smpp#my_param#` would be replaced with the value 'my_param' from the group 'smpp' coming inside the meta_data field.

Some sample 'sms-service' groups:

```
group = sms-service
keyword = nop
text = "You asked nothing and I did it!"
catch-all = true

group = sms-service
keyword = complex
get-url = "http://host/service?sender=%p&text=%r"
accept-x-kannel-headers = true
max-messages = 3
concatenation = true

group = sms-service
keyword = default
text = "No action specified"
```

How sms-service interprets the HTTP response

When an `sms-service` requests a document via HTTP, it will accept one of four types of content types:

`text/plain`	Blanks are squeezed into one, rest is chopped to fit an SMS message.
`text/html`	Tags are removed, rest is chopped to fit an SMS message.
`text/vnd.wap.wml`	Processed like HTML.
`text/xml`	Processed as a POST-XML. See *XML Post*
`application/octet-stream`	The body will be transmitted as the SMS message, as 8-bit data. This can be avoided by setting `assume-plain-text` variable on for the SMS-service.

Extended headers

Kannel uses and accepts several X-Kannel headers to be used with SMS-services, if option `accept-x-kannel-headers` was provided in the relevant 'sms-service' group.

Table 7-11. X-Kannel Headers

SMSPush equivalent	X-Kannel Header
`username`	`X-Kannel-Username`
`password`	`X-Kannel-Password`
`from`	`X-Kannel-From`
`to`	`X-Kannel-To`
`text`	request body
`charset`	charset as in `Content-Type: text/html; charset=ISO-8859-1`
`udh`	`X-Kannel-UDH`
`smsc`	`X-Kannel-SMSC`
`flash`	`X-Kannel-Flash` (deprecated, see `X-Kannel-MClass`
`mclass`	`X-Kannel-MClass`
`mwi`	`X-Kannel-MWI`
`compress`	`X-Kannel-Compress`
`coding`	`X-Kannel-Coding`. If unset, defaults to 0 (7 bits) if `Content-Type` is `text/plain` , `text/html` or `text/vnd.wap.wml`. On `application/octet-stream`, defaults to 8 bits (1). All other `Content-Type` values are rejected.

SMSPush equivalent	X-Kannel Header
validity	X-Kannel-Validity
deferred	X-Kannel-Deferred
dlr-mask	X-Kannel-DLR-Mask
dlr-url	X-Kannel-DLR-Url
account	X-Kannel-Account
pid	X-Kannel-PID
alt-dcs	X-Kannel-Alt-DCS
binfo	X-Kannel-BInfo
rpi	X-Kannel-RPI
priority	X-Kannel-Priority
meta-data	X-Kannel-Meta-Data

Kannel POST

Kannel can do POST if `service` is contains a `post-url="..."`.

Table 7-12. X-Kannel Post Headers

Parameter (escape code) equivalent	X-Kannel Header	Notes
%p (from)	X-Kannel-From	Only sent if send true
%P (to)	X-Kannel-To	
%t (time)	X-Kannel-Time	
%u (udh)	X-Kannel-UDH	in hex format: 060504158200
%i (smsc)	X-Kannel-SMSC	
- (mclass)	X-Kannel-MClass	
- (pid)	X-Kannel-PID	
- (alt-dcs)	X-Kannel-Alt-DCS	
- (mwi)	X-Kannel-MWI	
%c (coding)	X-Kannel-Coding	0=7 Bits, 1=8 Bit
- (compress)	X-Kannel-Compress	
- (validity)	X-Kannel-Validity	
- (deferred)	X-Kannel-Deferred	
%n (service name)	X-Kannel-Service	
%D (meta-data)	X-Kannel-Meta-Data	Meta-Data (Only supported at the r
%a or %r (text)	request body	Kannel send all w unless strip-ke

Parameter (escape code) equivalent	X-Kannel Header	Notes
`%C (charset)`	present in `Content-Type HTTP`	Example: `Conte` `text/plain; charset=ISO-8`

XML Post

Kannel can send and receive XML POST with the following format:

```
<?xml version="1.0"?>
<!DOCTYPE ...>
<message>
  <submit>
    <da><number>destination number (to)</number></da>
    <oa><number>originating number (from)</number></oa>
    <ud>user data (text)</ud>
    <udh>user data header (udh)</udh>
    <meta-data>meta-data</meta-data>
    <dcs>
      <mclass>mclass</mclass>
      <coding>coding</coding>
      <mwi>mwi</mwi>
      <compress>compress</compress>
      <alt-dcs>alt-dcs</alt-dcs>
    </dcs>
    <pid>pid</pid>
    <rpi>rpi</rpi>
    <vp>
      <delay>validity time in minutes</delay>
    </vp>
    <timing>
      <delay>deferred time in minutes</delay>
    </timing>
    <statusrequest>
      <dlr-mask>dlr-mask</dlr-mask>
      <dlr-url>dlr-url</dlr-url>
    </statusrequest>

    <!-- request from application to Kannel -->
    <from>
      <user>username</user>
      <username>username</username>
      <pass>password</pass>
      <password>password</password>
      <account>account</account>
    </from>
    <to>smsc-id</to>

    <!-- request from Kannel to application -->
    <from>smsc-id</from>
```

```
    <to>service-name</to>

  </submit>
</message>
```

Note: Don't forget to set POST Content-Type to `text/xml`!

There could be several `da` entries for `sendsms-user` to enable multi-recipient messages. `da` doesn't make sense in `sms-service`.

`udh` is the same format as `X-Kannel-UDH`. Example: <udh>06050415820000</udh>.

On Kannel->application, `from` is the `smsc-id` that message arrives and `to` is the service name.

On application->Kannel, `from` contains the credentials (`user/username`, `pass/password` and `account` and `to` corresponds to the `smsc-id` to submit the message.

`user` and `username` are equivalent and only one of them should be used. (same for `pass` and `password`.

When application POST in Kannel, as in GET, only `user`, `pass` and `da` are required. Everything else is optional. (`oa` could be needed too is there's no `default-sender` or `forced-sender`.

Warning

This is experimental code. XML format could and should change to fully met IETF's sms-xml standard (yet in draft) and additional tags needed by Kannel should be pondered.

SendSMS-user configurations

To enable an SMS push, you must set `sendsms-port` into the 'smsbox' group and define one or more 'sendsms-user' groups. Each of these groups define one account, which can be used for the SMS push, via HTTP interface (see below)

Table 7-13. SendSMS-User Group Variables

Variable	Value	Description
`group (m)`	`sendsms-user`	This is a mandatory variable
`username (m)`	string	Name for the user/account.
`password (m)`	string	Password for the user (see HTTP interface, below)
`name`	string	As in 'sms-service' groups.
`user-deny-ip`	IP-list	As other deny/allow IP lists, but for this user (i.e. this user is not allowed to do the SMS push HTTP request from other IPs than allowed ones). If not set, there is no limitations.

Variable	Value	Description
user-allow-IP	IP-list	
forced-smsc	string	Force SMSC ID as a 'string' (linked to SMS routing, see 'smsc' groups)
default-smsc	string	If no SMSC ID is given with the send-sms request (see below), use this one. No idea to use with forced-smsc.
forced-priority	number	Force priority of any message send by this user, in the range (0..3).
max-priority	number	Maximum allowed priority for any message send by this user, in the range (0..3).
default-sender	phone-number	This number is set as sender if not set by `from` get/post parameter
faked-sender	phone-number	As in 'sms-service' groups
max-messages	number	
concatenation	bool	
split-chars	string	
split-suffix	string	
omit-empty	bool	
header	string	
footer	string	
allowed-prefix	prefix-list	A list of phone number prefixes which are accepted to be sent using this username. Multiple entries are separated with semicolon (';'). For example, "040;050" prevents sending of any SMS message with prefix of 040 or 050 through this SMSC. If denied-prefix is unset, only this numbers are allowed. If set, number are allowed if present in allowed or not in denied list.
denied-prefix	prefix-list	A list of phone number prefixes which are NOT accepted to be sent using this username.

Variable	Value	Description
white-list	URL	Load a list of accepted destinations of SMS messages. If a destination of an SMS message is not in this list, any message received from the HTTP interface is rejected. See notes of phone number format from numhash.h header file.
black-list	URL	As white-list, but SMS messages from these numbers are automatically rejected.
dlr-url	URL	Optional. Defines a default URL which is fetched for DLR event, if no specific dlr-url CGI parameter is set in the HTTP request.
dlr-mask	number (bit mask)	Optional. Defines a default DLR event bit mask which is used in combination with the dlr-url config directive, if no specific dlr-url CGI parameter is set in the HTTP request.
allowed-prefix-regex	POSIX regular expression	A set of phone numbers which are accepted to be sent using this username. See section on Regular Expressions for details.
denied-prefix-regex	POSIX regular expression	A set of phone numbers which may not send using this username. See section on Regular Expressions for details.
white-list-regex	POSIX regular expression	Defines a set of accepted destinations of SMS messages. If a destination of an SMS message is not in this list, any message received from the HTTP interface is rejected. See section on Regular Expressions for details.
black-list-regex	POSIX regular expression	As white-list-regex, but SMS messages originating from a number matching the pattern are discarded. See section on Regular Expressions for details.

Some sample 'sendsms-user' groups:

```
group = sendsms-user
username = simple
password = elpmis

group = sendsms-user
username = complex
password = 76ftY
user-deny-ip = "*.*.*.*"
user-allow-ip = "123.234.123.234"
max-messages = 3
concatenation = true
forced-smsc = SOL
```

The second one is very limited and only allows a user from IP "123.234.123.234". On the other hand, the user can send a longer message, up to 3 SMSes long, which is sent as concatenated SMS.

Over-The-Air configurations

To enable Over-The-Air configuration of phones or other client devices that support the protocol you need to configure a `sendsms-user.ota-setting` group is not necessary, you can send settings to the phone as a XML document, but this method is perhaps more suitable for continuous provisioning.

If you want to send multiple OTA configurations through the smsbox and you do not want to send XML documents, you will have to declare a `ota-id` string to the different `ota-setting` groups.

Table 7-14. OTA Setting Group Variables

Variable	Value	Description
`group`	`ota-setting`	This is a mandatory variable
`ota-id`	`string`	An optional name or id for the ota-setting. Any string is acceptable, but semicolon ';' may cause problems, so avoid it and any other special non-alphabet characters.
`location`	`URL`	The address of the HTTP server for your WAP services, i.e. `http://wap.company.com`
`service`	`string`	Description of the service
`ipaddress`	`IP`	IP address of your WAP gateway
`phonenumber`	`phone-number`	Phone number used to establish the PPP connection
`speed`	`number`	Connection speed: 9600 or 14400. Defaults to 9600.
`bearer`	`string`	Bearer type: data or sms. Defaults to data.

Variable	Value	Description
calltype	string	Call type: isdn or analog. Defaults to isdn.
connection	string	Connection type: cont or temp. Cont uses TCP port 9201 and Temp uses UDP port 9200. Defaults to cont.
pppsecurity	on or off	Enable CHAP authentication if set to on, PAP otherwise
authentication		normal or secure. Indicates whether WTLS should be used or not. Defaults to normal.
login	string	Login name.
secret	string	Login password

A sample 'ota-setting' group:

```
group = ota-setting
location = http://wap.company.com
service = "Our company's WAP site"
ipaddress = 10.11.12.13
phonenumber = 013456789
bearer = data
calltype = analog
connection = cont
pppsecurity = off
authentication = normal
login = wapusr
secret = thepasswd
```

And a 'sendsms-user' to use with it. With concatenation enabled:

```
group = sendsms-user
username = otauser
password = foo
max-messages = 2
concatenation = 1
```

Table 7-15. OTA Bookmark Group Variables

Variable	Value	Description
group	ota-bookmark	This is a mandatory variable

Variable	Value	Description
ota-id	string	An optional name or id for the ota-bookmark. Any string is acceptable, but semicolon ';' may cause problems, so avoid it and any other special non-alphabet characters.
url	URL	The address of the HTTP server for your WAP services, i.e. http://wap.company.com
name	string	Description of the service

A sample 'ota-bookmark' group:

```
group = ota-bookmark
ota-id = wap-link
url = "http://wap.company.com"
service = "Our company's WAP site"
```

And a 'sendsms-user' to use with it, with the same conditions as for the 'ota-setting' group.

Setting up more complex services

The basic service system is very limited - it can only answer to original requester and it cannot send UDH data, for example. This chapter explains some more sophisticated and complex SMS service setups.

Redirected replies

The basic service system always sends the answer back to original requester, but sometimes the content server needs to send something to other terminals or delay the answer. To create such systems, an SMS push is used.

The idea is to get the initial request, but then send no reply. Instead, the reply (if any) is sent via HTTP sendsms-interface as SMS Push. This way the service application has full control of the return content, and can do all needed formatting beforehand.

Note that when no reply is wanted, remember to set the variable `max-messages` to zero (0) so that no reply is sent, unless an error occurs. Simple sample:

```
group = sms-service
keyword = talk
get-url = "http://my.applet.machine/Servlet/talk?sender=%p&text=%r"
max-messages = 0
```

Setting up operator specific services

Those running Kannel with several SMS centers might need to define services according to the relying SMS center. To achieve this, first you need to give an ID name for SMS center connections (see above). Then use the `accepted-smsc` variable to define which messages can use that service.

```
group = sms-service
keyword = weather
accepted-smsc = SOL
get-url = "http://my.applet.machine/Servlet/weather?sender=%p&operator=SOL&text=%r"
```

Setting up multi-operator Kannel

Sometimes there is a need for Kannel to listen to two (or more) distinct SMS centers, and messages must be routed to services according to where they came from, and replies likewise must return to same SMSC. This is done via `smsc-id` magic. Here is a shortened sample configuration, which handles to distinct SMS servers and services:

```
group = smsc
smsc-id = A
denied-smsc-id = B
...

group = smsc
smsc-id = B
denied-smsc-id = A
...

group = sms-service
accepted-smsc = A
get-url = "..."

group = sms-service
accepted-smsc = B
get-url = "..."
```

As can be seen, the `smsc-id` is used to identify the SMS center from which the message came. Then, the `denied-smsc-id` variable is used to prevent messages originally from the other SMS center from being sent through the other one. Finally 'sms-service' groups are defined with `accepted-smsc` so that they only accept messages from certain SMS center.

If you want to use SMS push services, requesters should then set the `smsc` request parameter, or 'sendsms-user' groups should be defined like this:

```
group = sendsms-user
username = operator_A
password = foo
forced-smsc = A

group = sendsms-user
username = operator_B
password = bar
```

```
forced-smsc = B
```

Note that if your SMS centers do not set the sender phone number but rely on number transmitted, you should set `faked-sender` to all 'sendsms-user' groups.

Running SMS gateway

Using the HTTP interface to send SMS messages

After you have configured Kannel to allow the sendsms service, you can send SMS messages via HTTP, e.g., using a WWW browser. The URL looks something like this:

```
http://smsbox.host.name:13013/cgi-bin/sendsms?
username=foo&password=bar&to=0123456&text=Hello+world
```

Thus, technically, you make an HTTP GET request. This means that all the information is stuffed into the URL. If you want to use this often via a browser, you probably want to make an HTML form for this.

Kannel will answer to sendsms request with following codes and body texts:

Table 7-16. SMS Push reply codes

Status	Body	Meaning
202	`0: Accepted for delivery`	The message has been accepted and is delivered onward to a SMSC driver. Note that this status does not ensure that the intended recipient receives the message.
202	`3: Queued for later delivery`	The bearerbox accepted and stored the message, but there was temporarily no SMSC driver to accept the message so it was queued. However, it should be delivered later on.
4xx	(varies)	There was something wrong in the request or Kannel was so configured that the message cannot be in any circumstances delivered. Check the request and Kannel configuration.
503	`Temporal failure, try again later.`	There was temporal failure in Kannel. Try again later.

Table 7-17. SMS Push (send-sms) CGI Variables

`username` (or `user`)	`string`	Username or account name. Must be `username` of the one 'sendsms-user' group in the Kannel configuration, or results in 'Authorization failed' reply.
`password` (or `pass`)	`string`	Password associated with given `username`. Must match corresponding field in the 'sendsms-user' group of the Kannel configuration, or 'Authorization failed' is returned.
`from`	`string`	Phone number of the sender. This field is usually overridden by the SMS Center, or it can be overridden by `faked-sender` variable in the `sendsms-user` group. If this variable is not set, smsbox `global-sender` is used.
`to`	`phone number list`	Phone number of the receiver. To send to multiple receivers, separate each entry with *space* (' ', '+' url-encoded) - but note that this can be deactivated via `sendsms-chars` in the 'smsbox' group.
`text`	`string`	Contents of the message, URL encoded as necessary. The content can be more than 160 characters, but then `sendsms-user` group must have `max-messages` set more than 1.
`charset`	`string`	Charset of text message. Used to convert to a format suitable for 7 bits or to UCS-2. Defaults to UTF-8 if coding is 7 bits and UTF-16BE if coding is UCS-2.
`udh`	`string`	Optional User Data Header (UDH) part of the message. Must be URL encoded.

`smsc`	`string`	Optional virtual smsc-id from which the message is supposed to have arrived. This is used for routing purposes, if any denied or preferred SMS centers are set up in SMS center configuration. This variable can be overridden with a `forced-smsc` configuration variable. Likewise, the `default-smsc` variable can be used to set the SMSC if it is not set otherwise.
`flash`	`number`	Deprecated. See `mclass`.
`mclass`	`number`	Optional. Sets the Message Class in DCS field. Accepts values between 0 and 3, for Message Class 0 to 3, A value of 0 sends the message directly to display, 1 sends to mobile, 2 to SIM and 3 to SIM toolkit.
`mwi`	`number`	Optional. Sets Message Waiting Indicator bits in DCS field. If given, the message will be encoded as a Message Waiting Indicator. The accepted values are 0,1,2 and 3 for activating the voice, fax, email and other indicator, or 4,5,6,7 for deactivating, respectively. This option excludes the `flash` option. [a]
`compress`	`number`	Optional. Sets the Compression bit in DCS Field.
`coding`	`number`	Optional. Sets the coding scheme bits in DCS field. Accepts values 0 to 2, for 7bit, 8bit or UCS-2. If unset, defaults to 7 bits unless a udh is defined, which sets coding to 8bits.

`validity`	`number (minutes)`	Optional. If given, Kannel will inform SMS Center that it should only try to send the message for this many minutes. If the destination mobile is off other situation that it cannot receive the sms, the smsc discards the message. Note: you must have your Kannel box time synchronized with the SMS Center.
`deferred`	`number (minutes)`	Optional. If given, the SMS center will postpone the message to be delivered at now plus this many minutes. Note: you must have your Kannel box time synchronized with the SMS Center.
`dlrmask`	`number (bit mask)`	Deprecated. See `dlr-mask`.
`dlr-mask`	`number (bit mask)`	Optional. Request for delivery reports with the state of the sent message. The value is a bit mask composed of: 1: Delivered to phone, 2: Non-Delivered to Phone, 4: Queued on SMSC, 8: Delivered to SMSC, 16: Non-Delivered to SMSC. Must set `dlr-url` on `sendsms-user` group or use the `dlr-url` CGI variable.
`dlrurl`	`string (url)`	Deprecated. See `dlr-url`.
`dlr-url`	`string (url)`	Optional. If `dlr-mask` is given, this is the url to be fetched. (Must be url-encoded)
`pid`	`byte`	Optional. Sets the PID value. (See ETSI Documentation). Ex: SIM Toolkit messages would use something like `&pid=127&coding=1&alt-dcs=1&mclass=3`
`alt-dcs`	`number`	Optional. If unset, Kannel uses the alt-dcs defined on smsc configuration, or 0X per default. If equals to 1, uses FX. If equals to 0, force 0X.

`rpi`	`number`	Optional. Sets the Return Path Indicator (RPI) value. (See ETSI Documentation).
`account`	string	Optional. Account name or number to carry forward for billing purposes. This field is logged as ACT in the log file so it allows you to do some accounting on it if your front end uses the same username for all services but wants to distinguish them in the log. In the case of a HTTP SMSC type the account name is prepended with the service-name (username) and a colon (:) and forwarded to the next instance of Kannel. This allows hierarchical accounting.
`binfo`	string	Optional. Billing identifier/information proxy field used to pass arbitrary billing transaction IDs or information to the specific SMSC modules. For EMI2 this is encapsulated into the XSer 0c field, for SMPP this is encapsulated into the service_type of the submit_sm PDU.
`priority`	number	Optional. Sets the Priority value (range 0-3 is allowed). (Defaults to 0, which is the lowest priority).

Notes:

a. To set number of messages, use `mwi=[0-3]&coding=0&udh=%04%01%02%<XX>%<YY>`, where YY are the number of messages, in HEX, and XX are `mwi` plus 0xC0 if `text` field is not empty.

Using the HTTP interface to send OTA configuration messages

OTA messages can be sent to mobile phones or devices to auto-configure the settings for WAP. They are actually complex SMS messages with UDH and sent as concatenated messages if too long (and compiled if necessary).

You may either pass an HTTP request as GET method or POST method to the HTTP interface.

If you want to send a configuration that is defined within Kannel's configuration file itself you have to pass a valid `ota-id` value otherwise the content of the request will be compiled to as OTA message.

OTA settings and bookmark documents

Here is an example XML document (this one contains CSD settings for logging in to a mobile service; note that you must store DTD locally):

```
<?xml version="1.0"?>
<!DOCTYPE CHARACTERISTIC-LIST SYSTEM "file:///gw/settings.dtd">
<CHARACTERISTIC-LIST>

  <CHARACTERISTIC TYPE="ADDRESS">
    <PARM NAME="BEARER" VALUE="GSM/CSD"/>
    <PARM NAME="PROXY" VALUE="10.11.12.13"/>
    <PARM NAME="PORT" VALUE="9201"/>
    <PARM NAME="CSD_DIALSTRING" VALUE="+12345678"/>
    <PARM NAME="PPP_AUTHTYPE" VALUE="PAP"/>
    <PARM NAME="PPP_AUTHNAME" VALUE="yourusername"/>
    <PARM NAME="PPP_AUTHSECRET" VALUE="yourauthsecret"/>
    <PARM NAME="CSD_CALLTYPE" VALUE="ISDN"/>
    <PARM NAME="CSD_CALLSPEED" VALUE="9600"/>
  </CHARACTERISTIC>

  <CHARACTERISTIC TYPE="URL"
                  VALUE="http://wap.company.com/"/>

  <CHARACTERISTIC TYPE="NAME">
    <PARM NAME="NAME" VALUE="Your WAP Company"/>
  </CHARACTERISTIC>

</CHARACTERISTIC-LIST>
```

A bookmark document looks like this:

```
<?xml version="1.0"?>
<!DOCTYPE CHARACTERISTIC_LIST SYSTEM "file:///gw/settings.dtd">
<CHARACTERISTIC-LIST>
  <CHARACTERISTIC TYPE="BOOKMARK">
    <PARM NAME="NAME" VALUE="WAP Company"/>
    <PARM NAME="URL" VALUE="http://wap.company.com/"/>
  </CHARACTERISTIC>
</CHARACTERISTIC-LIST>
```

Document type definition (DTD) for these documents is not available from Internet, you must supply it as a file. Kannel gw directory contains an example, `settings.dtd`.

OTA syncsettings documents

Used for the provisioning of sync configuration to SyncMl enabled handsets. Best supported by sonyericcsson terminals.

Sample syncsettings documents to set contacts, connection data and authentication:

```
<SyncSettings>
<Version>1.0</Version>
<HostAddr>http://syncml.server.com</HostAddr>
<RemoteDB>
  <CTType>text/x-vcard</CTType>
  <CTVer>2.1</CTVer>
  <URI>contact</URI>
  <Name>Address Book</Name>
</RemoteDB>
<Name>Synchonization</Name>
<Auth>
  <AuthLevel>1</AuthLevel>
  <AuthScheme>1</AuthScheme>
  <Username>yourusername</Username>
  <Cred>passwordbase64encoded</Cred>
</Auth>
</SyncSettings>
```

OMA provisioning content

OMA provisioning allows the configuration of a wider range of settings than previously available in OTA terminals. Refer to OMA-WAP-ProvCont-v1_1-20021112-C (at http://www.openmobilealliance.org/tech/docs/) for details.

A shared secret (i.e. a pin or the phone IMSI) can be used to authenticate the settings. Defaults are 'userpin' and '1234' a. See the cgi variables 'sec' and 'pin' for available authentication options.

GET method for the OTA HTTP interface

An example URL (OTA configuration defined in the Kannel configuration file):

```
http://smsbox.host.name:13013/cgi-bin/sendota?
    otaid=myconfig&username=foo&password=bar&to=0123456
```

URL containing XML document looks like this (you must URL encode it before sending it over HTTP):

```
http://smsbox.host.name:13013/cgi-bin/sendota?
    username=foo&password=bar&to=0123456&
    text=MyURLEncodedXMLdocument&type=settings
```

You can send either settings, bookmark, syncsettings, or oma-settings, set CGI variable type accordingly. Default for this variable is settings.

Table 7-18. OTA CGI Variables

`otaid`	`string`	Name or ID of the 'ota-setting' group in Kannel configuration that should be sent to the phone. This variable is optional. If it is not given the first 'ota-setting' group is sent. This is unnecessary when a XML document is send to the phone.
`username`	`string`	Username of the 'sendsms-user' group in Kannel configuration, that has been configured to send OTA messages.
`password`	`string`	Password associated with given `username`. Must match corresponding field in 'sendsms-user' group in Kannel configuration, or 'Authorization failed' is returned.
`to`	`number`	Number of the phone that is to receive the OTA configuration message.
`from`	`string`	Phone number of the sender. This field is usually overridden by the SMS Center, or it can be overridden by faked-sender variable in the sendsms-user group. If this variable is not set, smsbox global-sender is used.
`smsc`	`string`	Optional virtual smsc-id from which the message is supposed to have arrived. This is used for routing purposes, if any denied or preferred SMS centers are set up in SMS center configuration. This variable can be overridden with a `forced-smsc` configuration variable. Likewise, the `default-smsc` variable can be used to set the SMSC if it is not set otherwise.

text	XML document	An URL encoded XML document, containing either settings or bookmarks.
type	string	Type of the XML document, supported values are "settings", "bookmarks", "syncsettings", and "oma-settings". Default is "settings".
sec	string	Security model used to authenticate oma-settings. One of: "userpin", "netwpin", "usernetwpin", or "userpinmac".
pin	number	Authentication token.

Chapter 8. Setting up Push Proxy Gateway

This chapter explains how to set up a push proxy gateway (PPG). An example configuration file are given. A working push proxy gateway is described. Routing wap pushes to a certain smsc and asking for *SMS level* delivery reports are described.

Configuring ppg core group, for push initiator (PI) interface

PPG configuration group defines gateway's service interface. Configuring a PPG working with a trusted PI is easiest. Actually, you need no configuration at all: in this case a PPG with default values will be set up. Do not rely on this, default values may change. For PPG core configuration variables, see table 7.1.

An example of a core configuration for PPG working only with specific addresses follows. Note that ppg-deny-ip-list is not actually necessary, but does make configuring simpler: IPs here are always denied, even when they are mentioned in the allowed IPs list.

Ppg-url is a simple stamp, used for routing requests to the right service. You can change this stamp by setting `push-url` configuration variable.

```
group = ppg
ppg-url = /wappush
ppg-port = 8080
concurrent-pushes = 100
users = 1024
ppg-allow-ip = 194.100.32.125;127.0.0.1
ppg-deny-ip = 194.100.32.89;194.100.32.103
trusted-pi = false
```

Table 8-1. PPG core group configuration variables

Variable	Value	Description
`group`	*ppg*	Mandatory value. Tells that we are configuring the PPG group.
`ppg-port`	*number*	The port PPG is listening at. Default 8080.
`ppg-ssl-port (o)`	*number*	Mandatory value for PPG HTTPS support. The port at which PPG listens for HTTPS requests. There are no defaults; you must set the value separately.
`ssl-server-cert-file`	*string*	Mandatory value for PPG HTTPS support. The file containing server's ssl certificate.

Variable	Value	Description
`ssl-server-key-file`	*string*	Mandatory value for PPG HTTPS support. The file containing server's ssl private key.
`ppg-url`	*url*	URL locating PPG services. Default `/wappush` .
`global-sender`	*string*	Sender phone number required by some protocols.
`concurrent-pushes`	*number*	Number of concurrent pushes expected. Note that PPG *does* work even value is too low; it will only be slower. Default 100.
`users`	*number*	Number of actually configured user accounts. Note that PPG *does* work even value is too low; it will only be slower. Default 1024.
`trusted-pi`	*boolean*	If true, PI does authentication for PPG. Obviously, both of them must reside inside same firewall. Default true. If this variable is true, all security variables are ignored (even though they may be present).
`ppg-deny-ip`	*ip-list*	PPG will not accept pushes from these IPs. Wild-cards are allowed. If this attribute is missing, no IP is denied *by this list* .
`ppg-allow-ip`	*ip-list*	PPG will accept pushes from these, and only these, IPs. Wild-cards are allowed. Adding this list means that IPs not mentioned are denied, too.
`default-smsc`	*string*	If no SMSC ID is given with the wappush HTTP request (see below), use this one as default route for all push messages.
`default-dlr`	*string*	If no dlr url is given with the wappush HTTP request (see below), use this one as default route for all push messages.

Variable	Value	Description
ppg-smsbox-id	*string*	All ppg delivery reports are handled by this smsbox. This routes all DLR messages inside beaerbox to the specified smsbox for processing the HTTP requests to the DLR-URL.
service-name	*string*	This is sms service name used by smsbox for wap push.
default-dlr	*string*	If no dlr url is given with the wappush HTTP request (see below), use this one as default route for all push messages.
service-name	*string*	This is sms service name used by smsbox for wap push..
concatenation	*boolean*	Segment wap push binary sms. Default on. You need not normally set this value.
max-messages	*integer*	Maximum number of sm messages generated. Default 10. You need not set this variable, expect when your push documents are *very* long.

Configuring PPG user group variables

In addition of pi lists similar to the core group, ppg configuration specific to a certain user contains variables used for authentication and enforcing restrictions to phone numbers pi may contact. All variables are elaborated in table 7.2.

As an example, let us see how to configure a ppg user (a pi, named here 'picom') allowed to send pushes only from a specified ip.

```
group = wap-push-user
wap-push-user = picom
ppg-username = foo
ppg-password = bar
allow-ip = 62.254.217.163
```

It goes without saying that in real systems you must use more complex passwords than bar.

Table 8-2. PPG user group configuration variables

Variable	Value	Description
group	*wap-push-user*	Mandatory value. Tells that we are configuring the users group.

Variable	Value	Description
wap-push-user	*string*	(More) human readable name of an user.
ppg-username	*string*	Username for this user.
ppg-password	*string*	Password for this user.
allowed-prefix	*number-list*	Phone number prefixes allowed in pushes coming from this pi. These prefixes must conform international phone number format.
denied-prefix	*number-list*	Phone number prefixes denied in pushes coming from this pi. These prefixes must conform international phone number format.
white-list	*url*	Defines an url where from the white-list can be fetched. White list itself contains list of phone numbers accepting pushes from this pi.
black-list	*url*	Defines an url where from the blacklist can be fetched. Blacklist itself contains list of phone number not accepting pushes from this pi.
allow-ip	*ip-list*	Defines IPs where from this pi can do pushes. Adding this list means that IPs not mentioned are denied.
deny-ip	*ip-list*	Defines IPs where from this pi cannot do pushes. IPs not mentioned in either list are denied, too.
default-smsc	*string*	If no SMSC ID is given with the wappush HTTP request (see below), use this one as default route for this specific push user.
forced-smsc	*string*	Allow only routing to a defined SMSC ID for this specific push user.
dlr-url	*string*	If no dlr is given with the wappush HTTP request (see below), use this one as default route for this specific push user.

Variable	Value	Description
smsbox-id	*string*	Smsbox handling delivery reports fro this user.
forced-smsc	*string*	Allow only routing to a defined SMSC ID for this specific push user.
white-list-regex	POSIX regular expression	Defines the set of phone-numbers accepting pushes from this pi. See section on Regular Expressions for details.
black-list-regex	POSIX regular expression	Defines the set of phone-numbers rejecting pushes from this pi. See section on Regular Expressions for details.
allowed-prefix-regex	POSIX regular expression	Set of phone number prefixes allowed in pushes coming from this pi. See section on Regular Expressions for details.
denied-list-regex	POSIX regular expression	Set of phone number prefixes denied in pushes coming from this pi. See section on Regular Expressions for details.

Finishing ppg configuration

PPG uses SMS for sending SI to the phone and an IP bearer to fetch content specified by it (see chapter Overview of WAP Push). This means both wapbox and bearer smsc connections are in use. So your push proxy gateway configuration file must contain groups core, wapbox, smsc and smsbox. These are configured normal way, only smsc group may have push-specific variables. Note that following configurations are only an example, you may need more complex ones.

Bearerbox setup does not require any new variables:

```
group = core
admin-port = 13000
smsbox-port = 13001
wapbox-port = 13002
admin-password = b
wdp-interface-name = "*"
log-file = "filename"
log-level = 1
box-deny-ip = "*.*.*.*"
box-allow-ip = "127.0.0.1"
unified-prefix = "00358,0"
```

You must set up wapbox, for pulling (fetching) the wap data, and of course starting the push itself. No new variables here, either.

```
group = wapbox
bearerbox-host = localhost
log-file = "filename"
log-level = 0
syslog-level = none
```

To set up smsc connections, for pushing SI or SL over SMS. Here HTTP SMSC is used as an example. Variables no-sender and no-coding simplify HTTP request generated by Kannel. Send-url specifies content gateway, or sendsms service.

```
group = smsc
smsc = http
smsc-id = HTTP
port = 10000
system-type = kannel
smsc-username = foo
smsc-password = bar
no-sender = true
no-coding = true
send-url = http://host:port/path
```

To set up smsbox. This is used for ppg delivery reports, see later.

```
group = smsbox
bearerbox-host = localhost
smsbox-id = dlrbox
```

Kannel sources contain a sample push configuration file `gw/pushkannel.conf`.

Running a push proxy gateway

Push proxy gateway is started by simply typing, using separate windows:

```
gw/bearerbox [config-file]
gw/wapbox [config-file]
```

You can, of course, use more complex command line options.

An example using HTTP SMSC

An easy way to test and implement push services is to put ppg in the front of an existing sendsms service capable to send SMS data messages and to understand HTTP requests generated by HTTP SMSC. (See next chapter.) Then you need only configure SMSC configuration send-url to point to sendsms service.

An example of minimum SI document

Service indication (SI) should work with all types of phones, service loading does not. URL to be loaded is the main content of the document in both cases, however. Here an example (this is a minimum si document, not usable for testing, probably, but you want PPG to generate only one one SM per SI, if at all possible):

```
<?xml version="1.0"?>
<!DOCTYPE si PUBLIC "-//WAPFORUM//DTD SI 1.0//EN"
"http://www.wapforum.org/DTD/si.dtd">
<si>
    <indication href="http://www.gni.ch"
        si-id="1@gni.ch">
            You have 4 new emails
    </indication>
</si>
```

Note following points: a) Every SI must have different si-id. If there are none, href is used as an (very unsatisfactory) id. b) this si should not interrupt the phone's current action.

An example push (tokenized SI) document

HTTP SMSC generates a HTTP get request when it get a send-message event, expressed in Unicode. The content gateway, or the sendsms service must, of course, understand this URL. So here is an example, cgi variable text contains the url escaped form of a SI document. It is usable for testing prototype phones.

```
http://matrix:8080/phplib/kannelgw.php?user=*deleted*&
pass=*deleted*=to=%2B358408676001&text=3D%02%06%17%AE%96localhost
%3A8080%00%AF%80%8D%CF%B4%80%02%05j%00E%C6%0C%03wap.iobox.fi%00%11%03
1%40wiral.com%00%07%0A%C3%07%19%99%06%25%15%23%15%10%C3%04+%02%060%01
%03Want+to+test+a+fetch%3F%00%01%01&udh=%06%05%04%0B%84%23%F0
```

Default network and bearer used by push proxy gateway

If network and bearer attributes of the pap control document are missing or set any, Kannel uses address type for routing purposes: if the address type is a phone number (TYPE=PLMN), network defaults to GSM and bearer to SMS; if it is a IP-address (TYPE=IPv4), network defaults to GSM and bearer to CSD. So following minimal pap document works:

```
<?xml version="1.0"?>
<!DOCTYPE pap PUBLIC "-//WAPFORUM//DTD PAP//EN"
          "http://www.wapforum.org/DTD/pap_1.0.dtd">
<pap>
  <push-message push-id="9fjeo39jf084@pi.com">
    <address address-value="WAPPUSH=+358408676001/TYPE=PLMN@ppg.carrier.com"/>
  </push-message>
</pap>
```

Push related Kannel headers

This chapter recapitulates Kannel headers used by ppg.

PPG uses many Kannel headers. These are very similar as ones used by smsbox. (Both send sms to to the phone, after all.)

Table 8-3. Kannel headers used by PPG

Variable	Value	Description
`X-Kannel-SMSC`	*string*	Name of smsc used to deliver this push. Smsc configuration must contain some of corresponding variables, see
`X-Kannel-DLR-URL`	*url*	Url smsbox would call this url when doing the delivery report. Note that it can contain all Kannel escapes. See table 6.7 for details.
`X-Kannel-DLR-Mask`	*number*	Mask telling smsbox when it should do the delivery reports. Values are same as used by smsbox, see chapter Delivery Reports for details.
`X-Kannel-Smsbox-Id`	*string*	Tells which smsbox does the delivery report. Smsbox configuration must contain corresponding variable.

Requesting SMS level delivery reports for WAP pushes

Push content is a normal binary SM, so you can ask delivery reports for them. These are useful for testing purposes (did the phone get the content at all, or did it just reject it.) Another use is create fall-back services for phones not supporting displaying a specific push content. (MMS one being perhaps currently most obvious.) Generally speaking, this service is very similar to smsbox one. See chapter Delivery Reports for details.

For ppg sms delivery reports you will need a fully working smsbox. Add configuration variable `smsbox-id` to `smsbox` group (it is necessary, because there is no MT from any smsbox corresponding the delivery report bearerbox is receiving):

```
group = smsbox
smsbox-id = dlrbox
```

```
bearerbox-host = localhost
log-file = "/var/log/kannel/smsbox-core.log"
log-level = 0
access-log = "/var/log/kannel/smsbox-access.log"
```

Start smsbox normal way after updating the configuration file.

You must add to PPG configuration file two less obvious variables: `ppg-smsbox-id` and `service-name`. `Ppg-smsbox-id` defines smsbox through which you want to route delivery reports, `service-name` makes possible for smsbox to handle wap pushes as a separate sms service.

Setting `ppg-smsbox-id` will route all ppg messages through same smsbox. You can route delivery reports of separate user through separate smsboxes, by using configuration variable `smsbox-id` in group wap push user. Or you can use `X-Kannel-Smsbox-Id` . This means routing every message separately.

You can supply dlr url and dlr mask as kannel header, or as a configuration variable in ppg user or ppg core group. (This is order of precedence, too.) First one is valid for only one message, second for a specific user, and third for every ppg user.

So following is a minimum ppg core group for sms delivery reports:

```
group = ppg
ppg-url = /wappush
ppg-port = 8080
concurrent-pushes = 100
trusted-pi = true
users = 1024
service-name = ppg
ppg-smsbox-id = dlrbox
```

And you can add Kannel headers to http post request. Here an example code snippet (C using Kannel gwlib; this example asks for all possible delivery reports).

```
http_header_add(push_headers, "X-Kannel-SMSC", "link0");
http_header_add(push_headers, "X-Kannel-DLR-Url",
                "http://193.53.0.56:8001/notification-dlr?smsc-id=%i"
                "&status=%d&answer=%A&to=%P&from=%p&ts=%t");
http_header_add(push_headers, "X-Kannel-DLR-Mask",
                octstr_get_cstr(dos = octstr_format("%d", 31)));
http_header_add(push_headers, "X-Kannel-Smsbox-Id", "dlrbox"));
```

Here status=%d tells the type of the delivery report and answer=%A the delivery report itself (sms content of it). Other ones are needed to map delivery report to original wap push.

With these you can use with following http post request

```
http://193.53.0.56:8080/phplib/kannelgw.php?user=*deleted*&
pass=*deleted*=to=%2B358408676001&text=3D%02%06%17%AE%96localhost
%3A8080%00%AF%80%8D%CF%B4%80%02%05j%00E%C6%0C%03wap.iobox.fi%00%11%03
1%40wiral.com%00%07%0A%C3%07%19%99%06%25%15%23%15%10%C3%04+%02%060%01
%03Want+to+test+a+fetch%3F%00%01%01&udh=%06%05%04%0B%84%23%F0
```

Note that you can use all sms service escapes in dlrurl, see Parameters (Escape Codes) for details.

If you want to set dlr url for a specific user, you must set configuration variable `dlr-url`in `wap-push-user`, if for entire ppg, `default-dlr-url` in group `ppg`. Value must naturally match with one used in group smsc.

Routing WAP pushes to a specific smsc

This chapter explains how to route wap push to a specific smsc.

Smsc routing for wap pushes is similar to sms pushes. So, firstly you must define configuration variable `smsc-id` in smsc group (or groups) in question. Say you used value `link0`. You can send this as a Kannel header:

```
http_header_add(push_headers, "X-Kannel-SMSC", "link0");
```

Then you can issue a request:

```
http://193.53.0.56:8080/phplib/kannelgw.php?user=*deleted*&
pass=*deleted*=to=%2B358408676001&text=3D%02%06%17%AE%96localhost
%3A8080%00%AF%80%8D%CF%B4%80%02%05j%00E%C6%0C%03wap.iobox.fi%00%11%03
1%40wiral.com%00%07%0A%C3%07%19%99%06%25%15%23%15%10%C3%04+%02%060%01
%03Want+to+test+a+fetch%3F%00%01%01&udh=%06%05%04%0B%84%23%F0
```

You can use configuration variables to route all messages of a specific user or all ppg messages. Set `default-smsc` in group wap-push-user or in group ppg.

Again precedence of various methods of setting the smsc is kannel header, then configuration variable in group wap push user and then in group ppg.

Chapter 9. Using SSL for HTTP

This chapter explains how you can use SSL to ensure secure HTTP communication on both, client and server side.

Beware that the gateway, is acting in both roles of the HTTP model:

1. as HTTP client, i.e. for requesting URLs while acting as WAP gateway and while fetching information for the SMS services.
2. as HTTP server, i.e. for the administration HTTP interface, the PPG and for the sendsms HTTP interface.

That is why you can specify separate certification files within the core group to be used for the HTTP sides.

You can use one or both sides of the SSL support. There is no mandatory to use both if only one is desired.

Using SSL client support

To use the client support please use the following configuration directive within the core group

```
group = core
...
ssl-client-certkey-file = "filename"
```

Now you are able to use https:// scheme URLs within your WML decks and SMS services.

Using SSL server support for the administration HTTP interface

To use the SSL-enabled HTTP server please use the following configuration directive within the core group

```
group = core
...
admin-port-ssl = true
...
ssl-server-cert-file = "filename"
ssl-server-key-file = "filename"
```

Using SSL server support for the sendsms HTTP interface

To use the SSL-enabled HTTP server please use the following configuration directive within the core and smsbox groups

```
group = core
...
ssl-server-cert-file = "filename"
ssl-server-key-file = "filename"

group = smsbox
...
sendsms-port-ssl = true
```

Using SSL server support for PPG HTTPS interface

If you want use PAP over HTTPS, (it is, a https scheme) add following directives to the ppg core group:

```
group = ppg
...
ppg-ssl-port = 8090
ssl-server-cert-file = "/etc/kannel/cert1.pem"
ssl-server-key-file = "/etc/kannel/key1.pem"
```

PPG uses a separate port for HTTPS traffic, so so you must define it. This means that you can use both HTTP and HTTPS, when needed.

Chapter 10. SMS Delivery Reports

This chapter explains how to set up Kannel to deliver delivery reports.

Delivery reports are a method to tell your system if the message has arrived on the destination phone. There are different things which can happen to a message on the way to the phone which are:

- Message gets rejected by the SMSC (unknown subscriber, invalid destination number etc).
- Message gets accepted by the SMSC but the phone rejects the message.
- Message gets accepted by the SMSC but the phone is off or out of reach. The message gets buffered.
- Message gets successfully delivered.

Note: If you want to use delivery reports, you must define a `smsc-id` for each smsc group.

When you deliver SMS to Kannel you have to indicate what kind of delivery report messages you would like to receive back from the system. The delivery report types currently implemented are:

- 1: delivery success
- 2: delivery failure
- 4: message buffered
- 8: smsc submit
- 16: smsc reject

If you want multiple report types, you simply add the values together. For example if you want to get delivery success and/or failure you set the `dlr-mask` value to 1+2. and so on. If you specify `dlr-mask` on the URL you pass on to Kannel you also need to specify `dlr-url`. `dlr-url` should contain the URL to which Kannel should place a HTTP requests once the delivery report is ready to be delivered back to your system.

An example transaction would work as following.

- 1. you send a message using dlr-mask=7 and dlr-url=http://www.xyz.com/cgi/dlr.php?myId=123456&type=%d
- 2. Kannel forwards the message to the SMSC and keeps track of the message
- 3. The SMSC can not reach the phone and thus returns a buffered message
- 4. Kannel calls http://www.xyz.com/cgi/dlr.php?myId=123456&type=4 to indicate the message being buffered
- 5. The phone is switched on and the SMS gets delivered from the SMSC. The SMSC reports this to Kannel
- 6. Kannel calls http://www.xyz.com/cgi/dlr.php?myId=123456&type=1 to indicate the final success

Note: If you put your own message ID in the dlr-url like in the example above, you can then use this ID to update your database with the message status.

Depending on the SMSC type not all type of messages are supported. For example a CIMD SMSC does not support buffered messages. Also some SMSC drivers have not implemented all DLR types.

Chapter 11. Getting help and reporting bugs

This chapter explains where to find help with problems related to the gateway, and the preferred procedure for reporting bugs and sending corrections to them.

The Kannel development mailing list is devel@kannel.org. To subscribe, send mail to devel-subscribe@kannel.org (mailto:devel-subscribe@kannel.org). This is currently the best location for asking help and reporting bugs. Please include configuration file and version number.

Appendix A. Upgrading notes

This appendix includes pertinent information about required changes on upgrades.

From 1.2.x or 1.3.1 to 1.3.2 and later

- 1. `at` module was dropped and `at2` module is now called `at`
- 2. `emi` module was renamed to `emi_x25`, `emi_ip` sub-module was dropped and `emi2` is now called `emi`.

Appendix B. Certificate generation

This appendix includes pertinent information about required SSL certificate genaration, where needed.

Self-signed 1024-bit RSA SSL certificates using openssl

1. Generate private key:

   ```
   openssl genrsa -des3 -out server.key 1024
   ```

   You will be asked for a passphrase.

2. Generate a certificate request:

   ```
   openssl req -new -key server.key -out server.csr
   ```

   Several questions follow. At the end you may send server.csr to a certificate authority, which in turn will sign it and generate the certificate for you, or you can sign it yourself.

3. Remove passphrase from key:

   ```
   cp server.key server.key.org
   ```

   ```
   openssl rsa -in server.key.org -out server.key
   ```

   ```
   rm server.key.org
   ```

4. Self-sign the certificate:

   If you chose not to send the request to a Certificate Authority, you will need to sign it yourself. This one is good for 1 year:

   ```
   openssl x509 -req -days 365 -in server.csr -signkey server.key -out
   server.crt
   ```

5. Move keys to desired location:

   ```
   mv server.crt /etc/kannel/public/server.crt
   ```

   ```
   mv server.key /etc/kannel/private/server.key
   ```

   ```
   mv server.csr /etc/ianwap/private/ianwap.csr (key request)
   ```

Update configuration accordingly

Appendix C. Using the fake WAP sender

This appendix explains how to use the fake WAP sender to test the gateway.

Appendix D. Using the fake SMS center

Fakesmsc is a simple testing tool to test out Kannel and its SMS services. It *cannot* be used to send messages to mobile terminals, it is just a simulated SMS center with no connection to real terminals.

Setting up fakesmsc

This section sums up needed steps to set up system for fakesmsc use.

Compiling fakesmsc

The fake SMS center should compile at the same time as main Kannel compiles. The outcome binary, `fakesmsc`, is in `test` directory. The source code is quite simple and trivial, and is easily edited.

Configuring Kannel

To use `fakesmsc` to test out Kannel, you have to add it to main configuration file (see above). The simplest form for this configuration group is like this:

```
group = smsc
smsc = fake
port = 10000
```

The fakesmsc configuration group accepts all common 'smsc' configuration group variables, like `smsc-id`, `preferred-smsc-id` or `denied-smsc-id`, which can be used to test out routing systems and diverted services, before setting up real SMS center connections. If you include a fakesmsc group when bearerbox is connected to real SMS centers, you should add the `connect-allow-ip` variable to prevent unauthorized use.

To set up multiple fakesmsc'es, just add new groups. Remember to put a different port number to each one.

Running Kannel with fakesmsc connections

After configuring Kannel, you can start testing it. The bearerbox will listen for fakesmsc client connections to the port(s) specified in the configuration file.

Starting fake SMS center

Each fakesmsc is started from command line, with all sent messages after command name. If any options are used (see below), they are put between the command and the messages. The usage is as follows:

```
test/fakesmsc [options] <message1> [message2 ...]
```

Options and messages are explained below, but as a quick example, a typical startup can go like this:

```
test/fakesmsc -i 0.1 -m 100 "100 200 text nop" "100 300 text echo this"
```

This tells fakesmsc to connect to bearerbox at localhost:10000 (default) and send a hundred messages with an interval of 0.1 seconds. Each message is from number 100, and is either to number 200 with message 'nop' or to 300 with message 'echo this'.

Messages received from bearerbox are shown in the same format (described below).

Fake messages

Each message consists of four or five parts: sender number, receiver number, type, udh (if present) and main message itself. Sender and receiver numbers do not mean anything except for log files and number-based routing in Kannel.

The parts of a message are separated with spaces. As each message is taken as one argument, it must be put in quotation marks.

Message type must be one of the following: "text", "data", "ucs2", "udh-data", "udh-text", "route" and "dlr-mask". Here are some examples:

```
test/fakesmsc -i 0.01 -v 1 -m 1000 "100 300 text echo this message"
test/fakesmsc -i 0.01 -m 1000 "100 300 data echo+these+chars%03%04%7f"
test/fakesmsc -m 1 "100 500 udh-data %0eudh+stuff+here main+message"
```

For "text", the rest of the argument is taken as the literal message. For "data", the next part must be the url-encoded version of the message. Space is coded as '+'. For "ucs2", the next part must be the url-encoded version of the message. For "udh-data", the next 2 parts are the UDH and main message. Both must be in url-encoded form. For "udh-text", the UDH part must be url-encoded, and the message is taken as literals. For "route" both values are literal, where the first value defines the smsbox route destination. For "dlr-mask" noth values are literal, where the first value defines the numerical DLR event.

If multiple messages are given, fakesmsc randomly chooses one for each sending.

Interactive mode

If no messages are passed on the command line when starting fakesmsc, it runs on "interactive mode".

When running on this mode, fakesmsc accept single messages typed on stdin (without quotation marks) and displays the responses on stdout. This allows for a more finely controlled interaction with the backend applications.

Fakesmsc command line options

Fake SMS center can be started with various optional command line arguments.

Table D-1. Fakesmsc command line options

Switch	Value	Description

Switch	Value	Description
`-H`	*host*	Use host *host* instead of default `localhost`.
`-r`	*port*	Use port number *port* instead of default 10000.
`-i`	*interval*	Use message interval *interval* (in seconds, fractions accepted) instead of default interval 1.0 seconds.
`-m`	*max*	Send a maximum of *max* messages. Value -1 means that an unlimited number of messages is sent. Default -1. Using 0 can be useful to listen for messages sent via other channels.

In addition, fakesmsc accepts all common Kannel *Command line options* like `--verbosity`.

Appendix E. Setting up a test environment for Push Proxy Gateway

This appendix explains how to set a test environment for PPG. This contains a simulated SMSC, for instance a http server simulation (this is used as example, because it is simplest) and a simulated push initiator. Between them, there is the push proxy gateway to be tested. This means that you must configure HTTP SMSC.

Creating push content and control document for testing

Here is an example of a push control document, which gives PPG instructions how to do the pushing.

```
<?xml version="1.0"?>
<!DOCTYPE pap PUBLIC "-//WAPFORUM//DTD PAP//EN"
          "http://www.wapforum.org/DTD/pap_1.0.dtd">
<pap>
  <push-message push-id="9fjeo39jf084@pi.com"
                deliver-before-timestamp="2001-09-28T06:45:00Z"
                deliver-after-timestamp="2001-02-28T06:45:00Z"
                progress-notes-requested="false">
    <address address-value="WAPPUSH=+358408676001/TYPE=PLMN@ppg.carrier.com"/>
    <quality-of-service priority="low"
                        delivery-method="unconfirmed"
                        network-required="true"
                        network="GSM"
                        bearer-required="true"
                        bearer="SMS"/>
  </push-message>
</pap>
```

Because the push content is send to the phone over SMS, right value for `network-required` and `bearer-required` is true, for `network` GSM and for `bearer` SMS. However, you can omit these values altogether, if you use a phone number as an address. Address value is international phone number and it must start with plus. It is used here as an unique identifier, SMSC, or sendsms script must transform it to an usable phone number.

Here is an example of Service Indication, a type of push content. Essentially, the phone displays, when it receives this SI, the text "Want to test a fetch" and if the user wants, fetches the content located by URL `http://wap.iobox.fi`.

```
<?xml version="1.0"?>
<!DOCTYPE si PUBLIC "-//WAPFORUM//DTD SI 1.0//EN"
          "http://www.wapforum.org/DTD/si.dtd">
<si>
  <indication href="http://wap.iobox.fi"
              si-id="1@wiral.com"
              action="signal-high"
              created="1999-06-25T15:23:15Z"
```

```
              si-expires="2002-06-30T00:00:00Z">
    Want to test a fetch?
  </indication>
</si>
```

Note that the date value of the si-expires attribute contains trailing zeroes. They are OK here, because SI tokenizer removes them. But phones does not accept them in the final SMS data message. You should probably use `action="signal-high"` for testing purposes, for it causes an immediate presentation of the push message. Production usage is a quite another matter.

Another example of push content is Service Loading. In principle, the phone should fetch immediately content from URL `http://wap.iobox.fi` when it receives this document. This sounds quite insecure, and indeed, user invention is probably required before fetching.

```
<?xml version="1.0"?>
<!DOCTYPE sl PUBLIC "-//WAPFORUM//DTD SL 1.0//EN"
          "http://www.wapforum.org/DTD/sl.dtd">
<sl href="http://wap.iobox.fi"
    action="execute-high">
</sl>
```

Starting necessary programs

PPG test environment contains, in addition of wapbox and bearerbox, two test programs, `test_ppg` (simulating push initiator) and `test_http_server` (simulating a SMSC center accepting pushed content send over SMS. You can find both of these programs in `test` directory, and they both are short and easily editable.

To set up a test environment, you must first configure a push proxy gateway (setting flag trusted-pi true makes testing easier). This explained in Chapter "Setting up push proxy gateway". Then issue following commands, in Kannel's root directory and in separate windows:

```
gw/bearerbox [config-file]
gw/wapbox [config-file]
```

Of course you can use more complicated wapbox and bearerbox command line options, if necessary.

To run a http smsc, start http server simulation:

```
test/test_http_server -p port
```

You can, of course, select the port at will. Remember, though, that PPG listens at the port defined in the ppg configuration file. Other `test_http_server` options are irrelevant here.

Lastly, start making push requests, for instance with a test program `test_ppg`. Its first argument is a URL specifying location of push services. Other arguments are two file names, first one push content and

second one pap control document. (For command line options, see Table C.1.). For example doing one push(you can simplify push url by setting a ppg configuration variable, see "Setting up push proxy gateway"; q flag here prevents dumping of test_ppg program debugging information):

```
test/test_ppg -q http://ppg-host-name:ppg-port/ppg-url [content_file]
[control_file]
```

This presumes that you have set trusted-pi true.

If you want use authentication in a test environment, you can pass username and password either using headers (setting flag -b) or url (you must have set trusted-pi false and added wap-push-user configuration group):

```
test/test_ppg -q http://ppg-host-name:ppg-port?username=ppg-username'&'
password=ppg-password [content_file] [control_file]
```

Table E-1. Test_ppg's command line options

Switch	Value	Description
`-c`	*string*	Use content qualifier *string* instead of default `si` (service indication). Allowed values are wml, si, sl, sia, multipart, nil and scrap. Nil and scrap are used for debugging purposes. Wml does work with some older phone simulators.
`-a`	*string*	Use application id *string* instead of default `any`. Application identifies the application in the phone that should handle the push request. Sia, ua, mms, nil and scrap are accepted. Nil and scrap are used for debugging purposes.
`-e`	*string*	Use transfer encoding when sending a push content. Only base64 is currently supported.
`-b`	*boolean*	Use headers for authentication, instead of url. Default off.
`-i`	*number*	Wait interval *number* instead of default `0` between pushes.
`-r`	*number*	Do *number* requests instead of default `1`.
`-t`	*number*	Use *number* threads instead of default `1`.

Switch	Value	Description
-s	*string*	Use `string` as a X-WAP-Application-Id header (You must supply whole header).
-I	*string*	Use `string` as a X-WAP-Initiator-URI header (You must supply whole header).
-B	*boolean*	If set, accept binary content. Default is off. `1`.
-d	*enumerated string*	Set delimiter to be used. Acceptable values are crlf and lf. Default is `crlf`.
-p	*boolean*	If set, add an preamble (hard-coded one). Default is off.
-E	*boolean*	if set, add an epilogue (hard-coded). Default is off.

Using Nokia Toolkit as a part of a developing environment

This chapter describes a developing environment using Nokia Toolkit instead of `test_http_server` program.

You cannot use a real phone for testing a push server. Sending random messages to a phone does not work, because its only feedback (if it works properly) in error situations is dropping the offending message.

Nokia Toolkit, instead, displays push headers, decompiles tokenized documents and outputs debugging information. It is not, of course, a carbon copy of a real phone. But it is still useful for checking spec conformance of push servers.

Toolkit runs on Windows, the first thing you must is to install a virtual machine (VMWare is one possibility) in the machine where Kannel runs. Then you must configure Toolkit for working with a push gateway.

Then start `bearerbox` and `wapbox` similar way as told before. You must set the correct client address in the push document send by `test_ppg` program. Use IP address of our virtual machine (easiest way to get this is to ping your virtual machine name in the dos prompt window). Your bearer is in this case IP. An example pap document follows:

```
<?xml version="1.0"?>
<!DOCTYPE pap PUBLIC "-//WAPFORUM//DTD PAP//EN"
          "http://www.wapforum.org/DTD/pap_1.0.dtd">
<pap>
  <push-message push-id="9fjeo39jf084@pi.com"
```

```
                deliver-before-timestamp="2001-09-28T06:45:00Z"
                deliver-after-timestamp="2001-02-28T06:45:00Z"
                progress-notes-requested="false">
    <address address-value="WAPPUSH=192.168.214.1/TYPE=IPV4@ppg.carrier.com"/>
    <quality-of-service priority="low"
                        delivery-method="unconfirmed"
    </quality-of-service>
  </push-message>
</pap>
```

Note address-value format. It is contains type and value, because PAP protocol supports different address formats.

You must use `test_ppg`'s -a and -c flags when pushing messages to Toolkit. -A defines the client application handling pushes, right value for it is ua. -C defines the content type of your push message. SI works with all Toolkits, wml only with some older versions.

Testing PAP protocol over HTTPS

When testing HTTPS connection to PPG, you probably want use test_ppg's configuration file, because number of required parameters is quite high. Here is a example test_ppg configuration file:

```
group = test-ppg
retries = 2
pi-ssl = yes
ssl-client-certkey-file = /etc/kannel/certkey.pem

group = configuration
push-url = https://localhost:8900/wappush
pap-file = /etc/kannel/ipnoqos.txt
content-file = /etc/kannel/si.txt
username = foo
password = bar
```

With a configuration file, you can do a push by typing:

```
test/test_ppg -q [configuration_file]
```

Table E-2. Test_ppg's configuration file directives

Directive	Value	Description
`group`	*test_ppg*	Mandatory parameter. Start of test_ppg's core group.
`retries`	*number*	The client tries to log in to PPG *number* times before discarding the push request. Default is 2.

Directive	Value	Description
`pi-ssl`	*boolean*	Mandatory parameter for HTTPS connection. Does the client use HTTPS connection. Default is no.
`ssl-client-certkey-file`	*filename*	Mandatory parameter for HTTPS connection. File containing the client's ssl certificate and private key.
`ssl-trusted-ca-file`	filename	Mandatory parameter for HTTPS connection.This file contains the certificates test_ppg is willing to trust. If this directive is not set, certificates are not validated and HTTPS would not be tested.
`group`	*configuration*	Mandatory parameter. Start of test_ppg's test group.
`push-url`	*url*	Mandatory value. URL locating PPG's services.
`pap-file`	*filename*	Mandatory value. File containing pap request's control document.
`content-file`	*filename*	Mandatory value. File containing pap request's content document.
`username`	*string*	Mandatory value. PPG service user's username.
`password`	*string*	Mandatory value. PPG service user's password.

Appendix F. Setting up a dial-up line

This appendix explains how to set up a dial-up line in Linux for use with the Kannel WAP gateway. In order for it to work you need a Linux kernel with PPP capabilities. Most distributions provides PPP kernel support by default. For more information how to compile PPP support into the kernel please read the "Linux Kernel HOWTO" at http://www.linuxdoc.org/.

Analog modem

This section explains how to set up a dial-up line with an analog modem.

Download and install the mgetty package.

```
rpm -ivh mgetty-VERSION-rpm
```

To run mgetty as a daemon, add the following line to /etc/inittab.

Read man inittab for more detailed information. In this example we assume your modem is connected to the serial port ttyS0 (COM 1).

```
S0:2345:respawn:/sbin/mgetty ttyS0 -x 6 -D /dev/ttyS0
```

We need to start the pppd automatically when mgetty receives an AutoPPP request. Add the next line to /etc/mgetty+sendfax/login.config

```
/AutoPPP/ - - /usr/sbin/pppd file /etc/ppp/options.server
```

In /etc/mgetty+sendfax/mgetty.config you might need to change the connect speed between the computer and the modem. Note: this is not the connect speed between the WAP client and the server modem. If you are e.g. going to use a Nokia 7110 as the server side modem you need to change the speed to 19200. Usually you can just leave the speed to the default value (38400).

```
speed 38400
```

Add the following lines to /etc/ppp/options.server

```
refuse-chap
require-pap
lock
modem
crtscts
passive
192.168.1.10:192.168.1.20
debug
```

In /etc/ppp/pap-secrets add the username and password for the ppp account. The IP address is the one assigned to the phone.

```
wapuser * wappswd 192.168.0.20
```

Configure your phone (this example is for Nokia 7110)

```
homepage http:/yourhost/hello.wml
connection type continuous
connection security off
bearer data
dial up number (your phone number)
ip address (IP of host running bearerbox)
auth type normal
data call type analogue
data call speed 9600
username wapuser
password wappswd
```

ISDN terminal

This section needs to be written

Appendix G. Regular Expressions

Note: this is not intended to be an introduction to regular expressions (short: regex) but a description of their application within Kannel. For general information regarding regexes please refer to [Biblio-regexp].

Syntax and semantics of the regex configuration parameter

This section describes the regex-configuration parameters and their effects in combination with the respective non-regex-parameter, e.g. `white-list` and `white-list-regex`.

How-to setup the regex-parameters

examples, short syntax, what happens on errors Regex-parameters are configured just as every other parameter is configured. Regular expressions are supported as defined by POSIX Extended Regular Expressions. Suppose a configuration where only SMS messages originating from a sender using a number with a prefix of "040", "050", "070" or "090" are accepted. Without regexes the configuration would read

```
allowed-prefix="040;050;070;090"
```

Using regular expressions yields a more concise configuration

```
allowed-prefix-regex=^0[4579]0
```

The following table gives an overview over some regex-operators and their meaning, the POSIX Regular Expressions manual page (regex(7)). Once again, the extended regex-syntax is used and the table is just meant as a means to give a quick-start to regular expressions, the next section features some more complex examples.

Operator	Meaning
`|`	or, for example "dog|hog" matches dog or hog.
`{number,number}`	repetition, for example "a{2,5}" matches - among others - "aa", "aaa" and "baaaaad"
`*`	shorthand for {0,}
`?`	shorthand for {0,1}
`+`	shorthand for {1,}

Operator	Meaning
`[]`	bracket expression, defines a class of possible single character matches. For example "[hb]og" matches "hog" and "bog". If the expression starts with ^ then the class is negated, e.g. "[^hb]og" does not match "hog" and "bog" but matches for example "dog".
`()`	groups patterns, e.g. "[hb]o(glld)" matches "hog", "hold", "bog", "bold"
`[:class:]`	A character class such as digit, space etc. See wctype(3) for details.
`^`	Start of line anchor.
`$`	End of line anchor.

The advantages of regular expressions are at hand

- Regexes are easier to understand, if one is fluent in POSIX Regular Expressions. Although simple expressions as shown above should be clear to everyone who has ever used a standard UN*X shell.
- Regexes are easier to maintain. Suppose the example above needed to cope with dozens of different prefixes each with subtle differences, in such cases using a - carefully constructed - regular expression could help to keep things in apple pie order. Furthermore regexes help reducing redundancy within the configuration.
- Regexes more flexible than standard parameters.

Nevertheless, it must be mentioned that - in addition to the overhead involved - complexity is an issue, too. Although the syntactic correctness of each used regular expression is ensured (see below) the semantic correctness cannot be automatically proofed.

Expressions that are not compilable, which means they are not valid POSIX regexes, force Kannel to panic with a message like (note the missing "]")

```
ERROR: gwlib/regex.c:106: gw_regex_comp_real: regex compilation ‘[hbo(g|ld)’ failed:
PANIC: Could not compile pattern ’[hbo(g|ld)’
```

As shown the erroneous pattern is reported in the error message.

Regex and non-regex-parameters

Using the regex and non-regex version of a parameter at the same time should be done with caution. Both are combined in a boolean-or sense, for example

```
white-list=01234
white-list-regex=^5(23)?$
```

implies that a number is accepted either if it is "01234", "5" or "523" - note the use of anchors! The same goes for all the other parameters, thus both mechanisms can be used without problems in parallel, but care should be taken that the implications are understood and wanted.

Performance issues

While there is some overhead involved, the actual performance degradation is negligible. At startup - e.g. when the configuration files are parsed - the regular expressions are pre-compiled and stored in the pre-compiled fashion, thus future comparisons involve executing the expression on some string only. To be on the sure side, before using regexes extensively some benchmarking should be performed, to verify that the loss of performance is acceptable.

Examples

This section discusses some simple scenarios and potential solutions based on regexes. The examples are not meant to be comprehensive but rather informative.

Example 1: core-configuration

The bearerbox must only accept SMS messages from three costumers. The first costumer uses numbers that always start with "0824" the second one uses numbers that start with either "0123" and end in "314" or start with "0882" and end in "666". The third costumer uses numbers starting with "0167" and ending in a number between "30" and "57".

Important in this and in the following examples is the use of anchors, otherwise a "string contains" semantic instead of a "string is equal" semantic would be used.

```
group=core
...
white-list-regex=^((0824[0-9]+)|(0123[0-9]+314)|(0882[0-9]+666)|(0167[0-9]+
...
```

Example 3: smsc-configuration

Only SMS messages originating from certain SMSCs (`smsc-id` is either "foo", "bar" or "blah") are preferably forwarded to this smsc. Furthermore all SMSCs with an id containing "vodafone" must never be forwarded to the smsc. Not the missing anchors around "vodafone".

```
group=smsc
...
preferred-smsc-id-regex=^(foo|bar|blah)$
denied-prefix-regex=vodafone
...
```

Example 4: sms-service-configuration

Please note that there are a mandatory `keyword` and an optional `keyword-regex` fields. That means that service selection can be simplified as in the following example. Suppose that some Web-content should be delivered to the mobile. Different costumers use the same service but they rely on different keywords. Whenever a sms-service is requested, Kannel first checks whether a regex has been defined, if not a literal match based on `keyword` is performed. If a regex is configured then the literal match is never tried.

```
group=sms-service
...
keyword=web_service
keyword-regex=^(data|www|text|net)$
get-url=http://someserver.net/getContent.jsp
...
```

Appendix H. Meta Data

This appendix describes the elements and syntax required to access meta-data coming from and going to SMSC connections.

Overview

Many SMSC services have means to exchange data that doesn't conform to the established format

For example, the SMPP protocol defines the TLV (Tag, Length, Value) mechanism to allow adding arbitrary parameters to it's PDU's, thus extending it's functionality.

To achieve this, a special group and parameters were added to abstract the meta-data into something that could be used disregarding the SMSC type.

Currently only the SMPP module is implemented, but the format chosen allows for other modules to be added in the future.

SMPP Implementation

A special group type `smpp-tlv` is needed for each TLV. Each group defines the address, data type and length of the TLV

Example configuration:

```
group = smpp-tlv
name = my_tlv_name
tag = 0x1601
type = octetstring
length = 20
smsc-id = a;b;c
```

Variable	Data Type	Description
`tag`	`hex`	32-Bit HEX address for the given TLV. Must match SMPP specifications for the TLV (or be agreed with the SMSC operator).
`type`	`string`	Accepted data type. Accepted values are `integer`, `nulterminated` and `octetstring`. Must match SMPP specifications for the TLV (or be agreed with the SMSC operator).

Variable	Data Type	Description
length	integer	Maximum data length expected in bytes. Must match SMPP specifications for the TLV (or be agreed with the SMSC operator).
smsc-id	string	An optional smsc-id for which this TLV is valid. If smsc-id is not given then this TLV is valid for all SMSCs. To define list of smsc-id, just use ; as split char.
const	string	An optional constant value which is used if no value is set for this TLV via meta-data.

Beside defined in the configuration `smpp-tlv` group, SMPP also supports some hardcoded `meta-data` values:

- data_coding
- source_addr_ton
- source_addr_npi
- dest_addr_ton
- dest_addr_npi

MO Messages

Meta Data coming from MO messages comes urlencoded into the `%D` parameter. Only TLV parameters defined on the configuration file are honored.

Example

```
http://localhost/myscript?...&meta-data=%D&...
```

MT Messages

To send into an MT messages the `meta-data` parameter should be used.

The format used to pass the data has 2 parts: The `?smsc-type?` (surrounded by question marks), which specify the kind of smsc receiving the data (at the moment only `smpp` is implemented) and then a set of key/value pairs with the data to be transmitted. Extra smsc-types can be added surrounded by question marks and followed by the key/value pairs.

In other words, the data should be coded using the following format, where URLENCODE() implies that the data between parentheses should be urlencoded:

```
?URLENCODE(<smsc-type1>)?URLENCODE(key1)=URLENCODE(value1)&URLENCODE(key2)=URLENCODE(value2
```

```
?URLENCODE(<smsc-type2>)?URLENCODE(key1)=URLENCODE(value1)&...
```

Only the parameters defined on the configuration file are honored.

Important: For the sendsms interface, you must URLENCODE the resulting string again to pass it as a single parameter on the URL.

Examples

If we want to send the parameter "my-data" with value "Hello World" to an SMSC over SMPP, we'd use:

sendsms get-url:

```
http://localhost:13013/cgi-bin/sendsms?...&meta-data=%3Fsmpp%3Fmy-data%3DHello%2BWorld
```

sendsms post:

Send `?smpp?my-data=Hello+World` on the `X-Kannel-Meta-Data` header

XML:

```
<meta-data>?smpp?my-data=Hello+World</meta-data>
```

Appendix I. Log files

This appendix describes the log file format.

Bearerbox Access Log

```
2001-01-01 12:00:00 Sent SMS [SMSC:smsc] [SVC:sms] [ACT:account] [BINFO:binfo] [FID:1234567890] [META
          [from:12345] [to:67890] [flags:0:1:0:0:0] [msg:11:Hello World] [udh:0]
```

Variable	Value	Description
Date	2001-01-01 12:00:00	Date
Result	Sent SMS	Result: Send, failed, DLR (deliver report), Received, etc.
SMSC	smsc	Smsc id (`smsc-id`) defined in configuration group `smsc`
SVC	sms	Service name (`name`) defined in configuration group `sendsms-user`
ACT	account	Account name (`name`) defined in configuration group `sendsms-user`
BINF	binfo	Billing information used to send this message.
Foreign ID	1234567890	Foreign (SMSC-provided) Message ID (only available on MT messages)
META	meta-data	Meta-data used to send this message. See Meta Data for an explanation about meta-data.
from	12345	Sender
to	67890	Recipient
Flags	0:1:0:0:0	Flags: MClass, Coding, MWI, Compress, DLRMask
Message Text	11:Hello World	Size of message and message dump (in text or hex if it's binary)
User Data Header	0:	Size of UDH and UDH Hex dump

Log rotation

If Kannel is configured so that the bearerbox, wapbox and/or smsbox log to file each of these log files will continue to grow unless administered in some way (this is specially true if access logs are created and/or the log level is set to debug).

A typical way of administering log files is to 'rotate' the logs on a regular basis using a tool such as logrotate. A sample logrotate script (to be added to /etc/logrotate.d) is shown below. In this example the Kannel log files found in /var/log/kannel are rotated and compressed daily over 365 days. See the documentation for logrotate for more details. Of particular note however is the postrotate command, this killall -HUP issues a SIGHUP signal to each Kannel box process running. The SIGHUP signal has the effect of reopening the log file, and performing graceful restart of SMSC connections. Without this command Kannel will continue to write to the rotated log file. Kannel bearerbox can use the SIGUSR2 to reopen the logs only, without performing the graceful restart. This may be more applicable for logrotate usage.

```
/var/log/kannel/*.log {
      daily
      missingok
      rotate 365
      compress
      delaycompress
      notifempty
      create 640 kannel adm
      sharedscripts
      postrotate
            killall -USR2 bearerbox && killall -HUP smsbox wapbox || true > /dev/null 2> /dev/
      endscript
}
```

Glossary

M

MO

Mobile Originated - a SMS from mobile to application

MT

Mobile Terminated - a SMS from application to mobile

MWI

Message Waiting Indicator (See [Biblio-3GPP-23038])

MClass

Message Class (See [Biblio-3GPP-23038])

Coding

Message Coding (See [Biblio-3GPP-23038])

Bibliography

RFC 2616 - Hypertext Transfer Protocol -- HTTP/1.1, http://www.w3.org/Protocols/rfc2616/rfc2616.html, *Request for Comments: 2616*, The Internet Society, 1999.

3GPP 23.038, http://www.3gpp.org/ftp/Specs/latest/Rel-5/23_series/23038-500.zip, ..., 3GPP, ?.

3GPP 23.040, http://www.3gpp.org/ftp/Specs/latest/Rel-5/23_series/23040-530.zip, ..., 3GPP, ?.

regex(7), GNU regex manual: , GNU, 1998.