Kannel 1.3.1 User's Guide

Open Source WAP and SMS gateway

Lars Wirzenius
Gateway architect
Wapit Ltd

liw@wapit.com
http://www.wapit.com
http://www.kannel.org

Kalle Marjola
Manager
Wapit Ltd

rpr@wapit.com
http://www.wapit.com
http://www.kannel.org

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

CTO & CIO
Wapme Systems AG

tolj@wapme-systems.de
http://www.wapme.de
http://www.kannel.org

Aarno Syvänen

Chief MMS Developer
Global Networks Inc.

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

Kannel 1.3.1 User's Guide: Open Source WAP and SMS gateway

by Lars Wirzenius, Kalle Marjola, Andreas Fink, Bruno Rodrigues, Stipe Tolj, and Aarno Syvänen

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 1.3.1 2006.07.01

Table of Contents

List of Tables

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.

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.

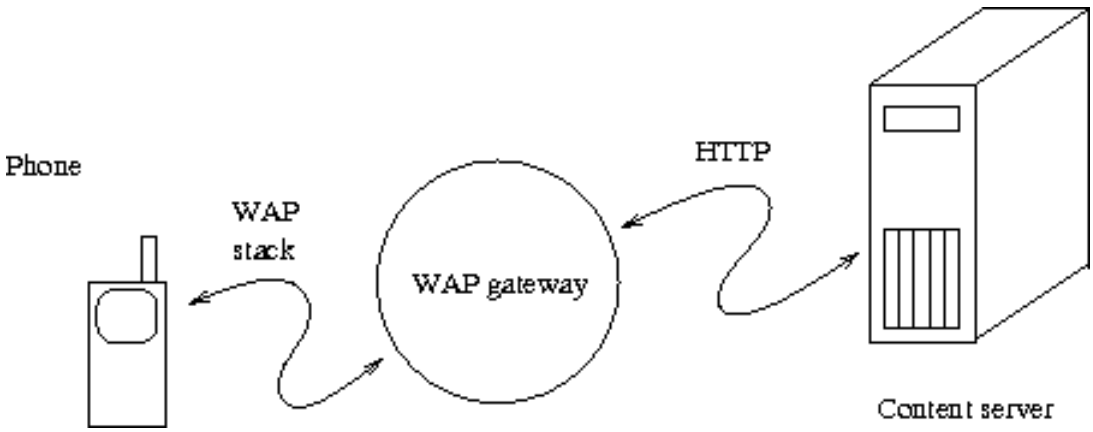
Unlike marketers claim, WAP does not bring the existing content of the Internet directly to the phone. There are too many technical and other problems for this to ever work properly. The main problem is that Internet content is mainly in the form of HTML pages, and they are written in such way that they 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 bytecode format.

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 iniatiates 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 Wapforum 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 becomed 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 sended 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

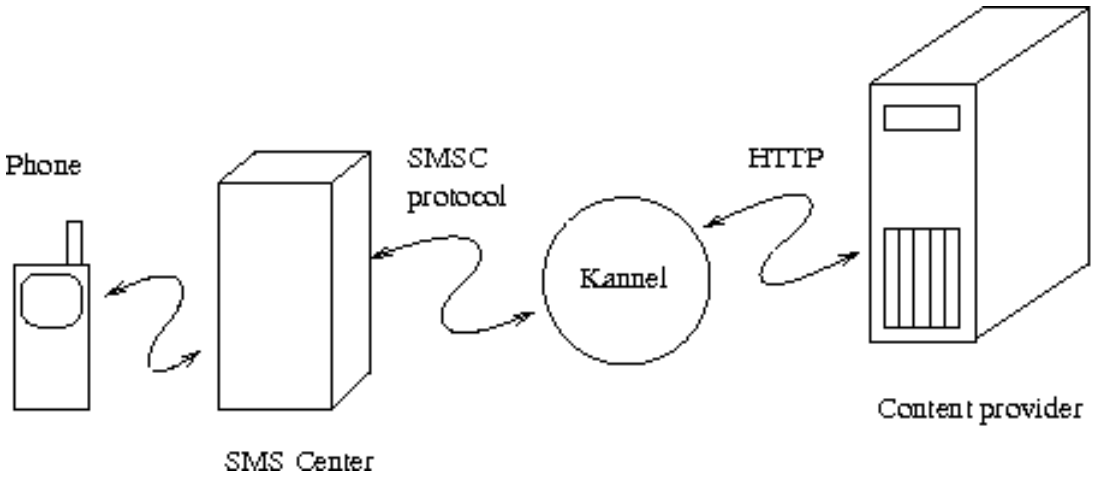
SMS, short messaging service, is a way to send short (160 character) messages from one GSM phone to another. It can also be used to send operator logos, ringing tones, business cards and phone configurations.

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.

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

This section needs to be written.

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. Kannel requires the following software environment:

- C compiler, development libraries and related tools.

- The Gnome XML library (known as gnome-xml and libxml), version 2.2.5 or newer. See http://xmlsoft.org/xml.html.
- GNU Make.
- Posix threads (pthread.h).
- GNU Bison 1.28 if you modify the WMLScript compiler.
- DocBook markup language tools (jade, jadetex, DocBook stylesheets, etc; see README.docbook), if you want to format the documentation (pre-formatted versions are available).

Hardware requirements are fluffier. We haven't benchmarked Kannel yet, so there are no hard numbers, but a reasonably fast PC workstation (400 MHz Pentium II, 128 MB RAM) should serve several concurrent users or tens of SMS messages per second without problems.

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.

Getting the source code

The source code to Kannel is available for download at http://www.kannel.3glab.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.3glab.org/arch.shtml (http://www.kannel.3glab.org/arch.shtml)
3. The `README` and various other text files in the source tree.

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.

You need the following tools to compile Kannel:

- C compiler and libraries for ANSI C, with normal Unix extensions such as BSD sockets.
- 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 stylesheets, jade, jadetex, etc; see `README.docbook` 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.

Compiling the gateway

If you are using Kannel on a supported platform, or one that is similar enough to one, compiling Kannel is trivial. After you have unpacked the source package of your choosing, 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 10 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 `debug`.
- `--enable-docs (default)` Build documentation, b.e., converting the User Guide and the Architecture Guide from the DocBook markup language to PostScript and HTML.
- `--disable-docs` Don't build documentation.
- `--enable-drafts` When building documentation, include the sections marked as `draft`.
- `--disable-drafts (default)` When building documentation, don't include the sections marked as `draft`.
- `--enable-debug` Enable non-reentrant development time debugging of WMLScript compiler.
- `--enable-localtime` Write log file time stamps in local time, not GMT.
- `--disable-assertions` Turn off runtime assertion checking. This makes Kannel faster, but gives less information if it crashes.
- `--with-malloc=`*`type`* Select memory allocation module to use: *`type`* is `native`, `checking` (the default), or `slow`. For production use you probably want `native`. The `slow` module is more thorough than `checking`, but much slower.
- `--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.

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. Log in as root:

```
su -
```

3. Install the RPM package:

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

Upgrading Kannel

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

2. Log in as root

3. Upgrade the RPM package:

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

Removing Kannel

1. Log in as root:

2. Remove the RPM package:

```
rpm -e kannel
```

After you have installed Kannel from the RPM packages you x 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 the same directory there is an example file called smskannel.conf. It has 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 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/share/doc/kannel-VERSION/ or /usr/doc/kannel-VERSION/ depending on if you used the RedHat 7.x or 6.x 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.

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

```
dpkg -l libxml2
```

Installing or upgrading Kannel using APT

1. Log in as root:

```
su -
```

3. Install or upgrade the package:

```
apt-get install kannel
```

See http://kannel.org/download.shtml#debian_repository for informations 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 runing 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.

Aditionally 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 combined gateway. These chapters explain the configuration and other aspects of gateway of that type.

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 whitespace 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 whitespace 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'.

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 www server, just invent any port, but it must be over 1023 unless you are running Kannel as a root process (not recommended)
`admin-port-ssl (o)`	bool	If set to true a SSL-enabled administration HTTP server will be used instead of the default unsecure plain HTTP server. To access the administration pacges 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

Variable	Value	Description
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 wildcard in a place of any ONE number, so *.*.*.* matches any IP.
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.
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 seperate 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 seperate networks operated and the TCP communication is not secured on a lower network layer. Defaults to "no".

Variable	Value	Description
`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 wildcard in place of any ONE number, so *.*.*.* matches any IP.
`udp-deny-ip` `udp-allow-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`.
`wdp-interface-name (c)`	IP or '*'	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.
`log-file`	filename	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-level`	number 0..5	Minimum level of logfile events logged. 0 is for 'debug', 1 'info', 2 'warning, 3 'error' and 4 'panic' (see Command Line Options)
`access-log`	filename	A file in which information about received/sent SMS messages is stored. Access-log in 'core' group is only used by the bearerbox.

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".*
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 SMS Center is discarded. 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!
black-list	URL	As white-list, but SMS messages to these numbers are automatically discarded

Variable	Value	Description
`store-file`	filename	A file in which any received SMS messages are stored until they are successfully handled. By using this variable, no SMS messages are lost in Kannel, but theoretically some messages can duplicate when system is taken down violently.
`http-proxy-host`	hostname	Enable the use of an HTTP proxy for all HTTP requests.
`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-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-enabed 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.

Variable	Value	Description
`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.
`dlr-storage`	type	Defines the way DLRs are stored. 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, mysql. By default this is set to 'internal'.
`maximum-queue-length`	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).

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 [conffile]
```

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.

[conffile] 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` subdirectory), 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 [conffile]
```

or the SMS box:

```
./smsbox -v 1 [conffile]
```

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

`-v <level>`	Set verbosity level for stdout (screen) logging. Default is 0, which means 'debug'. 1 is 'info, 2 'warning', 3 'error' and 4 'panic'

`--verbosity <level>`	
`-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 logfile defined in configuration file.
`-V <level>` `--fileverbosity <level>`	Set verbosity level for that extra logfile (default 0, which means 'debug'). Does not affect verbosity level of the logfile 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)

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 queueing 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-id` should be re-started.
`stop-smsc`	Shutdown a single SMSC link. Password required. Additionally the `smsc` parameter must be given (see above).

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 configuration

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

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	
`syslog-level`	number	Messages of this log level or higher will also be sent to syslog, the UNIX system log daemon. The wapbox logs under the 'daemon' category. The default is not to use syslog, and you can set that explicitly by setting `syslog-level` to 'none'.
`force-sar`	bool	If set wapbox will force to process WTP-SAR connections even while Kannel does not support this feature now. Some real phones seem to break connection if fallback to non SAR communication is being tried by the gateway.
`smart-errorsr`	bool	If set wapbox will return a valid WML deck describing the eror that occured while processing an WSP request. This may be used to have a smarter gateway and let the user know what happend actually.

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 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 5-1. SMSC Group Variables

Variable	Value	Description

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.
`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 (';')

Variable	Value	Description
`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 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 through this SMSC.
`preferred-prefix`	`prefix-list`	As `denied-prefix`, but SMS messages with receiver starting with any of these prefixes is 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
		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	
		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 tyles. Please see your SMSC module type you want to use for more details.
`alt-charset`	`number`	

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
sender-prefix = "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
`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 minutes. The right value to use depends on how eager the SMSC is to close idle connections. 5 minutes is a good guess. If you see many unexplained reconnects, try lowering this value. Set it to 0 to disable this feature.

Variable	Value	Description
sender-prefix	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 `sender-prefix` 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 `""`.

CMG UCP/EMI 4.0

Kannel supports two types of connections with CMG SMS centers: direct TCP/IP connections (`emi_ip` or `emi2`) and ISDN/modem (X.25 over D channel ISDN is called X.31) connection (`emi`). `emi2` is a new implementation of the EMI protocol that supports more features and should work more reliably than the old one. It is the recommended one to use with TCP/IP connections. Sample configurations for these are:

```
group = smsc
smsc = emi2
#smsc = emi_ip to use the old implementation
host = 103.102.101.100
port = 600
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
host = 100.102.100.102
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)
`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-host`	`hostname`	Optional hostname in which to bind the connection in our end. TCP/IP connection only.
`our-port`	`port-number`	Optional port number in which to bind the connection in our end. 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.

Variable	Value	Description
connect-allow-ip	IP-list	If set, only connections from these IP addresses are accepted to receive-port. TCP/IP connection only.
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.

Variable	Value	Description
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, requeue, 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 mst accurate)
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)
throughput	number (messages/sec)	If SMSC requires that kannel limits the number of messages per second, use this variable. (optional)
retry	boolean	Assuming that kannel is well configured and we had one sucessful connection, if retry is true, kannel will always retry the connection even if some related error ocur.

Variable	Value	Description
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.
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.

SMPP 3.4

This implements Short Message Peer to Peer (SMPP) Protocol 3.4 in a manner that should also be compatible with 3.3. Sample configuration:

```
group = smsc
smsc = smpp
host = 123.123.123.123
port = 600
receive-port = 700
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. May be the same as receive-port. Use value 0 to disable this I/O thread.

Variable	Value	Description
transceiver-mode	bool	Attempt to use a TRANSCEIVER mode connection to the SM-SC. It uses the standard transmit 'port', there is no need to set 'receive-port'. This is a SMPP 3.4 only feature and will not work on an earlier SM-SC. This will try 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. May be the same as port. Use value 0 to disable this I/O thread.
smsc-username (m)	string	The 'username' of the Messaging Entity connecting to the SM-SC. If the SM-SC 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.
smsc-password (m)	string	The password matching the "smsc-username" your teleoperator provided you with.
system-type (m)	string	Usually you can get away with "VMA" which stands for Voice Mail Activation.

Variable	Value	Description
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".
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.
our-host	string	Specicy the outgoing IP address for connections from a multi-homed machine. If this is not defined the default device of the machine will be used.
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 explicity in the SMPP Protocol Specification and will be goverened 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.

Variable	Value	Description
reconnect-delay	number	Optional the time between attemps to connect an ESME to an SMSC having failed to connect initating or during an SMPP session. The default is 10 seconds.
source-addr-ton	number	Optional, source address TON setting for the link. (Defaults to 0).
source-addr-npi	number	Optional, source address NPI setting for the link. (Defaults to 1).
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 0).
dest-addr-npi	number	Optional, destination address NPI setting for the link. (Defaults to 1).

Variable	Value	Description
		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.
`msg-id-type`	`number`	
`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.

Sema Group SMS2000 OIS 4.0 and 5.0

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.

```
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
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

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 attemps to connect to an SMSC having failed to connect initating 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 highest priority)

GSM modem

Kannel can use a GSM modem as an SMS center.

```
group = smsc
smsc = at
modemtype = wavecom
device = /dev/ttyS0
```

```
pin = 2345
```

Variable	Value	Description
modemtype	string	Modems from different manufacturers have slightly different behaviour. We need to know what type of modem is used.
device (m)	device-name	The device the modem is connected to, like /dev/ttyS0.
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.
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.
alt-dcs	boolean	When encoding DCS field internally, there are two formats with similar functionality. The 0x0X (alt-dcs = false or non-present) or the 0xFX (alt-dcs = true). If you have a buggy modem (like Siemens M20) that don't like to send binary messages, try setting alt-dcs to true.

Modem Type	Modems
wavecom	Wavecom
premicell	Nokia Premicell
siemens	Siemens M20 (this modem have some bugs)
siemens-tc35	Siemens TC35
falcom	Falcom

Modem Type	Modems
nokiaphone	Nokia 6210, 7110, 8210 (tested). Probably other Nokia phones too.
ericsson	Ericsson

GSM modem 2

This new driver is replacing the old GSM Modem driver from Kannel. It allows a GSM Modem or Phone to be connected to Kannel and work as a virtual SMSC

```
group = smsc
smsc = at2
modemtype = auto
device = /dev/ttyS0
speed = 9600
pin = 2345
```

Variable	Value	Description
modemtype	string	Modems from different manufacturers have slightly different behaviour. 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 autodetected like the Nokia Premicell).
device (m)	device-name	The device the modem is connected to, like /dev/ttyS0.
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.
retry	boolean	Assuming that kannel is well configured and we had one sucessful connection, if retry is true, kannel will always retry the connection even if some related error ocur.
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 behaviour" 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 behaviour, 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 behaviour is known to cause minor or major hicups for a few buggy modems. Modems known to work with this setting are Wavecom WM02/M1200 and the Siemens M20.

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
`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-strign2` "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"
`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)

Variable	Value	Description
sendline-sleep	number (miliseconds)	Optional, defaults to 100 miliseconds. The sleep time after sending a AT command.
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 batery, 64% full) and "+CSQ: 14,99" (0-31, 0-7: signal strenght 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 suppoerted 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").
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 quickier sending the messages.

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 unsolicted 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.

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. 'kannel' is only system currently supported.
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 hexdump. For instance, text=%2b123 is represented as text=2b313233.

Variable	Value	Description
port (m)	port-number	Port number in which Kannel listens to (MO) messages from other gateway
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.

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 5-2. SMSC driver features

Feature	cimd	cimd2	emi	emi_ip	emi2	smpp	sema	ois	at	at2	http	fake
Can use DLR	n	y?	n	n	y	y?	n	n	n	n	n	n
Can set DCS[a]	?	?	?	?	y	?	?	?	?	y	?	?
Can set Alt-DCS												

Feature	cimd	cimd2	emi	emi_ip	emi2	smpp	sema	ois	at	at2	http	fake
	n	n	n	n	y	n	n	n	n	y	n	n
Can set Validity												
	?	?	?	?	y	?	?	?	?	y	?	?
Can set Deferred												
	?	?	?	?	y	?	?	?	?	n	?	?
Can set PID												
	n	n	n	n	y	y	n	n	n	y	n	n
Can set RPI												
	n	n	n	n	y	y	n	n	n	n	n	n
Can send Unicode												
	?	?	?	?	y	?	?	?	?	y	?	?
Can send 8 bits												
	?	?	?	?	y	?	?	?	?	y	?	?
Correctly send GSM alphabet												
	?	?	?	?	y	?	?	?	?	?	?	?

Notes:
a. To use `mclass`, `mwi`, `coding` and `compress` fields.

Table 5-3. SMSC driver internal features

Feature	cimd	cimd2	emi	emi_ip	emi2	smpp	sema	ois	at2	at	http	fake
Can keep idle connections alive												
	n	y?	n	n	y	y?	?	?	y	?	?	?
Can send octet data without UDH												
	n	y?	y?	y?	y	n	n	y?	y?	?	n	y?[a]
Can send octet data with UDH												
	N	y?	y?	y?	y	y?	n	?	y?	?	y?	y?[a]
Can send text messages with UDH												
	n	y?	y?	y?	y	n	n	?	y?	?	n	y?
Can receive octet data without UDH												
	n	y?	n	n	y	n	y?[b]	y?	y?	?	n	n
Can receive unicode messages												
	n	n	n	n	n	n	n	n	n	?	n	n
Can receive octet data with UDH												
	n	y?	n	n	y	n	n	N	y?	?	y?	y?
Can receive text messages with UDH												
	n	y?	n	n	y	n	n	N	y?	?	n	n

Feature	cimd	emi	emi2	sema	ois	at2	at	http	fake
	cimd2	emi_ip	smpp						
Correctly encodes `@` when sending	y?	y?	?	?	y	y?	?	y?	y?	?	y?	y?
Correctly encodes `ä` when sending	y?	y?	?	?	y	y?	?	y?	y?	?	y?	y?
Correctly encodes `{` when sending	n	y?	?	?	y	y?	?	n	N[c]	?	y?	y?
Can receive `@` in text messages	y?	y?	?	?	y	y?	?	y?	y?	?	y?	y?
Can receive `ä` in text messages	y?	y?	?	?	y	y?	?	y?	y?	?	y?	y?
Can receive `{` in text messages	n	y?	?	?	y	y?	?	n	y?	?	y?	y?
Can shut down idle connections	n	n	n	n	y	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

Smsbox configuration

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
ssl-certkey-file
```

Table 5-4. 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.
`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-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`.
`sendota-url (o)`	url	URL locating the sendota service. Defaults to `/cgi-bin/sendota`.

Variable	Value	Description
`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	
`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.`.

Variable	Value	Description
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>.
mo-recode	boolean	If true, kannel will try to convert UCS2 messages received to ISO-8859-1. If it's possible, the message will have coding equal to 7 bits and charset equal to iso-8859-1.
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 queueing is done.
http-queue-delay	integer	If set, specifies how many seconds should pass within the HTTP queueing 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.

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 determinated 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 our 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 proxys that can be written easly using smsbox routing facility.

If you smppbox handles the SMPP specific communication to your EMSEs, and if an client send a submit_sm PDU, smppbox 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 shortcuts for certain networks or route any inbound traffic from a certain smsc link connected to bearerbox, you need to seperate 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
shortcuts = "1111;2222;3333"
```

which means and 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

smsbox-route inherits from core the following fields:

Table 5-5. 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-ids	word-list	If set, specifies from which smsc-ids all inbound messages should be routed to this smsbox instance. List contains smsc-ids seperated by semilon (";"). This rule may be used to pull any smsc specific message stream to an smsbox instance.
shortcuts	number-list	If set, specifies which receiver numbers for inbound messages should be routed to this smsbox instance. List contains numbers seperated by semilon (";"). This rule may be used to pull receiver number specific message streams to an smsbox instance.

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 5-6. SMS-Service Group Variables

Variable	Value	Description
group (m)	sms-service	This is a mandatory variable

Variable	Value	Description
`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.
`aliases`	word-list	If the service has aliases, they are listed as a list with each entry separated with a semicolon (';')
`name`	string	Optional name to identify the service in logs. If unset, `keyword` is used.
`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 otherwhere.
`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 otherwhere 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.

Variable	Value	Description
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.
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'
allowed-prefix	prefix-list	A list of phone number prefixes of the sender number which are accepted to be received by this service. [b] 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.

Variable	Value	Description
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.
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*.

Variable	Value	Description
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.
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.

Variable	Value	Description
suffix	string	
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

Notes:

a. Even if this service is denied, kannel still searches for other service which accepts the message, or default service.

b. 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.

Table 5-7. 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
%t	the time the message was sent, formatted as "YYYY-MM-DD HH:MM", e.g., "1999-09-21 14:18"
%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
%d	the delivery report value
%A	the delivery report SMSC reply, if any
%n	the sendsms-user or sms-service name
%c	message coding: 0 (default, 7 bits), 1 (7 bits), 2 (8 bits) or 3 (unicode)
%C	message charset: for a "normal" message, it will be "gsm" (coding=1), "binary" (coding=2) or "UTF16-BE" (coding=3). If the message was sucessfully recoded from unicode, it will be "ISO-8859-1"
%u	udh of incoming message

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.

Table 5-8. 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
coding	X-Kannel-Coding. If unset, defaults to 1 (7 bits) if Content-Type is text/plain , text/html or text/vnd.wap.wml. On application/octet-stream, defaults to 8 bits (2). All other Content-Type values are rejected.
validity	X-Kannel-Validity
deferred	X-Kannel-Deferred
dlrmask	X-Kannel-DLR-Mask
dlrurl	X-Kannel-DLR-Url
account	X-Kannel-Account
pid	X-Kannel-PID
alt-dcs	X-Kannel-Alt-DCS

Kannel POST

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

Table 5-9. 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`	1=7 Bits, 2=8 Bit
`- (compress)`	`X-Kannel-Compress`	
`- (validity)`	`X-Kannel-Validity`	
`- (deferred)`	`X-Kannel-Deferred`	
`%n (service name)`	`X-Kannel-Service`	
`%a or %r (text)`	request body	kannel send all w unless `strip-ke`
`%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)</text>
    <udh>user data header (udh)</udh>
    <dcs>
      <mclass>mclass</mclass>
      <coding>coding</coding>
      <mwi>mwi</mwi>
      <compress>compress</compress>
```

```
      <alt-dcs>alt-dcs</alt-dcs>
    </dcs>
    <pid>pid</pid>
    <statusrequest>
      <dlr-mask>dlr-mask</dlr-mask>
      <dlr-url>dlr-url</dlr-url>
    </statusrequest>
    <from>
      <user>username</user>
      <username>username</username>
      <pass>password</pass>
      <password>password</password>
      <account>account</account>
    </from>
    <to>smsc-id</to>
    <from>smsc-id</from>
    <to>service-name</to>
  </submit>
</message>
```

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

`ud`

Note: Davi: I still have to test binary and unicode <ud> content

`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 5-10. 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.
`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.
`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	

Variable	Value	Description
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.
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	URL to be fetched if a `dlrmask` CGI parameter is present.

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.

External delivery report (DLR) storage

Delivery reports are supported by default internaly, 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
mysql-username = foo
mysql-password = bar
database = dlr

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

LibSDB DLR storage

To store DLR information into a LibSDB ressource (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 ressource itself.

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

```
group = sdb-connection
id = mydlr
url = "mysql:host=localhost:db=dlr:uid=foo:pwd=bar"

group = dlr-db
id = mydlr
table = dlr
field-smsc = smsc
field-timestamp = ts
field-destination = destination
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.

DLR database field configuration

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

Table 5-11. DLR Database Field Configuration Group Variables

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

Variable	Value	Description
`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-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 garantee 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-destination = destination
field-service = service
field-url = url
field-mask = mask
field-status = status
```

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.

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 5-12. 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.
`host (m)`	hostname or IP	Hostname or IP of a server running a MySQL database to connect to.
`mysql-username (m)`	username	User name for connecting to MySQL database.
`mysql-password (m)`	password	Password for connecting to MySQL database.
`database (m)`	string	Name of database in MySQL database server to connect to.

A sample 'mysql-connection' group:

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

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 seperate `mysql.conf` file.

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 continous 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 5-13. 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.
`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 wether 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 5-14. OTA Bookmark Group Variables

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

Table 5-15. 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 UCS2. Defaults to ISO-8859-1 if coding is 7bits and UTF16BE if coding is UCS2.
`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 1 and 4, for Message Class 0 to 3, A value of 1 sends the message directly to display. mclass=2 sends to mobile, 3 do SIM and 4 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 1,2,3 and 4 for activating the voice, fax, email and other indicator, or 5,6,7,8 for deactivating, respectivly. This option excludes the `flash` option. [a]
`coding`	`number`	Optional. Sets the coding scheme bits in DCS field. Accepts values 1 to 3, for 7bit, 8bit or UCS2. 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 syncronized 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 syncronized with the SMS Center.
dlrmask	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 `dlrurl` CGI variable.
dlrurl	string (url)	Optional. If `dlrmask` is given, this is the url to be fetched. (Must be urlencoded)
pid	byte	Optional. Sets the PID value. (See ETSI Documentation). Ex: SIM Toolkit messages would use something like `&pid=127&coding=2&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 2, force 0X.
rpi	number	Optional. Sets the Return Path Indicator (RPI) value. (See ETSI Documentation).

`account`	string	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 servicename (username) and a colon (:) and forwarded to the next insta ce of kannel. This allows hierarchical accounting.

Notes:

a. To set number of messages, use `mwi=[1-4]&coding=1&udh=%04%01%02%<XX>%<YY>`, where YY are the number of messages, in HEX, and XX are `mwi-1` 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.

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 or bookmark, set CGI variable type accordingly. Default for this variable is settings.

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`.

Table 5-16. 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 sended 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, either "settings" or "bookmarks". Default is "settings".

Chapter 6. Setting up a SMS&WAP gateway

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

SMS&WAP gateway configuration

Configuration is done as explained in previous chapters, you simply have to include all the data from both chapters into the configuration file.

Running SMS&WAP gateway

There are no special tricks to this, just launch both the smsbox and the wapbox in addition to the bearerbox, using multiple hosts if needed.

Chapter 7. 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.

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 7-1. 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. Wildcards 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. Wildcards 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.

Configuring PPG user group variables

In addition of pi lists similar to the core group, ppg configuration spesific 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 7-2. PPG user group configuration variables

Variable	Value	Description
`group`	*wap-push-user*	Mandatory value. Tells that we are configuring the users group.
`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 wherefrom the whitelist can be fetched. White list itself contains list of phone numbers accepting pushes from this pi.
`black-list`	*url*	Defines an url wherefrom the blacklist can be fetched. Blacklist itself contains list of phone number not accepting pushes from this pi.
`allow-ip`	*ip-list*	Defines ips wherefrom this pi can do pushes. Adding this list means that ips not mentioned are denied.

Variable	Value	Description
deny-ip	*ip-list*	Defines ips wherefrom 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.

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 mut 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 group will eventually disappear, use here only necessary configuration variables.

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

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 [conffile]
gw/wapbox [conffile]
```

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 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 push (tokenised SI) document

HTTP SMSC generates a HTTP get request when it get a sendmessage 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>
```

Chapter 8. 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 seperate 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 = "filenane"
```

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 = "filenane"

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 = "/home/aarno/kannelcvs/gateway/gw/cert1.pem"
ssl-server-key-file = "/home/aarno/kannelcvs/gateway/gw/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 9. 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.

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 currrently 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 togeter. For example if you want to get delivery success and/or failure you set the dlrmask value to 1+2. and so on. If you specify dlrmask on the URL you pass on to kannel you also need to specify dlrurl. dlrurlshould 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 dlrmaks=7 and dlrurl=www.xyz.com/cgi/dlr.php?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?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
- 4. Kannel calls http://www.xyz.com/cgi/dlr.php?type=2 to indicate the final success

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 10. 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.3glab.org. To subscribe, send mail to devel-subscribe@kannel.3glab.org (mailto:devel-subscribe@kannel.3glab.org). This is currently the best location for asking help and reporting bugs. Please include configuration file and version number.

Appendix A. Using the fake WAP sender

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

Appendix B. 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 outcoming 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" and "udh". Here's an example of using each:

```
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 %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 urlcoded version of the message. Space is coded as '+'. For "udh", the next 2 parts are the UDH and main message. Both must be in urlcoded form.

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

Fakesmsc command line options

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

Table B-1. Fakesmsc command line options

Switch	Value	Description
`-H`	*host*	Use host *host* instead of default `localhost`.
`-p`	*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.

Switch	Value	Description
`-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 C. 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 sended to the phone over SMS, rigth value for `network-required` and `bearer-required` is true, for `network` GSM and for `bearer` SMS. However, you can omit these values alltogether, 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 unsecure, 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 sended 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 [conffile]
gw/wapbox [conffile]
```

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 C-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 tranfer 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`.

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 tokenised documents and outputs debugging information. It is not, of course, a carbon copy of a real phone. But it is still usefull 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 sended 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 = /home/aarno/kannelcvs/gateway/gw/certkey.pem

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

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

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

Table C-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.
`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 paramenter 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.

Directive	Value	Description
`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 D. 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 E. Log files

This appendix describes the log file format.

Bearerbox Access Log

```
2001-01-01 12:00:00 Sent SMS [SMSC:smsc] [SVC:sms] [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`
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 especially 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 HUP command to each kannel box running. The HUP signal has the effect of reopening the log file, without this command Kannel will continue to write to the rotated log file.

```
/var/log/kannel/*.log {
```

```
        daily
        missingok
        rotate 365
        compress
        delaycompress
        notifempty
        create 640 kannel adm
        sharedscripts
        postrotate
              killall -HUP bearerbox smsbox wapbox || true > /dev/null 2> /dev/null
        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, ?.