

# **SuiteSMPP**

Version 1.0.2

---

## **Developer's Guide**

---

This guide contains information on how to use SuiteSMPP from a developer's perspective. It includes information, examples and guides which will hopefully assist developers achieve their goals.

## Table of Contents

---

1. Overview of SuiteSMPP .....	3
1.1. Features .....	3
1.2. Supported Platforms .....	3
1.3. Supported Programming Languages .....	3
1.4. SMPP protocol compliance .....	3
1.5. Robustness .....	4
2. Introduction for Developers .....	4
2.1. Things to have Handy .....	4
2.2. SMPP Types .....	5
2.3. Namespaces and Package layout .....	6
2.4. Implementation of PDUs in SuiteSMPP .....	6
2.5. General PDU methods .....	7
2.6. Exceptions .....	7
2.7. Extra Octets in PDUs .....	8
2.8. Unknown TLVs .....	8
2.9. Using the ByteBuffer .....	8
3. Quick Start .....	9
3.1. Basic usage steps .....	9
3.2. Code examples .....	9
4. Feedback .....	11
4.1. Bug Reports .....	11
4.2. Suggestions .....	11
4.3. Support .....	11

# Overview of SuiteSMPP

---

## Features

- Full SMPP 3.4 and 5.0 protocol compliance.
- Support for platforms which do not adhere completely to the specification or have customized PDUs.
- Fast and efficient.
- Support for multiple programming languages and platforms.
- Tried and tested in the field. We use it in our core products which process millions of SMSes every day.

## Supported Platforms

**C++.** Anything with a standard GNU or GNU-compatible toolchain (binutils, coreutils and g++ or compatible compiler). The library builds cleanly on Linux and Solaris. Other platforms are untested but should build without any issues. The library has been built against SUSv2 so any platform which is a variant of UNIX should work. Please let us know if you have success on your platform.

**Java.** A standard Java Virtual Machine supporting Java 5+ is required.

## Supported Programming Languages

- C++
- Java 5+

## SMPP protocol compliance

In summary of the below table of SMPP PDUs, "complete compliance". All PDUs and each parameter of the PDUs are supported. If you find anything missing, let us know.

Name	3.4	5.0
BindTransmitter	Yes	Yes
BindTransmitterResp	Yes	Yes
BindReceiver	Yes	Yes
BindReceiverResp	Yes	Yes
BindTransceiver	Yes	Yes
BindTransceiverResp	Yes	Yes
Outbind	Yes	Yes
Unbind	Yes	Yes
UnbindResp	Yes	Yes
EnquireLink	Yes	Yes
EnquireLinkResp	Yes	Yes
AlertNotification	Yes	Yes
GenericNack	Yes	Yes
SubmitSm	Yes	Yes

Name	3.4	5.0
SubmitSmResp	Yes	Yes
DataSm	Yes	Yes
DataSmResp	Yes	Yes
SubmitMultiSm	Yes	Yes
SubmitMultiSmResp	Yes	Yes
DeliverSm	Yes	Yes
DeliverSmResp	Yes	Yes
CancelSm	Yes	Yes
CancelSmResp	Yes	Yes
QuerySm	Yes	Yes
QuerySmResp	Yes	Yes
ReplaceSm	Yes	Yes
ReplaceSmResp	Yes	Yes
BroadcastSm		Yes
BroadcastSmResp		Yes
QueryBroadcastSm		Yes
QueryBroadcastSmResp		Yes
CancelBroadcastSm		Yes
CancelBroadcastSmResp		Yes

## Robustness

Through our experience in the field, we have discovered certain SMSCs which deviate slightly from the standard in ways which break the protocol specification. SuiteSMPP caters for these deviations with:

**Support for additional octets padded to each PDU.** Some systems send through valid PDUs with rubbish octets appended to the PDU, even though the PDU length calculation is accurate and includes these bytes. SuiteSMPP receives these octets and makes them available to the developer, should the developer wish to use them. The developer may also append random octets to the PDU using SuiteSMPP should he or she wish to do so.

**Support for unknown TLV values.** The SMPP protocol specification has a collection of TLV values which are described in the spec. Some SMSCs have customized TLV values appended in each PDU which are not a part of the spec. SuiteSMPP receives these unknown TLVs and make them available to the developer, should the developer wish to use them. (See unknown TLV method calls) The developer may also append their own TLV values to a PDU using SuiteSMPP should he or she wish to do so.

## Introduction for Developers

---

### Things to have Handy

- The SMPP protocol specifications. Available here [<http://www.smsforum.net/>].
- A network sniffing tool. We recommend Wireshark [<http://www.wireshark.org/>].
- A simple SMPP server simulator for testing. You can use one of ours which is available from our website [<http://www.smksoftware.com/>].

## SMPP Types

The SMPP protocol has 4 types and they are described below, taken directly from the 5.0 protocol specification:

Integer	<p>An unsigned integer value, which can be 1, 2 or 4 octets in size. The octets are always encoded in Most Significant Byte (MSB) first order, otherwise known as Big Endian Encoding.</p> <p>A 1-octet Integer with a value 5, would be encoded in a single octet with the value 0x05</p> <p>A 2-octet integer with the decimal value of 41746 would be encoded as 2 octets with the value 0xA312</p> <p>A 4-octet integer with the decimal value of 31022623 would be encoded as 4 octets with the value 0x1D95E1F</p>
C-Octet String	<p>A C-Octet String is a sequence of ASCII characters terminated with a NULL octet (0x00). The string “Hello” would be encoded in 6 octets (5 characters of “Hello” and NULL octet) as follows: 0x48656C6C6F00</p> <p>Two special variants exist for use within SMPP. These are C-octet String (Decimal) and C-Octet String (Hexadecimal), which are used to carry decimal and hexadecimal digit sequences respectively. These fields are encoded the same way as any ASCII string, but are specifically used to designate decimal and hexadecimal numbers when presented in string format.</p> <p>A Decimal C-Octet String “123456789” would be encoded as follows: 0x31323334353637383900</p> <p>A Hexadecimal C-Octet String “A2F5ED278FC” would be encoded as follows: 0x413246354544323738464300</p>
Octet String	<p>An Octet String is a sequence of octets not necessarily terminated with a NULL octet. Such fields using Octet String encoding, typically represent fields that can be used to encode raw binary data. In all circumstances, the field will be either a fixed length field or explicit length field where another field indicates the length of the Octet String field. An example of this is the short_message field of the submit_sm PDU that is Octet String encoded and the previous message_length field specifies its length.</p>
Tagged Length Value (TLV)	<p>A Tagged Length Value Field is a special composite field that comprises of three parts:</p> <ul style="list-style-type: none"><li>• A 2-octet Integer (Tag) The tag identifies the parameter.</li><li>• A 2-octet Integer (Length) The length field indicates the length of the value field in octets. Note that this length does not include the length of the tag and length fields.</li><li>• An Octet String (Value)</li></ul> <p>The value field contains the actual data for the TLV field. The Tag identifies the parameter. The Length indicates the size of the Value field in octets.</p>

An example of a TLV is the `dest_bearer_type`. Its Tag is 0x0007 and has a value size of 1 octet. The value 0x04 indicates USSD as a bearer type. In its encoded form, this TLV would appear as follows: 0x0007000104

The first 2 octets 0x0007 identifies the Tag `dest_bearer_type`. The next two octets 0x0001 indicate the 1-octet length of the value field. The value field 0x04 indicates USSD ref. 4.8.4.64

## Namespaces and Package layout

SuiteSMPP keeps all classes in appropriate namespaces and separate packages depending on the programming languages being used. In the following listing, we're going to list some important SuiteSMPP subsystems along with the location in their respective languages.

Common Classes	<p>These contain the base PDU, TLV and supporting ByteBuffer classes. These are used by all PDU protocol versions as well as the supporting classes and structures. That's why they're common.</p> <p><b>C++.</b> - <code>suitesmpp::common</code></p> <p><b>Java.</b> - <code>com.smksoftware.suitesmpp.common</code></p>
Structures	<p>The structures are specific to the multiple operations (<code>SubmitMultiSM</code> and <code>SubmitMultiResp</code>). These classes represent a single destination or recipient in a list of recipients held by the respective PDUs.</p> <p><b>C++.</b> - <code>suitesmpp::structs</code></p> <p><b>Java.</b> - <code>com.smksoftware.suitesmpp.structs</code></p>
Exceptions	<p>The exceptions are thrown depending on both use, system and data errors. They have their own namespace.</p> <p><b>C++.</b> - <code>suitesmpp::exceptions</code></p> <p><b>Java.</b> - <code>com.smksoftware.suitesmpp.exceptions</code></p>
PDUs	<p>The PDUs are divided into two namespaces depending on their protocol version: <code>smpp34</code> and <code>smpp50</code>. The PDU classes are present in both with the same name if the PDU is supported in both versions.</p> <p><b>C++.</b> - <code>suitesmpp::(smpp34/smpp50)::pdu</code></p> <p><b>Java.</b> - <code>com.smksoftware.suitesmpp.(smpp34/smpp50).pdu</code></p>
Constants and Enums	<p>Enums and constants have their own namespace to keep things separate. These are also based on the protocol version.</p> <p><b>C++.</b> - <code>suitesmpp::(smpp34/smpp50)::constants</code></p> <p><b>Java.</b> - <code>com.smksoftware.suitesmpp.(smpp34/smpp50).constants</code></p>

## Implementation of PDUs in SuiteSMPP

There is a base PDU from which all implemented PDUs derive. This base class represents the SMPP PDU header which is present in all derived classes. So it includes the command length, command id, command status and the sequence number.

The base PDU also keeps the extra unknown TLV values and the extra octets. See the sections below for this.

## General PDU methods

### Getters and Setters

When attempting to set and get data members of the PDU, you should keep the SMPP specifications handy because the method names of the classes are named directly from the specification. We'll take, for example, the command length field. The field is called "command\_length" in the specification. The getters and setters will appear as:

```
PDU& setCommandLength(uint32_t commandLength);

uint32_t getCommandLength(void);
```

Mandatory parameters are expected to be present as per the specification. For TLV and optional parameters, there is an additional method to determine if the field has been set.

```
bool isMsValiditySet(void);
```

### Clearing and Sizing

Each PDU has a clear and size method.

```
void clear(void);

size_t size(void);
```

These methods do exactly as expected. The value returned by size is the number of octets required to serialize this PDU into a ByteBuffer.

## Exceptions

These are some exceptions defined for the use, parsing and generation of the PDUs. These are defined below.

BaseException	The base exception in SuiteSMPP from which all other exceptions derive. These are never thrown directly.
BufferInsufficientException	In the case of a ByteBuffer having insufficient data while a PDU is parsing it or generating into it. Normally it's thrown if the ByteBuffer doesn't have the required amount of data.
InvalidException	A field being set by the programmer is invalid.
OutOfMemoryException	In the case of the system running out of memory. If you manage to receive one of these, there's a big issue at play.
RangeException	A piece of data used to set a field in the PDU by the programmer is out of range or exceeds the bounds of the data field, according to the specification.
UnknownValueException	There is a specification-defined field which has been set to an unknown value or value outside the specification.
UnsetException	A piece of data which is mandatory or expected has not been set in the PDU.

Each exception class has a few relevant methods to retrieve information related to the exception. For example: the method in which the exception occurred, the field name, the supported range, etc.

## Extra Octets in PDUs

As mentioned, this is a robustness feature of SuiteSMPP. Some rogue SMSCs in the field actually append some extraneous octets (for reasons still unknown but most likely bug related) which are included in the command length calculation. These methods below give a programmer access to the extra octets and allow him or her to create their own deviant packets.

```
const std::vector<uint8_t> retrieveExtraOctets(void);

PDU& addExtraOctet(uint8_t val);

PDU& addExtraOctets(const char* val, size_t len);

void clearExtraOctets(void);
```

## Unknown TLVs

An additional robustness feature is to allow the presence of unknown TLVs in the PDUs. To be "unknown", the TLV tag value must not have been defined in the SMPP specification. The following methods allow one to access and create unknown TLVs at one's whim.

```
const std::vector<suitesmpp::common::TLV> retrieveUnknownTLVs(void);

PDU& addUnknownTLV(const suitesmpp::common::TLV& utlv);

void clearUnknownTLVs(void);
```

## Using the ByteBuffer

The ByteBuffer is an optimized, efficient implementation of a buffer managing octets. This is the container which the PDU classes use when parsing or generating themselves.

```
void parse(suitesmpp::common::ByteBuffer& buffer);

void generate(suitesmpp::common::ByteBuffer& out);
```

## Internal buffer management

The ByteBuffer uses a FIFO queue of octets. An object may write to the end of the queue. The first octets read off are at the front of the queue. As the queue grows longer, the internal queue buffer size is increased on a 1-1 basis for each incoming octet. The initial queue size is 0 octets. The programmer can, and it is recommended to do so, instantiate the ByteBuffer with an initial minimum capacity sufficient for their purposes so that unnecessary buffer reallocations are avoided. (A few kb perhaps?)

Reading positions in the ByteBuffer can be manipulated with calls to `mark()`, `reset()` and `cancel()`. These record a reading position, return to the previously recorded mark and forget the previously recorded mark respectively. Multiple calls to `mark` are cumulative and without limit. The number of existing marks can be determined with `markCount()`. The number of octets read since the last mark can be determined with `octetsSinceMark()`. Data which is present after an existing mark is not deleted until all marks preceding it have been cleared.

`skip()` is also available to skip past several octets in the queue.

## ByteBufferMark for RAII (C++ only)

There is a `ByteBufferMark` class for RAII paradigm in the C++ classes. It will create and reset a `mark()` on a `ByteBuffer` on construction and destruction, respectively of course.

# Quick Start

---

## Basic usage steps

### Parsing from a network socket

#### Procedure 1. Receiving data and parsing PDUs

1. Instantiate a ByteBuffer with a decent min capacity.
2. Bind and listen on a server socket.
3. Receive data on the server socket and append() into the ByteBuffer.
4. Call `suitesmpp::smpp34/smpp50/support::parse()` using the ByteBuffer. If the PDU was parsable, then a PDU will be returned. Otherwise, NULL is returned. If the incoming data is garbage, an exception will be thrown.
5. Process the PDU, if present, and then go back to receiving more data from the socket.

### Generating network data for a PDU

#### Procedure 2. Generating a PDU for network transfer

1. Instantiate and populate your PDU with the data required.
2. Instantiate a ByteBuffer with a size of `PDU->size()`.
3. Call `PDU->generate( ByteBuffer )` to make the PDU generate the raw octets.
4. Transmit the ByteBuffer data over the network to the other SMPP side. You can use the `raw()` and `available()` methods of the ByteBuffer to access the underlying bytes.

## Code examples

### Generating and parsing a BindReceiver request

#### C++

```
#include <stdlib.h>
#include <iostream>
#include <suitesmpp/smpp34/support.hpp>

using namespace suitesmpp::smpp34;

int main()
{
    pdu::BindTransceiver* bindreq = new pdu::BindTransceiver;
    bindreq->setCommandStatus( constants::PduCommandStatus::ROK );
    bindreq->setSequenceNumber( 1 );
    bindreq->setSystemId( "SYSTEMID" );
    bindreq->setPassword( "password" );
```

```
bindreq->setSystemType( "VMS" );
bindreq->setInterfaceVersion( 0x34 );
bindreq->setAddrTon( 0 );
bindreq->setAddrNpi( 0 );
bindreq->setAddressRange("");
bindreq->setCommandLength( bindreq->size() );

suitesmpp::ByteBuffer buffer( bindreq->size() );

bindreq->generate( buffer );

// Now we re-parse the buffer we just generated.

suitesmpp::PDU* pdu = support::parse( buffer );

pdu->generatehr( std::cout );

return EXIT_SUCCESS;
}
```

## Java

```
import com.smksoftware.suitesmpp.common.*;
import com.smksoftware.suitesmpp.exceptions.*;
import com.smksoftware.suitesmpp.smpp50.constants.*;
import com.smksoftware.suitesmpp.smpp50.pdu.*;

import java.io.*;

public class Main
{
    public static void main( String[] args )
        throws
            RangeException, BufferInsufficientException,
            UnknownValueException, IOException
    {
        BindTransceiver bindreq = new BindTransceiver();

        bindreq.setCommandStatus( PduCommandStatus.ROK.getValue() );
        bindreq.setSequenceNumber( 1 );
        bindreq.setSystemId( "SYSTEMID" );
        bindreq.setPassword( "password" );
        bindreq.setSystemType( "VMS" );
        bindreq.setInterfaceVersion( 0x34 );
        bindreq.setAddrTon( 0 );
        bindreq.setAddrNpi( 0 );
        bindreq.setCommandLength( bindreq.size() );

        ByteBuffer buffer = new ByteBuffer( (int)bindreq.size() );

        bindreq.generate( buffer );

        // Now we re-parse the buffer we just generated.
    }
}
```

```
PDU pdu = Parser.parse( buffer );

PrintWriter myOut = new PrintWriter( System.out, true );
pdu.generatehr( myOut );
    }
}
```

## Feedback

---

### Bug Reports

All bug reports to email address: support <at> smksoftware <dot> com

### Suggestions

All suggestions to email address: support <at> smksoftware <dot> com

### Support

Open source users (GPL / non-paying clients) must try the community forums first.

OSSLA clients and open source users who really really deem their support query valid, can send requests to email address: support <at> smksoftware <dot> com