

**An Introduction to...**

# **The Noctor Java SMS Development Kit**

---

Hay Systems Ltd.  
Version 0.1, October 16<sup>th</sup> 2001

## What Is Noctor?

The Noctor Java SMS Development Kit (hereafter referred to as the “Noctor JDK”) constitutes a simple, approachable interface to the functionality of the SMPP protocol in its revision 3.3 guise. The Noctor JDK is best suited to low throughput client-side solutions or the rapid prototyping phase of a project. If more functionality or scalability is required then libraries such as Logica’s SMPP SDKs should be considered [[http://www.logica.com/sector/telecoms/what\\_we\\_do/vass\\_developer.asp](http://www.logica.com/sector/telecoms/what_we_do/vass_developer.asp)].

## Before We Begin: Please Note

Some *important* caveats and Good Things™ are listed in the section titled “Hints & Tips” towards the end of this document.

## Step 0: Preparation

Before the example code used in this document or any other code that uses the Noctor JDK can be used, it is vital that the Noctor JDK JAR file is listed in the classpath of the machine to be used. This goes not only for development, but also for the distribution of such code.

## Step 1: Connecting

The first action that must be taken is the establishment of the client-SMSC connection. In the Noctor JDK this is accomplished by means of the `Smpp` object’s constructor, as follows:

```
Connecting to the SMSC
...
0  Smpp smpp;
1  try {
2      smpp = new Smpp(address, port);
3  } catch(UnknownHostException uhe) {
4      error that the SMSC address is (or appears to be) invalid
5  } catch(IOException ioe) {
6      error that there was a communications error
7  } catch(socksException se) {
8      error that there was an a SOCKS proxy error
9  }
...

```

Things to Note:

1. The value “**address**” can be any string containing a valid internet address, be it host name (i.e. “some.host.somewhere”) or IP address (i.e. “123.123.123.123”). This *really* ought to be the one given to you by whoever is providing you with your SMPP connection (e.g. HSL’s Advanced Service: <http://www.hsl.uk.com/information/>).
2. The value “**port**” should be the port number on the host you’ve specified upon which the SMPP service operates. Again, you’ll have been given this information and should stick to it.
3. The code given above deals explicitly with each of the three types of exception that the constructor we’re calling can throw. This is general good practice, but if you’re feeling lazy you can replace the three separate `catch` statements a single one that catches only “`Exception`” as it is the superclass of all exceptions in Java and will match onto all three.
4. You need to replace the contents of each `catch` statement with something that actually works... the simplest case being nothing.

## Step 2: Binding In

The second step in an interaction with an SMSC (or the HSL systems) is the act of “binding” to the SMSC. Binding can be thought of as logging in and will generally require that you have a valid connection details.

The simpler of the two types of bind operation is the *transmitter binding* which allows only the sending of short messages. Binding as a *receiver* is covered below, in the section titled “Step 3b: Receiving Messages”. In the Noctor JDK the process of performing a transmitter binding is fairly straightforward

and although the asynchronous bind described here is more complicated than its synchronous sibling, it is often the more useful of the two.

```
Performing a Transmitter Binding
...
0 // A simple class that shows how to implement the BindResponse
1 // interface and how to make the bind attempt.
2
3 public class TXBindExample implements BindResponse {
4     public static void main(String[] args) {
5         Smpp smpp;
6         try {
7             smpp = new Smpp(address, port);
8         } catch(UnknownHostException uhe) {
9             error; the SMSC address seems to be invalid
10        } catch(IOException ioe) {
11            error; there was a communications error
12        } catch(socksException se) {
13            error; there was a SOCKS proxy error
14        }
15
16        try {
17            Binding binding = new Binding(system_id,
18                                        system_type,
19                                        password);
20
21            smpp.bind(this, binding);
22        } catch(IOException ioe) {
23            error; there was a communications error
24        } catch(SMSException se) {
25            error; the SMSC said there was an error - the
26            bind has essentially failed
27        }
28
29        if(smpp != null)
30            smpp.close();
31    }
32
33    void bindResponse(Exception e, Binding binding) {
34        if(e == null)
35            success! perform appropriate action
36        else
37            error; we could not bind
38    }
39 }
```

#### Things to Note:

1. The initialisation of the Smpp object is subject to the same set of “Things to Note” as is given in the previous section.
2. The class we’ve created here implements the BindResponse interface, which is necessary because that interface is the means by which the Noctor JDK “calls back” to our class with the result of the bind operation.
3. When the “bindResponse” callback method is invoked, the exception passed to it (e) indicates the success or failure of the operation and the Binding object can be used to identify which bind operation the callback is being made in respect to (you may have several outstanding).
4. The whole point of asynchronous binds is to allow more control over the activity of your program rather than be stuck in blocking operations for (potentially) long periods of time. If such control doesn’t matter in a particular instance, then use the Smpp.bind(Binding) method instead.

### Step 3a: Sending a Message

Once bound as a transmitter, we can start submitting messages to the SMSC. Sending is fairly straightforward and can be accomplished with some code along the lines of:

## Sending a Message

```
...
0 // A simple class that shows how to implement the BindResponse
1 // interface and how to make the bind attempt.
2
3 public class SendExample implements BindResponse, SendResponse {
4     public static void main(String[] args) {
5         Smpplib smpplib;
6         try {
7             smpplib = new Smpplib(address, port);
8         } catch(UnknownHostException uhe) {
9             error; the SMSC address seems to be invalid
10        } catch(IOException ioe) {
11            error; there was a communications error
12        } catch(socksException se) {
13            error; there was a SOCKS proxy error
14        }
15
16        try {
17            Binding binding = new Binding(system_id,
18                                        system_type,
19                                        password);
20
21            smpplib.bind(this, binding);
22        } catch(IOException ioe) {
23            error; there was a communications error
24        } catch(SMSEException se) {
25            error; the SMSC said there was an error - the
26            bind has essentially failed
27        }
28
29        try {
30            Address destination = new Address(number,
31                                             TON,
32                                             NPI);
33
34            Message message = new Message(destination,
35                                         "Hello World!");
36
37            smpplib.send(this, message);
38        } catch(IOException ioe) {
39            error; there was a communications error
40        } catch(SMSEException se) {
41            error; the SMSC said there was an error - the
42            bind has essentially failed
43        }
44    }
45
46    void sendResponse(Exception e, Message message) {
47        if(e == null)
48            success! we've managed to send the message!
49        else
50            error; ack! The send failed :(
51    }
52
53    void bindResponse(Exception e, Binding binding) {
54        if(e == null)
55            success! perform appropriate action
56        else
57            error; we could not bind
58    }
59 }
60 }
```

Things to Note:

1. The code given here bears a striking resemblance to that given for the transmitter binding example, this should come as no surprise, as there are essentially only one additional piece of code (lines 26-42).
2. The `Message` class provides fairly comprehensive access to the configurable aspects of an SMPP message, allowing delivery time, time-to-live, multiple recipients, delivery receipts etc. to be specified or requested. It is important to note that some of these functions are unsupported with certain operators and most of the advanced functionality requires a special arrangement with your service provider.
3. It is important to ensure that the Type of Number (TON) and Numbering Plan Indicator (NPI) that you specify in the address are valid in terms of the address being specified. If there is a mismatch, then the address will be rejected by the SMSC. The SMPP 3.3 or 3.4 specifications are invaluable as reference for the values that are valid for these (and all other) fields used in SMPP.

## Step 3b: Receiving Messages

To receive messages using the Noctor JDK takes relatively little effort – only slightly more complex than the transmitter binding. The added complexity stems from the necessity of asynchronous delivery of messages to our application and the option of specifying the address range for which the binding should receive messages. The code given above for the transmitter binding (Step 2) can be altered for receiver binding and message reception as follows (numbers repeated from above denote lines that are replacements, new numbers are new lines):

| Performing a Receiver Binding |   |
|-------------------------------|---|
| ...                           |   |
| 3                             | <code>public class RXBindExample implements BindResponse, Receiver {</code>   |
| ...                           |   |
| 17                            | <code>    Binding binding = new Binding(system_id,<br/>                                  system_type,<br/>                                  password,<br/>                                  this);</code> |
| ...                           |   |
| 35                            | <code>    void receive(Exception e, Message message) {</code>   |
| 36                            | <code>        if(e == null)</code>  |
| 37                            | <code>            <b>success! we're free to process the received message</b></code>   |
| 38                            | <code>        else</code>   |
| 39                            | <code>            <b>error; there was an error while receiving the message</b></code>   |
| 40                            | <code>    }</code>  |
| 41                            | <code>}</code>  |

### Things to Note:

1. Our class definition has now been changed to include implementation of the `Receiver` interface, which is the means by which the Noctor JDK delivers messages to our application. The changes are the addition of the interface name to the “implements” list of the class (line 3) and the addition of the “receive” method (lines 35 to 41).
2. The actual act of asking the Noctor JDK to perform a receiver binding is undertaken by the revised line 17, which now passes our class as the target of the receive callbacks.
3. It is possible to specify an address range that the receiver binding should “listen” on, allowing only those messages destined for specific addresses to be received by a given receiver binding. This method requires that one use the 5 argument constructor of the `Binding` class, which has the form `Binding(system_id, system_type, password, address, receiver);` and where the `address` parameter is a valid address range. Please note that this form of the receiver binding requires special configuration at the SMSC to allow this and even then, only subsets of the address range allocated to you by your service provider can be specified.

## Step 4: Unbinding/Closing the Connection

Once we have completed the work that the SMPP connection was created for, we must shut it down in an orderly fashion. To do this, we use code similar to the following:

| Unbinding/Closing the Connection |                            |
|----------------------------------|----------------------------|
|                                  | ...                        |
| 0                                | <code>smpp.close();</code> |
|                                  | ...                        |

Things to Note:

1. Not much, really, since this method throws no exceptions. Just remember that this method is the clean way to shut down the connection.

## Appendix A: Code listings

### Connecting to the SMSC

```
0 import sms.Smpp;
1 import utils.socksException;
2 import java.io.IOException;
3 import java.net.UnknownHostException;
4
5 class ConnectExample {
6     public static void main(String[] args) {
7         Smpp smpp;
8         try {
9             smpp = new Smpp(args[0], Integer.parseInt(args[1]));
10        } catch(UnknownHostException uhe) {
11            System.err.println("SMSC address is invalid");
12        } catch(IOException ioe) {
13            System.err.println("IO error while connecting");
14        } catch(socksException se) {
15            System.err.println("SOCKS error while connecting");
16        }
17    }
18 }
```

#### Notes:

- Save this code as “ConnectExample.java” somewhere in your classpath and compile it as per normal.
- To run it type `java ConnectExample host port` where host and port relate to the SMSC you wish to connect to.

### Performing a Transmitter Binding

```
0 import sms.*;
1 import utils.socksException;
2 import java.io.IOException;
3 import java.net.UnknownHostException;
4
5 // A simple class that shows how to implement the BindResponse
6 // interface and how to make the bind attempt.
7
8 public class TXBindExample implements BindResponse {
9     public static void main(String[] args) {
10        Smpp smpp;
11        try {
12            smpp = new Smpp(args[0], Integer.parseInt(args[1]));
13        } catch(UnknownHostException uhe) {
14            System.err.println("SMSC address is invalid");
15        } catch(IOException ioe) {
16            System.err.println("IO error while connecting");
17        } catch(socksException se) {
18            System.err.println("SOCKS error while connecting");
19        }
20
21        try {
22            Binding binding = new Binding(args[2],
23                                         args[3],
24                                         args[4]);
23
24            smpp.bind(this, binding);
25        } catch(IOException ioe) {
26            System.err.println("IO error while binding");
27        } catch(SMSEException se) {
28            System.err.println("SMS error while binding");
29        }
30
31        if(smpp != null)
32            smpp.close();
33    }
34 }
```

```

33
34     void bindResponse(Exception e, Binding binding) {
35         if(e == null)
36             System.out.println("Bound successfully!");
37         else
38             System.err.println("Bind failed (" + e.getMessage() +
39         ")");
40     }

```

#### Notes:

- Save this code as “TXBindExample.java” somewhere in your classpath and compile it as per normal.
- To run it type `java TXBindExample host port system_id system_type password` where host and port relate to the SMSC you wish to connect to and system\_id, system\_type & password match the details given to you by your service provider.

### Sending a Message

```

0  import sms.*;
1  import utils.socksException;
2  import java.io.IOException;
3  import java.net.UnknownHostException;
4
5  // A simple class that shows how to implement the SendResponse
6  // interface and how to send a message.
7
8  public class SendExample implements BindResponse, SendResponse {
9      public static void main(String[] args) {
10         Smpp smpp;
11         try {
12             smpp = new Smpp(args[0], Integer.parseInt(args[1]));
13         } catch(UnknownHostException uhe) {
14             System.err.println("SMSC address is invalid");
15         } catch(IOException ioe) {
16             System.err.println("IO error while connecting");
17         } catch(socksException se) {
18             System.err.println("SOCKS error while connecting");
19         }
20
21         try {
22             Binding binding = new Binding(args[2],
23                                         args[3],
24                                         args[4]);
25
26             smpp.bind(this, binding);
27         } catch(IOException ioe) {
28             System.err.println("IO error while binding");
29         } catch(SMSEException se) {
30             System.err.println("SMS error while binding");
31         }
32
33         try {
34             int TON = Integer.parseInt(args[6]);
35             int NPI = Integer.parseInt(args[7]);
36             Address destination = new Address(args[5],
37                                             TON,
38                                             NPI);
39
40             Message message = new Message(destination,
41                                           "Hello World!");
42
43             smpp.send(this, message);
44         } catch(IOException ioe) {
45             System.err.println("IO error while sending");
46         } catch(SMSEException se) {
47             System.err.println("SMS error while sending");
48         }
49     }
50 }

```

```

43     void sendResponse(Exception e, Message message) {
44         if(e == null)
45             System.out.println("Message sent successfully!");
46         else
47             System.err.println("Send failed (" + e.getMessage() +
");
48     }
49
50     void bindResponse(Exception e, Binding binding) {
51         if(e == null)
52             System.out.println("Bound successfully!");
53         else
54             System.err.println("Bind failed (" + e.getMessage() +
");");
55     }
56 }

```

#### Notes:

- Save this code as "SendExample.java" somewhere in your classpath and compile it as per normal.
- To run it type `java SendExample host port system_id system_type password number TON NPI`, where host and port relate to the SMSC you wish to connect to, `system_id`, `system_type` & `password` match the details given to you by your service provider and number, TON and NPI together constitute some valid destination address.

#### Performing a Receiver Binding

```

0  import sms.*;
1  import utils.socksException;
2  import java.io.IOException;
3  import java.net.UnknownHostException;
4
5  // A simple class that shows how to implement the BindResponse
6  // interface and how to make the bind attempt.
7
8  public class RXBindExample implements BindResponse, Receiver {
9      public static void main(String[] args) {
10         Smppt smpp;
11         try {
12             smpp = new Smppt(args[0], Integer.parseInt(args[1]));
13         } catch(UnknownHostException uhe) {
14             System.err.println("SMSC address is invalid");
15         } catch(IOException ioe) {
16             System.err.println("IO error while connecting");
17         } catch(socksException se) {
18             System.err.println("SOCKS error while connecting");
19         }
20
21         try {
22             Binding binding = new Binding(args[2],
                                           args[3],
                                           args[4],
                                           this);
23
24             smpp.bind(this, binding);
25         } catch(IOException ioe) {
26             System.err.println("IO error while binding");
27         } catch(SMSException se) {
28             System.err.println("SMS error while binding");
29         }
30
31         if(smpp != null)
32             smpp.close();
33
34         void bindResponse(Exception e, Binding binding) {
35             if(e == null)
36                 System.out.println("Bound successfully!");

```

```
37         else
38             System.err.println("Bind failed (" + e.getMessage() +
39 ")");
40     }
41     void receive(Exception e, Message message) {
42         if(e == null)
43             System.out.println("Message received (" +
44 message.getText() + ")");
45         else
46             System.err.println("Message reception failed (" +
47 e.getMessage() + ")");
48     }
49 }
```

Notes:

- Save this code as "RXBindExample.java" somewhere in your classpath and compile it as per normal.
- To run it type `java RXBindExample host port system_id system_type password` where host and port relate to the SMSC you wish to connect to and system\_id, system\_type & password match the details given to you by your service provider.