SMPP Resource Adaptor v2.2.x

Developer's Guide

6 APRIL 2016

OpenCloud

Copyright and Disclaimers

Copyright © 2016 OpenCloud Limited. All rights reserved.

OpenCloud™ is a trademark of OpenCloud Limited.

Edinburgh House, St John's Innovation Park, Cowley Road, Cambridge CB4 0DS, United Kingdom

Documentation is provided "as is" and all express or implied conditions, representations and warranties, including any implied warranty of merchantability, fitness for a particular purpose or non infringement, are disclaimed, except to the extent that such disclaimers are held to be legally invalid.

The information in this document is confidential and meant for use only by the intended recipient and only in connection with and subject to the terms of its contractual relationship with OpenCloud.

Acceptance and/or use of any of the information contained in this document indicates agreement not to disclose or otherwise make available to any person who is not an employee of the intended recipient, or to any other entity, any of the information contained herein. This documentation has the sole purpose of providing information regarding OpenCloud software products and/or services and shall be disclosed only to those individuals who have a need to know.

Any entity or person with access to this information shall be subject to this confidentiality statement. No part of this publication may be reproduced or transmitted in any form or by any means for any purpose without the express written permission of OpenCloud.

Strictly Confidential

The information in this document is confidential and only available under OpenCloud's Non-disclosure Agreement (NDA). It must not be distributed or disclosed to any 3rd party. OpenCloud reserves the right to change the Product Portfolio information without notice or consultation.

No part of this publication may be reproduced or transmitted in any form or by any means for any purpose.

Contents

Developer's Guide

1 About the SMPP RAs....1

2 SMPP Bound Resource Adaptor....3

3 SMPP Generic Resource Adaptor....5

Configuration Properties....6

Events....8

Public Interface SmppSession....10

Simple Message Center Example....12

4 SMPP PDUS....14

© 2016 OpenCloud. All right reserved.

OpenCloud

OpenCloud's Rhino is a real-time application server for agile development, deployment and efficient management of person-to-person communication services across current and next generation technology. Rhino is a high performance, genuinely carrier-grade service execution environment for realizing a Next Generation Service Delivery Platform (NG-SDP). It uses commercial-off-theshelf (COTS) hardware and software to deliver service layer agility to TDM and IP-based networks at a radically lower price-point than traditional solutions from network equipment providers.

OpenCloud headquarters are in Cambridge, United Kingdom. R&D, Engineering and Support are located in New Zealand, Spain and there are OpenCloud branch offices in the United States, Singapore and Japan.

For more information go to:

www.opencloud.com

http://developer.opencloud.com

© 2016 OpenCloud. All right reserved.

SMPP Resource Adaptor Developers Guide

This document details basic procedures for system administrators deploying, managing and configuring the SMPP Resource Adaptor **v2.2.x**.

Topics

This document includes the following topics:

- About the SMPP RAs — an overview of the SMPP Generic and Bound RAs
- SMPP Generic Resource Adaptor — the SMPP Generic RA's SBB interface, activities, events, configuration properties, and examples
- SMPP Bound Resource Adaptor — the SMPP Bound RA's SBB interface, activities, events, life cycle, and configuration properties
- SMPP PDUs — classes for SMPP PDU types.

Audience and Scope

Intended Audience

This document is aimed at readers who:

- have a general, high-level understanding of telecommunication protocols
- are familiar with the OpenCloud SMPP Resource Adaptor
- want to install and administer the SMPP Resource Adaptor.

Scope

This document covers procedures for administering and deploying the SMPP Resource Adaptor.

This document does not focus on:

- SMPP Resource Adaptor APIs — see the SMPP Generic and SMPP Bound Resource Adaptor Javadocs.

About the SMPP RAs

Generic and Bound

There are two SMPP resource adaptors:

- SMPP Resource Adaptor (also known as the SMPP "Generic" Resource Adaptor)
- SMPP Bound Resource Adaptor.

SMPP Generic RA

The SMPP Generic resource adaptor supports generic SMPP services, in which SBBs are responsible for opening and closing SMPP sessions, and performing all session-related logic for an External Short Message Entity (ESME) or Message Center (MC) role.

SMPP Bound RA

The SMPP Bound resource adaptor supports a single, persistent connection to a Message Centre. The SBBs using this resource adaptor may only function as ESME nodes, and cannot open or close SMPP sessions. Services can support connections to multiple Message Centres by creating multiple resource adaptor entities, each one bound to a specific MC.

For more information, see:

- SMPP Generic Resource Adaptor
- SMPP Bound Resource Adaptor
- SMPP PDUs.

SMPP Bound Resource Adaptor

Supporting a single, persistent connection

This resource adaptor supports a single, persistent connection to a Message Centre. The SBBs using this resource adaptor may only function as ESME nodes, and cannot open or close SMPP sessions. Services can support connections to multiple Message Centres by creating multiple resource adaptor entities, each one bound to a specific MC.

Each SMPP Bound resource adaptor entity represents a single SMPP session with a Message Centre. The Message Centre host/port to connect to is specified by passing configuration property arguments when creating the resource adaptor entity. When the resource adaptor entity is activated it automatically attempts to connect to the Message Centre and bind. If the session dies for some reason, such as the Message Centre going down, the resource adaptor entity will attempt to reconnect at regular intervals until successful, or until the entity is deactivated.

Below are details of the SBB Interface, Activities, Events, Resource Adaptor Life Cycle, and Resource Adaptor Configuration Properties.

SBB Interface

The SBB interface object that is bound into the SBB's JNDI namespace is an instance of `SmppBoundSession`. The `SmppBoundSession` interface only permits the SBB to send requests and responses on the single session. All session setup and tear down is handled by the SMPP Bound resource adaptor entity.

SBBs that want to be able to select between multiple SMPP sessions must ensure that they have JNDI bindings for the resource adaptor entities that they want to use. It is entirely up to the SBB developer and deployer how multiple SMPP Bound resource adaptor entities are used. An example of an SBB using a "routing table" profile to select from a number of resource adaptor entities is described below.

Activities

The SMPP Bound resource adaptor uses the `SmppTransaction` object to represent an activity in the SLEE. The `SmppTransaction` activity is (typically) a short-lived activity, as it represents just a single SMPP request/response "transaction".

New activities are created when the resource adaptor receives a request, or an SBB sends a request using `SmppBoundSession.sendRequest(com.opencloud.slee.resources.smpp.pdu.Request)`. Activities end when the SBB sends a response, or when the resource adaptor has received a response and fired an event.

Events

The event types for this resource adaptor are the same as those defined for the generic SMPP resource adaptor (See `SmppResourceAdaptor`). Each SMPP request PDU is a different event type, so SBBs must subscribe to all the events that they expect to receive.

The resource adaptor will not emit any bind request or response events since the session is already bound. The events `SMPP_TIMEOUT_SESSION_INIT`, `SMPP_TIMEOUT_INACTIVITY`, `SMPP_TIMEOUT_ENQUIRE_LINK` are not emitted by this resource adaptor since these timeouts are not relevant to SBBs using this single, persistent SMPP session.

The `SMPP_NEW_SESSION` and `SMPP_END_SESSION` events are also not emitted.

The `SMPP_TIMEOUT_RESPONSE_RECEIVED` event will be emitted if a response to a request is not received in time (see `SmppResponseReceivedTimeout` in the configuration properties table below).

Resource Adaptor Life Cycle

Each resource adaptor entity connects to a single SMSC. The SMSC is specified using config properties supplied during resource adaptor entity creation. The resource adaptor entity does not attempt to connect to the SMSC until it is activated. Deactivating the resource adaptor entity will close the connection to the SMSC.

Resource Adaptor Configuration Properties

Property Name	Type	Default	Comment
`SMSCHost`	String	localhost	IP address or hostname of SMSC that this entity will connect to
`SMSCPort`	Integer	2775	TCP port to connect to.
`BindMode`	String	TRX	Type of bind request to use, must be one of TX, RX or TRX (transmitter, receiver or transceiver).
`SystemID`	String	rhino	SystemID in bind request for authenticating to SMSC.
`Password`	String	rhino	Password in bind request for authenticating to SMSC.
`RebindInterval`	Long	5000	Interval (in ms) between re-bind attempts after connection dies.
`EnquireLinkInterval`	Long	5000	Interval (in ms) between `ENQUIRE_LINK` requests. The resource adaptor periodically sends these to check the session is still active.
`EnquireLinkThreshold`	Integer	5	Number of consecutive failed `ENQUIRE_LINK` requests, after which the resource adaptor will drop the connection and attempt to reconnect to the SMSC.
`SmppResponseReceivedTimeout`	Long	11000	Time (in ms) that stack waits for a response to be received
`SmppResponseSentTimeout`	Long	10000	Time (in ms) that stack waits for a response to be sent (to an incoming request). The resource adaptor will automatically send a `GENERIC_NACK` if the SBB has not sent a response.

SMPP Generic Resource Adaptor

Supporting generic SMPP services

This resource adaptor supports generic SMPP services, in which SBBs are responsible for opening and closing SMPP sessions, and performing all session-related logic for an External Short Message Entity (ESME) or Message Center (MC) role.

Services using this resource adaptor can function as ESME or MC nodes, or both (an RE or Routing Entity). The type of node is determined automatically when the service sends or receives its first PDUs in a session. For example, if a service creates a session and then sends an OUTBIND request, then this is MC behaviour; so the node is treated as an MC from that point on, until the end of the session. Similarly if a node creates a session, and then sends a BIND_TRANSMITTER request, then this is ESME behaviour. The node type determines what PDUs are allowed to be sent or received in a session. For example, an MC cannot send SUBMIT_SM requests because this is not allowed by the SMPP protocol.

(The SMPP Bound resource adaptor type API consists of some classes in this package, and some in the `com.opencloud.slee.resources.smpp.boundra` package. The documentation for the Bound RA is in that package.)

SBB Interface

The SBB interface object that is bound into the SBB's JNDI namespace is an instance of `SmppProvider`. SBBs must not use the methods on `SmppProvider` which are inherited from `SmppStackProvider` — these are only visible for historic reasons and will be removed in a future release. Some `SmppStackProvider` methods will throw a `SecurityException` if called from a SBB, and others will not behave as expected.

Activities

The activity for the generic resource adaptor is an SMPP Session, which is a TCP connection between 2 SMPP peers, and some associated session state. This is represented by the `SmppSession` interface.

An `SmppSession` activity is created when an incoming connection is received, or when an SBB calls `SmppProvider.openSession()` . The SMPP RA does not generate an `SMPP_NEW_SESSION` event if an SBB opened the session.

If an SBB calls `SmppSession.closeSession()` then the SmppSession activity will be ended. The SMPP RA does not generate an `SMPP_END_SESSION` event in this case, but the SLEE will of course generate an `ActivityEndEvent`.

See also:

- Configuration Properties
- Events
- Public Interface SmppSession
- Simple Message Center Example

Configuration Properties

Configuring the SMPP Generic Resource Adaptor

Below are details of the SMPP Generic Resource Adaptor configuration properties.

Property Name	Type	Default	Comment
`SmppBindAddresses`	String		Allows a different bind address (network address and port combination) to be specified for each node in a cluster. See note below.
`SmppListenAddress`	String	null	Local IP address to listen on (null = listen on all addresses). `SmppBindAddresses` should be used instead of this.
`SmppListenPort`	Integer	2775	Local TCP port to listen on. `SmppBindAddresses` should be used instead of this.
`SmppSessionInitTimeout`	Long	30000	Timeout (in ms) for session init, an `SMPP_TIMEOUT_SESSION_INIT` event will be fired if session was not bound this amount of time after TCP connection opened.
`SmppEnquireLinkTimeout`	Long	60000	Timeout (in ms) for link enquiry, an `SMPP_TIMEOUT_ENQUIRE_LINK` event will be fired periodically using this interval.
`SmppInactivityTimeout`	Long	120000	Timeout (in ms) for session inactivity, an `SMPP_TIMEOUT_INACTIVITY` event will be fired if session is inactive for this amount of time.
`SmppResponseReceivedTimeout`	Long	10000	Timeout (in ms) for receiving a response from the a peer, an `SMPP_TIMEOUT_RESPONSE_RECEIVED` will be fired if this amount of time passes.
`SmppResponseSentTimeout`	Long	10000	Timeout (in ms) for a service to send a response to an incoming request, resource adaptor will send a `GENERIC_NACK` it this amount of time passes.
`SmppStateCheckingEnabled`	Boolean	true	Determines whether stack checks for correct session state when sending/receiving messages.

`SmppBindAddresses`	String		Allows a different bind address (network address and port combination) to be specified for each node in a cluster. Takes the following format:
			`{node}address:port,{node}address:port,....`
			This allows entities running on two nodes on the same host to use different ports, for example:
			`{101}0.0.0.0:2775,{102}0.0.0.0:2776`

			It also allows entities running on different hosts to specify an interface to listen on that is specific to each host, for example:
			
			`{101}192.168.1.100:2775,`
			`{102}192.168.1.101:2775`
			
			If empty, `SmppListenAddress` and `SmppListenPort` will be used, and will have the same value on all nodes.

Events

Generic SMPP Events

Each type of SMPP Protocol Data Unit (PDU) is a separate SLEE event type. In addition, there are several more event types that represent session opening/closing and communication error events (timeouts, I/O errors). For the complete list of event names see below.

The SMPP RA does not emit a `SMPP_TIMEOUT_RESPONSE_SENT` event, even though this event type is defined by `SmppErrorEvent`. This event is handled internally by the RA, as it means that the application (SBB) has not sent a response to an incoming request within the stack's ResponseSentTimeout. This may indicate that there was some error in the SLEE during event processing. The RA sends a GENERIC_NACK PDU with status ESME_RSYSERR in this case. The RA also sends a GENERIC_NACK if the event was processed successfully but not handled by any SBBs, or if the event processing failed for some other reason.

There are four event classes:

- SMPP Session Events
- SMPP Request Events
- SMPP Response Events
- SMPP Error Events

SMPP Session Events

Class: `SmppSessionEvent`

- `com.opencloud.slee.resources.smpp.SMPP_NEW_SESSION`
- `com.opencloud.slee.resources.smpp.SMPP_END_SESSION`

SMPP Request Events

Class: `SmppRequestEvent`

- `com.opencloud.slee.resources.smpp.pdu.ALERT_NOTIFICATION`
- `com.opencloud.slee.resources.smpp.pdu.BIND_TRANSMITTER`
- `com.opencloud.slee.resources.smpp.pdu.BIND_RECEIVER`
- `com.opencloud.slee.resources.smpp.pdu.BIND_TRANSCEIVER`
- `com.opencloud.slee.resources.smpp.pdu.DELIVER_SM`
- `com.opencloud.slee.resources.smpp.pdu.SUBMIT_SM`
- `com.opencloud.slee.resources.smpp.pdu.DATA_SM`
- `com.opencloud.slee.resources.smpp.pdu.CANCEL_SM`
- `com.opencloud.slee.resources.smpp.pdu.QUERY_SM`
- `com.opencloud.slee.resources.smpp.pdu.REPLACE_SM`
- `com.opencloud.slee.resources.smpp.pdu.ENQUIRE_LINK`
- `com.opencloud.slee.resources.smpp.pdu.OUTBIND`
- `com.opencloud.slee.resources.smpp.pdu.UNBIND`

SMPP Response Events

Class: `SmppResponseEvent`

- `com.opencloud.slee.resources.smpp.pdu.GENERIC_NACK`
- `com.opencloud.slee.resources.smpp.pdu.BIND_TRANSMITTER_RESP`
- `com.opencloud.slee.resources.smpp.pdu.BIND_RECEIVER_RESP`
- `com.opencloud.slee.resources.smpp.pdu.BIND_TRANSCEIVER_RESP`
- `com.opencloud.slee.resources.smpp.pdu.DELIVER_SM_RESP`
- `com.opencloud.slee.resources.smpp.pdu.SUBMIT_SM_RESP`
- `com.opencloud.slee.resources.smpp.pdu.DATA_SM_RESP`
- `com.opencloud.slee.resources.smpp.pdu.CANCEL_SM_RESP`

- com.opencloud.slee.resources.smpp.pdu.QUERY_SM_RESP
- com.opencloud.slee.resources.smpp.pdu.REPLACE_SM_RESP
- com.opencloud.slee.resources.smpp.pdu.ENQUIRE_LINK_RESP
- com.opencloud.slee.resources.smpp.pdu.UNBIND_RESP

SMPP Error Events

Class: SmppErrorEvent

- com.opencloud.slee.resources.smpp.SMPP_ERROR
- com.opencloud.slee.resources.smpp.SMPP_TIMEOUT_SESSION_INIT
- com.opencloud.slee.resources.smpp.SMPP_TIMEOUT_ENQUIRE_LINK
- com.opencloud.slee.resources.smpp.SMPP_TIMEOUT_INACTIVITY
- com.opencloud.slee.resources.smpp.SMPP_TIMEOUT_RESPONSE_RECEIVED
- com.opencloud.slee.resources.smpp.SMPP_TIMEOUT_RESPONSE_SENT

Public Interface SmppSession

SmppSession

This interface represents an SBB's view of an SMPP session. The session is created when an SMPP node connects to another node using `SmppProvider.openSession()`.

The session represents a point-to-point connection between two SMPP nodes. One node is always behaving as an ESME (External Short Message Entity) and the other is an MC (Message Center, also known as SMSC). The type of node that the service is behaving as is determined by how the session is initiated. For example, if the service opens a connection and then attempts to send a `BIND_TRANSMITTER` request, then the session type is set to ESME (as this is ESME behaviour). Similarly, if the service opens a connection and then sends an `OUTBIND`, it will be treated as a MC node. The state of the session changes automatically as messages are processed by the implementation. The session type (ESME or MC) and state determine what SMPP operations are permitted.

There are methods to close the session, create outgoing request activities for responses to be fired on, and send requests whose responses will be fired on this activity. The SMPP protocol is inherently asynchronous, and as such this implementation uses an asynchronous model as well. Responses to requests are received as SLEE events.

For convenience, a blocking `sendSyncRequest` method has been provided for simpler services that do not need to be asynchronous.

> ⚠ The synchronous API should only be used for simple test services that are not expected to run at high loads. Services that use the synchronous model also do not scale well.

Examples

An ESME connecting and binding to a Message Center

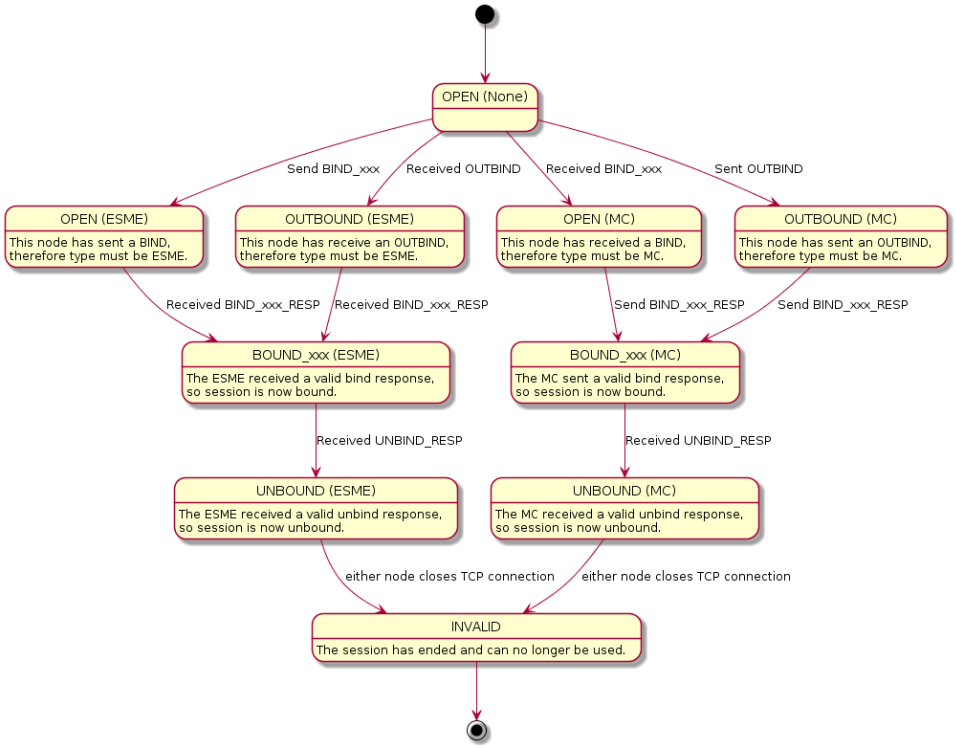
```
BindTransmitter bindRequest = new BindTransmitter();
bindRequest.setInterfaceVersion((byte)0x34);  // SMPP V3.4
bindRequest.setSystemID("esme");
bindRequest.setPassword("secret");
session.sendRequest(bindRequest); // asynchronous send ...

// ... response received by a "handleResponse" event handler method:
void handleResponse(SmppResponseEvent event, ActivityContextInterface aci) {
    Response r = event.getResponse();
    if ((r.getCommandID == CommandID.BIND_TRANSMITTER_RESP) &&
        (r.getCommandStatus == CommandStatus.ESME_ROK)) {
        // MC successfully authenticated bind, session
        // is now in BOUND_TX state.
    }
}
```

SMPP Session States

The state diagram shows the states and transitions that an SMPP Session can go through.

Note that for brevity, each type of bind request has not been shown; instead `BIND_xxx` is used to denote `BIND_TRANSMITTER`, `BIND_RECEIVER`, and `BIND_TRANSCEIVER`.

Note that, in any of the above states, either node closing the session will cause a transition to the `INVALID` state.


OPEN (None)
Send BIND_xxx
Received OUTBIND
Received BIND_xxx
Sent OUTBIND
OPEN (ESME)
This node has sent a BIND, therefore type must be ESME.
OUTBOUND (ESME)
This node has receive an OUTBIND, therefore type must be ESME.
OPEN (MC)
This node has received a BIND, therefore type must be MC.
OUTBOUND (MC)
This node has sent an OUTBIND, therefore type must be MC.
Received BIND_xxx_RESP
Received BIND_xxx_RESP
Send BIND_xxx_RESP
Send BIND_xxx_RESP
BOUND_xxx (ESME)
The ESME received a valid bind response, so session is now bound.
BOUND_xxx (MC)
The MC sent a valid bind response, so session is now bound.
Received UNBIND_RESP
Received UNBIND_RESP
UNBOUND (ESME)
The ESME received a valid unbind response, so session is now unbound.
UNBOUND (MC)
The MC received a valid unbind response, so session is now unbound.
either node closes TCP connection
either node closes TCP connection
INVALID
The session has ended and can no longer be used.

Simple Message Center Example

Simple Example of the SMPP Generic Resource Adaptor Type

Below are examples of three types of requests, and a diagram of session states, using the SMPP Generic Resource Adaptor.

Receiving bind_transmitter request

```
public void onBindTransmitterEvent(SmppRequestEvent event, ActivityContextInterface aci) {
    SmppSession smppSession = event.getSession();
    BindTransmitter bindRequest = (BindTransmitter) event.getRequest();
    BindTransmitterResp bindResponse = (BindTransmitterResp)
 bindRequest.createResponse(CommandStatus.ESME_ROK);
    // add sc_interface_version TLV
    byte[] version = {0x50};
    bindResponse.addTLV(new TLV(TLV.SC_INTERFACE_VERSION, version));
    smppSession.sendResponse(bindResponse);
}
```

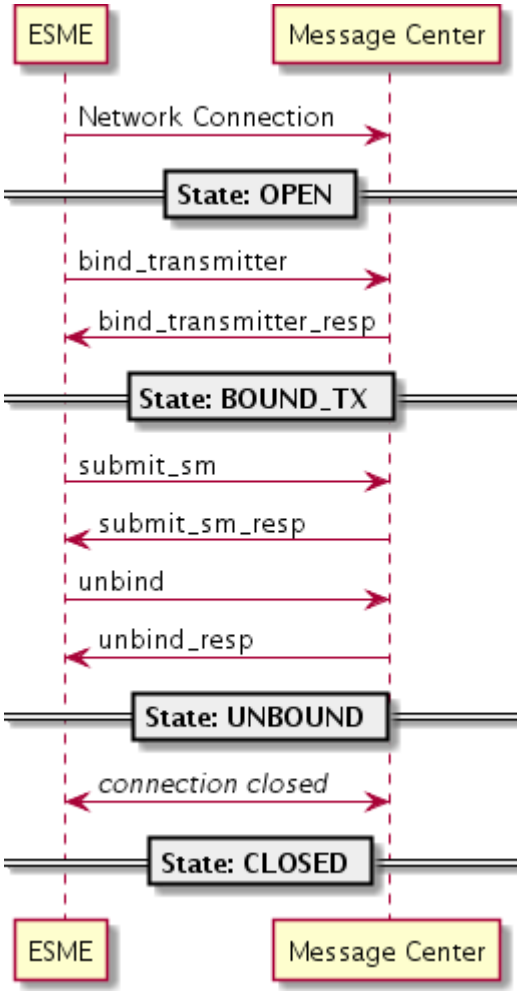
Receiving submit_sm request

```
public void onSubmitSMEvent(SmppRequestEvent event, ActivityContextInterface aci) {
    SubmitSM smRequest = (SubmitSM) event.getRequest();
    SubmitSMResp smResponse = (SubmitSMResp) smRequest.createResponse(CommandStatus.ESME_ROK);
    smResponse.setMessageID("12345");
    ((IncomingSmppRequestActivity)aci.getActivity()).sendResponse(smResponse);
}
```

Receiving unbind request

```
public void onUnbindEvent(SmppRequestEvent event, ActivityContextInterface aci) {
    SmppSession smppSession = event.getSession();
    Unbind unbindRequest = (Unbind) event.getRequest();
    UnbindResp unbindResponse = (UnbindResp) unbindRequest.createResponse(CommandStatus.ESME_ROK);
    smppSession.sendResponse(unbindResponse);
}
```

Sequence diagram illustrating session states



SMPP PDUs

Classes for SMPP PDU types

The API contains classes that represent each of the SMPP PDU types, as well as classes for other data types and some helper utilities.

The PDU classes match the SMPP PDU name, but use Java class naming conventions.

PDU fields are represented by instance variables, with set/get methods for each. To send a `SUBMIT_SM` request, the application creates an instance of the `SubmitSM` class, sets fields such as the source and destination addresses, and then sends the request using the appropriate `SmppProvider` method:

```
import com.opencloud.slee.resources.smpp.SmppProvider;
import com.opencloud.slee.resources.smpp.pdu.SubmitSM;
import com.opencloud.slee.resources.smpp.pdu.Address;
import com.opencloud.slee.resources.smpp.pdu.ShortMessage;

// [...]
    SubmitSM request = new SubmitSM();
    request.setSourceAddress(new Address("1234"));
    request.setDestAddress(new Address("6421555666"));
    request.setShortMessage(new ShortMessage("Your entry was received. Thank you."));
    provider.sendRequest(sessionID, request);
// [...]
```