Question: What is the minimum required version of WildFly for ESME?
Answer: ESME requires at least WildFly 16 and JDK 1.8 for smooth operation.
Example:
To ensure compatibility:
WildFly version: 16+
JDK version: 1.8

Question: How do you configure the JDK path in WildFly?
Answer: You need to set the JAVA_HOME variable in the standalone.conf file located inside the bin directory of WildFly.
Example:
Edit the file at:
$WILDFLY_HOME/bin/standalone.conf
Add or modify the line:

JAVA_HOME="/opt/java/jdk"

Question: How do you configure memory allocation in WildFly?
Answer: Memory settings can be configured in the same standalone.conf file by updating the JAVA_OPTS variable with -Xms and -Xmx values.
Example:
Set both initial and max heap memory to 1024MB:

JAVA_OPTS="-Xms1024m -Xmx1024m"

Question: Where is the standalone.xml configuration file located?
Answer: It is located at $WILDFLY_HOME/standalone/configuration/standalone.xml.
Example:
Path to edit:

/home/user/wildfly/standalone/configuration/standalone.xml

Question: What network ports can be configured in the standalone.xml file?
Answer: Ports that can be configured include:
ajp port
https port
http port (default 8080)
management http port
management https port
Example:
To change the HTTP port from 8080 to 8181:

<socket-binding name=“http” port=“8181”/>

Question: How can you check if a port (e.g., 8080) is already in use?
Answer: Use the netstat command with grep to filter results for the desired port.
Example:

netstat -antp | grep 8080

Example:

```
       <socket-binding name="management-http" interface="management"
port="${jboss.management.http.port:9990}"/>
        <socket-binding name="management-https" interface="management"
port="${jboss.management.https.port:9993}"/>
        <socket-binding name="ajp" port="${jboss.ajp.port:8009}"/>
        <socket-binding name="http" port="${jboss.http.port:8080}"/>
        <socket-binding name="https" port="${jboss.https.port:8443}"/>
        <socket-binding name="txn-recovery-environment" port="4712"/>
        <socket-binding name="txn-status-manager" port="4713"/>
        <outbound-socket-binding name="mail-smtp">
            <remote-destination host="localhost" port="25"/>
        </outbound-socket-binding>
    </socket-binding-group>```


 Adding MySQL/Oracle JAR to WildFly:
```

Question: How can you add a MySQL or Oracle JDBC driver to WildFly?
Answer: You need to add the JDBC driver JAR file to the correct module path in WildFly and define a corresponding module.xml file.
Example:
For MySQL, create the directory:
$WILDFLY_HOME/modules/system/layers/base/com/mysql/jdbc/Driver/main
Place the MySQL JAR file (e.g., mysql-connector-java-8.0.26.jar) in the main folder.

Question: What should the module.xml contain when adding a MySQL driver to WildFly?
Answer: The module.xml should define the module name (matching the folder path) and reference the JAR file in the resources section.
Example:
$WILDFLY_HOME/modules/system/layers/base/com/mysql/jdbc/Driver/main/module.xml

```
   <resources>
        <resource-root path="mysql-connector-java-8.0.26.jar"/>
    </resources>
    <dependencies>
        <module name="javax.api"/>
        <module name="javax.transaction.api"/>
    </dependencies>
```

```
</module>```

Question: Why should the module name in module.xml match the path name?
Answer: WildFly uses the module path and name to load the driver
correctly; mismatched names will result in deployment errors.
Example:
Path:
com/mysql/jdbc/Driver
Module name in module.xml:

<module name="com.mysql.jdbc.Driver">
Question: Where should the MySQL JAR file be placed in the WildFly
directory structure?
Answer: Inside the main folder under the JDBC driver module path.
Example:
Full path:

$WILDFLY_HOME/modules/system/layers/base/com/mysql/jdbc/Driver/main/mysql-
connector-java-8.0.26.ja
```

ESME Application SMPP.XML file Configuration

Question: how to configure the smscid in smpp.xml file ?
Answer: The smpp.xml file holds the essential configuration settings required for ESME to establish a successful SMPP bind with the SMSC.
Example:
File location:
$WILDFLY_HOME/standalone/deployments/ESME.war/WEB-INF/classes/smpp.xml

Question: What should be add in the <smsc id> tag in smpp.xml?
Answer: It should contain a unique identifier for the SMSC.
Example:

``` <smsc id="SMSC_ID" protocol="smpp">
    :
    :
  </smsc> '''

Question: How to connect with the SMSC?
Answer: The <ip-address> and <port> tags specify the network location of the SMSC that the ESME should connect to SMSC.
<system-id>It is the unique ID for the ESME system that wants to bind to the SMSC.
```

<password> Password of the SMSC.
Example:

``` <ip-address>SMSC_IP</ip-address>
<port>SMSC_PORT</port>
<system-id>test1</system-id>
<password>test1</password> ```

Question: What is the password requirement in smpp.xml for SMPP binding?
Answer: The password must be less than 9 characters, as per the SMPP standard.

Question: What values can the <bind-mode> take in smpp.xml, and what do they mean?
Answer: The bind-mode can be:
t → Transmitter
r → Receiver
tr → Transceiver
Example:

<bind-mode>tr</bind-mode>

Question: What does <REGISTERED-DELIVERY> mean in smpp.xml?
Answer: It specifies whether delivery receipts are needed from the SMSC.
Example:

<REGISTERED-DELIVERY>1</REGISTERED-DELIVERY>
<!-- 1 = Yes, 0 = No -->

Question: What should be the <interface-version> for SMPP 3.4?
Answer: The version tag should be set to 4, which corresponds to SMPP version 3.4.
Example:

<interface-version>4</interface-version>
Question: What are TON and NPI, and how are they used in smpp.xml?
Answer:

TON (Type of Number) and NPI (Numbering Plan Indicator) define how source and destination addresses are interpreted.
Example:

<sourceAddrTon>5</sourceAddrTon>  <!-- Alphanumeric -->
```

<sourceAddrNpi>0</sourceAddrNpi>
<destAddrTon>1</destAddrTon> <!-- International -->
<destAddrNpi>1</destAddrNpi>

Question: What is the purpose of the service-type parameter in smpp.xml?
Answer: The service-type parameter is used to indicate the SMS application service associated with the message. It allows the ESME to leverage enhanced messaging services and control the tele-service used.
Example:

<service-type></service-type> <!-- Default (NULL) for SMSC default settings -->
Question: What is dcs and how is it used?
Answer: dcs stands for Data Coding Scheme. It defines the encoding used in the message.
0 → English (default)

8 → Unicode (for other languages)

16 → Flash Message
Example:

<dcs>8</dcs> <!-- For Unicode message encoding -->
Question: What does messageReceiverListenerImpl define?
Answer: It specifies the full class path of the implementation that listens to received messages. This should only be changed if suggested by the Development Team.
Example:

<messageReceiverListenerImpl>com.sixdee.imp.receiver.MessageReceiverListenerImpl</messageReceiverListenerImpl>
Question: What does TPS mean in the ESME context?
Answer: TPS (Transactions Per Second) defines how many transactions are allowed per second. Default is -1 (unlimited), and it can be set to 1 when using retry mechanisms.
Example:

<TPS>1</TPS>
Question: What is the purpose of enquireLinkTimer?
Answer: This defines the interval (in seconds) at which the ESME sends an enquire_link command to check the SMPP session status.
Example:

<enquireLinkTimer>5</enquireLinkTimer>
Question: What is receive-timeout in smpp.xml?
Answer: It sets how long (in seconds) the ESME will wait for a response message from the SMSC in a synchronous way.
Example:

<receive-timeout>10</receive-timeout>
Question: What does pdu-processors define?
Answer: It defines the number of threads that can process SMPP requests simultaneously. Maximum value is 5.
Example:
<pdu-processors>3</pdu-processors>
Question: What is pdu-maxQueueSize used for?
Answer: It sets the maximum size of the message queue waiting to be processed by PDU processors. Use -1 for unlimited.
Example:

<pdu-maxQueueSize>1000</pdu-maxQueueSize>
Question: What does allowedNumSeries do?
Answer: It lists the allowed number series from which incoming messages are accepted. Default is "*" meaning all are allowed.
Example:

<allowedNumSeries>*</allowedNumSeries>

Connector

What is the purpose of the <connector> section in smpp.xml?
Answer: The <connector> section defines how the ESME communicates with external systems (like MO_Router or other third-party applications) through protocols like HTTP. It includes details like thread count, protocol, method, URLs, and message format.
Example:

```
   <max_connection>5</max_connection>
    <transportprotocol>http</transportprotocol>
    <transportmethod>post</transportmethod>
    <transporturl1>Mo_router_URL</transporturl1>
</connector>```

Question: What does the <max_connection> tag define in connector?
Answer: It specifies the number of threads that can simultaneously send messages to the next generation (NG) platform or any third-party system. Valid range is 1–5.
Example:

<max_connection>5</max_connection>
Question: What is the role of <transportprotocol> in the connector section?
```

Answer: It sets the transport protocol used for communication, typically http for most ESME to MO_Router interactions.
Example:

<transportprotocol>http</transportprotocol>
Question: What does <transportmethod> represent in the connector section in smpp.xml?
Answer: It defines the HTTP method used for sending the message—usually post.
Example:

<transportmethod>post</transportmethod>

Question:ESME will support primary and secondary url? What are <transporturl1> and <transporturl2> used for?
Answer: Yes, These specify the URLs where the message will be posted. ESME can use one as primary and the other as fallback or for load distribution.
Example:
<transporturl1>Mo_router_URL</transporturl1>
<transporturl2>Mo_router_URL2</transporturl2>

Question: What is <messageformat> in the connector section and why is it important?
Answer: It defines the structure of the message (in XML or JSON) that will be sent from ESME to the downstream system like MOR. The structure varies depending on the type and length of the message.
Example:

```<messageformat>
  "timeStamp": "{0}",
  "transactionId": "{1}",
  "smscID": "{2}",
  "esmClass": "{4}",
  "orginatingAddress": "{5}",
  "destinationAddress": "{6}",
  "dcs": "{7}",
  "sms": "{8}",
  "byteSms": "{9}",
  "optionalParamText":"{11}",
  "message":"{17}",
  "messageIdentifier":"{18}",
  "messageParts":"{19}",
  "messagePartNumber":"{20}"```
</messageformat>

Question: What happens when a long message is received from SMSC?
Answer: A long message gets split into multiple parts, each sent separately with UDH (User Data
```

Header) to help identify and reassemble the full message at the MOR (Message Orchestration Router) side.

Example:

<MESSAGE-IDENTIFIER><![CDATA[85]]></MESSAGE-IDENTIFIER>
<MESSAGE-PARTS><![CDATA[2]]></MESSAGE-PARTS>
<MESSAGE-PART-NUMBER><![CDATA[1]]></MESSAGE-PART-NUMBER>
This means the message has two parts and this is the first part. UDH headers (unreadable characters) may be present in <SMS>, but removed in <MESSAGE>.

Question: What is the purpose of the <MESSAGE-IDENTIFIER> tag?
Answer: It contains a unique identifier (like 85) to link all parts of a multipart message. MOR uses this to concatenate the parts.

Example:

<MESSAGE-IDENTIFIER><![CDATA[85]]></MESSAGE-IDENTIFIER>
Used in both message parts for linking them together.

Question: What does <MESSAGE-PARTS> indicate?
Answer: It shows the total number of message parts.

Example:

<MESSAGE-PARTS><![CDATA[2]]></MESSAGE-PARTS>
Indicates that there are 2 parts in the complete message.

Question: What is the significance of <MESSAGE-PART-NUMBER>?
Answer: It specifies the sequence number of each part in the multipart message, helping MOR to reassemble them in order.

Example:

<MESSAGE-PART-NUMBER><![CDATA[2]]></MESSAGE-PART-NUMBER>
This indicates the current part is the second segment of the total message.

Question: What does it mean if <ESM-CLASS> is 0?
Answer: It means the message is a single-part message (usually less than 160 characters), and there are no UDH headers involved.

Example:

<ESM-CLASS><![CDATA[0]]></ESM-CLASS>
<MESSAGE><![CDATA[Hello from SMPPSim]]></MESSAGE>

<MESSAGE-IDENTIFIER><![CDATA[]]></MESSAGE-IDENTIFIER>
A simple message without multipart handling.

Question: How is a newline character handled when coming from the SMSC?
Answer: If the message contains newline (\n) characters, they are preserved in the <MESSAGE> field if properly decoded. UDH headers still apply only if the message is multipart (ESM-CLASS = 64).

Example:

<MESSAGE><![CDATA[Hello\nWorld]]></MESSAGE>
The decoded message contains a newline. This is common when ESME receives SMS with formatting or line breaks.

Question: What happens to <MESSAGE-IDENTIFIER>, <MESSAGE-PARTS>, and <MESSAGE-PART-NUMBER> in single message scenarios?
Answer: These fields remain empty because no UDH headers are present, and the message is not split.

Example:

```<MESSAGE-IDENTIFIER><![CDATA[]]></MESSAGE-IDENTIFIER>
<MESSAGE-PARTS><![CDATA[]]></MESSAGE-PARTS>
<MESSAGE-PART-NUMBER><![CDATA[]]></MESSAGE-PART-NUMBER>```

Delivery connector

Question: how to configure the <deliveryconnector> section in smpp.xml?
Answer: It defines how the ESME sends delivery receipts to third-party systems (like UMS/NG) when RegisteredDelivery = 1. It includes transport protocol details, target URLs, and the message format.

Example:

```<deliveryconnector>
<max_connection>5</max_connection>
<transportprotocol>http</transportprotocol>
<transportmethod>post</transportmethod>
<transporturl1>Mo_router_URL</transporturl1>
<transporturl2>Mo_router_URL</transporturl2>
<messageformat>
“featureId”: “DELIVERY-RECEIPT”,“timeStamp”: “{0}”,“transactionId”: “{1}”,
“smscID”: “{2}”,“commandId”: “{3}”,“orginatingAddress”: “{5}”,
“destinationAddress”: “{6}”,“dcs”: “{7}”,“sms”: “{8}”,“messageId”: “{10}”

</messageformat>
</deliveryconnector>```

Question: When is a delivery receipt triggered in ESME?
Answer: A delivery receipt is triggered when <REGISTERED-DELIVERY> is set to 1 in the SMPP bind request, indicating that the SMSC should send back delivery status info.

Example:

<REGISTERED-DELIVERY>1</REGISTERED-DELIVERY>
Question: What is the role of ESM-CLASS in the delivery receipt?
Answer: ESM-CLASS value is 4 for delivery receipts, indicating that the incoming message is a receipt and not an actual SMS.

Example:

<ESM-CLASS><![CDATA[4]]></ESM-CLASS>
Question: What is included in the <SMS> tag of a delivery receipt?
Answer: The <SMS> tag includes delivery details such as message ID, submission and delivery timestamps, status, and part of the original text message.

Example:

```<SMS><![CDATA[
id:0 sub:001 dlvrd:001
submit date:2104091601
done date:2104091601
stat:DELIVRD err:000
Text:I called You at Wed,
]]></SMS>```
Question: What does the delivery connector message format contain?
Answer: It's a JSON/XML formatted string that includes tags like timestamp, transaction ID, smsc ID, command ID, sender and receiver info, DCS, message content, and message ID.

Example (JSON format):

```{
"featureId": "DELIVERY-RECEIPT",
"timeStamp": "20210409160127",
"transactionId": "13600",
"smscID": "airtel",
"commandId": "5",
"orginatingAddress": "9611726551",
"destinationAddress": "8688492301",
"dcs": "2",

“sms”: “id:0 sub:001 dlvrd:001 submit date:2104091601 done date:2104091601 stat:DELIVRD err:000 Text:I called You at Wed,”,
“messageId”: “0”
}```
Question: How does ESME send the delivery receipt to 3rd party applications?
Answer: Using HTTP POST method defined in the delivery connector. The receipt is sent to one of the configured URLs like transporturl1 or transporturl2.

Example:

<transportmethod>post</transportmethod>
<transporturl1>Mo_router_URL</transporturl1>

Optional Parameters

Question: What are optional parameters in smpp.xml used for?
Answer: Optional parameters are additional tags (TLVs) used in special scenarios, such as handling message payloads, advanced routing, or app-specific metadata. These tags are not mandatory for basic message delivery but enhance message handling capabilities.

Question: Where are optional parameters defined in smpp.xml?
Answer: They are defined under the <optionalparameterstag> section in the smpp.xml configuration file.

Question: What does a typical optional parameter tag look like?
Answer: Each optional parameter is declared as a <tlv> block, containing a unique tag identifier and a corresponding name.

Example:

```<optionalparameterstag>
<tlv1>
<tag>5126</tag>
<name>OptionalParam_OA</name>
</tlv1>
</optionalparameterstag>```
Question: What does the tag 5126 represent in the example?
Answer: 5126 is a unique tag identifier used by the SMPP protocol to reference the optional parameter. It must match what is expected or used on both ESME and SMSC ends.

Question: What is the purpose of OptionalParam_OA in the example?
Answer: OptionalParam_OA is a custom optional parameter name which could be used to indicate
```

an alternate Originating Address or any custom logic defined in the application that processes these parameters.

API Packets

ESME Xml Request Packet
Adapters:
Example:
http://localhost:8081/ESME/HttpAdapter
http://localhost:8081/ESME/HttpNoRespAdapter

Sample ESME XML Request packet :

Submit_SM

Here, in submit_sm, Acknowledgement is given when the request hits the ESME

URL and after processing, the original response is given to the resp URL attached in

the request API. For synch_submit_sm, Request and Response will generate in the

same cycle simultaneously, so no acknowledgement is given in that case.

Submit_sm

```
<REQ>
<FEATURE>submit_sm</FEATURE>
<TIME-STAMP>25062009103510</TIME-STAMP>
<RESPONSE-URL>http://localhost:8080/ESME/TestServlet</RESPONSE-URL>
<PARAMETERS>
<REQ-TRANSACTION-ID>0</REQ-TRANSACTION-ID>
<SMSC-ID>airtel</SMSC-ID>
<SUBMIT-SM>
<SOURCE-ADDR-TON>0</SOURCE-ADDR-TON>
<SOURCE-ADDR-NPI>0</SOURCE-ADDR-NPI>
<SOURCE-ADDR>1234</SOURCE-ADDR>
<DEST-ADDR-TON>0</DEST-ADDR-TON>
<DEST-ADDR-NPI>0</DEST-ADDR-NPI>
<DESTINATION-ADDR>9945626828</DESTINATION-ADDR>
<ESM-CLASS>0</ESM-CLASS>
<concatenatedSMesmClass>0</concatenatedSMesmClass>
<SHORT-MESSAGE>UkNIUiAyMjIy</SHORT-MESSAGE>
<DATA-CODING>0</DATA-CODING>
```

</SUBMIT-SM>
</PARAMETERS>
</REQ>
Instant response
<RESP>
<TIME-STAMP>23072021202222</TIME-STAMP>
<STATUS>0</STATUS>
</RESP>
Final response
<RESP>
<STATUS>0</STATUS>
<STATUS-CODE>0</STATUS-CODE>
<TIME-STAMP>30072021192109</TIME-STAMP>
<PARAMETERS>
<REQ-TRANSACTION-ID>0</REQ-TRANSACTION-ID>
<RESP-TRANSACTION-ID>0</RESP-TRANSACTION-ID>
<SMSC-ID>airtel</SMSC-ID>
</PARAMETERS>
</RESP>

Synch_Submit_sm packet

Synch_Submit_SM

In synch_submit_sm, the response is synchronized with the request. A sample packet of request, response and the adapter of synch_submit_sm is shown below.

Synch_Submit_sm
<REQ>
<FEATURE>synch_submit_sm</FEATURE>
<TIME-STAMP>09092015184542</TIME-STAMP>
<PARAMETERS>
<REQ-TRANSACTION-ID>2</REQ-TRANSACTION-ID>
<SMSC-ID>airtel</SMSC-ID>
<SUBMIT-SM>
<SOURCE-ADDR-TON>0</SOURCE-ADDR-TON>
<SOURCE-ADDR-NPI>0</SOURCE-ADDR-NPI>
<SOURCE-ADDR>8688492301</SOURCE-ADDR>
<DEST-ADDR-TON>0</DEST-ADDR-TON>
<DEST-ADDR-NPI>0</DEST-ADDR-NPI>
<DESTINATION-ADDR>222222222</DESTINATION-ADDR>
<ESM-CLASS>0</ESM-CLASS>
<concatenatedSMesmClass>0</concatenatedSMesmClass>
<SHORTMESSAGE>SSBjYWxsZWQgWW91IGF0IFdlZCwgOSBTZXAgMjAxNSAxODoxMDo
yNQ==</SHORT-MESSAGE>
<REGISTERED-DELIVERY>1</REGISTERED-DELIVERY>

<DATA-CODING>0</DATA-CODING>
</SUBMIT-SM>
</PARAMETERS>
</REQ>
Synch_submit_response
<RESP>
<STATUS>0</STATUS>
<STATUS-CODE>0</STATUS-CODE>
<TIME-STAMP>23072021202301</TIME-STAMP>
<PARAMETERS>
<REQ-TRANSACTION-ID>2</REQ-TRANSACTION-ID>
<RESP-TRANSACTION-ID>7</RESP-TRANSACTION-ID>
<SMSC-ID>airtel</SMSC-ID>
<MESSAGE-ID>5</MESSAGE-ID>
</PARAMETERS>
</RESP>

MESSAGE PAYLOAD & UDH

UDH Support [Primary Method]
<REQ>
<FEATURE>submit_sm</FEATURE>
<TIME-STAMP>09092015184542</TIME-STAMP>
<PARAMETERS>
<REQ-TRANSACTION-ID>2</REQ-TRANSACTION-ID>
<SMSC-ID>airtel</SMSC-ID>
<SUBMIT-SM>

<SOURCE-ADDR-TON>0</SOURCE-ADDR-TON>
<SOURCE-ADDR-NPI>0</SOURCE-ADDR-NPI>
<SOURCE-ADDR>8688492301</SOURCE-ADDR>
<DEST-ADDR-TON>0</DEST-ADDR-TON>
<DEST-ADDR-NPI>0</DEST-ADDR-NPI>
<DESTINATION-ADDR>222222222</DESTINATION-ADDR>
<ESM-CLASS>64</ESM-CLASS>
<concatenatedSMesmClass>64</concatenatedSMesmClass>
<SHORTMESSAGE>SSBjYWxsZWQgWW91IGFoIFdlZCwgOSBTZXAgMjAxNSAxODoxMDo
yNQ==</SHORT-MESSAGE>
<DATA-CODING>0</DATA-CODING>
</SUBMIT-SM>
</PARAMETERS>
</REQ>

Message Payload [SMSC Support Required]

<REQ>
<FEATURE>submit_sm</FEATURE>

<TIME-STAMP>09092015184542</TIME-STAMP>
<PARAMETERS>
<REQ-TRANSACTION-ID>2</REQ-TRANSACTION-ID>
<SMSC-ID>airtel</SMSC-ID>
<SUBMIT-SM>
<SOURCE-ADDR-TON>0</SOURCE-ADDR-TON>
<SOURCE-ADDR-NPI>0</SOURCE-ADDR-NPI>
<SOURCE-ADDR>8688492301</SOURCE-ADDR>
<DEST-ADDR-TON>0</DEST-ADDR-TON>
<DEST-ADDR-NPI>0</DEST-ADDR-NPI>
<DESTINATION-ADDR>222222222</DESTINATION-ADDR>
<ESM-CLASS>0</ESM-CLASS>
<concatenatedSMesmClass>0</concatenatedSMesmClass>
<SHORTMESSAGE>SSBjYWxsZWQgWW91IGF0IFdlZCwgOSBTZXAgMjAxNSAxODoxMDo
yNQ==</SHORT-MESSAGE>
<DATA-CODING>0</DATA-CODING>
<MESSAGE-PAYLOAD>true</MESSAGE-PAYLOAD>
</SUBMIT-SM>
</PARAMETERS>
</REQ>

ESME MO & Delivery Packets [in smpp.xml]

MO Flow [Connector url]
<XML>
<TIME-STAMP>{0}</TIME-STAMP>
<TRANSACTION-ID>{1}</TRANSACTION-ID>
<SMPP-ID>{2}</SMPP-ID>

<ESM-CLASS>{4}</ESM-CLASS>
<OA>{5}</OA>
<DA>{6}</DA>
<DCS>{7}</DCS>
<SMS>{8}</SMS>
<SMS-BYTE>{9}</SMS-BYTE>
<OPTIONAL-PARAM>{11}</OPTIONAL-PARAM>
<MESSAGE>{17}</MESSAGE>
<MESSAGE-IDENTIFIER>{18}</MESSAGE-IDENTIFIER>
<MESSAGE-PARTS>{19}</MESSAGE-PARTS>
<MESSAGE-PART-NUMBER>{20}</MESSAGE-PART-NUMBER>
</XML>
Sample
<XML>
<TIME-STAMP><![CDATA[20210730194044]]></TIME-STAMP>
<TRANSACTION-ID><![CDATA[74376]]></TRANSACTION-ID>
<SMPP-ID><![CDATA[airtel]]></SMPP-ID>
<OA><![CDATA[4477665544]]></OA>

<DA><![CDATA[337788665522]]></DA>
<DCS><![CDATA[0]]></DCS>
<SMS><![CDATA[Hello from SMPPSim Hello from SMPPSim Hello from SMPPSim
Hello from SMPPSimHello from SMPPSimHello from SMPPSimHello from
SMPPSim]]></SMS>
<SMSBYTE><![CDATA[IEhlbGxvIGZyb20gU01QUFNpbSBIZWxsbyBmcm9tIFNNUFBTaW
0gSGVsbG8gZnJvbSBTTVBQU2ltIEhlbGxvIGZyb20gU01QUFNpbUhlbGxvIGZyb20
gU01QUFNpbUhlbGxvIGZyb20gU01QUFNpbUhlbGxvIGZyb20gU01QUFNpbQ==]]>
</SMS-BYTE>
<OPTIONAL-PARAM><![CDATA[]]></OPTIONAL-PARAM>
<MESSAGE><![CDATA[from SMPPSim Hello from SMPPSim Hello from SMPPSim
Hello from SMPPSim Hello from SMPPSim Hello from SMPPSim]]></MESSAGE>
<MESSAGE-IDENTIFIER><![CDATA[111]]></MESSAGE-IDENTIFIER>
<MESSAGE-PARTS><![CDATA[2]]></MESSAGE-PARTS>
<MESSAGE-PART-NUMBER><![CDATA[1]]></MESSAGE-PART-NUMBER>
</XML>

Delivery Receipt [Delivery connector url]

<REQ>
<FEATURE>DELIVERY-RECEIPT</FEATURE>
<TIME-STAMP>{0}</TIME-STAMP>
<TRANSACTION-ID>{1}</TRANSACTION-ID>
<SMPP-ID>{2}</SMPP-ID>
<COMMAND-ID>{3}</COMMAND-ID>
<ESM-CLASS>{4}</ESM-CLASS>
<OA>{5}</OA>
<DA>{6}</DA>
<DCS>{7}</DCS>
<SMS>{8}</SMS>
<MESSAGE-ID>{10}</MESSAGE-ID>

</REQ>
Sample
<REQ>
<FEATURE>DELIVERY-RECEIPT</FEATURE>
<TIME-STAMP><![CDATA[20210730194217]]></TIME-STAMP>
<TRANSACTION-ID><![CDATA[74378]]></TRANSACTION-ID>
<SMPP-ID><![CDATA[airtel]]></SMPP-ID>
<COMMAND-ID><![CDATA[5]]></COMMAND-ID>
<OA><![CDATA[222222222]]></OA>
<DA><![CDATA[8688492301]]></DA>
<DCS><![CDATA[0]]></DCS>
<SMS><![CDATA[id:1 sub:001 dlvrd:001 submit date:2107301942 done
date:2107301942 stat:DELIVRD err:000 Text:I called You at Wed,]]></SMS>

<MESSAGE-ID><![CDATA[1]]></MESSAGE-ID>
</REQ>

ESME Json Request Packet

Adapter:
Example:
http://localhost:8081/ESME/HttpJsonAdapter

Sample ESME JSON Request packet :

Submit_sm
{
“featureId”:“submit_sm”,
“timeStamp”:25062009103510,
“respUrl”:"http://localhost:8080/ESME/TestServlet",
“parameters”:{
“reqTransactionId”:0,
“smscId”:“airtel”,
“submitSm”:{
“sourceAddrTon”: “0”,
“sourceAddrNpi”: “0”,
“sourceAddr”:1234,
“destAddrTon”: “0”,
“destAddrNpi”: “0”,
“destinationAddr”:11111111,
“esmClass”: “0”,
“concatenatedSMesmClass”: “0”,
“dataCoding”:0,
“shortMessage”:“UkNIUiAyMjIy”
}
}
}
Instant response
{
“Response”: {
“timeStamp”: “23072021104848”,

“status”: “0”
}
}
Final response
{
“status”:“0”,
“statusCode”:“0”,

“timeStamp”:“23072021122642”,
“parameters”:{
“reqTransactionId”:“0”,
“respTransactionId”:“0”,
“smscId”:“airtel”
}
}

Synch_Submit_sm

{
“featureId”: “synch_submit_sm”,
“timeStamp”: “09092015184542”,
“parameters”: {
“reqTransactionId”: “1”,
“smscId”: “airtel”,
“submitSm”: {
“sourceAddrTon”: “0”,
“sourceAddrNpi”: “0”,
“sourceAddr”: “8688492301”,
“destAddrTon”: “0”,
“destAddrNpi”: “0”,
“destinationAddr”: “222222222”,
“esmClass”: “0”,
“concatenatedSMesmClass”: “0”,
“dataCoding”:0,
“shortMessage”:
“SSBjYWxsZWQgWW91IGF0IFdlZCwgOSBTZXAgMjAxNSAxODoxMDoyNQ==”,
“registeredDelivery”: “1”
}
}
}
Synch_submit_response
{
“status”: “0”,
“statusCode”: “0”,
“timeStamp”: “23072021123007”,
“parameters”: {
“reqTransactionId”: “2”,
“respTransactionId”: “1”,
“smscId”: “airtel”,

“messageId”: “38”
}
}

MESSAGE PAYLOAD & UDH
UDH Support Support [Primary Method]


```
{
“featureId”: “synch_submit_sm”,
“timeStamp”: “09092015184542”,
“parameters”: {
“reqTransactionId”: “1”,
“smscId”: “airtel”,
“submitSm”: {
“sourceAddrTon”: “0”,
“sourceAddrNpi”: “0”,
“sourceAddr”: “8688492301”,
“destAddrTon”: “0”,
“destAddrNpi”: “0”,
“destinationAddr”: “222222222”,
“esmClass”: “64”,
“concatenatedSMesmClass”: “64”,
“shortMessage”:
“SSBjYWxsZWQgWW91IGF0IFdlZCwgOSBTZXAgMjAxNSAxODoxMDoyNQ==”,
“registeredDelivery”: “1”
}
}
}
```


Message Payload [SMSC Support Required]


```
{
“featureId”: “synch_submit_sm”,
“timeStamp”: “09092015184542”,
“parameters”: {
“reqTransactionId”: “1”,
“smscId”: “airtel”,
“submitSm”: {
“sourceAddrTon”: “0”,
“sourceAddrNpi”: “0”,
“sourceAddr”: “8688492301”,
“destAddrTon”: “0”,
“destAddrNpi”: “0”,
“destinationAddr”: “222222222”,
“esmClass”: “0”,
“concatenatedSMesmClass”: “0”,
“shortMessage”:
“SSBjYWxsZWQgWW91IGF0IFdlZCwgOSBTZXAgMjAxNSAxODoxMDoyNQ==”,
“messagePayload”:“true”,

“registeredDelivery”: “1”
}
```

}
}

Message Payload (MT)

Question: How can we send large messages (more than 254 octets) from ESME to SMSC in SMPP?
Answer:
To send messages larger than 254 octets, we use Multipart messaging with UDH (User Data Header). The message is split into segments, and each part is sent as a separate SMPP request with a UDH indicating its position in the sequence.

Question: What value must esm_class be set to when sending a multipart message using UDH?
Answer:
Set esm_class to 0x40 (which is 64 in decimal). This signals that the message contains a UDH.

Example:

<ESM-CLASS>64</ESM-CLASS>
<concatenatedSMesmClass>64</concatenatedSMesmClass>
Question: What is the structure of the UDH for multipart messages?
Answer:
A typical UDH for concatenated SMS looks like:
0x05 0x00 0x03 <msg_id> <total_parts> <part_number>
0x05 – Length of UDH
0x00 – Information element identifier
0x03 – Length of the header
<msg_id> – Unique ID for this message
<total_parts> – Total number of parts
<part_number> – The sequence number of this part

Question: How does the short_message field look in a multipart message?
Answer:
Each short_message starts with the UDH bytes followed by the text content for that part.

Example:

Message 1:
esm_class = 0x40
short_message = 0x05 0x00 0x03 0x05 0x02 0x01 Barcelona are to appeal against…

Message 2:
esm_class = 0x40
short_message = 0x05 0x00 0x03 0x05 0x02 0x02 26-year-old Spain midfielder…
Question: What is the maximum number of characters allowed per segment in a concatenated message?
Answer:

Encoding Max Characters per Segment
7-bit (GSM) 153
8-bit 134
16-bit (UCS2) 67
Question: For Multipart message in the esme request what values we need to set or enable?
Answer:
In the request, the esm_class is set to 64, and optionally, a separate tag like concatenatedSMesmClass can also carry the same value to indicate multipart handling.

Example:

<ESM-CLASS>64</ESM-CLASS>
<concatenatedSMesmClass>64</concatenatedSMesmClass>

Message Payload Concept in smpp.xml file

Question: How do we enable the message payload feature in SMPP for sending long messages?
Answer:
To enable the message payload feature (as an alternative to multipart UDH), we must:

Configure the optionalparameterstag in smpp.xml to include the tag for message_payload (0424).

Include extra XML tags in the request from the MO Router to ESME indicating the use of message payload.

This allows the system to send long messages using the message_payload TLV parameter instead of splitting them manually with UDH.

Question: What tag and value are required in the smpp.xml to enable message payload support?
Answer:
You must add the following block inside <optionalparameterstag>:

Example:

```<optionalparameterstag>
<tlv1>
<tag>0424</tag>
<name>message_payload</name>
</tlv1>
</optionalparameterstag>```

Question: What should the MO Router include in its request XML to indicate use of the message payload method?
```

Answer:
The MO Router must set the ESM-CLASS to 0 (no UDH), disable concatenatedSMesmClass, and explicitly enable MESSAGE-PAYLOAD.

Example:

<ESM-CLASS>0</ESM-CLASS>
<concatenatedSMesmClass>0</concatenatedSMesmClass>
<MESSAGE-PAYLOAD>true</MESSAGE-PAYLOAD>
Question: What does setting <MESSAGE-PAYLOAD>true</MESSAGE-PAYLOAD> achieve?
Answer:
It instructs the ESME to send the complete message as a single unit using the message_payload TLV, rather than breaking it into multiple segments. This method is cleaner and avoids message part headers like in UDH.

Multiple SMSC Support

Question: ESME will support Multiple SMSC connection?
Answer:
Yes,Multiple SMSC Support allows the ESME to connect with two or more SMSC accounts simultaneously. Each account is independently configured in the smpp.xml file using separate <smsc> blocks, enabling features like load distribution, redundancy, and differentiated routing.

Question: What is the purpose of smsc id in the config?
Answer:
The smsc id uniquely identifies each SMSC configuration block within the smpp.xml. For instance, you may have:

<smsc id="SMSC1" ... >
...
</smsc>
<smsc id="SMSC2" ... >
...
</smsc>
Question : Can each SMSC connection have different settings?
Answer:
Yes. Each SMSC connection can have its own:
IP address and port
System ID and password
Bind mode (t, r, or tr)
TPS (transactions per second)
Queue size
Delivery settings
Encoding, timeout values, etc.
Example:
<smsc id="SMSC1" protocol="smpp">

<ip-address>127.0.0.1</ip-address>
<port>8056</port>
<system-id>j</system-id>
<password>jpwd</password>
<bind-mode>tr</bind-mode>

Question: How is the bind mode defined for different SMSCs?
Answer:
The <bind-mode> tag is used to define how ESME connects to an SMSC. Possible values:

t → Transmitter (sending only)
r → Receiver (receiving only)
tr → Transceiver (both send and receive)
Example:
<bind-mode>tr</bind-mode>

Question: What is a sample configuration for supporting two SMSCs?
Answer:
Sample:
Here's a simplified structure for two SMSCs:

<config>
<smsc id="SMSC1" protocol="smpp">
<ip-address>127.0.0.1</ip-address>
<port>8056</port>
<system-id>j</system-id>
<password>jpwd</password>
<bind-mode>tr</bind-mode>
...
</smsc>
<smsc id="SMSC2" protocol="smpp">
<ip-address>127.0.0.1</ip-address>
<port>8056</port>
<system-id>j</system-id>
<password>jpwd</password>
<bind-mode>tr</bind-mode>
...
</smsc>
</config>
Question: What happens if <load-from-db> is set to true?
Answer:
If <load-from-db> is set to true, the system loads SMSC connection details from a database instead of the XML file. By default, it is false.

Retry Feature

how to enable the retry feature in Esme?
Step 1: Update web.xml to enable Retry
Location:
$WILDFLY_HOME/standalone/deployments/Esme.war/WEB-INF/web.xml

What to change:
Find the servlet with <servlet-name>Applicationconfig</servlet-name> and update this line:

<param-name>Retry</param-name>
<param-value>N</param-value>
Change it to:
<param-name>Retry</param-name>
<param-value>Y</param-value>

Step 2: Modify Retry Configuration Properties
In your retry configuration properties file (usually retry_config.properties or similar), set the following:

synchSubmitSm = retrySubmitSm
roundRobinSupport = false
synchSubmitSm=retrySubmitSm: Switches the submission logic to support retry.
roundRobinSupport=false: Ensures retry goes to the same SMSC instead of rotating.

Question: Does ESME support message retry if the SMSC is temporarily unavailable or the message fails to send?
Answer:
Yes, ESME supports a retry mechanism. When enabled, the system will attempt to resend the message if the initial attempt fails, improving delivery success rate.

Question: How do you enable the retry feature in ESME configuration?
Answer:
To enable retry, two main configuration changes are required:

Update the Retry flag in web.xml.

Modify values in the retry config properties.

Example (1) — Update web.xml:

Change the Retry <param-value> from N to Y:

```<servlet>
<servlet-name>Applicationconfig</servlet-name>
<servlet-class>com.sixdee.imp.common.config.ContextLoaderServlet</servlet-class>
<init-param>
```

<param-name>Retry</param-name>
<param-value>Y</param-value>
</init-param>
<init-param>
<param-name>SmscDetailLoader</param-name>
<param-value>N</param-value>
</init-param>
<load-on-startup>2</load-on-startup>
</servlet>```
Location: $WILDFLY_HOME/standalone/deployments/Esme.war/WEB-INF/web.xml

Example (2) — Retry Config Properties:

Set the following in the retry config:

synchSubmitSm = retrySubmitSm
roundRobinSupport = false
Question: What does setting roundRobinSupport to false mean?
Answer:
Setting roundRobinSupport=false disables round-robin SMSC selection during retry. Instead, the retry logic will attempt to resend the message using the originally assigned or prioritized SMSC.

ELK support

Question:
How does ESME support ELK logging?

Answer:
ESME supports logging into the ELK stack (Elasticsearch, Logstash, Kibana) by adding a <logs> configuration in the Smpp.xml file. This enables structured logging in a format that ELK can parse and visualize. Logstash collects and processes the logs, Elasticsearch stores them, and Kibana provides the visual interface for monitoring and analysis.

Example:
Below is a sample configuration added to the Smpp.xml file for ELK logging:

```
 <formatPattern><![CDATA[

    TimeStamp::{1,date,yyyy-MM-dd'T'HH:mm:ss XXX}||

    ClientTransactionId::{3}||
```

```
    MSISDN::{15}||

    TransactionId::{2}||

    OfferCode::0||

    TransactionDate::{4}||

    statusCode::{5}||

    Feature::{6}||

    SmscId::{0}||

    commandStatus::{7}||

    SeqNo::{8}||

    ServiceType::{9}||

    SourceAddrTon::{10}||

    SourceAddrNpi::{11}||

    DestAddrTon::{13}||

    DestAddrNpi::{14}||

    SourceAddr::{12}||

    EsmClass::{16}||

    ProtocolId::{17}||

    RegisteredDelivery::{21}||

    DataCoding::{23}||

    ShortMessageHex::{25}||

    ShortMessage::{32}

  ]]></formatPattern>
```

```
</logs>'''
```

```
This format ensures that each important data point from a transaction
(like MSISDN, SmscId, EsmClass, etc.) is captured and structured in the
logs, ready to be shipped to Logstash and visualized in Kibana.
```

Question:
What are the common ESME response error codes and their meanings?

Answer:
ESME defines a range of error codes to identify specific issues during processing. These error codes help in troubleshooting failed transactions, malformed requests, or system issues.

Example:

ErrorCode ErrorDesc
-1001 Unknown feature or Illegal feature
-1002 Overload rejection
-1003 Internal server error (check feature_id)
-1004 Unknown Operation or Exception
1000 Illegal request Message
1005 Mandatory field missing
1006 Illegal data type
1007 Username or password does not match
1008 Other Error (e.g., failed to send response)
1009 Unknown CommandId
1010 Execute Exception
1011 CMD_FIELD_MISSING_ERRORCODE
1012 UNKNOWN_COMMANDID_ERRORCODE
1013 UNKNOWN_SMSC_ERRORCODE
1014 NOT_BOUND_WITH_SMSC_ERRORCODE
1015 ALREADY_BOUND_WITH_SMSC_ERRORCODE
1016 IOEXCEPTION_ERRORCODE
1017 NEGATIVE_RESPONSE_ERRORCODE
1018 RESPONSE_TIMEOUT_ERRORCODE
1019 UNSUC_DELIVERY_SME_ERRORCODE
1020 THROTTLING_LIMIT_ERRORCODE
Question:
How would you interpret error code 1005?

Answer:
Error code 1005 indicates a Mandatory field is missing in the request payload. The request should be reviewed to ensure all required XML tags or fields are correctly included.

Example:
If a <TRANSACTION-ID> tag is missing from a delivery request, the system may respond with error 1005.